A PATTERN-BASED RELEASE PLANNING METHODOLOGY FOR MARKET-DRIVEN SOFTWARE

AMIR SEYED DANESH

THESIS SUBMITTED IN FULFILMENT
OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2016

UNIVERSITY MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: **AMIR SEYED DANESH**

Registration/Matric No: **WHA070031**

Name of Degree: **DOCTOR OF PHILOSOPHY**

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"):

A PATTERN-BASED RELEASE PLANNING METHODOLOGY FOR MARKET-DRIVEN SOFTWARE

Field of Study: **SOFTWARE RELEASE PLANNING**

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor ought I reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date

Subscribed and solemnly declared before,

Witness's Signature Date

Name:

Designation:

ABSTRACT

In software development, release planning is performed in order to select important features and requirements based on resource and technical constraints and the relationships between requirements. Several methods have been developed for release planning, but the most challenging part of release planning methods is dealing with unique complexities and varying characteristics of problem scope and domain. This has made the existing methods and approaches only applicable to a limited range of software projects. As a result, a more flexible and adaptive approach is required for release planning that can be customized in accordance to different domains and can be employed in a variety of software development projects. Achieving a highly customized release planning methodology requires an effective planning process that covers all release planning tasks and can be customized according to project specifications for each parameter. This study aims to, firstly, assemble a general and comprehensive process for release planning which covers all the important tasks and, secondly, identify parameters which can be used to customize the process for various projects, and thirdly introduce and apply release planning patterns based on process model steps and the identified parameters to facilitate customization. Thus, available release planning approaches have been studied and four common tasks i.e. requirements prioritization, resource estimation, release pre-planning, and trade-off analysis were identified and assembled in the form of a process model. Subsequently, various relevant parameters for each step which are related to project specifications were identified. In order to customize each step of the process model for various projects, each parameter was determined precisely and its current instances were identified. Some of the parameters are shared between process model steps. The notion of "pattern" was employed in order to facilitate the customization of the steps of the process model and several patterns were identified. Every release planning pattern has constraints based upon parameter

instances of the step of process model, and suggests a solution as the selected method to apply to the step. Using this notion, the release planner only has to select a release planning pattern based on selected parameters that suit his circumstance. To validate the proposed methodology, two methods are used. At first, five software companies with 31 projects were used to implement method to their projects. The companies were asked to apply the pattern-based methodology to at least two releases and, for each step of the methodology, apply PRP tool suggested method. Secondly, numbers of 13 experts in the software development domain are answered questions in a survey on the results of the methodology and their satisfaction. Results showed that in most cases -more than 87% of the cases, the methodology suggested by the pattern release planner produce better releases and make the release planning process easier and faster than those used previously.

ABSTRAK

Dalam pembangunan perisian, perancangan keluaran dilakukan untuk memilih ciri dan keperluan yang penting berdasarkan kekangan sumber dan teknikal serta perkaitan antara keperluan-keperluan. Beberapa metod telah dibangunkan untuk perancangan keluaran tetapi tugas paling mencabar bagi metodologi perancangan keluaran ialah menangani kerumitan yang unik dan pelbagai ciri skop dan domain masalah. Ini menjadikan metodologi sedia ada hanya boleh digunakan kepada projek perisian yang terbatas. Akibatnya, metod yang lebih adaptif dan anjal diperlukan untuk perancangan keluaran yang boleh di selaraskan kepada domain yang berbeza-beza dan digunakan dalam pelbagai projek pembangunan perisian. Dalam keberhasilan metodologi perancangan keluaran yang beradaptasi tinggi memerlukan proses perancangan yang berkesan yang mengandungi semua tugas perancangan keluaran dan boleh diadaptasikan berdasarkan spesifikasi projek untuk setiap satu parameter. Kajian ini bermatlamat untuk pertamanya; menghimpunkan satu proses yang umum dan komprehensif yang mencakupi semua tugas tugas penting bagi perancangan keluaran, keduanya; mengenalpasti parameter parameter yang boleh digunakan untuk mengadaptasikan proses perancangan keluaran bagi pelbagai projek and ketiganya; memperkenalkan dan menggunapakai pattern berdasarkan peringkat peringkat proses umum dan parameter-parameter yang dikenalpasti untuk membantu adaptasi. Oleh itu, metodologi perancangan keluaran sedia ada telah dikaji dan empat tugas am iaitu pengutamaan keperluan, penganggaran sumber, pra-perancangan keluaran dan analisis timpal-balik telah dikenalpasti dan dihimpunkan dalam bentuk proses umum. Kemudian, berjenis-jenis parameter penting bagi setiap peringkat yang berkaitan dengan spesifikasi projek telah dikenalpasti. Untuk mengadaptasi setiap peringkat bagi proses umum untuk pelbagai projek, setiap parameter ditentukan dengan tepat dan contoh terkini dikenalpasti. Pengertian "pattern" digunakan untuk memudahkan adaptasi setiap

peringkat bagi proses umum itu dan beberapa pattern telah ditakrifkan. Takrifan pattern dikemukakan menggunakan contoh parameter yang ditentukan dalam peringkat adaptasi dan metod perancangan keluaran untuk perlaksanaan setiap peringkat dicadangkan. Dalam kata lain, setiap pattern perancangan keluaran mengandungi kekangan kekangan berdasarkan contoh parameter bagi setiap peringkat dan mencadangkan penyelesaian iaitu metod untuk perlaksanaan peringkat tersebut. Menggunakan idea ini, perancang keluaran cuma perlu memilih pattern perancangan keluaran berdasarkan parameter yang sesuai dengan keadaannya. Untuk mengesahsahihkan kaedah yang dicadangkan, dua metod digunakan. Pertamanya, lima syarikat perisian dengan 31 projek digunakan untuk melaksanakan kaedah tersebut untuk projek-projek mereka. Syarikat-syarikat tersebut diminta untuk menggunakan kaedah 'pattern-based' untuk sekurang-kurangnya dua lepasan dan untuk setiap langkah kaedah , menggunakan metod yang dicadangkan oleh alatan PRP. Keduanya, 13 pakar dalam domain pembangunan perisian diminta untuk menjawab set soalan tinjauan tentang kepuasan mereka menggunakan kaedah tersebut. Keputusan menunjukkan yang dalam kebanyakn kes, lebih 87% kes, metod yang dicadangkan oleh PRP menghasilkan lepasan yang lebih baik dan menjadikan proses perancangan lepasan lebih mudah dan cepat daripada proses yang tidak menggunakan cadangan daripada PRP.

ACKNOWLEDGEMENT

First, I would like to thank Dr. Rodina Ahmad, with whom I have enjoyed working and who is a great advisor. Thank you for your encouragement, support, and directions you have provided during the past four years. I have been very grateful to have you as a counsellor and advisor.

I am deeply grateful to my wife, Mahgol, for her on-going moral support, and acceptance of my long hours away from our family. Thank you so much for your constant encouragement and never-ceasing patience.

I wish to thank my parents for the encouragement and for providing me with so many opportunities to progress in academics. Thank you for the invaluable support you have given me in the course of my life and studies.

My special thanks go to my friends in the Faculty of Computer Science and Information Technology (FSKTM) at University of Malaya for the helpful discussions we had about my project.

Kuala Lumpur, 8th September

Amir Seyed Danesh

TABLE OF CONTENTS

Abstract iii

Abstrak v

Acknowledgement vii

Table of Contents viii

List of Figures xiii

List of Tables xvi

CHAPTER 1: INTRODUCTION 1

1.1. Background and motivation 1

1.2. Problem statements 3

1.3. Research objective 7

1.4. Research Methodology 9

1.4.1. A review on literature 10

1.4.2. Developing the new release planning methodology 11

1.4.3. Developing the tool 12

1.4.4. Validating the methodology by case studies 13

1.4.5. Presenting results 14

1.5. Research contribution 15

1.6. Results 16

1.7. Thesis structure 17

CHAPTER 2: LITERATURE REVIEW 19

2.1. Introduction 19

2.2. Release planning methods 20

2.2.1. Ad-Hoc Approach 20

2.2.2. Planning Game Approach 21

2.2.3. Incremental Funding Method (IFM) 22

2.2.4. Optimization-Based Techniques 23

2.2.5. Hybrid Intelligence Approach 23

2.2.6. Lightweight Re-planning 24

2.3. Release planning researches 25

2.4. Requirements prioritization methods 30
2.4.1. Nominal Scale 33
2.4.2. Ordinal Scale 34
2.4.3. Ratio Scale 36
2.4.4. Compound techniques 38
2.5. Resource estimation methods 39
2.5.1. Non-algorithmic methods 42
2.5.2. Algorithmic methods 43
2.6. Software development patterns 46
2.7. Summary 50
CHAPTER 3: SOFTWARE RELEASE PLANNING IN INDUSTRY 53
3.1. Overview 54
3.2. Software release planning challenges in software development 55
3.2.1. Related Work 57
3.2.2. Research Design 59
3.2.3. Software Projects 62
3.2.4. Challenges identification 66
3.2.5. Challenges identified in industry 70
3.2.6. Discussion and validity of results 82
3.2.7. Conclusion 85
3.3. Companies' Approaches in Software Release Planning 86
3.3.1. Companies and software releases 86
3.3.2. Objective 88
3.3.3. Research design 88
3.3.4. Results 90
3.3.5. Findings and discussions 94
3.3.6. Discussion on the findings 100
CHAPTER 4: SOFTWARE RELEASE MANAGEMENT IN INDUSTRY 101
4.1. Overview 101
4.2. Software release management 101
4.3. Objective 104
4.4. Research Method 104

4.4.1. Research question 105
4.5. Challenges in industry 106
4.5.1. Categorization of releases 106
4.5.2. The need for some support tool in release management 107
4.5.3. Appropriate tools 108
4.5.4. Foreseeing a new release before real execution 109
4.5.5. Release manager's role 109
4.5.6. Proper understating of Request for Changes (RFC) 110
4.5.7. Release policy 111
4.6. Threats to validity 111
4.7. Summary and conclusion 114
CHAPTER 5: DESCRIPTION OF THE PATTERN-BASED RELEASE PLANNING METHODOLOGY 115
5.1. Introduction 115
5.2. The process model of release planning 115
5.2.1. Requirements prioritization 121
5.2.2. Resource estimation 123
5.2.3. Pre-release planning 124
5.2.4. Analysis of pre-release plan and selecting the final release 127
5.2.5. The process model of release planning 127
5.3. Release planning process model customization 130
5.3.1. Customization of requirements prioritization 130
5.3.2. Customizing resource estimation 150
5.3.3. Customization of pre-release plan 160
5.4. Pattern-based release planning methodology 175
5.4.1. Definition of release planning patterns 178
5.4.2. The structure of release planning patterns 179
5.4.3. Development method of release planning patterns 181
5.4.4. Algorithms of using release planning patterns 181
5.5. Release planning patterns 188
5.5.1. Requirements prioritization patterns 188
5.5.2. Resource estimation patterns 197
5.5.3. Patterns of pre-release planning 204

5.5.4. Release planning patterns 210
5.6. Release planning anti-patterns 218
5.6.1. Anti-patterns of requirements prioritization 220
5.6.2. Anti-patterns of resource estimation 222
5.6.3. Anti-patterns of pre-release planning 225
5.7. Summary 227
CHAPTER 6: EVALUATION OF THE METHODOLOGY 228
6.1. Introduction 228
6.2. Evaluation Objectives 228
6.3. Evaluation method 229
6.3.1. Case studies evaluation method 230
6.3.2. Experts review evaluation method 232
6.4. Evaluation Success Factors 233
6.5. Case studies Selection 234
6.5.1. Company A description 235
6.5.2. Company B description 237
6.5.3. Company C description 239
6.5.4. Company D description 241
6.5.5. Company E description 244
6.6. Effective parameters in case studies 245
6.7. Case studies patterns usage 252
6.8. Experts Demography 254
6.9. Summary 255
CHAPTER 7: EVALUATION RESULTS 256
7.1. Introduction 256
7.2. Requirements prioritization pattern evaluation 256
7.3. Resource estimation pattern evaluation 261
7.4. Pre-release planning pattern evaluation 266
7.5. Release planning pattern evaluation 272
7.6. Overall evaluation results 277
7.7. Summary of evolution results 279

CHAPTER 8: RESULTS AND FUTURE WORK 280

8.1. Achievement objectives 280

8.2. Research findings and contribution 280

8.3. Research executive constraints 284

8.4. Future works 284

References 285

Appendix A: Questionaries 296

Appendix B: Pattern-based release planning methodology questionaries 298

Appendix C: Pattern Release Planning (PRP) tool 299

LIST OF FIGURES

Figure 1-1: Main steps of the research 10
Figure 2-1: Planning software releases 19
Figure 2-2: Generic process model for the Light-Weight Re-planning 25
Figure 2-3: Computation of Function Point 41
Figure 3-1: The Research Procedure 60
Figure 3-2: Challenges during the discussion, interview and meeting 69
Figure 3-3: Categorization of the challenges 83
Figure 3-4: Categorization by view of planning 95
Figure 3-5: Tendency of companies to re-planning and new planning 97
Figure 3-6: Grouping human resources 98
Figure 3-7: Tendency of companies to human or systematic approaches 99
Figure 4-1: Overview of release planning process 102
Figure 4-2: Respondents 108
Figure 4-3: Level of importance 113
Figure 5-1: Taxonomy of requirements selection factors 128
Figure 5-2: Inputs, outputs and activities in the release planning process model 130
Figure 5-3: Relations between requirements prioritization parameters instances in tool .. 149
Figure 5-4: Relations between resource estimation parameters instances in tool 159
Figure 5-5: Relations between pre-release planning parameters instances in tool 174
Figure 5-6: Relationships between release planning process model and the method .. 176
Figure 5-7: Using pattern in release planning 177
Figure 5-8: Requirements prioritization pattern for large projects 189
Figure 5-9: Requirement prioritization pattern with medium level of requirements ... 192
Figure 5-10: Requirements prioritization pattern with huge number of customers 194
Figure 5-11: Requirements prioritization pattern for small projects 196
Figure 5-12: Resource estimation pattern in large projects 199
Figure 5-13: Resource estimation pattern for projects with unlimited customers 201
Figure 5-14: Resource estimation pattern in small projects 203
Figure 5-15: Release planning pattern in large projects 205
Figure 5-16: Pre-release planning pattern with large number of customers 207
Figure 5-17: Pre-release planning pattern with single-variable methods 209
Figure 5-18: Release planning pattern in large projects 212
Figure 5-19: Release planning pattern with large number of customers 215
Figure 5-20: Release planning pattern in small projects 217
Figure 6-1: Method of peforming case studies 232
Figure 6-2: Development environment in studied projects 246

Figure 6-3: Development methodology in studied projects 246
Figure 6-4: Input and output volume in studied projects 247
Figure 6-5: Market type in studied projects 247
Figure 6-6: Requirements prioritization input number in studied projects 248
Figure 6-7: Team size in studied projects 248
Figure 6-8: Number of release planning parameters in studied projects 249
Figure 6-9: Requirements level in studied projects 250
Figure 6-10: Requirements number in studied projects 250
Figure 6-11: Team experience in studied projects 251
Figure 6-12: Team size in studied projects 252
Figure 6-13: Usage of different patterns in the studied projects 253
Figure 7-1: Results of using requirements prioritization patterns in the projects 256
Figure 7-2: Experts' reviews for the requirements prioritization patterns 257
Figure 7-3: Results of using PR1 in the projects 258
Figure 7-4: Experts' reviews for PR1 pattern 258
Figure 7-5: Results of using PR2 in the projects 259
Figure 7-6: Experts' reviews for PR2 pattern 259
Figure 7-7: Results of using PR3 in the projects 260
Figure 7-8: Experts' reviews for PR3 pattern 261
Figure 7-9: Results of using resource estimation patterns in the projects 262
Figure 7-10: Experts' reviews for the resource estimation patterns 262
Figure 7-11: Results of employing PE1 in the projects 263
Figure 7-12: Experts' reviews for PE1 pattern 263
Figure 7-13: Results of employing PE2 in the projects 264
Figure 7-14: Experts' reviews for PE2 pattern 265
Figure 7-15: Results of employing PE3 in the projects 265
Figure 7-16: Experts' reviews for PE3 pattern 266
Figure 7-17: Results of using pre-release planning patterns in the projects 267
Figure 7-18: Experts' reviews for the pre-release planning patterns 267
Figure 7-19: Results of employing PP1 in the projects 268
Figure 7-20: Experts' reviews for PP1 pattern 269
Figure 7-21: Results of employing PP2 in the projects 269
Figure 7-22: Experts' reviews for PP2 pattern 270
Figure 7-23: Results of employing PP3 in the projects 271
Figure 7-24: Experts' reviews for PP3 pattern 271
Figure 7-25: Results of using release planning patterns in the projects 272
Figure 7-26: Experts' reviews for the release planning patterns 273

Figure 7-27: Results of using P1 in the projects..... 273
Figure 7-28: Experts' reviews for P1 pattern..... 274
Figure 7-29: Results of using P2 in the projects..... 274
Figure 7-30: Experts' reviews for P2 pattern..... 275
Figure 7-31: Results of employing P3 in the projects 276
Figure 7-32: Experts' reviews for P3 pattern..... 276
Figure 7-33: Results of employing patterns in the projects 277
Figure 7-34: Experts' reviews for the patterns 278
Figure 7-35: Experts' reviews for the methodology 278

LIST OF TABLES

Table 1-1: Main inputs, activities and outputs in reviewing literature 11
Table 1-2: Main inputs, activities and outputs in developing methodology 12
Table 1-3: Main inputs, activities and outputs in tool development 13
Table 1-4: Main inputs, activities and outputs in validating the method 14
Table 2-1: Essential scales used for AHP 37
Table 2-2: Categorization of algorithmic methods 44
Table 3-1: Summary of the conducted interviews 61
Table 3-2: Projects and their application areas 66
Table 3-3: Participants in the survey 89
Table 3-4: Companies characterized 90
Table 3-5: Individual's responses in companies 96
Table 3-6: Individual's responses in companies 99
Table 4-1: The responders' numbers 105
Table 4-2: A Questioner sample 112
Table 5-1: Steps of implementing release planning methodologies 117
Table 5-2: Classification of prioritization methods 133
Table 5-3: Comparison parameters of requirements prioritization methods 134
Table 5-4: Parameters of requirements prioritization 135
Table 5-5: Effect of instances on the requirements prioritization method 137
Table 5-6: Relations between "market type" instances and other parameters 140
Table 5-7: Relations between "development methodology" and other parameters 142
Table 5-8: Relations between "team size" and other parameters 144
Table 5-9: Relations between "requirements number" and other parameters 145
Table 5-10: Relations between "requirements level" and other parameters 146
Table 5-11: Relations between "number of prioritization inputs" and other parameters 147
Table 5-12: Relations between "team experience" and other parameters 148
Table 5-13: Classification of different resource estimation methods 152
Table 5-14: Characteristics of resource estimation method 153
Table 5-15: Effective parameters on resource estimation 154
Table 5-16: Effect of parameters and instances on resource estimation method 155
Table 5-17: Relations between "development methodology" and other parameters 156
Table 5-18: Relations between "requirements number" and other parameters 158
Table 5-19: Relations between "team experience" and other parameters 158
Table 5-20: Characteristics of release planning methods 164
Table 5-21: Effective parameters on pre-release planning 165
Table 5-22: Effect of parameters and instances on pre-release planning method 166

Table 5-23: Relations between "market type" and other parameters..... 168
Table 5-24: Relations between "development methodology" and other parameters.... 170
Table 5-25: Relations between "project size" and other parameters 171
Table 5-26: Relations between "requirements number" and other parameters 172
Table 5-27: Relations between "number of plan generation parameters" and other parameters..... 173
Table 6-1: Characteristics of projects in company **A** 236
Table 6-2: Characteristics of projects in Company B..... 238
Table 6-3: Characteristics of projects in Company C..... 241
Table 6-4: Characteristics of projects in Company D..... 243
Table 6-5: Characteristics of projects in Company E..... 245
Table 6-6: Pattern codes..... 252
Table 6-7: Expert demographic data..... 254

CHAPTER 1: INTRODUCTION

1.1. Background and motivation

In two recent decades, software development processes and methodologies have been significantly enhanced and from among the new activities introduced to the software development teams, release planning is the most important. The history of release planning originates in the old concept of version in classic software development methodologies(Sommerville, 2010). In the waterfall model, that is the best known classic methodology in software development, requirements, design, implementation, and verification were performed sequentially, and the whole software was its final output. The output was then known as a version of the software and every accomplishment of this classic cycle resulted in a newer version(Sommerville, 2010). By introducing the 'iteration' concept and iterative development, the classic waterfall methodology was replaced by newer methodologies and there was no necessity to develop a whole final output every time the cycle was over. Thereafter the concept of 'release' was introduced into software development.

A release in software development domain is known as a product output that meets certain features and requirements determined by customers' demands. In most cases, releases are developed iteratively and incrementally and finally lead to a software version or an external release for customers. In each release, new features and requirements are inserted. In current software development methodologies, numerous releases are developed and the specifications of each release should be elaborated; hence, release planning is considered as a key task in software development. Briefly, it is selecting a series of features and requirements in sequential time period, considering resources and technical constraints(A. G. Jadallah, 2010). The gradual increase in the size and complexity of software systems adds to the importance of release planning, and

it is considered as a main task at the beginning of every software project. Release planning is a trade-off analysis task due to the need for continuous changes to later releases of software which meet customers' needs and are effective regarding the development time and costs. Hence, determining which features need to be considered in which release has been called a wicked problem by many experts(A. G. Jadallah, 2010). A worthy release plan can easily wipe out weeks or even months of activities of a team, impose heavy expenditures on developers and make teams exhausted.

Release planning starts by emerging new features or requirements, and then a project manager or domain expert estimates the time and cost required for each one to prioritize them and finally selects requirements with higher priorities in accordance with technical and human resource constraints to generate a release. Some release planning methods consider the score of requirements from users' viewpoint and also the relationship between different requirements since they may be interdependent. Sorted requirements based on one or more parameters are selected until budget, time or human resource constrains inhibit adding a new requirement. This set of requirements forms a release with specific time and costs.

Generally, there are two main methods for release planning based on Ruhe's views which are presented in (Ruhe & Saliu, 2005a). The first is the manual method -art of release planning- which, in fact, relies on human judgment and is used when small numbers of features are available. Individuals make decisions on release features through negotiations and meetings. However, considering the increasing number of features and users, it is difficult to rely on manual methods to generate proper solutions (Dybå & Dingsøyr, 2008). The second method is the hybrid release planning which relies on both human and computational intelligence to systematically generate release planning solutions. In recent years, various formal models, such as Planning Game (Ruhe & Saliu, 2005b), Incremental Funding Method (IFM) (Denne & Cleland-Huang,

2004), optimization-based techniques (Bagnall, Rayward-Smith, & Whittley, 2001), Hybrid Intelligence approach (Przepiora, Karimpour, & Ruhe, 2012; G. Ruhe & A. Ngo, 2004; Saliu & Ruhe, 2005a), and Lightweight Re-planning (AlBourae, Ruhe, & Moussavi, 2006), have been developed in which systematic methods are able to present several alternative solutions. Reading through several of these models mentioned in Stahlberg's survey (Svahnberg et al., 2010), one notices that the process is getting more and more systematic. But Mohebzada (Mohebzada, 2012) proposes a recommendation system for release planning in which human play the main role besides the presented systematic method and the recommendation system assists the product manager or release planner.

Most release planning methods present various releases in order to select the best one. Generally, choosing the final release plan is a selection among multiple plans in which management viewpoints are more effective than technical parameters. On the other hand, it should be emphasized that release planning cannot be a totally automatic task and primary values and parameters based on which the task is performed are entered by users and experts.

1.2. Problem statements

As part of any incremental and iterative software development, the question which new features should be offered in upcoming releases is of key importance for product success (Felderer, Beer, Ho, & Ruhe, 2014). The most important consideration in release planning is to find the best set of features and requirements to generate releases and release plans. The definition of the best set is different in every project and depends upon the project's characteristics. This definition leads into a description of the best release plan for every certain project. The best release plan for a project is one which, for example, covers most requirements in every release, distributes costs equally in all releases, has the optimal distribution of the human resources in all releases, or is a

combination of these or other cases. Product release decisions (what to release?, when to release?, how good is the release?) are inherently complex because of their comprehensive information needs, the diversity of criteria, the variety of stakeholders involvement, and the presence of all types of resource and dependency constraints that have to be taken into(Ho, Shahnewaz, & Ruhe, 2014). Therefore, generating different release plans is not the main goal of release planning; it is rather to generate the best plans based on the project's characteristics or features considered by the project manager.

Despite the presence of incremental and iterative methods, the release planning process is complicated due to the possible influence of various factors such as: types of requirements, implementation strategies, value for the developing company, urgency for the client, risk management and personal decisions (Ruhe, 2005). In addition, deciding on features to be included in a specific release is a complicated task and is referred to as a wicked problem which is difficult to clearly define and often has no clear-cut solution (Carlshamre, 2002; Rittel & Webber, 1984). Bagnall *et al.* (Bagnall et al., 2001) discuss release planning factors and show that the problem of selecting an optimal next release is NP-hard. Svahnberg (Svahnberg et al., 2010) studies release planning methods and parameterizes the process. The author introduces some requirement selection factors and concludes that it may be difficult to find a release planning model that suits a company's needs while addressing the desired requirements selection factors. Moreover, a fully systematic approach is never sufficient and needs to be combined with the experience of professional practitioners. For this reason, Ruhe and Saliu (Ruhe & Saliu, 2005a) have discussed release planning from two different dimensions, art and science. The art of release planning refers to human and his capabilities while the science refers to the algorithms and methods.

Various studies are performed on the field of release planning trying to create the best release plans. Based on literature, the most important weaknesses of these methods include:

- **Lack of attention to project characteristics**

  One of the most significant aspects neglected in most release planning methods is paying attention to project characteristics. Release planning methods and approaches usually notice requirements and their related features such as priority, time and costs and do not attend characteristics of the project or the project's team. As an example, suppose suggesting AHP method (Perini, Susi, Ricca, & Bazzanella, 2007) for an agile team with less than 5 members and a huge number of requirements. Generally, human resource is the great issue for agile teams and they prefer to use rapid and effective methods. However, the huge number of requirements makes the use of manual methods impossible and thus the optimal method of release planning should be specified according to the team and project characteristics. Methods that focus on cost and resource optimization, in a different way, cannot be used in every project since they are not solely focusing on projects and are performed in respect to other features(Ngo-The & Ruhe, 2009). It must be noted that project and team features are sometimes opposite, such as in the above-mentioned example of a team with a small number of members who are supposed to deal with a huge number of requirements. Insufficient attention to this issue results a method which is impractical or encounters many problems during implementation.

- **Weak customization**

  In its simplest form, customization of release planning methods is the capability of the method to be parametric. Although most presented methods have tried to be parameterized, the parametric form was only used to display the problems and

used in problem areas, not to present different solutions. In other words, parameterization does not affect the solution selection and is only used to show different parameters of the problem. To be effective, the users are expected to reach other solutions or proposed methods through changing the inputs. This issue has not been addressed in most studies and the only research dealing with this is (Slooten, 2012), which is actually not about presenting a release planning method and presents a framework to evaluate the maturity of a release planning process. Therefore, most methodologies only emphasize the method they present and do not care about the variety of input parameter values, for example, when the number of requirements changes. In case of parameter variation, the release planning method must take it into consideration and make possible customizations while most methods cannot be customized and only look at the method regardless of input parameters values. Hence, parameter variety does not influence the concluding method and is not used in the development of the solution. In addition, project characteristics should be considered in the method customization and the concluding solution presented by that method.

- **Lack of attention to problem domain complexity**

Release planning, like most tasks in software engineering, is an executive task that involves different, and sometimes unknown, parameters and hence creates various practical complications. Identifying these parameters and their roles help to recognize release planning process better and, hence, the presented solution will be more tended towards reality. In most existing release planning methods, only the requirements prioritization and resource constraints are considered and therefore, lack of real parameters makes the planning problem simplistic and the presented solution not practical. (Ruhe, 2005) and (Marjaie & Kulkarni, 2010) investigated effective parameters on release planning and hidden parameters of

requirements' prioritization, which is a main step in release planning. Parameters such as the risk of implementing every requirement, requirement difficulty degree and requirement repetition rate in users' demands are those neglected in most research and thus the presented methods are not practical (to some extent) and software developing teams prefer to use simple and manual methods. This is one of the major challenges facing release planning (Seyed Danesh & Ahmad, 2012).

Weaknesses of current release planning methods make many of them inapplicable in various environments. Besides, differences of software projects and their characteristics make these methods unsuitable for various project types. Therefore, a release planning methodology is needed which removes or decreases the weaknesses mentioned above while being applicable in various projects.

1.3. Research objective

Complications and difficulties of available release planning methods mentioned earlier led the present research toward developing an inclusive, highly customizable methodology for a release planning to be used in various projects. Thus, this research seeks a release planning methodology based on effective parameters in every project with the use of successful experiences in this field. The leading objective of this research is to present a release planning methodology to be used in a variety of project types. Considering the main objective, following goals are met throughout this thesis:

- Developing a process model for Release Planning in order to cover the procedures and tasks involved in current methods and approaches. This process model is based on the common steps in current methodologies.

- Customizing every step of the process model by identifying parameters, objectives, inputs, outputs, and the relation between them in order to increase flexibility for individual projects.

- Developing and introducing the concept of release planning pattern to customize a process model to reach the desirable method for every step.

- Validating the proposed release planning methodology using different case studies in the PRP developed tool.

There is no need to mention that the proposed methodology has a significant effect on the quality of release planning since a "pattern" of successful past experiences has been observed in this methodology. In addition, this methodology significantly decreases time, which is an important factor in current release planning methodologies. The reason is that many of the defined parameters can be saved for future planning.

The main advantage of this methodology is its capability to a release based on the characteristics of the project. In other words, solutions are proposed according to the project characteristics. In contrast to other current methodologies, which are not flexible and customizable, this method provides solutions based on values, resources and other project parameters.

As we know, there are two Software development situations. In the first type, which is called bespoke development, the main purpose is to meet a specific company's needs to facilitate running its business. Here, the customers are known and there is usually only one release because it is targeting a specific group. Of course, there is always maintenance for that single release.

On the other hand, there are market-driven products; similar to what we are aiming in this research. In this type of development, there are often a larger number of releases as long as there is a market for the product. Customers are unknown here, and the main goal of the product is long term. In this kind of development, releases and their plays are significant because there are always market demands for new and unpredicted

releases for a product. The present dissertation mainly focuses on market-driven products, which have various releases for a single product.

1.4. Research Methodology

Every research requires an integrated and well-structured methodology to achieve the main goals. In this research has, in fact, two main sections: the first one deals with challenges in software release planning. In this section, the challenges have been discussed from the point of view of software companies. Qualitative approach has been used in this section of the study using direct interviews, emails, and questionnaires. Due to the ambiguousness and lack of clarity of parameters significant in release planning, an empirical study was felt needed.

The second section introduces, develops, and validates the PBRP methodology in five different steps that will be discussed further below. A specific procedure is conducted, which is composed of five main steps and some sub-steps as shown in Figure 1-1. This section uses case studies in all the subsections of introduction, development, and validation.

These steps are explained in detail below.

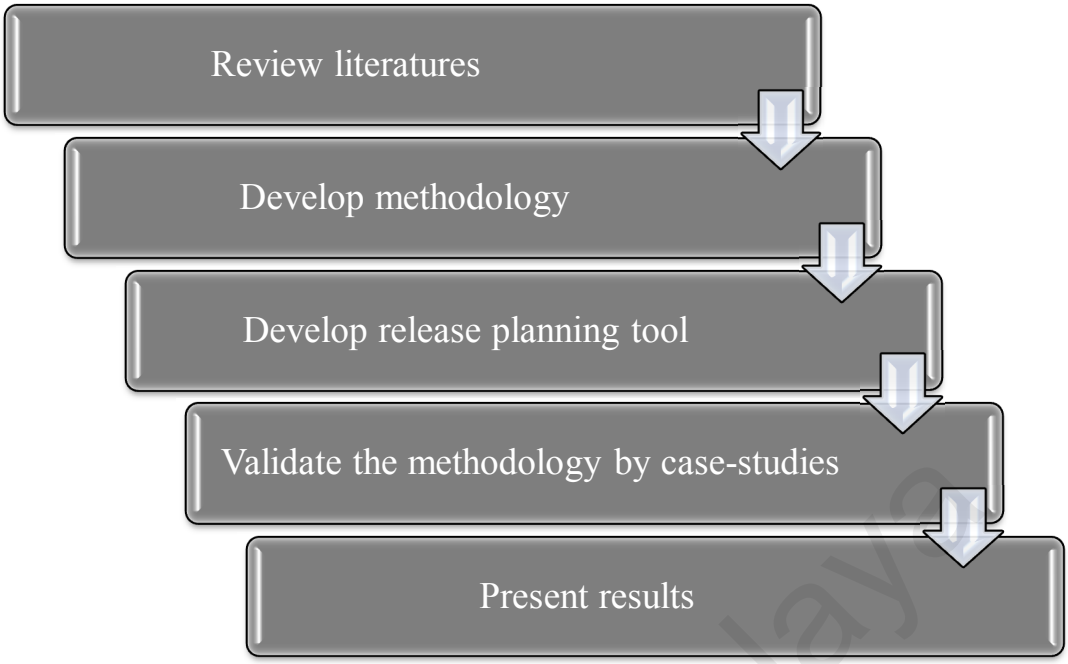


Figure 1-1: Main steps of the research

1.4.1. A review on literature

The research initially began with a review on previously implemented methods, studies and reports in the area of release planning. Naturally, the large number of these studies necessitated the study to be performed in a systematic manner. To do so, release planning area is divided into two main parts: release planning and release management. Methods presented in each of these are evaluated independently. In order to investigate every method, it is tried to study basic methods first and then go to newer methods or methods based upon old ones or their modifications. Besides, the specific challenges that these methods were trying to solve are classified and then common challenges are studied.

The main expectation of the literature review is to obtain an almost complete knowledge of release planning and management, their challenges and methods presented to solve them. As a result, an ordered set of all challenges, problems and their solutions is provided. The solutions cover some issues but still not able to solve the others. Moreover, the review shows issues not being considered so far in the field of release

planning and also the reasons they are neglected. It also provides an initial evaluation of the release planning and its efficacy to identify more applicable methods in the field. The final report of this review indicates the core problems of release planning and how the presented methodology tries to solve them. Table 1-1 shows a summary of the main inputs, activities and outputs of this step.

Table 1-1: Main inputs, activities and outputs in reviewing literature

Main Inputs	Main Activities	Main Outputs
- Academic and professional books, periodical journals, conference proceedings, technical reports, online documentation	- Studying concepts of release planning - Studying release planning methods - Reviewing challenges and weaknesses of release planning methods - Investigating the origins of release planning problems - Evaluating release planning methods	- Report on review of release planning methods - Report on release planning challenges

1.4.2. Developing the new release planning methodology

Having literature reviewed and current release planning methods evaluated, it is necessary to develop a new methodology based on findings in order to solve previous problems. This step aims at presenting a customizable methodology based upon project features and team characteristics. To do so, common tasks in implementing release planning are gathered from various methods, and after that a process model is generated which covers all main activities of release planning. Then, the process is customized using specifications concluded through the review. These specifications include project or team characteristics and other effective features on release planning, which can affect one or more tasks. Having identified these effects, they are specified as much as possible at the parameter level to obtain a comprehensive image of every parameter range. This helps to understand parameter variety and see the influence of parameters on the final solution. A new concept, release planning pattern, is used in the process of methodology development, which increases the customization speed and makes use of

previous experiences to improve the quality of the method and guarantee its applicability and functionality. Moreover, with the help of effective parameters on every task of release planning process, the methodology maintains expandability and new parameters and patterns can be easily added. At the end of the process, the presented methodology is generated in association with a set of release planning patterns and patterns of every step of the process model to be used in release planning. Table 1-2 shows a summary of the main inputs, activities and outputs of this step.

Table 1-2: Main inputs, activities and outputs in developing methodology

Main Inputs	Main Activities	Main Outputs
- Academic and professional books, periodical journals, conference proceedings, technical reports, online documentation - Report on review of release planning methods - Report on release planning challenges	- Reviewing common tasks of release planning methods - Presenting the release planning process model - Investigating effective parameters on release planning - Investigating the effect of parameters on every task of the process model - Investigating the effect of release planning parameters on selecting the method to perform every step of the process model - Investigating the interaction of planning's effective parameters - Defining the concept of release planning and its application - Presenting the Pattern based release planning methodology	- Effective parameters on the process model - Release planning patterns - Pattern based release planning methodology

1.4.3. Developing the tool

The presented release planning methodology needs to be used in practice. For this reason, a certain tool is developed. The tool must be capable of keeping record of all steps, parameters and instances of the process model and their relationships as well as

the relation between distinct interacting parameters. Furthermore, the tool must be able to define a release planning pattern and assign different steps based on various parameters. It should also feature searching different patterns using parameters to enable users to find and employ suitable patterns through entering their data and project specifications.

Table 1-3: Main inputs, activities and outputs in tool development

Main Inputs	Main Activities	Main Outputs
- The general methodology of release planning - Effective parameters on the process model - Release planning patterns - Pattern based release planning methodology	- Developing a tool based on the process model - Expanding the tool to enter relationships between parameters and their instances - Expanding the tool to enter release planning patterns - Expanding the tool to search for patterns based upon parameters and instances	- Pattern based release planning tool

1.4.4. Validating the methodology by case studies

It is necessary to evaluate applicability and effectiveness of the methodology after developing its suited tool. Among all evaluation methods, performing case studies is the most effective and efficient one, especially in the area of software engineering and release planning, which are considered executive and practical fields. The number of case studies is determined by the method and the probable errors. Since the pattern based release planning methodology has various patterns and each of them must be evaluated as much as possible, more than one case study are performed and various projects are tried as case studies. Therefore, companies selected for case studies are those which have numerous projects to find common characteristics (project and team) and this is considered as the main criterion for selecting case studies. This is important

because identical patterns are more likely to be used in these companies, and more data is available. Every project must have the following characteristics to be selected: there are at least 4 members in every team, each team has an approach, and the project is repeated at least twice. Every team selected as the case study is trained with the prepared release planning tool, and members are asked to enter specifications and parameters to perform the planning according to the pattern proposed by the software. Then, they are asked to evaluate the proposed pattern and the pattern based release planning methodology and to state its strengths and weaknesses.

Table 1-4: Main inputs, activities and outputs in validating the method

Main Inputs	Main Activities	Main Outputs
- The pattern-based release planning methodology - Pattern based release planning tool	- Making use of the tool for planning - Searching for the considered pattern in the tool based on project parameters - Performing release planning according to tool proposed method - Recording results of using tool proposed method	- Results of evaluating the use of every pattern - Results of evaluating the use of pattern based release planning methodology

1.4.5. Presenting results

Having case studies performed and the methodology evaluation results obtained from different teams, it is time to evaluate all results and achievements to identify strengths and weakness of the methods and their reasons. Results are usually conducted and documented in the form of charts and recorded as research results.

1.5. Research contribution

Release planning has become one of the most important tasks in software development projects since it shows the costs and time required to accomplish the project. Various methods are proposed for release planning but none of them are customizable and they mostly have paid no attention to effective parameters on release planning originating from project specifications. For this reason, this study introduces a pattern based methodology for release planning which is, first, highly customizable and can be used in a wide range of software projects. Second, the methodology uses project and team specifications effective on release planning to determine the exact method in order to enhance efficacy and applicability, and third, the methodology builds upon previous experiences and uses them along with the concept of pattern for release planning. Currently, there is no methodology available having these capabilities and providing teams with the same possibilities and hence the present thesis can be used by various software teams. Moreover, the methodology can be expanded, and more parameters and patterns can be added to it, thus teams can develop the methodology according to their own needs.

In summary, here are the contributions of the research for:

- Practitioners
    - Agility to choose best fitted method for release planning of each project
    - Using pattern based development to select the release planning method
    - Enhancing the method to add new release planning patterns
- Project Managers
    - Attaining the best release plan for a project
    - Incorporating user requests in release planning as required
- Knowledge and research people

- Enhancing an adaptive release planning method

1.6. Results

The presented methodology is generated by planning patterns and characteristics and parameters effective on every stage of the process model. It is accompanied by a set of achievements, including:

- **Improvement in quality of release plans**

  Pattern-based release planning methodology enables making use of previous successful experiences in the field of release planning, and this enhances the quality of developed plans.

- **Reduction in time spent on release planning**

  Pattern-based release planning methodology provides (simply and rapidly) the release planner or project manager with successful experiences of other projects considering a set of predetermined parameters and, as a result, reduces release planning time.

- **Release planning relative to project characteristics**

  Pattern-based release planning methodology uses a project's specific features and the best of past experiences to present a methodology suiting the project and hence increases the success rate of accomplished projects.

- **Making use of the best experiences in release planning**

  It is possible, using pattern-based release planning and developing new release plans, to transfer the experience and acquired knowledge of successful projects.

- **Omission of decisions made without technical support**

  Pattern-based release planning methodology aids project managers or release planners to achieve a better understanding of release planning and its different

methods introduced in the form of release planning patterns and prevents wrong or technically unsupported decisions.

1.7. Thesis structure

Chapter 1 includes thesis introduction, background and the overall image of the presented methodology as well as the thesis objectives.

Chapter 2 introduces a state of the art release planning in software release planning and argues current methods. The overall view of this chapter is based upon common steps and parameters.

Chapter 3 investigates the problem statement in details, including definitions, hypotheses and research questions.

Chapter 4 evaluates challenges in software developing companies and how to deal with them while preparing for new releases. Employed methods are also discussed here.

Chapter 5 includes release planning management and investigation of management challenges in this area.

In Chapter 6, a set of common steps and tasks of various methodologies are gathered as the process model of release planning. This process model covers all the tasks to be accomplished in release planning but it should be customized for every individual project. Hence, customization of the process and the parameters involved in the customization of every step are described. Having customization accomplished, due to the large number of parameters in every step to select the required method of release planning, a new concept, pattern, is used to narrow down the parameters. The patterns are based upon the concept and are conducted according to the parameters in every step (considering all the tasks in the process model) to use previous experiences and determine the most effective method for performing every step of release planning by

simply using parameter regulation. A set of three patterns are selected as examples for every step.

Chapter 7 evaluates pattern-based release planning methodology. To do so, five big software companies with various projects are selected. The results of implementing these methods were then evaluated. Besides, the results for every specific pattern and patterns of every step of the process model indicated that patterns pertaining to requirements prioritization step were more precise than the others.

Chapter 8 presents the results of the case studies and findings of the thesis.

Chapter 9 discusses the good characteristics or advantages of the methodology and suggestions for future improvements.

CHAPTER 2: LITERATURE REVIEW

2.1. Introduction

Release planning is one of the main tasks in software development which plays an important role in forming releases and the outputs of a developer team. In a highly abstracted level, release planning is the process of selecting the best set of assigned features in a certain release. This means that instead of developing new and perfect software, it is tried to plan to develop a number of new features in incremental series which improve functionality and performance as shown in Figure 2-1.

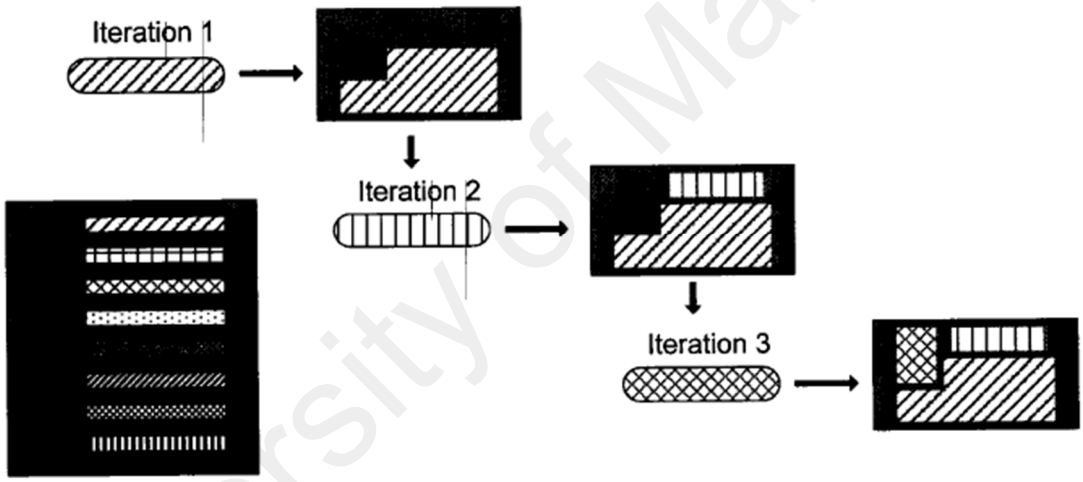


Figure 2-1: Planning software releases (Ruhe & Saliu, 2005a)

In this section, we review the current status of research and studies on release planning and, especially, methods, models and characteristics of planning in order to provide a clear insight into the task and its techniques. Below, we first explain common and widely used release planning methods and describe release re-planning techniques, and then characterize methods of requirements prioritization and resource estimation. In addition, software development patterns are discussed.

2.2. Release planning methods

Today, large companies in software development face many problems in their new releases. Various methodologies addressing the release planning challenges have been used in industry and many more approaches have been proposed in academic research. Although there are various methods and approaches to develop a new release, a new version has always its own problems and challenges. The next section studies and reviews these available methods.

2.2.1. Ad-Hoc Approach

Some organizations do not see release planning activity as independent, and they think basic decisions in their organization are based on business rule and needs. Decision-making process is mostly haphazard with an emphasis on guessing, discussion, business rules, and customers' needs instead of a systematic way based on formulation and quantitative ways.

In Ad-hoc approach, that is the easiest form to practice, there is no clear order for scheduling, planning and prioritization of requirements and findings are manual and usually based on negotiation but Anton in(Ruhe & Saliu, 2005a) reported that a complex project would likely fail without a plan. Many release plans focus only on the target release contents (Anton, 2003) rather than on defining incrementally releasable products.

Ruhe in(Nejmeh & Thomas, 2002) did a comparison between Ad-hoc planning and Systematic planning to know the level of reliability and validity of their research and found out that Systematic planning based on tools is more reliable than Ad-hoc planning.

The Ad-hoc manner is more common and maybe suitable for a relatively small in-house project that includes a few features only and no serious constraints. It can be said that prioritization in this approach is based on many factors such as project managers' judgment and ideas of stakeholders. Therefore, this approach depends more on individuals.

2.2.2. Planning Game Approach

The planning game (PG) refers to the process of planning and deciding what to develop in extreme programming (XP) project (Du, 2006). The main goal of XP is to lower the costs of change in software requirements (Du, 2006). With traditional system development methodologies, like Waterfall Methodology, the requirements for the system are determined often at the beginning of the project .The first Extreme Programming project was started on March 6, 1996. Extreme Programming is one of several popular agile development processes, and it has already proved to be very successful at many companies of all different sizes and industries worldwide.

Extreme planning (Ruhe & Saliu, 2005b) conducts release planning by performing planning games techniques. It has been successful in many companies because it emphasizes on customer satisfaction. Its aim is to deliver maximum value to the customer in the shortest time possible.

Customers write story cards describing the features they want, and developers assign important features and estimate the time required to develop those features. The most promising story cards are chosen for the next release by either setting a release date and adding the cards until the estimated total matches the release date, or selecting the highest value cards first and setting the release date based on the estimates given on the cards.

PG's strength is in the simplistic and straightforward approach it adopts, which works well in smaller projects. However, as the size and the complexity of the projects increase, the decisions involved in planning releases become very complex.

According to (Anton, 2003), XP (and thus PG) does not provide guidance on some key issues. XP does not:

- Address how a development group should interact with key stakeholders
- Describe how to produce consistent features and priorities that satisfy multiple stakeholders
- Provide a suggested technique to balance conflicting demands of multiple stakeholders
- Provide a technique for managers to assess the value of proposed features.

2.2.3. Incremental Funding Method (IFM)

These days, software companies' investigations in software development need to return in much shorter time and take more revenues. These companies do not invest in software development without clear returns. That means, they first look at the market and its demands and start their developments based on that. IFM is a financially informed approach to software development, designed to maximize returns through delivering functionality in 'chunks' of customer valued features, and carefully sequenced to optimize Net Present Value (NPV) (Beck, 2001). The Incremental Funding Method is a software engineering method that emphasizes financial considerations in a software project. The method was introduced by Mark Denne and Jane Huang.

This method decomposes the system into units of customer-value functionality known as minimum marketable feature that can be delivered quickly and provides market value

to the customer (Denne & Cleland-Huang, 2004). An MMF's value is typically measured in terms of both tangible and intangible factors such as revenue generation, cost savings, competitive differentiation, brand name projection, and enhanced customer loyalty. MMFs are identified by customers, developers, and business stakeholders according to the adopted software development process (Denne & Cleland-Huang, 2003).

2.2.4. Optimization-Based Techniques

Considering the complex nature of RP, several studies have modelled the problem of selecting release features as a specialized optimization problem. In formulating an optimization model for RP, Bagnall *et al.* (Bagnall et al., 2001) assign weights to customers based on their importance to the software company. The objective is to find a subset of customers whose demands are to be satisfied within the available cost. Ho-Won Jung (Ho-Won, 1998) follows a similar footing with the goal of selecting features that give maximum value for minimum costs within allowable cost limits of a software system.

These optimization approaches cope better with bigger problems, but customers are not given an opportunity to participate in RP decisions. The importance of customers to a company is not the only issue RP should try to satisfy. Most of the problems discussed above for "planning games" arise equally here.

2.2.5. Hybrid Intelligence Approach

The hybrid intelligence approach for RP proposed by Ruhe *et al.* (G. Ruhe & A. Ngo, 2004) is an extension of the optimization-based techniques. The belief that computational intelligence cannot replace a human decision maker was the driving force of this approach. A synergy between the two decision strategies was explored.

The overall architecture of this approach called EVOLVE* (G. Ruhe & A. Ngo, 2004) is designed as an iterative and evolutionary procedure mediating between the real world problem of software RP, the available tools of computational intelligence for handling explicit knowledge and crisp data, and the involvement of human intelligence for tackling tacit knowledge and fuzzy data. At all iterations of EVOLVE*, three phases are passed: modelling, exploration, and consolidation. We will later illustrate the approach in a case study example.

2.2.6. Lightweight Re-planning

Lightweight re-planning was first introduced by Thamer AlBourae in (AlBourae et al., 2006) and emphasizes adding new features in re-planning. In fact, in this process model old features are compared with newly added ones using Analytical hierarchy process. In Incremental software development, changes are very important and new change requests arrive during the process. These changes include modification of some features or addition of new ones.

The main goal of the light re-planning model is to develop a new product plan that achieves higher stakeholder satisfaction. Figure 2-2 shows a generic process model that is describing main release re-planning activities including their input and output.

In this model, three main roles are considered: Product manager, who is responsible for the whole development process; Stakeholders, which include any team member who are concerned with the product development; and the supporting environment, which facilitates the achievement of the process's goals.

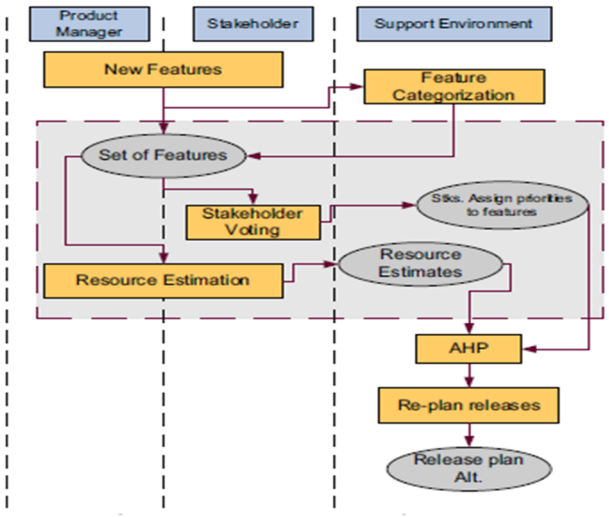


Figure 2-2: Generic process model for the Light-Weight Re-planning

In the next section, we will explain each activity in detail. The main steps are (AlBourae et al., 2006):

New features: When developments are going to start, the new features are collected. The requested changes are added to the old sets of features and are categorized by the feature categorization process.

Feature categorization: Adopted framework changes requested should be categorized to distinguish between duplicated features, on-going, or newly-added ones.

Stakeholders' vote: Stakeholders are people who are effective in the development process and can include different groups, such as managers, developers or end users.

Resource estimation: Resource capacities are one of the areas that should be considered when re-planning product releases. The main goal is to determine the likely usage of effort Effort (*fi*), and time Time (*fi*) for each feature *fi* for the next release.

2.3. Release planning researches

Release planning is an important part of development activity, specially in large software development organizations. A release plan contains features and needs which

must be developed in the next iteration and is influenced by several factors such as the types of requirements, implementation strategies, value for the developing company, urgency for the client, etc. (Ruhe, 2005). Generally, software development organizations perform release planning as an ad-hoc task by project managers, when the number of requirements is small (Seyed Danesh & Ahmad, 2012). When there is a large number of requirements, the uncertainty of factors causes the release planning be classified as NP-hard problem and, like other such problems, it often needs a search-based approach to find the optimal or near optimal solution (Bagnall et al., 2001; Durillo, Zhang, Alba, Harman, & Nebro, 2011; Mohebzada, 2012; Saleem & Shafique, 2008). Researchers address a release planning problem in two levels. First, they present a systematic model based on factors and parameters and/or find out effective factors; next, they focus on proposing new algorithms to optimize the results of release planning.

The uncertainty of factors in release planning is a major concern in the first class of studies. Al-Emran *et al.* (Al-Emran, Kapur, Pfahl, & Ruhe, 2010) examine the impact of uncertainty in operational release planning and present a method for analyzing and estimating the impact of uncertainty on planning parameters. They find out that the uncertainty of release planning parameters increases – both in magnitude and variance – with an increase of pessimism level as well as an increase of the number of uncertainty factors. Also, Lindgren *et al.* (Lindgren, Land, Norstrom, & Wall, 2008) and Saliu *et al.* (Saliu & Ruhe, 2005b) study the key aspects of software release planning in industries and list them as: objectives, resource constraints, technology constraints, system constraints, time horizon, stakeholder involvement, and short- and long-term planning. In addition, Wohlin and Aurum (Wohlin & Aurum, 2005) study decision-making criteria to choose the best parameters for a release and conclude that the business-oriented (customer and market focused criteria) and management-oriented criteria

(related to cost-benefit and timeliness of delivery) are more important than technical concerns (related to software architecture and interdependency of requirements).

Bagnall *et al.* (Bagnall et al., 2001) define a mathematical model to describe the parameters' effect on release planning and try to find a systematic approach to overcome the next release problem (NRP). They compare three general approaches and conclude that on the large scale, yielding an optimal solution in reasonable time may fail, but with a small set of requirements, the techniques will be sufficient. Also, Carlshamre (Carlshamre, 2002) proposes a pragmatic tool utilizing a selection algorithm which, based on value, estimated resources and interdependencies between requirements, presents a number of valid release suggestions. He emphasizes, though, that this tool has several serious shortcomings because in the study, the size of the product is small, companies involved have a fairly good picture of their customers, and all results are organization-dependent to some extent. Colares *et al.* (Colares, Souza, Carmo, Padua, & Mateus, 2009) present a mathematical model that takes into account several important aspects of release planning, such as stakeholders' satisfaction, costs, deadlines, available resources, efforts needed, risk management and requirements interdependencies. This model is validated by experimental data and is not empirical.

On the other hand, the algorithmic solutions try to handle the specified factors by using heuristic algorithms. Greer and Ruhe (Greer & Ruhe, 2004) present the EVOLVE method based on genetic algorithm that generates a typically small set of most promising candidate solutions from which the actual decision-makers can choose. In fact, the EVOLVE searches the requirements, constraints, and priorities based on defined parameters and the data provided by users and presents candidate release plans. The EVOLVE is a well-known method for release planning that resolves the release planning problem by using the genetic algorithm (Maurice, Ruhe, Ngo-The, & Saliu, 2005; G. Ruhe & A. Ngo, 2004). AlBourae *et al.* (AlBourae et al., 2006) propose a

lightweight re-planning process model based on AHP and greedy algorithm. During the AHP process, the Weight Average Satisfaction (WAS) method is used to justify requirements' importance. In reality, this process model provides a basis for incorporating changes instantly into the development lifecycle. It has not been validated empirically and needs real-world industrial experiments. Also, Freitas *et al.* (Freitas, Coutinho, & Souza, 2011) study release planning techniques, especially heuristic algorithms that are search-based, and propose to use exact optimization techniques based on Simplex method instead of using meta heuristic genetic algorithms and Simulated Annealing.

Hybrid methods handle the release planning problem by breaking down the problem to various sub-problems or various views. Because of the complexity of effective factors in release planning, Ruhe and Saliu (Ruhe & Saliu, 2005a) introduce a hybrid release planning framework that features both human and computational intelligence (Ruhe, 2005). The human intelligence may overcome the release planning methodology's weakness around the relationship between various factors as well as unknown factors. Saliu and Ruhe (Saliu & Ruhe, 2007) also present Bi-Objective Release Planning for Evolving Systems (BORPES) to optimize the value of release plans from both the business perspective and the implementation perspective by a trade-off between the two perspectives and feature coupling detection method to reduce cognitive effort during implementation. Jadallah *et al.* (A. Jadallah, Al-Emran, Moussavi, & Ruhe, 2009) break down the release re-planning problem into "How", "When" and "What" (H2W) and propose an algorithm for each one. H2W provides a new approach for re-planning of an existing product, but the quality of release plans is not validated by industrial evaluation. Another hybrid method to overcome the uncertainty of planning factors is the recommendation system. Mohebzada (Mohebzada, 2012) presents a recommendation system named SRP-Plugin 2.0 to assist product managers with better

release decisions. Because of the large volume of requirements data and dependency of release planning factors to input parameters, machine learning is used in the SRP-Plugin that is realized through four techniques. The recommendation system is based on EVOLVE II release planning that can learn the human intelligence and use it in decision making.

Related studies on release planning methods show that using human intelligence can lead to better decision making in release planning and methods that use it can demonstrate a better performance (Greer & Ruhe, 2004; Maurice et al., 2005; Ruhe, 2005; G. Ruhe & A. Ngo, 2004; Ruhe & Saliu, 2005a), but this is not intended to solve every problem of release planning (Mohebzada, 2012). Human intelligence as applied in the release planning method is used in two ways. In some methods, human intelligence is used to rank input requirements or select the best plan, but in Mohebzada(Mohebzada, 2012), it is used to train the machine in recommendation systems. Therefore, human intelligence can be used as the driver of release planning method to make decision making. This idea can lead to a better method for release planning.

Looking at the studies done on release planning and examining current methodologies and approaches, one can see that various parameters, that are sometimes unclear or even unknown, affect a project. Although the proposed solutions almost always improve the release, they may not be the optimal ones.

In fact, present studies can be divided into two categories: the first group, which attempt to identify and solve the unexpected problems, often deals with parameters and the problems afflicting them. There are numerous valuable examples of empirical studies in this category.

The second group's main focus is on introducing methodologies or general frameworks to improve or modify release planning. This group of studies has shown less flexibility in general and requires new methodologies and frameworks.

One important element which is missing in these studies and in practical software business is providing solutions based on a project's characteristics and parameters, which can significantly increase flexibility and can customize the solution for individual projects. In other words, specific solutions can be drawn on a project's characteristics that are often the optimal ones since they originate from the parameters of a specific project.

2.4. Requirements prioritization methods

Requirements prioritization is performed in order to identify and recognize more prior requirements. Sommerville (Sommerville, 2010) identifies requirements prioritization as a task performed to identify important requirements but Firesmith (Firesmith, 2004) describes this task as the process of determining requirements implementation order for the sake of system implementation. A comparison of the two definitions demonstrates that Sommervile (Sommerville, 2010) emphasizes the importance of requirements for users. Of course, it must be born in mind that requirements also have interdependencies which influence the system implementation order in stakeholders' viewpoint. However, Firesmith considers requirements interdependencies in a systematic manner in his definition since he emphasizes implementation and notices their implementation order.

Different aspects of features can be considered in requirements prioritization. These are known as "requirements prioritization measures" the most important of which are mentioned below:

- **Importance**

Stakeholders can observe the requirements and determine which one is more important to them. This is usually considered as a multi-dimensional measure and can lead to various perceptions using different views. For example, importance can be considered in terms of the value for the market, the value for accomplishing routine tasks of a stakeholder, or the value for product quality. It is of high significance to define 'importance' well to the stakeholders to be able to process their data.

- **Time**

Time is one of the most important and widely used measures of requirements prioritization. It is estimated and entered into the prioritization procedures using certain methods. Of course, it is influenced by other factors such as implementation techniques or team experience. In most cases, time is estimated by the project manager or requirements manager or experimentally by the development team, unless the resource estimation technique differs from the requirements prioritization technique. In this case, time estimation is omitted from prioritization activity.

- **Costs**

Cost can also be considered as one of the important parameters in requirements prioritization. In fact, cost is a computation parameter that is directly influenced by the time needed for implementation of a certain requirement. It is also influenced by other factors such as extra resources, predicted and even unpredicted costs not related to human resources. Moreover, expenditures on purchasing licenses for some components or prerequisite software can affect costs of a requirement. For instance, to implement a requirement like sending SMS in

the software it is necessary to purchase the software and components related to SMS sending system and then embed this capability in the main software.

- **Penalty**

  Penalty is, simply, the costs of delay in implementing a certain requirement. This can have great significance in developers' work if the penalty resulted from unimplemented requirements which can be lead to financial or physical costs. Furthermore, penalty can originate from not presenting a product adequately to the market which causes financial loss to the developer team or the user (customer).

- **Risk**

  The risk originating from every requirement is related to the risks of a project. Requirement-originated risk encompasses different aspects such as the market value, the required resources and the requirement variability. Requirement's risk is an estimated value that is assigned by project manager or requirement manager.

Other aspects of a requirement, such as stability, market value and available resources, may also be considered as requirement prioritization measures. Requirements are mostly prioritized based on only one parameter, but this depends on project features and expectations of requirement prioritization. Clearly, prioritization based on multiple measures is more difficult than that based on only one. Moreover, it must be kept in mind that requirement variations and interdependencies can also influence measures, and this complicates multiple-measure prioritization.

Followings are some essential requirements prioritization techniques, which are classified into three groups: nominal scales, ordinal scales and ratio scales.

2.4.1. Nominal Scale

Requirements in this technique are assigned to various priority groups, and all requirements in a group have the same priority. The technique is usually very simple and requirements are often categorized in several defined levels.

2.4.1.1. Numerical Assignment

This method is mentioned in studies such as Berander and Andrews (Berander & Andrews, 2005) and Karlsson *et al.*(L. Karlsson, Host, & Regnell, 2006). This is a simple technique based on categorizing requirements into various priority groups. The number of priority groups can vary but there are commonly three groups: critical, standard and optional requirements. The outcome of numerical assignment is a type of nominal scale and the groups are prior only in terms of their name-based categorization and there is no extra information on higher or lower priority of a certain requirement to others in the same group.

2.4.1.2. MoScoW

MoScoW is a type of numerical assignment suggested by DSDM[1] consortium and Hatton (Hatton, 2007), (Hatton, 2008). Currently, it is employed in DSDM software development method. The main idea of MoScoW is to classify all requirements into four groups: "MUST have", "SHOULD have", "COULD have" and "WON'T have".

- "MUST have" means requirements of this group must be present in the project. Lack of these requirements results into the failure of the project.
- "SHOULD have" means the project succeeds if it includes requirements of this group.

[1]Dynamic System Development Method

- "COULD have" means the project succeeds if it includes requirements of this group. But, this group is less prior than the previous one.

- "WON'T have" is like an "interest list". This means that requirements of this group are good but are not implemented in the current stage and may be employed in the next version.

Results of MoScoW are obtained in nominal scale. All requirements in a certain group are of the same priority and there is no excess information on higher or lower priority of a certain requirement to others in a single group.

2.4.1.3. Top-10 requirements

This is a simple and coarse technique in terms of complexity and granularity, respectively (Berander & Andrews, 2005)and determines top-10 more prior requirements from a bigger set. The technique does not specify an internal order for requirements and this is considered as its main weakness. However, it can be useful in cases with numerous stakeholders of the same importance (Lausen, 2002). Throughout prioritization, procedure interferences lead to numerous states. Therefore, it is important to neglect averaging since some requirements may be omitted consequently (Berander & Andrews, 2005). This method should be used when the lowest level of interference is present.

2.4.2. Ordinal Scale

Methods of the ordinal scale lead to an ordered and arranged list of requirements. The list can be arranged based on a certain parameter (usually preference). This method develops requirements more precise than methods of nominal scale, but can result in more errors if not performed carefully.

2.4.2.1. Simple Ranking

Ranking elements can be easily understood by most individuals and can occur in daily life and that is why the method is well-accepted. Bernarder and Adnrews (Berander & Andrews, 2005) and Hatton (Hatton, 2008) showed that in this method N requirements can be easily ranked in the form of 1….N, in which 1 is the most prior requirements and N is the least prior one. This is the most widely used method in ordinal scale.

2.4.2.2. Bubble Sort

This method is described by Aho, Hopcroft and Ullman (Aho, Hopcroft, & Ullman, 1983) and is used to sort different factors. Karlsson et al.(J. Karlsson, Wohlin, & Regnell, 1998) first introduced the method for the requirements prioritization field. The main idea in Bubble Sort is to enable users to compare two requirements and change their places if in a wrong order. The comparison continues until no more replacements are needed. Its outcome is a list of requirements prioritization. The average complexity of Bubble Sort is $O(n^2)$.

2.4.2.3. Binary Search Tree

This is another technique used to sort factors which is described by Aho *et al.*(Aho et al., 1983). In this tree, every node has at last two sub-trees. The method was introduced by Karlsson *et al.*(J. Karlsson et al., 1998) in order requirements prioritization and sorting. The idea in this method is that every node represents a requirement and all requirements under the left sub-tree of the node are less priori and all requirements under the right sub-tree of the node are more prior than the node. During the implementation of this method, a requirement is first selected as an initial node. Then, an unsorted requirement is compared to the initial node and placed under left sub-tree if it is les prior than the upper node. If it is more prior than the node, it is placed under the right sub-tree. This continues until no other node requires comparison and all

requirements are placed correctly. Average complexity of binary search tree is O (*n* log *n*).

The three above-mentioned techniques are used to rank requirements. Ranking is very simple and understandable to individuals. But, Bubble Sort and Binary Search Tree seem to be more difficult. Simple ranking can be employed when a relatively small number of requirements are being prioritized but an increase in their number leads individuals to fail to remember all the requirements and their priorities. If a huge number of requirements are being prioritized, Bubble Sort and Binary Search Tree seem more efficient in achieving a high precision.

2.4.3. Ratio Scale

Results of this scale can represent relative differences between necessities. In fact, methods of this scale compare requirements and try to measure and quantify their priorities to one another.

2.4.3.1. 100-Dollar method

The 100-Dollar method (cumulative voting) is proposed by Bernarder and Andrews (Berander & Andrews, 2005) and Hatton (Hatton, 2008) and is considered as a simple method of requirements prioritization. Its main idea is that every stakeholder assumes he/she has 100 $ to distribute among requirements. Results are obtained in ratio scale and show the importance of a requirement relative to another.

2.4.3.2. Analytic Hierarchy Process

Analytic Hierarchy Process (AHP) is another well-known prioritization method which bases upon the results of ratio scale. Developed by Saaty (Saaty, 1980), the method is designed for complicated decision-making. Its main idea is to compare all requirement pairs to determine their priorities. While using AHP, the user first specifies substitute items and features for every certain requirement and employs them to build the

hierarchy. Then, he/she clarifies his/her preference for every feature pair by attributing a priority ranging from 1 to nine, where 1 represents equal value and 9 represents supreme value. The scale is shown in Table 2-1. After AHP changed users' evaluation to numerical values, the numerical priority of every factor of the hierarchy is generated. If n requirements are being prioritized, then N*(N-1) / 2 measures are required. Thus, the method complexity equals $O(n^2)$.

Empirical studies by Karlsson and Ryan (J. Karlsson & Ryan, 1997) and Karlsson *et al.* (J. Karlsson et al., 1998) indicate that AHP is highly time-consuming. Some methods are proposed and developed to reduce the number of comparisons and to minimize AHP required time; two important ones include hierarchical AHP and the Minimum Spanning Tree.

Table 2-1: Essential scales used for AHP (J. Karlsson & Ryan, 1997)

Relative Intensity	Definition	Explanation
1	Of equal value	Two requirements are of equal value
3	Slightly more value	Experience slightly favors one requirement over another
5	Essential or strong value	Experience strongly favors one requirement over another
7	Very strong value	A requirement is strongly favored and its dominance is demonstrated in practice
9	Extreme value	The evidence favoring one over another is of the highest possible order of affirmation
2, 4, 6, 8	Intermediate values between two adjacent judgments	When compromise is needed
Reciprocals	If requirement i has one of the above numbers assigned to it when compared with requirement j, then j has the reciprocal value when compared with i.	

2.4.3.3. Hierarchical AHP

Davis (Davis, 1993) suggests that requirements in large projects are mostly presented in a hierarchy in which more general requirements are on top and more precise ones are placed at lower levels. Hierarchical AHP, introduced by Karlsson *et al.* (J. Karlsson et al., 1998) uses AHP to prioritize requirements only in the same level of the hierarchy. This method can reduce the number of decisions compared to AHP. Since all

requirements are not paired together and are not compared, the method can decline repetitive comparisons but the ability to identify contrary judgments decreases.

2.4.3.4. Minimum Spanning Tree

This is another prioritization technique introduced by Karlsson *et al.* (J. Karlsson et al., 1998). Its main idea is that in the case of consistent decisions no excess is available. In such a case, the number of comparisons declines to N-1 (N = number of requirements). This tree is composed of unique requirement pairs and is a directional graph. Compared to AHP, the Minimum Spanning Tree considerably reduces the number of comparisons. However, the ability to identify contrary judgment is weak.

2.4.3.5. Cost–Value Approach

Karlsson and Ryan (J. Karlsson & Ryan, 1997) proposed a method for requirements prioritization called "Cost–Value Approach". The essential idea underlying this method is to examine every requirement from two aspects: its value to users and its implementation costs. The method employs AHP to compare requirement pairs consistent with cost and relative value. Empirical studies of Karlsson and Ryan (J. Karlsson & Ryan, 1997) demonstrate that the Cost–Value Approach is highly time-consuming.

2.4.4. Compound techniques

2.4.4.1. Planning Game

Beck (Beck, 1999) proposed a prioritization method called "Planning Game" which is based upon a combination of several requirements prioritization techniques. This method is mostly used in agile projects and its underlying idea is to employ a combination of Numerical Assignment and Ranking methods to prioritize requirements. Requirements are first divided into three groups: (1) those unable to function without the system, (2) those of less necessity but with justified business value, and (3) those

presence of which is good. After the classification, requirements are easily ranked in their groups.

2.5. Resource estimation methods

Correct estimation of the costs of a software production provides the project manager with a strong support to make different decisions during software's lifecycle. Project manager, analyst, release planner, programmer and other team members recognize the amount of work and time required to present a proper product. Without appropriate estimation of costs, the project manager often fails to identify the required time, work and required resources to accomplish a project. Wrong estimation leads the project fails totally.

One of the most important factors in estimating costs of a software system is its size. Five methods are proposed to estimate software size, which are described below.

- **Line of Code**

Number of lines of codes in the source of the program presented to user, except for explanations and empty lines, is known as LOC. This is independent from the programming language. The precise size of LOC is specified after the project is accomplished. A common method to estimate LOC is to use experience along with PERT technique. In this method, three variables are considered for program code size: L (the lowest possible size), H (the highest possible size) and M (the mean or medium possible size). The code size S is obtained by the following equation (Sommerville, 2010):

$$S = (L + H + 4M) / 6$$

It is possible to estimate and sum the code size of different components.

- **Software Science**

In this method, two measures are used to estimate software size: code length and volume (Sommerville, 2010). Code length is employed to calculate the code length of the program source and is obtained by following equation (Sommerville, 2010):

$$N = P + Q$$

Where P is the total count of operators and Q is the total count of operands. Volume represents the used space and is calculated by:

$$V = N \log (p + q)$$

Where p and q are the total number of independent operators and operands, respectively.

- **Function Point**

This estimation method is based upon software function (or functional value) and results for an empirical relationship based on measurable (direct) scales of software information and evaluations of its complexity. The measurable scales include (Sommerville, 2010):

- Number of user inputs
- Number of user outputs
- Number of user inquiries
- Number of files
- Number of external interfaces (files shared with or delivered to external systems)

Each of the above scales is assigned a complexity class of 1 (simple), 2 (medium), and 3 (complicated) and a weight value ranging from 3 (for simple inputs) to 15 (for complicated files). Computation of the total count is shown in Figure 2-3. Then, the function is calculated through the following equation (Pressman, 2001):

FP = count total × [0.65 + 0.01 ×Σ (F*i*)]

The main advantage of this method is that it can be employed based on system requirements and in initial steps of the project.

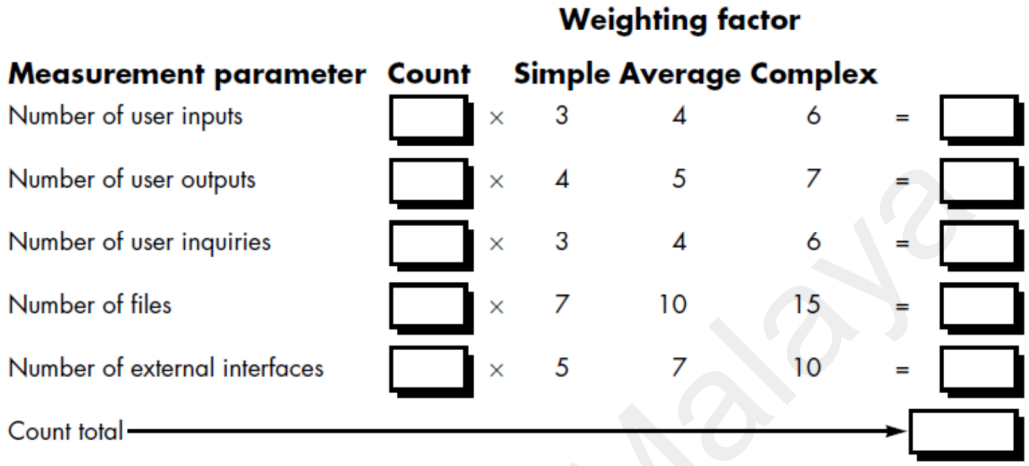


Figure 2-3: Computation of Function Point (Pressman, 2001)

- **Feature Point**

This method can both develop an algorithm and add it as a new class to the five classes of Function Point method. Every used algorithm is assigned a weight ranging from 1 (for simple algorithm) to 10 (for complicated algorithms). This method best suits programs with few inputs and outputs and high algorithmic complexity.

- **Object point**

This method acts based on number and complexity of forms, reports and software components of new generation languages. Each of these objects adopts a weight (according to its number) ranging from 1 (for simple forms) to 10 (for components of new generation languages) and the result is weight sum of these items. This method can also be implemented in initial steps of a project.

Methods of estimation software costs are divided into two general groups: algorithmic and non-algorithmic methods. Non-algorithmic methods are based on a set of steps and tasks which do not follow a certain algorithm and are mostly based upon experience.

Algorithmic methods follow certain algorithms and the estimation procedure is well specified in them. Although past experiences are also used here, the methods are mainly based upon algorithms.

2.5.1. Non-algorithmic methods

2.5.1.1. Analogy Costing

In this method, costs of a new project are estimated based on experiences of similar previous projects. The method can be employed in a whole project or subsystems. In the former, costs of all components are studied and in the latter, similarities and differences between the current system and previous ones are investigated, and hence the estimation is more precise. One of the advantages of this system is its implementation based on previous real experiences. The disadvantage is that because of the inconsistency of previous systems with the current one, the wrong comparison may turn aside the estimation.

2.5.1.2. Expert Judgment

In this method, costs are estimated based on personal methods and innovative techniques of experts of software development. Then, techniques such as Delphi and PERT which result in estimation aggregation are used to remove probable inconsistencies in the estimations of different individuals. For example, Delphi technique runs in this way:

1) System features are described for every individual.

2) Individuals present their estimations independently (without any consultancy).

3) Presented estimations are listed and communicated to individuals. Then, they are asked to present the estimation again and explain its rationale.

4) Steps 2 and 3 repeat until a proper result is achieved.

2.5.1.3. Parkinson

Software costs are not estimated in this method but are determined considering available resource (regardless of project objectives). For example, if the time required to accomplish is 12 months and there are 5 persons available, the rate of 60 persons per month is estimated. Although it provides acceptable estimation in some cases, it is not considered as a proper technique for costs estimation.

2.5.1.4. Price-to-Win

In this method, costs are estimated based on employer's budget instead of software, its capabilities and applications. For instance, if the real estimation of project equals 100 person-months but the employer has sufficient budget only for 60 person-months, the estimation is performed based on the latter.

2.5.1.5. Button-up

In this method, every system component is estimated independently and then the sum of estimations is considered as the total costs estimation. In order to employ this method, it is necessary to first have a primary plan of the system to identify its components.

2.5.1.6. Top-Down

This is contrary to the previous method and project costs are estimated by algorithmic and non-algorithmic methods in an integrated form based on general measures. The cost can be distributed between different system components in the next stage.

2.5.2. Algorithmic methods

Algorithmic methods use mathematical models to estimate project costs. Every algorithmic model is defined as a function of cost factors. Current algorithmic methods differ in two aspects: first, selection of cost factors and, second, definition of cost calculation function. Cost factors directly affect cost estimation and include the following groups:

1) Product Factors, such as: reliability, complexity rate, database volume, reusability, coordinating project documentations with its lifecycle needs.

2) Computer Factors, such as: time limit for system running, limited storage capacity, limitations in computer restarting, platform diversity.

3) Personnel Factors, such as: skills of the analysis team, skills of coders, fluency in the platform, fluency in programming language and its tools, coordination of team members.

4) Project Factors such as: using Multisite Development, using software tools.

Table 2-2 shows the categorization of algorithmic methods:

Table 2-2: Categorization of algorithmic methods (Briand & Wieczorek, 2002)

Algorithmic Models					
	Linear	**Multiplicative**	**Power Function**	**Discrete**	**Others**
Empirical	Nelson	Walston-Felix	COCOMOs	Aron Boeing Molverton	Price-S
Analytical			Putnam		Soft-Cost

2.5.2.1. COCOMO method

This method was first proposed by Bohem in 1981 (Boehm, 1981). In his model, Bohem defined the following factors as effective on costs of a software project:

1) Product's reliability

2) Product's complexity

3) Run-time limit

4) Main memory limitation

5) Machine availability

6) Analysis team's capability

7) Experience in developing applied software

8) Programming team's capability

9) Rate of using modern planning tools

10) Rate of using modern programming tools

In this method, the effect rate of every factor on the project is ranked from "low" to "very high" and they are assigned certain weights. In this way, a matrix is generated in which rows represent effective factors and columns represent their effect rates. Figures representing the weight of every factor are written in the table. The method considers numerous factors in estimations and hence it has a high error probability.

The general formula of this method is (Briand & Wieczorek, 2002):

$$\text{PersonMonth} = a(\text{KDSI})^b$$

Where "*a*" and "*b*" depend on COCOMO modeling level (simple, medium, detailed) and the state of estimated project (organic, semi-detached, embedded).

In addition to basic COCOMO, there are also other models available for this method, the newest of which is COCOMO II. This is a combination of "Applications Composition", "Early Design" and "Post Architecture" models. In this model, the exponent "b" of the formula changes based on factors such as project flexibility, software architecture, risk conflict methods, coherence and effective relationships in the project team. Besides, new cost factors are defined in order to integrate project architecture and risk reduction.

2.5.2.2. Putnam's Model

The main specification of this method is the software equation defined as:

$E = y(T) = 0.3945 * K$

K = area under curve [0, 1) measured in programmer year

T = optimal development time in years

$D = K / T^2$ *difficulty*

$P = c_i * D^{-2/3}$ *productivity*

$S = c * K^{-1/3} * T^{4/3}$ *lines of code*

Project accomplishment time, E, is an environmental factor representing development capability. S is based on LOC and represents person-year in the project. E is a parameter called "labor reinforcement" varying from 8 (for new software with high interface) to 27 (for rebuilt software).

2.6. Software development patterns

The current use of the term "pattern" is originated from writings of an architect named Christopher Alexander (Alexander, Ishikawa, & Sara Ishikawa, 1977) who has many books in urban planning and building architecture. In 1987, two researchers, Ward Cunningham and Kent Beck (Beck & W., 1987), who were working on designing user interface with Smalltalk programming language decided to use some of Alexander's ideas to develop a 5-pattern small language. Later, Jim Coplein (Coplien, 1992) used published results of the two researchers to prepare a catalogue of idioms in C++ language and published it in one of his books. Between 1990 and 1992 members of "Gang of Four" (Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides) met each other frequently and tried to prepare a catalogue of patterns and introduced it in a materialist workshop in 1991. In August 1993 Kent Beck and Grady Booch met in Colorado, a meeting which formed the basis of Hillside group. Not long afterward the famous book of *Designing Patterns* was published by the GoF (Gamma, Helm, Johnson, & Vlissides, 1995).

Several definitions of pattern are presented in various texts. Generally, it can be said that every pattern shows how a certain problem is solved through a specific solution. But pattern is more than a solution and indicates that the problem in consideration occurs in a certain context in which there are other interests. In fact, the solution proposed by a pattern contains a type of structure which brings about a balance between specific interests or constraints to present the best solution for a problem. A good pattern:

- solves the problem - Patterns show the solution not guidelines or notions.
- is a proved concept - Patterns are proved solutions of problems.
- is not the evident and obvious solution - Many solution methods try to differentiate solution and primary concepts. The best patterns develop indirect solutions for a problem.
- describes a relationship - Patterns not only describe modules but also deepen system's structures and mechanisms.

Patterns are generally divided into three classes but because of wide adoptions from the book of "Gang of Four" (Gamma et al., 1995) and the fact that it is the first book on architecture patterns, the focus has been on a certain type of pattern (namely designing patterns) only. Every pattern in the field of architecture is called a designing pattern. In addition to these classifications in architecture, other patterns are presented in the field of software engineering and information technology (e.g. implementation through a specific programming language and information system integration) to help use successful experiences.

Every pattern has its own form which shows that pattern's specifications. The Gang of Four (Gamma et al., 1995) presented a form for architecture patterns which is widely

used and is called "GoF form". Of course, there are also other patterns available but mostly consent on some basic specifications which must be clear for the pattern in each form. These include (Gamma et al., 1995):

- **Name:** Every pattern must have a specific name. Pattern's name allows a word or a short term to be used to refer to the pattern, its specifications and the knowledge it associates with.

- **Problem:** A sentence describing the intention of a problem considered by the pattern. In other words, pattern problem concerns objectives and issues the pattern is meant to address in a certain context or force. Most pattern forces are in contradiction with objectives proposed in the problem.

- **Context:** A precondition under which the problem and solution seem to be repeated and the presented solution fits the problem. Context shows pattern application. In other words, context indicates an atmosphere in which the pattern is repeated and pattern repetition means the frequency of problem and pattern solution.

- **Forces:** A description of relevant forces and how they interact / contradict each other and their considered objectives. In other words, a force is an objective scenario applied as a reason for using the considered pattern. Forces show the problem's difficulty and define a variety of equilibriums to be considered in presence of forces and external stresses.

- **Solution:** Permanent relationships and dynamic rules describing realization of a good result. In fact, this indicates certain commands which describe how required working products are made. The descriptions may be associated with images, charts and explanations on pattern structure, its participation mode and

colleagues. They also show how the problem is solved. Not only the solution must describe a permanent structure but also the dynamic behavior. The permanent structure shows the form and regulations of a pattern but usually it is the dynamic behavior which resurrects the pattern. Description of pattern solution may indicate guidelines to be considered or avoided while implementing the real pattern solution.

- **Resulting Context:** System's position or configuration after pattern implementation including results of pattern implementation and other troubles and patterns which may occur in the new context. This step is sometimes called "forces analysis" since it shows solved forces and those not solved or unable to implement.

- **Examples:** One or more pattern applications in a certain context which show how the pattern is implemented and transmitted to the resulting context.

- **Rationale:** It is the justified explanation of steps or rules in a pattern and describes how and why the pattern, as a whole, solves its forces in a certain manner and in consistency with desired philosophy, concepts and objectives. The pattern rationale explains how forces and constraints coordinate to achieve a harmony. In fact, this section suggests how the pattern really works, why it works and why it is good. The solution component of the pattern may specify visible structures and pattern's behavior based on appearance, but rationale describes basic structures and key mechanisms of the inner pattern.

- **Relevant Patterns:** show dynamic and static relationships between this pattern and others in that language or system. Most relevant patterns have common forces, initial context and a text consistent with other patterns.

- **Known use:** As the name clarifies, this section states known uses of the pattern and its applications in available systems.

Consequently, the patterns help to reuse solutions and prior experiences instead of rediscovering them. In software development, various patterns are used such as design, architecture, process, and project pattern. The process patterns are used to make a process or a set of activities, actions, and work tasks in software development (Khaari & Ramsin, 2010; Tran, Coulette, & Bich Thuy, 2007; Zhao, 2010). Often, the process pattern used to make a generic process involves the common stages of development such as software design and, to make that, a set of common activities or stages and their relations should be identified and determined.

2.7. Summary

In this chapter, major concepts in release planning, e.g. release planning methods, requirements prioritization methods and resource estimation methods are discussed. In addition, an overview of software development patterns is presented.

- There are some major methods for planning various releases, including ad-hoc approach, planning game approach, incremental funding method, optimization-based techniques, hybrid intelligence approach and lightweight re-planning.

- Requirements prioritization methods are discussed and categorized in three major categories: Nominal scale, ordinal scale, ratio scale. In the nominal scale, techniques are assigned to various priority groups, and all requirements in a group have the same priority. Methods like numerical assignment, MoScoW and Top-10 requirements are in this category. Methods of the ordinal scale lead to an ordered and arranged list of requirements. Methods like Simple Ranking, Bubble Sort, and Binary Search Tree are in this category. The ratio scale contains

methods like 100-Dollar method, Analytic Hierarchy Process, Hierarchical AHP, Minimum Spanning Tree, and cost–value approach.

- Resource estimation methods focus on correct estimation of costs of a software production. There are two categories of methods: non-algorithmic and algorithmic. Algorithmic methods use mathematical models to estimate project costs.

- Software development patterns are used to show how a certain problem is solved through a specific solution. A good pattern can solve the problem, has a proved concept, has more than the obvious solutions, and describes a relationship.

In release planning problem, the process pattern can be used and help it in three ways. At first, the process pattern can break down the release planning problem to solvable sub-problems. Breaking the problem to well-defined sub-problems can lead to smaller and highly reachable solutions based on the circumstance (A. Jadallah, Al-Emran, et al., 2009). Defining the process pattern for release planning and identifying the set of activities and their relations can be a basis for improving the release planning process, and the method of breaking can be based on a planning process such as H2W and EVOLVE (A. Jadallah, Al-Emran, et al., 2009; Maurice et al., 2005; G. Ruhe & A. Ngo, 2004; Ruhe & Saliu, 2005a) or another new method. Secondly, a process pattern can help to improve using past experiences. In comparison to Mohebzada (Mohebzada, 2012) and Ruhe and Saliu (Ruhe & Saliu, 2005a) that used past experiences by recommendation system or direct human involvement in planning, process patterns use the more specific experiences, because the planning problem is broken down to sub-problems that are more specific. Thirdly, in each activity, best experiences (method) can be used as a solution based on specific parameters that make the process pattern more adoptable. Unlike most of release planning methods that cannot be justified or adopted

by project specifications, every activity of the process pattern can be adopted based on project specifications and can lead to a more adjustable approach.

In the next chapters, current release planning methods are discussed and used to define the process model of release planning. Also, requirements prioritization and resource estimation methods are used to identify parameters to customize each step of the process model. In addition, software development patterns are used to define release planning patterns.

CHAPTER 3: SOFTWARE RELEASE PLANNING IN INDUSTRY

One of the most important issues of release planning is to explore challenges and problems of the process in industries. Various challenges may occur in this field due to time, budget and many other constraints. Most of the challenges can't be determined or specified by systematic methods or available approaches. New challenges are faced by software developers from an applied and empirical viewpoint. In general, when-to-release decisions are often made ad-hoc based on business needs and project manager's experience (Ho & Ruhe, 2013). It can be said that the existing systematic methods are not efficient without this empirical look or at least marginal problems specific to a certain product or organization must be considered in software release planning. A variety of papers are published on the subject, each of which investigates challenges with a different look. Accordingly, this chapter contains two main sections:

The first section is an empirical study which explores available challenges in software release planning. This is a qualitative approach and its main methodology is composed of an interview with employees. In fact, the main objective is to investigate and find release planning challenges, which are classified into two groups: human-originated and system-originated. The second section studies the subject from companies' viewpoint and categorizes available methods of software release planning considered by most companies. It also examines how companies treat developing a new release from an applied and empirical view. Here, their experience plays an important role.

In order to validate our approach, multiple case studies were conducted. The data were collected using questionnaires. Direct interviewers have also been conducted with representatives of seven software companies with different levels of software development experiences. Results showed that experienced companies prefer to improve their existing software products rather than creating a new plan. The reason for this was the invaluable existing trust among clients or customers in their products.

3.1. Overview

Software development is a complicated process and requires careful planning to produce high quality software. In large software development projects, release planning may involve a lot of unique challenges. Due to time, budget and some other constraints, potentially there are many problems that may occur. Subsequently, project managers have been trying to identify and understand release planning, challenges and possible resolutions which might help them in developing more effective and successful software products. Although there are several approaches for release planning, which are used by software companies to generate new releases, companies are still unable to determine which exact methods are more suitable for their release planning.

Variety and significant increase of companies active in software release and consequently the quantity of products bring about new challenges and confusions. These challenges either have not been addressed by systematic approaches or have not been accounted as a dominant parameter. This chapter includes two sections:

The first section presents findings from an empirical study which investigates release planning challenges. It takes a qualitative approach using interviews and observations with practitioners and project managers at five large banking software projects in Informatics Services Corporation (ISC) in Iran. The main objective of this part is to explore and increase the understanding of software release planning challenges in several software companies in a developing country. A number of challenges were elaborated and discussed in this study within the domain of banking software projects. These major challenges are classified into two main categories: the human-originated including people cooperation, disciplines and abilities, and the system-oriented including systematic approaches, resource constraints, complexity, and interdependency among the systems.

The second part investigates the methods companies use to plan for new software releases and the approach they use. In this work, the current approaches used in the software industry have been categorized in order to select the most appropriate approach. In order to validate our approach, multiple case studies were conducted. The data were collected using questionnaires. Direct interviewers have also been conducted with representatives of seven software companies with different level of software development experiences. Results showed that experienced companies prefer to improve their existing software products rather than creating a new plan. The reason for this was the invaluable existing trust among clients or customers in their products. Doing so, they intend to improve their condition by increasing the reliability of the software produced. These companies generally prefer to use systematic approaches, when they come to decide for a development process. In contrast, newer companies with less experience prefer to rely more on human experience for their releases. Newer companies, therefore, are not able to foresee the future of the software market. These companies do not just rely on systematic approaches for their development, but they rather use the best available plans to produce good quality products.

3.2. Software release planning challenges in software development

Release planning is considered a company-wide optimization problem involving many stakeholders in which the goal is to maximize utilization of the often limited resources, of a company and turn them into business benefit (Ruhe & Saliu, 2005a).

Release planning can also be perceived as a decision for selecting important and necessary features for a new product. Since implementation of all the features is impossible in one release, we need to know which features should be implemented in the subsequent release and which one can be further postponed. If there is no proper or sufficient planning for a new release, 'critical' features might be delayed into the release late in the cycle which might subsequently affect the overall release schedule. The

potential effect might result in unsatisfied customers, time and budget overruns, and a loss in market share (Penny, 2002). Delivering software in an incremental fashion suggests increased customer satisfaction and reduction of many risks associated with delivering large software projects (G. Ruhe & A. Ngo, 2004).

Release planning for a new release of software includes assigning important requirements by investigating time, resources, budget and constraints. Software release planning is a complex task because many different factors must be considered in order to have good quality software, and project managers always face many problems for a new release. According to Ruhe and Saliu (Ruhe & Saliu, 2005b), the complexity of release planning is partly due to the incompleteness and uncertainty of the information that characterizes the problems. They discuss product release planning and use the word "endless" to describe the challenges in any software development. This means that in every software development project these challenges exist and there should be some ways to manage and mitigate these problems.

Without a proper software release planning, software projects are prone to fail because of the problems with new features that are not really necessary and urgent in a new release. Generating a new release is difficult to plan and becomes even more difficult with an increase in demand. Sometimes, a new release may have less efficiency compared to the previous releases because of misunderstanding of challenges in the process of coming up with a new release. This study explains the findings of an industrial qualitative study in Iran, focusing on current practice and challenges of software release planning in five large banking software projects.

3.2.1. Related Work

In large software companies, besides focusing on the main steps of software development such as requirements definition, analysis, design, implementation and testing, a plan for a new release is needed to make sure that the produced software is able to cope with new demands and is able to evolve gradually with the increased understanding of the developers and the users. The ability to be agile and aggressive in the development team is becoming necessary to ensure that the product is able to meet the ever-changing needs of stakeholders.

However, performing the release planning process is not simple. The tools and methods that are used to support release planning are very intricate and complex. Since we do not have sufficient understanding or even fundamental mechanisms that cover most of the problems that may occur in this process, there is a necessity for a study to be performed in this area. Empirical studies are a key way to gather information and move towards well-known decision (Perry, Porter, & Votta, 2000). Surveys, experiments, case studies, are examples of empirical methods that are used to investigate software development processes. Empirical study is an attempt to learn something useful by comparing theory to reality and to improve our theories as a result(Perry et al., 2000).

According to Ruhe (Ruhe, 2005), a release plan is influenced by several factors such as: types of requirements, implementation strategies, urgency for the client, value for the developing company, risk management and personal decisions. Investigation and evaluation of these factors and proposed algorithms which can help us in decision making is very important. There are also tools implementing these algorithms, a comparison of them can be found in (Saliu & Ruhe, 2005b). These methods are based on a number of variables to be estimated by experts. In its most basic form, customer value and cost are estimated (Ho-Won, 1998), while other works consider more parameters (Ruhe & Saliu, 2005a). There are varying definitions in the literature on

what constitutes the release planning problem. Ruhe and Saliu in (Saliu & Ruhe, 2005b) have provided a set of key aspects for release planning methods to be able to compare and understand them. Their paper describes ten technical and non-technical aspects that are significant to provide an impact on release planning process. These aspects are a useful guideline for us to evaluate our challenges as well as to identify ways to overcome them. There are also various methodologies which aim at detecting release planning problems from industry and academic research which are categorized in (Ruhe & Saliu, 2005b). Saliu and Ruhe (Ruhe & Saliu, 2005b) discuss current challenges in release planning, main characteristics of a release plan and present a form description of a release planning process.

According to Carlshamre (Carlshamre, 2002), there is always a possibility that problems occur for a next release which are not predicted and are different in each software development project. Carlshamre (Carlshamre, 2002) has classified release planning as a "wicked problem". The concept of a wicked planning problem was first introduced by Rittel and Webber in (Rittel & Webber, 1984). Wicked problems are difficult to clearly define and there is often no clear-cut solution to wicked problems. For this reason, only a systematic approach can be used and we need human ability and experience of professional practitioners in the world of software as well. Ruhe and Saliu have discussed release planning from two different dimensions: art and science (Ruhe & Saliu, 2005a). The art of release planning refers to human and his capabilities and the science refers to the algorithms and methods. Based on their understanding, Ruhe and Saliu have designed, implemented and evaluated a support tool for release planning as a means of developing a rich understanding of the task.

Ruhe and An Ngo-The in (Ngo-The & Ruhe, 2009) proposed a systematic approach for solving the wicked problem of software release planning and a new method EVOLVE+ for decision support for software release planning.

To facilitate the release planning process, Wohlin and Aurum have recognized the importance of 13 criteria used in deciding when to include a software requirement in a release (Wohlin & Aurum, 2005). They show the motivation for the criteria and that there are indeed some criteria that are more important than others in the decision-making process when deciding which requirements to include in a specific project or release. Their work concludes that business and management criteria are ranked higher than system criteria and that this is not an indication of this area being less important, rather that there is a need for better tools and methods for addressing these issues. In another related research, the importance of software architecture in release planning process is investigated and release planning process is discussed (Lindgren, Norstrom, Wall, & Land, 2008).

Based on the noted works in this area, this research focuses on a deeper understanding of the release planning process. Hence, it intends to carry out an investigation to identify challenges and problems in release planning process. The main objective of the study is to explore the release planning problems and challenges in banking software projects specifically in Iran. The findings might be useful for others to make comparison and analysis with the current understanding of release planning process.

3.2.2. Research Design

Due to the fact that there are many problems and challenges in developing a new release in software projects, this research has focused on challenges associated with five large banking projects. Investigation of these challenges needs a proper and comprehensive study on software projects. A qualitative case study is performed to understand and identify challenges in banking software projects. Qualitative research methods are useful when the purpose is to explore an area of interest, to obtain an overview of a complex area, and to discover diversities and varieties rather than similarities (Robson, 2011). Qualitative data sources include observation and participant observation

(fieldwork), interviews and questionnaires, documents and texts, and the researcher's impressions and reactions (Myers, 2009). It is also preferable to use a qualitative approach when the aim is to improve the understanding of a phenomenon about which little is known. This is due to the fact that the case study focuses on gaining in-depth information (Hoepfl, 1997). The quality of a qualitative study relies on the quality of the investigator (Robson, 2011). For this study, we interviewed 27 experienced software developers, analysts and designers and 5 project managers. Release challenges and problems with current release cycles, have been identified. This study consists of three steps which are described in the following figure.

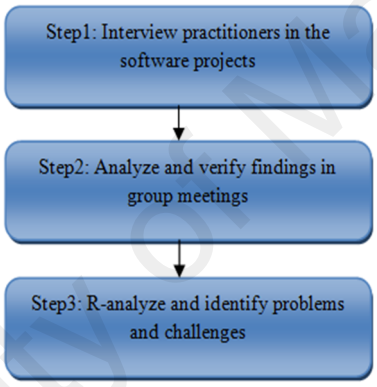


Figure 3-1: The Research Procedure

- **Step 1: Interview practitioners in the software projects**

  In the first step, semi-structured interview (Robson, 2011) and direct observation for data collection that included introductory and technical questions were performed with project team members. This was done through discussion among the interviewer and the interviewees. For each project, at least five persons attended the interview, but the number varies in each project. The interviews varied between 70 to 80 minutes in length. All their ideas were transcribed and later sent to them by e-mail to be approved and verified. Summary of interview questions are available in table A in Appendix.

- **Step 2: Analyse and verify in group meetings**

  The second step was investigating and analysing data collected from step1 in three long meetings with four project managers. Each meeting lasted between 120 to 180 minutes. In fact, the purpose of these meetings was to collect additional information, as well as to verify and confirm the information gained from the practitioners in the first step. Another objective was to obtain feedback from project managers' views. In this step, project managers' experiences were also of great value to us.

- **Step 3: Re-analyse and identify the challenges**

  In this final step, we had a comprehensive view of all projects and their characteristics. All data received in two previous steps were re-analysed and challenges were identified. We found 12 challenges that were faced in the new release process and were very important to all interviewees. All the staff agreed that these challenges existed in their projects. Summary of the conducted interviews is shown in Table 3-1.

Table 3-1: Summary of the conducted interviews

Project Name	Interviewees	Interview type	Additional information
Damoon	Project Manager	Meeting	-
Damoon	System Analyst, Developer	Individual	Sending e-mail to get approvals
Saba	Project Manager	Meeting	-
Saba	System Analyst, Developer	Individual	Sending e-mail to get approvals
PKI /CA	Project Manager	Meeting	-
PKI /CA	System Analyst, Developer, Security Manager	Individual	Sending e-mail to get approvals
EXIMBILLS	Project Manager	Meeting	-
EXIMBILLS	System Analyst, Developer, Designer	Individual	Sending e-mail to get approvals

Project Name	Interviewees	Interview type	Additional information
Islamic Loan systems	Project Manager	Meeting	-
Islamic Loan systems	System Analyst, Developer	Individual	Sending e-mail to get approvals

3.2.3. Software Projects

This research was conducted in five banking software projects in Informatics Services Corporation (ISC) Company in Iran. ISC was established in 1993 and a new phase of renovation and modernization of different sections in the banking system started in this company. In this section, projects were selected based on the team size and the number of releases in each project. These projects have had at least 8 members and developed at least 2 releases. Table 3-2 shows the projects descriptions as well as their applications. The old deficient structure of banking system was transformed to an efficient new one through the executive and technical power of ISC.

Here is an overview of the five projects that are still under development and have produced a few releases so far:

A. Project 1: Damoon

The PGS (Payment Gateway Solution) system is for both internet shops and offline shops. In case of an internet shop, it allows to set up a work place for credit card payments operator. When used in offline stores, it would be a good replacement for a regular POS machine for more modern payment terminal. The main functionalities of the systems are:

- Credit card transactions with manual data input using the keyboard;
- Credit card transactions using the second magnetic strip (provided that the PC has a reader connected to it);

- Getting cardholder information from PGS SYSTEM server provided that the cardholder is registered with PGS SYSTEM or has purchased earlier paying through PGS SYSTEM;
- Processing transactions data locally without requesting from the PGS SYSTEM servers.

B. Project 2: Saba

Internet banking systems (Saba) is a web-based Banking Application developed by ISC, on an Intranet/Internet environment. An intranet/internet banking application could be part of an e-banking application. In e-banking applications, the users can perform their banking needs through different kinds of channels, such as Mobile, ATM, POS, etc. internet banking systems. Currently, it can interact to any Retail Banking Systems and deliver appropriate transactions.

C. Project 3: PKI (public key infrastructure)

At present, many on-going processes are transaction-oriented. Transactions in the broadest sense of the word cover the elementary messaging for processes like Internet banking, B2B and B2C exchanges, online notary services, e-invoices, online tax declarations, etc. All of these business processes can be handled faster and more cost effectively electronically. Not only for security and privacy reasons but also because of government regulations, it has become increasingly important to guarantee the authenticity and integrity of these transactions by means of digital signatures. Performing transactions electronically can be a huge cost saver compared to traditional paper based procedures. It can also yield better and faster results and thus lead to higher customer satisfaction. Several industry and government initiatives stimulate businesses to increase the level of automation in their internal and external processes. Well-known examples are Bolero.net for global trading and Identrus and Swift TrustAct for global e-commerce.

On the government front, the European Council has prepared a new directive on VAT and invoicing that enables companies to replace paper based invoicing with exclusive electronic invoicing even for cross-border transactions. Electronic signatures are explicitly mentioned as one of the means to implement such a system. Although less fancy than other applications, e-invoicing is easier to implement and yields an immediate and predictable return on investment.

D. Project 4: EXIMBILLS

Trade Finance Systems (EXIMBILS) is an integrated system that audits and automates the complete cycle of trade finance transactions, in real time and in accordance with SWIFT and UCP standards. This allows for a rapid yet comprehensive installation for banks wishing to implement on a strict time scale. The system is able to save bank's time and money by straight processing to automate the creating of records from incoming SWIFT messages, passing accounting entries, producing customer advices, and making payments with little or no user intervention.

E. Project 5: Islamic Loan systems

Islamic Banking is now well into its stride and there is no longer any doubt that it has earned a respected place in the world of banking. Some leading international organizations, including the International Monetary Fund, have carried out extensive research in order to understand and evaluate the characteristics of this newly established banking practice. Islamic Banking, which bases itself on the principle of fair profit sharing and claims to be the most stable method of banking, provides a range of deposit and loan products to its customers.

Although there are similarities between Islamic Banking practices and the traditional western banking methods, the principles lying behind Islamic products as well as the technicalities involved in the day to day business are entirely different. This stems from

the fact that Islamic banks do not regard their customers purely as their creditors or debtors. An Islamic bank customer participates in all investment activities of the bank and shares the profits as well as the business risks involved. A certificate of deposit is a document of participation and investment and not purely a debt-reclaim document. A loan granted to a customer is regarded as an investment, which involves risk, and not a debt burden on the customer.

The Islamic Loan Products cover a range of customer financial requirements including consumer loans, commercial loans, mortgages, corporate loans and investments. They are divided into some major categories, which may differ from bank to bank although the main principles and procedures remain unchanged. The main categories, referred to as "*Aghd*", are "*Ghard-Ol-Hassane*", "*Mozaribbe*", "*Morabehe*", "*Joale*", "*Moshareka*", "*Ijare*", and "*Salaf*".

The widespread use of Islamic Banking both in Islamic and non-Islamic countries has created demand for computer based systems supporting Islamic Banking procedures. The traditional retail banking systems, which mainly cover the western banking products, do not encompass the requirements of an Islamic bank. The deficiencies of the existing software systems have forced Islamic banks to invest heavily on improving the traditional systems. However, this has not proved successful mainly due to the nature of the Islamic Loan products. On the other hand, modern international banks with local presence in Islamic countries have recently shown an interest in Islamic transactions and products. However, their limited knowledge of Islamic banking has hindered a powerful competition with local banks. The software projects and their application areas are introduces in Table 3-2.

Table 3-2: Projects and their application areas

Project Name	Number of project members	Application domain	Number of releases until now	End users	Long-term planning
Damoon	8	Internet payment systems	2	Banks and financial institute	Yes
Saba	15	Internet banking systems	6	Banks and financial institute	yes
PKI /CA	12	Public Key Infrastructure	2	Central bank	yes
EXIMBILLS	18	Trade finance systems	2	Banks and financial institute	yes
ILS	26	Retail banking systems	9	Banks and financial institute and central bank	yes

3.2.4. Challenges identification

In this study, the objective is to find and understand problems and challenges that team members and project managers usually face in the process of release planning. As it was mentioned earlier, this study includes three steps. In the first step, we had several face-to-face interviews and discussions with the team members, e.g. developers and analyzers, those who were inside the system and those who were in close contact with the system. In this step, we interviewed 27 interviewees and issues about development process and possible problems faced in release planning were discussed. Then, interview transcripts were refined and e-mailed to all the interviewees to be verified. During the interviews, two researchers were involved to ensure that the researchers are able to verify their collected information.

In the second step, interviews were conducted in three meetings with five project managers in which researchers verified the data collected from the first step and tried to identify release planning challenges related to the outside environment of the system. Besides, discussions were done on the requirements for durability and sustainability of

the systems. In this step, experiences of the managers and their concerns about the future of the systems were identified and illustrated in Figure 3-2.

The final step was about re-analysing and investigating the findings of the two previous steps to identify the challenges shown in Figure 3-2. For the identification process, all related transcripts for each step were compiled and arranged in a readable format as seen in afore-mentioned table.

Having an accurate project deadline means schedules can be planned. One of the most important factors both for the team members and the managers was discussing the deadlines for the future releases because it cannot be decided easily and estimated with accuracy and this estimation can trigger problems for the personnel. Sometimes, the managers were very strict in deciding on the deadlines because of the pressures from the organizations. They believed that the time of a new release was of importance because of the tough competition and the high demands of the stakeholders.

Proper communication and coordination among the members is significant in successful software development process. In the meetings with managers, they mentioned that communication and coordination are among the most important elements for the success of the product.

Chatzoglou in (Chatzoglou, 1997) discusses that a lack of resources i.e. people involved, time and money as the requirements of the activity. Lack of resources including human or financial was also discussed in meetings, but the discussions were more on the human resources. The survey by Hall et al. (Hall, Beecham, & Rainer, 2002) also confirms that lack of skilled personnel is one of the organizational problems in the process of software development. Lack of experienced and know-how personnel has a great impact on the release time in the future and sometimes it has been observed that because of this problem the new release may be delayed for a few months.

Ruhe and Saliu (Ruhe & Saliu, 2005a) mentioned that complexity is one of the difficulties in release planning. Complexity and constraints of the system are of the factors seen in every software development project and members know that there is no systematic approach to solve it. In this study, during the discussion with the members, creative and experimental approaches were suggested as possible ways to tackle these problems. Most of the managers agreed that they could not avoid the inherent complexity of the software system and the complexity can result in delays of the future releases.

Another challenge is in terms of the interdependency of the systems and the related sub-systems in a web-based project, especially banking projects. This is one of the main topics discussed in the interviews and meetings since a few of these systems are almost ready to be utilized but are delayed to be installed due to their non-adaptability with the related systems.

Identification and discovery of significant new features for a system which would be the base for new releases was also one of the major challenges to the team members and managers. There is always this question that how we can spot important features and prioritize them for future releases. Managers defined systematic approaches and tools as the basic way to cater this question. In all projects, some were worried about changes in strategies and lack of clarity in the goals. Sometimes, the pressure on staff for generating a new release was regulated due to the importance of the project, and sometimes the same project was halted for a long time. Many of these strategic changes were devised by the top level managers based on certain organizational policies which the team members were not aware of and may result in problems for the new release.

Ruhe in (Saliu & Ruhe, 2005b) has mentioned stakeholder involvement as one of the key aspects for release planning. Although there are many stakeholders who are

interested in the content of the release, the definition of stakeholder is different for different organizations. These types of different definitions will be further discussed in the next section.

Most of the time, manager is the main person in charge and has the full responsibility to think about the future of the products. Their concern is that the new release should have better quality and performance. Therefore, they need to foresee the future of the product and decide whether the new release can cover all the inefficiencies of previous releases. In our meetings with the managers, they mentioned that they usually paid special attention to the process of the development of projects. Most of the managers already had regular weekly and monthly meetings to evaluate the tasks of the personnel so that they are able to keep track of the project's progress. They should be the first one to know if anything goes wrong with the project. The following figure shows the topics discussed in each step.

As it was previously shown in Figure 3-1, the first step is the interview in which challenges in projects are identified first, and then analyzed and verified in meetings with managers. Finally, these challenges are re-analyzed and categorized for every project.

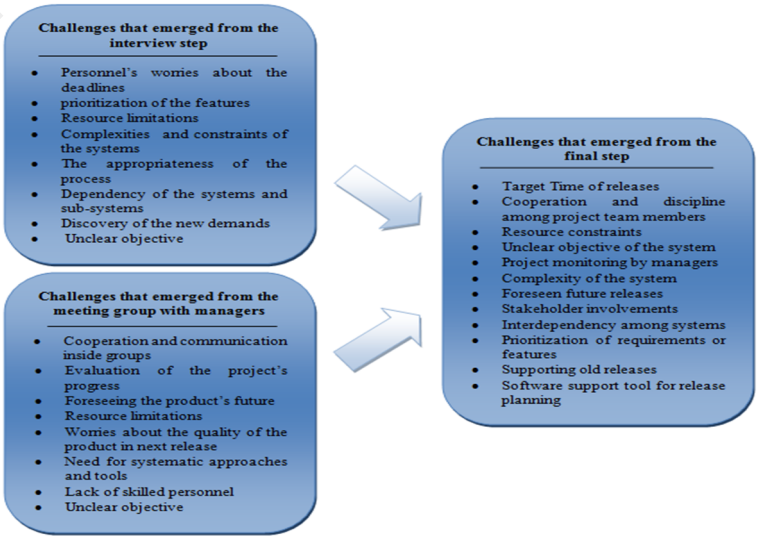


Figure 3-2: Challenges during the discussion, interview and meeting

3.2.5. Challenges identified in industry

A satisfactory release for software can be attributed to a well-organized and planned process. Software quality can be achieved through identification of real software defects and adding suitable features for the new release. This section presented the challenges identified during re-analysis in step 3. The twelve challenges presented in the following section are the findings from the discussions and the analyses made in the study. Projects and all their characteristics are available in Appendix A, Table B.

3.2.5.1. Target time of releases

One of the most important questions that project managers are challenged with in release planning is when to release the next software version. The time from when the software is conceptualized until it is being available for new version is important to be planned to ensure that the software is not outmoded in more than one release. This time refers to the time needed for a new release of product or project, and setting this period of time effectively is a particularly crucial ingredient in a successful release plan. The challenge is to determine an acceptable time of release for a project.

All the interviewees were mainly concerned about time scheduling and one of the developers mentioned that he always faced problem with the amount of time allocated to him to finish his work. Setting time for release planning can be with fixed intervals or flexible ones. For some projects, this time is fixed and pre-determined and in others, it is flexible or based on new demands or the condition of the project.

In Damoon project, the time for release is fixed and is determined twice a year. Based on the new requirements of the users, a new release is provided. In Saba, the release time is considered crucial and it is determined to be three times a year. Three new releases have been provided annually and so far they have had a total of 6 releases based on their customers' requests.

The release time for PKI/CA is fixed and is once a year. Its project manager intends to concentrate more on security aspects for each new release, because security is one of the most important considerations in this type of projects. In EXIMBILLS, the time of new release is flexible and depends on many factors. Creating a new release for EXIMBILLS is based on new functions and new requirements of banks and Swift organizations. There are many functions planned in this system that must be implemented in the future. In the ILS, a new release is flexible due to the changes in rules and regulations. For this project, they have already made 9 releases. To set the target time of releases is so much dependent on many realistic factors of the projects. Hence, the manager has to be aware of and sensible to the project they are handling.

3.2.5.2. Cooperation and discipline among project team members

During the group meetings, one of the shared concerns among project managers was the lack of proper cooperation and discipline among different members of a project team. Some of the developers show little interest in making a new release. The developers may not feel the need for a new release directly because they are not the main users. Most managers agree that if their team members or the developers are not able to work together and harmonically, they may have difficulties in meeting deadlines. Many problems may potentially arise due to the team disability to work together. Therefore, trying to improve on the cooperation and discipline of the team will be one of the important and crucial challenges in release planning as well.

In Damoon project, the developers did not get on very well together because of insufficient information about the requirements. Hence, to improve the situation more regular meetings with project managers were conducted to obtain more information about these requirements. This makes it possible to build a better relationship amongst them as the project progresses. In Saba project, the cooperation among developers is well-maintained due to the clarity in the definition of requirements. During the

interview meeting, the project manager of Saba told us that Saba is one of the successful projects at ISC and he is satisfied with all the members' work and behavior. He attributed the project success to the cooperation and the discipline of the project team members. In this project, the number of releases is 6, as it is shown in Appendix A table B. In PKI /CA, the cooperation and discipline of development team is satisfactory. This might be due to the fact that the project is new and the team is following the manager's instructions accordingly. In EXIMBILLS, enthusiasm among the members was not satisfactory and the work flow was very slow due to the lack of qualified staff which led to employment of new members. As the literature indicates, new members take some time to get used to the company's procedures and methodology. This slows down the software development process as a whole. IILS project is having a similar problem as in EXIMBILLS but for a different reason. In IILS, the problem is that changes during the construction phase are time consuming. It is a large project with a large number of staff and cooperation among all of the members is not that well. The management is taking extra effort to ensure that the project team members are able to work cooperatively together. At this point in time, ILS project is still struggling with the problems and still looking for ways to increase cooperation among its team members.

3.2.5.3. Resource constraints

One of the main issues that all of the interviewees complained about, was the problem of resource constraints. If the needed resources are available in abundance, the project duration can be shortened to achieve a new release. On the other hand, if the needed resources are severely limited, the project is more likely to be delayed. When a new requirement or feature is decided and planned for a next release, many constraints like time and effort must be faced and planned. Resource constraints are clearly a key aspect of release planning (Ruhe & Saliu, 2005b), since without considering resource constraints the consequence would be an unrealistic release. In all these projects, there

were no serious financial constraints, because most of the customers were banks and government institutes but sometimes payments to the client companies were delayed due to some avoidable circumstances. In Damoon and Saba projects they face expertise constraints. The projects have difficulties trying to find required expertise in the area. Project managers believed that they were "always behind technology" in these two projects. In EXIMBILLS, there is always the risk of being behind the new version of the system software, because EXIMBILLS is a new Trade Finance system for Iranian banks and it is not yet fully understandable in their requirements and directions.

In the PKL/CA project, the project manager perceives that the project's security aspects are hard to attain and achieve. Thus, the project manager is always willing to increase his investments to improve the overall security aspects of the system. Unavailability of the new technology was one of their problems in this area as well. In ILS because of the complexity of the systems, every change needed a lot of budget and time, either from financial or human resources aspects. The project manager is scared of new changes and sometimes tries to keep the old system. Developers always feel that they are working in an old technology environment and they wish either to change these old technologies or leave the project.

3.2.5.4. Unclear objective of the system

The objectives as stated in (Saliu & Ruhe, 2005b) describe the desired properties for a product, or stated differently, the goals of the product. Sometimes these objectives are related to a project strategy, features, content, quality, aims and satisfaction. In many large software projects, the ambiguity in the objectives can lead to many problems in generating releases. Unclear project goals and objectives, and frequent change of the objectives during the project are key factors in failures for release planning.

In Saba, the managers, initially were not sure of how secure their system would be. The reason was that the system was supposed to be the first Internet banking solution to be used in Iran and there happened to be many new changes which were unpredictable and unplanned for at the beginning of the project.

The bank which will be using the system is actually the largest bank in the country with over 40 million customers. So, many uncertainties and worries arise around the project that can lead to a poor progress. Like Saba, Damoon faced some changes in objectives which were not planned before. In Trade Finance (EXIMBILLS), all operations were performed manually in Iran before implementing this system. Therefore, they always fear the risk of customer dissatisfaction or reactions to the system. At this point, the project is expected to face many changing objectives which might be driven by the customers' responses to the system.

The stakeholders of ILS project have so much concern about its return on investment (ROI). At the same time, the project has many requirements which are changing regularly and the rules and regulations set by CBI are constantly being modified. Therefore, the project management has to endlessly put lots of man/days effort to ensure the project is able to meet the demands. ILS project eventually managed to break even financially this year. In PKI/CA, the security risks are always the main issue in the system, as the project management is not very sure how complete the project's security requirements are set up.

In general, it can be observed from the projects that frequent changes and unclear policies and strategies of the system can cause hindrance and difficulties in the process of development of future releases.

3.2.5.5. Project monitoring by managers

One of the main concerns of the managers in all these projects is monitoring the progress of the projects. It is crucial for project managers to have an accurate progress report to make release planning successful.

Almost all the project managers believe that project monitoring would have a significant effect on the quality of the new releases. The important element is the ability to identify or recognize a problem in software development process. Once a problem is detected, it can be tackled and it can be no longer present in the new release. If the monitoring is done properly and thoroughly, achieving the final goal would be much easier.

In all projects, after constructing a Gantt chart, the project managers is responsible to update the tasks and if any of the tasks are behind schedule, the required resources are called to overcome the shortfall.

The monitoring process in Damoon and Saba was taking place on regular weekly basis, with the exception that in Saba the resources could be modified according to project needs. In PKI/CA, that process was done regularly on a monthly and occasionally every two weeks. In EXIMBILLS, since it was a new system and the system's main structure was not defined yet, there was no fixed schedule for the monitoring or reporting process. In ILS, the monitoring process was regular and was performed once a month. In short, project managers monitored the work progress in order to evaluate the flow of the project under development, with the aim of improving future project functionalities. The managers emphasized that project monitoring was a challenge and the monitoring process helped them tremendously to plan more easily for the next release.

3.2.5.6. Complexity of the system

One of the important hindrances that can delay or cause problems in large projects for delivering a new release is the complexity of the system. This complexity can be innate and is usually seen in most large software projects. Most project complexities cannot be eliminated completely and can only be reduced. Sometimes, technical constraints can also cause complexity. Technical constraints refer to any of a number of technical issues and obstacles that will impact the new release. For example, a company might be trying to connect many banking branches to a central location via links and this can produce complexity to the system. Size of the project is another concept that affects the complexity in each system, because some projects may have hundreds to thousands of features.

In Saba project**,** the complexity of the system increased due to the need to connect the application server to the mainframe running on COBOL/CICS/IMS environment. Project managers strived hard to decrease this complexity by using the IBM CICS Transaction Gateway (CTG).

This connection problem was also observable in Damoon. In Trade Finance**,** no big complex issue was found in the system as the platform was on PC environment and the connectivity to mainframe was always on batch mode and via file transfer (FTP), but the swift messages in EXIMBILLS were not received on time. The complexity in the ILS was in its data base. They had two choices; one was to use the existing IMS and the other was to use a better and new engine such as Oracle, DB2 or Informix. Eventually, they decided to use the DB2. In PKI /CA**,** the complexity was the construction of the security room for their system as the room must had been designed in a particular setting and arrangement with specialized software and hardware platform with high level of security in mind. As it was a new platform they always felt the risk of things not going according to plan. PKI /CA is one of the largest projects in Iran with a lot of

requirements and new demand features, and this cause the project's complexity. This complexity is expected to delay the new release for a few months and even a year. For this reason, an innovative solution to decrease these complexities is required.

3.2.5.7. Foreseen future releases

Most software projects in long-term development process require new features or requirements that cannot be implemented in one release and must be considered for several future releases. In this case, pro-activity is needed to ensure a successful release planning in the future and it is advisable to have a plan for later releases. It has been observed that planning for only one release (i.e., next one) is usually not enough (Carlshamre, 2002).Sometimes, stakeholders' features may not be considered in the next release and a planned schedule is not available. This may result in dissatisfaction, so it is advisable to plan in advance for two or more releases to provide a clearer picture to the stakeholders.

Hence, necessity of current release management that is able to predict the issues and requirements in following releases arise. For example, there can be customer requests that can impact a release project, such as, the need to add more features or functionalities required by a specific customer.

In all projects, planning for the future is considered difficult for project managers due to many uncertainties in the industries and, more specifically, banking projects are greatly affected by customer and banking demands. One of the professional developers mentioned the uncertainties by saying that their prediction is subject to change every day. For Damoon project, the plan for one release ahead is always on calendar and the stakeholder knows about the details of the next release. For the personal Internet banking solution, Saba has also one release ahead and similarly the stakeholders know the details of the coming release. In PKI /CA, as the project only has one release a year,

the details are usually set by CBI and that is a well-known fact to all. Due to the flexibility and the newness of the releases in EXIMBILLS, no exact calendar is set and the project rolls out the releases whenever the timing is considered right. This situation is true for ILS too because of its flexible nature. However, most managers agree that it is much easier to plan for releases if the schedule is set ahead of time or there is a set direction towards producing several releases within the specified time.

3.2.5.8. Stakeholder involvements

A stakeholder is a person or group of people that may significantly influence the success of a project. It is clear that the stakeholders are interested in having their ideas being considered in the contents of a release. Thus, the presence of the stakeholders and trying to attend to requirements is effective in a new release. For all these projects, there are three types of stakeholders involved.

The Steering Committee meetings involve three people from the customers' side and three from developer's side. The meetings take place once a month and the strategic planning and resource planning for the projects are usually discussed there.

The Operational meetings usually take place once every week and in these meetings they discuss how to finalize the new requirements and how to reach the deadlines.

The Technical meetings usually happen whenever is needed and sometimes twice a day. In these meetings, they discuss the technical details of the requirements with the customers and business analysts which are usually from the customers' side. As illustrated in Appendix A, the main customers for these projects are banks and governmental organizations. Damon Project is an Internet shopping project with Saderat, Melli and Mine and Industry Banks as the major clients. These banks use the users and customers' viewpoints to improve the system. Saba project, an Internet Banking project in terms of banking transactions that are performed via a secured

Internet application, is running in Melli, Export and Development, Mine and Industry, and Saderat banks. Central bank of Iran with the most foreign transactions in the country uses PKI/CA to do so. EXIMBILLS which is foreign currency software is used by Saderat and Melli banks for their international transactions and trades. It can be said that all the foreign currency transactions in the country are done by these two banks and this system exclusively. Melli, Export and Development, Mine and Industry, and Saderat banks use ILS for their financial requirements.

In fact, there are generally three levels of stakeholders who are important to be considered as inputs to future releases. The first ones are developers and creators of the system. The second ones are banks and their experts whose opinions are very important. The third level is the customers and users of the banking services. Customers of each bank are shown in table B of Appendix A. Hence, it is critical to ensure there are sufficient involvements of stakeholders in the project development. The involvement will not only ensure the valid requirements have been understood but also enables better planning for the progress of the project and more specifically better plan for future releases.

3.2.5.9. Interdependency among systems

Many web applications are interdependent to each other. When looking after older applications or creating new ones, it seems very difficult to synchronize a system which relies on other systems. For example, a finance system depends on the wage and salary system, etc. Therefore, one of the important and considerable issues in a new release is to fully investigate and understand the relationships among systems and the related sub-systems. Most of these systems and sub-systems have to work integratively within the new release.

In Damoon and Saba, the interdependency among systems is high due to online/real time connections and interfaces to other systems such as card and core banking system. So, the understanding of these relationships and their mapping to the new release is of high importance.

In PKI/CA, the dependency degree is low due to its being a closed system. In fact, these systems are somehow independent and their transactions are not related to other systems. In EXIMBILLS, the dependency is medium. Only one interface is in existence and that is to General Ledger (GL) system. Dependency in ILS is high because the existing interface is with five other systems which cover customer financial requirements including consumer loans, commercial loans, mortgages, corporate loans and investments. Therefore, we can conclude that most large software projects which are related to other systems and sub-systems require full synchronization and adoption to those systems for generating a new release. This is actually a great challenge to project managers in terms of systems understanding and cooperation of various entities of the system. Producing a new release in this context usually requires not only cooperation from the technical groups and managerial people of the system but also full support from the users of the system.

3.2.5.10. Prioritization of requirements or features

Prioritizing requirements can be seen as the process of deriving an order relation on a given set of requirements, with the ultimate goal of obtaining a shared rationale for partitioning them into subsequent product releases (Perini, Susi, & Avesani, 2012).

A project manager has to balance the project scope against the constraints of the schedule, budget, resources, and goals. One balancing strategy is prioritization to drop or postpone low priority requirements to a later release when there are new, higher priority requirements. Therefore, it is very important to decide what the prioritization is

based on. Different prioritization techniques can be used in different projects depending on different parameters. In release planning tools, there have been a few techniques used for prioritizing the requirements. Some comparisons are made in (J. Karlsson et al., 1998). Requirement prioritization is used in software release planning for assigning which candidate requirements of a software project should be included in a certain release. When customer expectations are high, time is short, and budget is limited, you want to make sure the product contains the most necessary features only. So, it is important for managers to prioritize what to include in a next release. The team must collaborate on requirements prioritization. Damoon, Saba, EXIMBILLS and ILS are Customer centric. They allow the customers to dictate the priorities for the projects' requirements. These projects have many customers or end users for their banking operations, so the customers' demands are high and the necessity of prioritization is considered important.

PKI/CA project is more government centric. The government always has the upper hand in dictating the priorities. This system is crucial for Central Bank of Iran and hence they have the first word in setting the priorities. The project manager mentioned to us that usually during the meeting with the Central Bank, the bank will instruct them on what to do and the development team has to follow the orders obediently.

3.2.5.11. Supporting old releases

One of the issues that always worry project managers is the capability of a new release to support older releases. Most of the time, it is expected that a new release is expanded to cover all of the previous releases.

However, there are occasions that the new releases are less efficient than the older ones and the users might find out later on and demand to use the old releases. Therefore, managers are always striving to have the best possible features in the last release.

Usually, a new release is produced when there are many requests or requirements made by customers on the product. As a result, the teams may suggest bundling the appropriate features together and then constructing a new release to be deployed. On the other hand, according to the project managers, whenever there is a new release many possibilities might occur even though many testing and quality assurance procedures have been performed. The most concerning issue is to ensure that a new release always supports old releases.

3.2.5.12. Software support tool for release planning

Release planning is a complex process which needs intensive human expertise and knowledge. It includes many demanding tasks like resource estimation and setting objectives in release plan generation and decision making. These tasks altogether call for an intelligent tool support that would be of great value to a project manager who is making release decisions.

Most project managers agree that the whole process of preparing, constructing, resource allocating and so on are very formidable tasks that need to be well planned to be executed. Most of the time, they do not have a proper tool in order to assist them in these difficult operations. Most managers are looking for some support tools to assist them in this process. Many of them believe that software tools might give them extra advantages to possibly create a more effective plan for their releases.

3.2.6. Discussion and validity of results

This section is divided in two parts. The first part presents our experience and view from qualitative research in banking projects and the second part discusses the threats to the validity of our research.

The qualitative research approach is usually used for the investigation of social phenomena or, in other words, situations in which people are involved and different

kinds of processes take place (Dyba, Prikladnicki, Ronkko, Seaman, & Sillito, 2011). In software engineering that includes various domains for developing a software product, evaluation of every domain can give us some new insights and experiences. Thus, it is advisable to always go searching for new knowledge even though some of these findings cannot be generalized for all situations. In this study, banking software projects were investigated and a group of people were interviewed with regard to their experiences. We performed two rounds of interviews to provide us with different points of views and at the same time to increase the reliability and validity of the study. After performing the interviews and data collection, the data transcripts were almost 60 pages. The 12 challenges which were identified were later categorized into two main categories as seen in Figure 3-3. The first category is referred to as human-oriented challenges, which are related to stakeholders, customers, their duties and their cooperation, and the way the tasks are done. The second category is referred to as system-oriented challenges that are related to problems and the issues of the developed system, e.g. limitations and complexities. The duration of the time for a new release, which was introduced as the first challenge, can be grouped into both of these categories.

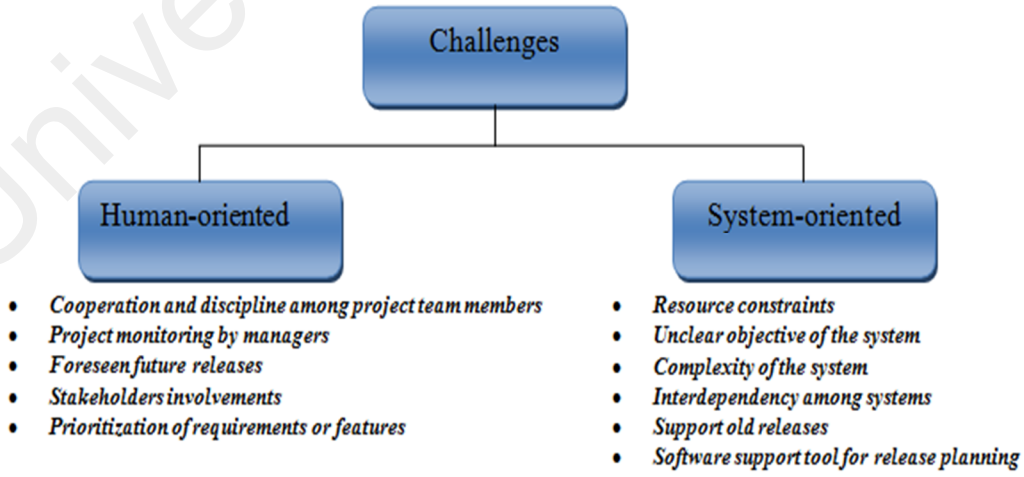


Figure 3-3: Categorization of the challenges

As we observed earlier, nearly all the challenges found in software development projects come under these two groups. In the first one, which is related to human,

people and their attributes have a significant impact on software release planning. Some of the roles which are performed by humans involved in release planning, like functional analyst, development lead, and quality assurance are very complex. Tasks in which they perform require innovation and previous project experience. As we acknowledge, innovation is a difficult ability to be measured and to be defined. Hence, these tasks which are essential in release planning are obviously difficult and might suggest to the research community to perform more studies to identify ways to facilitate these activities or tasks. From another aspect in the second group which is related to system, many challenges with regards to inherent qualities of the system and the environment are identified. Similarly, system and environment understanding requires a more systematic approach and support tools to be used. These two categories of challenges might lead the practitioners and the research communities to explore more opportunities and ways in order to overcome these challenges in the future with the aim to produce better quality product releases.

When a researcher performs a qualitative research, he or she must pay attention to the validity of the research. There are many different ways of establishing validity, including: interviewer corroboration, negative case analysis, and conformability. Most of these methods were described by Lincoln and Guba in (Lincoln & Guba, 1985).

Validity of the result needs to be examined by introducing proper counter measures. We have followed the recommendations by Yin (Yin, 2003) where he has chosen four possible ways for the validation. The first one is referred as construct validity. In this case, this study used two main researchers for interviews and discussions in order to reduce misconception or misunderstanding of the information gained from the interviewees. After each interview, both researchers and interviewers sat down to discuss the data and information gained. Resolution in terms of terminologies or words used by the interviewees was discussed. Data were collectively gathered and organized

by both researchers. Unclear data were also discussed and resolved before the next meeting with the interviewees.

The second important measurement was trying to demonstrate the internal validity of the study. There was always a meeting with project managers to confirm information from step 1. In this second meeting, the interviewees (developers) were not invited. This was purposely planned to counter check the validity of the data gathered from the developers. In step 3, the researchers re-analysed the data collected from step 1 and added in the data or information gathered from the project managers in step 2. The third validity measure is to show the reliability of this study. The first author actually is a full time employee in three of five projects from 2005 to 2007 at ISC. The author has had the opportunity of recognizing the issues that have been raised by project managers and developers in the projects. The author was aware of the possible problems in many of the projects. These entire have, in a way or another, helped to increase the external validity of the study results.

3.2.7. Conclusion

This study has presented 12 challenges in a release planning process from banking software project domain. Due to the fact that there are many different aspects to be considered for a new release, this study is conducted with the aim to better understand and identify main challenges faced by people in development teams.

Some of the challenges can be considered common problems faced by software development teams and some are quite rare. The study also exposes that there are cases in which, a new release is not always better in functionality than the older ones and may even have more problems than the previous release. This circumstance might indicate that there are certain unresolved issues in the release planning process. Even though, the findings of the study are not generalizable to all release planning processes which are

taking places in other companies, the findings can at least provide us with the possible understanding that these problems might occur during the process. Essentially, this study is able to identify 12 challenges and those challenges are divided into human-oriented and system-oriented categories.

Although these identified challenges have been observed in special domains (i.e. banking), the present study can be investigated further by increasing the number of sample projects in order to spot more detailed challenges in other areas from both human and system point of views. This work can be a useful guide for release planning process in order to have an improved product and more satisfied customers.

In the next section, we tried to find how companies deal with developing new releases. For instance, how do younger companies, which are less experienced, decide for developing their new releases? Do they choose systematic methods or decide according to the views of their managers and experts?

3.3. Companies' Approaches in Software Release Planning

Nowadays, numerous methods of release planning are available in software companies each of which use methods based on their own viewpoint. Despite this, managers still find themselves confused in developing a new release. This research project investigates the methods companies use to plan for new software releases and the approaches they take.

3.3.1. Companies and software releases

The concept of software improvement is still one of the worries that software managers commonly face. Companies try to improve their software products by understating the new requirements and then implementing them in their future software releases. It has been shown that investment in process improvement has had significant benefits for companies, including improvements in product quality, reduction in time-to-market

products and improvements in the productivity (Zahran, 1998). Release planning can be seen as company-wide optimization problem involving many stakeholders where the goal is to maximize utilization of the companies' often limited resources and turn them into business benefit (Ruhe & Saliu, 2005a). A major problem faced by companies, developing or maintaining large and complex systems, is to decide which features should be added to which release of the software (Bagnall et al., 2001). Although there are many existing release planning approaches for generating new releases, there are still lots of problems such as lack of a precise and systematic way to produce optimal products. Carlshamre (Carlshamre, 2002) has classified release planning as a "wicked problem". The concept of a wicked planning problem was first introduced by Rittel and Webber in (Rittel & Webber, 1984). Wicked problems are difficult to define and there is often no clear-cut solution to them. Companies are, therefore, always confused and have many problems with the generation of new releases. The aim of this research is to investigate how companies deal with new software releases and what methods are used by different companies with different experiences in software development to cut down on the level of confusion. For this reason, using a systematic approach alone is not enough. Companies also need human capability and the experience of professional practitioners to carry out such a task. One of the possible topics discussed within software companies' media is to what extent the human factor is important for release planning. Another question is whether systematic approach alone can be adequate to respond to various requests, including demands for new features; and is there still need for human creativity and resources for decision makings. For example, a new company, which has not yet defined all of its tasks and needs, cannot rely on a systematic approach for release planning. In order to fill the gap, this company needs human resources. Another significant point to consider is the type of planning used by software companies. This largely depends on their experience and type of product concerned. For

example, in some experienced companies, project managers do not have to concentrate much on new plans, because they usually have positive expectations on the future of their current product and its reliability. Instead, they always look for ways to improve the product quality. Project managers usually tend to avoid exposure to a risky market. In this research, we study different companies' approaches for generating new releases in a release planning concept based on multiple case-studies taken from different software companies.

3.3.2. Objective

The aim of this study is to investigate how companies plan new releases, and what approaches are used by software companies. Based on this, we categorized and analyzed approaches currently used by software companies. Choosing an appropriate approach can help with the quality of the product in a new release. For example, for some software companies, release planning depends solely on management and members of software developing team, but some other companies rely on the use of automation for their new release. Therefore, there are difficulties to identify suitable methods for a new software release because release planning does not have a specific rule and it is impossible to have a comprehensive method for release planning to cover all companies. This research compared software companies with different products and experiences in software development and found some useful results which can help software companies to choose an appropriate method in delivering a new release.

3.3.3. Research design

This research analysed seven cases (software companies) in industry. Further, we investigated their current release planning approaches and their plans for new releases. Software development is a complicated process, and requires careful planning and execution to meet the goals. Therefore, it is very important to see how companies choose methods for their optimal release planning. Some companies prefer to choose

only human-based approaches and rely on the judgment of management and staff, and others prefer to use just the systematic approach. However, we cannot say which one is better and more useful. This study was conducted in seven software companies in Iran with different experiences and products. The goal of this work was to identify ways companies deal with the release of a new generation of software. We have used semi-structured interviews as the primary data collection method, and asked the staff to fill out the questionnaires after we explained to them what exactly they were expected to do. We interviewed and discussed with fifty-one software professionals, working for software companies, including: 8 project managers, 17 developers, 15 analysts and 11 quality assurances managers. The role of the survey respondents are shown in Table 3-3. For the reason of confidentiality, we did not reveal names of participating companies. Instead, we referred to these companies as: A, B, C, D, E, F, and G.

Table 3-3: Participants in the survey

Role of respondent	Number	Companies
Project managers	8	A,B,C,D,E,F,G
Developers	17	A,B,C,D,E,F,G
Analyzers	15	A,B,C,E,F,G
Quality Assurance	11	A,B,C,D,G

All the companies studied in this research develop software with different experience in software development. As shown in Table 3-4, some of them are old companies and have great experiences in software development. Others are major but new international organizations who have established their activities in banking software in Iran. Their projects vary in team size, experience and duration as well.

Table 3-4: Companies characterized

Company	Type of product	experience
A	Banking software	1 year
B	Database systems	3 years
C	Web-based application	8 months
D	Banking software	9 years
E	Banking software	12 years
F	Database systems	1 year
G	Medical systems	2.5years

In this project, semi-structured interviews (Robson, 2011) with technical questions was conducted with the team members for data collection and sometimes we asked some complementary questions to have more elaboration on the answers. As mentioned before, 51 staff including project managers, developer analysers and quality managers attended the interviews. The time spent on interviews and filling forms varied between 30 to 45 minutes long for each one.

Here is a summary of questions that were asked:

1- When do you generate a new application?

2- What approaches do you prefer more to plan for new releases?

3- How do you rate the role of human in generating a new release? (Indicate your score with percentage)

4- How do you make decisions for a new release?

5- Does your previous experience affect a new release? If yes, how?

3.3.4. Results

Analysis of this research is presented here with the data collected from the companies. Following is a summary of the companies' behaviours in release planning:

3.3.4.1. Company A

This company has about one year experience in developing banking software. This new company develops cutting edge banking systems. The company's goal is to come up with new ideas and new plans in the banking domain, both offline and online. Based on the information gathered from practitioners and considering the newness of the company in software development, developing new application follows orders received from banks and governmental organizations. The role of human in creating a new plan is highlighted here because they do not rely on a systematic approach for new plans. The company's goal is to automate banking operations and add new relevant features. This company does not follow any specific regulations related to release planning. They make decisions based on the existing circumstances and needs. This company tends more towards the human resources and according to data collected from responders of company A (see Table 3-4), 85.7% of the responders believe that human resources are very essential for the process of release planning. 57% of them agree to use both human and systematic resources, while 28.6% of the responders just want to use human resources.

3.3.4.2. Company B

This company has been producing database systems for over 3 years. They usually take a systematic approach for their release planning and have been successful to some extent. Their main focus is on maintenance and improvement of current products rather than developing new products. The reason for this lies in the satisfaction they have with their current products. Without the specific need for any new and profitable product, they usually do not concentre on new plans. They have developed special regulations for a new release, including several meetings with the clients for every new plan. In company B, which has just passed its third year of being in business, 45% of the

responders like to use a combination of human and systematic resources, 22% of them prefer only systematic approach, while 33% of them like to use human resources.

3.3.4.3. Company C

This company produces and designs web-based applications and websites. Since this company is very new in the market, they do not have a comprehensive plan for new releases. Their activities are based on customers' orders and new releases are based on customers' requests. They prefer to have new contracts and adjust themselves to various plans to improve their development process. As a result of their newness in this industry, the role of human is high. With numerous meetings and discussions with customers they try to achieve more systematic approaches. But because they do not have enough feedback, most of the decisions are based on the agreements between the customers and system developers. They also do not have a specific regulation for a new release. Company C is new in business, and therefore human resources play a major role in their decision makings. 60% of the members use a blend composite of human and systematic methods, and 40% of them believe in human methods only. In this company, a new release without human intervention is baseless.

3.3.4.4. Company D, E

These two companies which are supported by the government have a long experience in banking projects and cooperation with each other. They have an interdependent relationship and work together to develop projects in the area of banking software. Nearly all the software that are used by the banks and financial institutions affiliated with the government have been developed by these companies. Therefore these companies are more inclined to support and improve their own software with an emphasis on re-planning new release. Systematic approaches are utilized in developing a new release; for example, parts like requirement elicitation, requirement estimation,

and requirement prioritization are defined to complete this systematic approach. Role of human is more dominant in requirement elicitation, and older releases are very important because improvements are done in those releases. Creating a new release is based on the customers' request and is usually done twice a year. In companies D and E, which are in the business of producing banking software relatively for a long time, as we can see from the Table 3-4, 40% to 50% of the responders from these companies see the use of systematic approach as a necessity, and only 18% of the responders from company D and 20% of the responders from company E still prefer to use human resources in their planning.

3.3.4.5. Company F

This company has one year of experience in database systems and information retrieval for governmental and private organizations and institutions. This is a private company that creates a new application based on the requests from clients. Since this company is relatively new, and it does not have a specific plan for a new release in the foreseeable future, the role of human in decision makings and investigating the requirements is very significant. They must decide what features to add in the new release and when it is going to be released. Since they have only one year experience in software development, developers try to create samples and libraries to move to a systematic approach. 42% of the responders from company F prefer to use human resources, and 25% of them just like systematic approach.

3.3.4.6. Company G

This company develops medical software. Due to the fact that the software is used in hospitals and medical centres, creating a new release is based on the requirements and features that are elicited from the real environment. It can be said that the method used in this company is like INF; it determines the value of each new feature that is going to

be added to the new system. The role of human in their release planning is fading and it is needed only for collecting the requirements from the environment. Previous experiences are actually the base for creating new releases. After reviewing the data we received from the companies we were able to achieve the following important findings that will be proved in the next section using statistical methods: older and experienced software companies tend to focus on improving their current products rather than to create a new product or a new release. For them improving the quality of their products and satisfaction of their customers are more important, and in fact they do not want to take the risk of developing a new product. This is because their products have reached an acceptable level of reliability in the market. A new release could be produced based on the feedback they get from their products. These kinds of companies use more systematic approach than human approach when making decisions. Newer companies usually have no experience in developing software, therefore they start from ground zero, and they come up with a new plan. For them, to be successful in the market means to be creative and their plan should cover the best capabilities. Manager of this kind of companies usually think that using the systematic approach only is not enough and they believe using human abilities bring stability to their companies. As we can see from Table 3-4, 50% of the responders from company G like to use a blend of human and systematic approach, 17% of them prefer human resources and 35% of them just want to use systematic approach.

3.3.5. Findings and discussions

This section presents our findings discovered during the interviews and discussions. Based on the data collected from different companies, there are two recognizable categories:

- **Categorization of software release planning approaches from the viewpoint of either re-planning or new plan**

Software development planning is crucial to the success of a project (Sommerville, 2010). A software development plan aims to define various tasks such as phases and releases of software development. One of the most important tasks of developing software is requirement elicitation from the customers. The extent of knowing these requirements and the percentage of implementation depends on the company's plan. Applying the requirements may lead to either creating a new plan or to improving the existing plan. As mentioned before, there are many release planning approaches for implementing requirements, some of them function based on new plans and new releases, and others are based on product background.

Figure 3-4 illustrates that collected data are divided into two main groups: the first one is called re-planning and is used for new release without any background, meaning there is no need for past details. Newly established companies use these methods, because they do not have any history or experience with their products, i.e. their products are new. They need to estimate resources, efforts and everything needed for defining a good plan for future production. These companies prefer to use new plans for their start, because they are in a risky situation; they do not know whether their product will be a success or not.

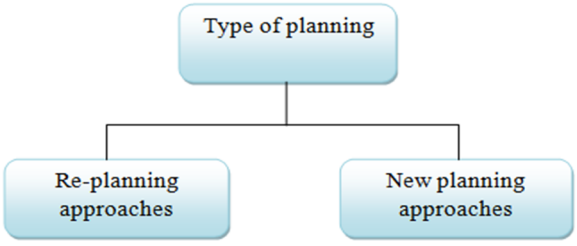


Figure 3-4: Categorization by view of planning

Second group, which is called new planning, are methods that have improved versions of the older releases and decision making in them is based on the improvement of previous releases and re-planning that most of older companies try to do. They get customers' feedback and attempt to improve current software. Re-planning means creating a new plan to solve an application's malfunctioning. It can be seen as a procedural use of past work, which is a special kind of plan reuse. In plan reuse, the current plan is used to solve a new problem by changing the initial goal.

Table 3-5 shows the individuals' preferences for planning of a new release. It shows that people in newer companies such as A, C and F prefer more to select new planning, and older companies such as B, D, and E that have a longer experience in software development, don't want to be exposed to any risk and try to improve their products because they have already paved their way into the market.

Table 3-5: Individual's responses in companies

Company	New planning	Re-planning
A	57%	42%
B	34%	66%
C	80%	20%
D	28.6%	71.4%
E	30%	70%
F	71.4%	28.6%
G	16.7%	83.3%

Reliable software products force managers to continue and focus more on their existing products. Based on the survey done on 51 people, including project managers, developers, etc. and from the following diagram, we can see that with the increase of companies' experience from C to E the tendency goes toward re-planning and company E shows the highest score. For example, in company C that is a newer company in our

research, 80% of respondents are in favour of re-planning and 20% of them prefer new planning.

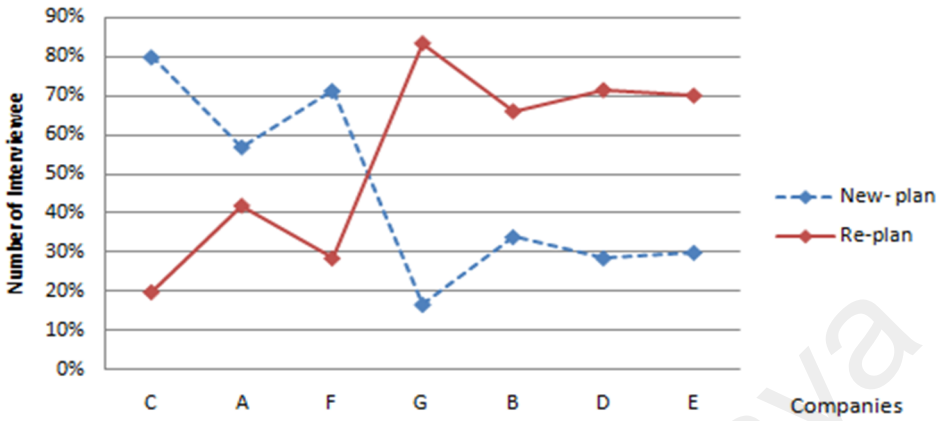


Figure 3-5: Tendency of companies to re-planning and new planning

As illustrated in Figure 3-5 (see the solid line), the tendency to re-planning goes up steadily with the increase of companies' experience. This means such companies like to focus on re-planning and improving their products. On the other hand, the dashed line, which shows the tendency to new planning, comes down with the increase of experience.

- **Categorization of software release planning approaches based on usage of human resource**

Nowadays, most software companies tend to develop their software by using a systematic and automated methodology, and this applies to release planning as well, which is a part of software development process, but in the real world this might be impossible to carry out.

As mentioned before, Carlshamre in (Carlshamre, 2002) has classified release planning as a wicked problem. This means there are different ambiguous issues that are unpredictable in software development process which may make moving to systematic approaches not always practical and may require human based approaches as well.

Release planning is known to be a cognitively and computationally difficult problem (Ngo-The & Ruhe, 2009). Real-world release planning problems may include several hundred features potentially offered in the next releases (Ngo-The & Ruhe, 2009).

Based on the case studies, we observed a pattern of the differing role that can be played by human in release planning; we categorized the release planning approaches used in all the studied companies into three groups: 100% human based approaches, 100% systematic approaches and a blend of human and systematic approaches. We also observed the correlation between the length of the experience of the company and the categories. Figure 3-6 shows three groups: first one denoted as "100% systematic based", represents the approach used by older and experienced companies. They use systematic methods only. Second group, named "100% human based", shows the methods used by new companies. They usually focus on the role of human creativity, and as such, they do not consider release planning as a separate activity. Therefore, they rely 100% on decisions made by leaders and human experience. Last group which is called "blends of human and systematic" shows a combination of systematic and humanistic role in release planning.

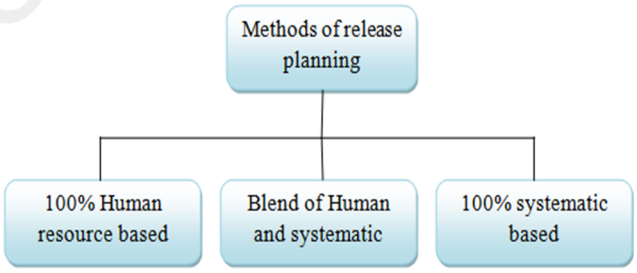


Figure 3-6: Grouping human resources

Table 3-6 illustrates that at company B, which has a relatively good experience in software development, only 22 % of the interviewees believe in the role of human and 33% of them believe in systematic approach, whereas 45% of them prefer a combination of both approaches for release planning. This table shows that new

companies consider the necessity of human role in release planning much higher than systematic approach.

Table 3-6: Individual's responses in companies

Company	Only systematic approaches	Only human based approaches	Blend of human and systematic
A	14.3%	28.6%	57.1%
B	33%	22%	45%
C	0%	40%	60%
D	40%	18%	42%
E	50%	20%	30%
F	23%	35%	42%
G	33%	17%	50%

As shown in Figure 3-7, we can also see that from company C (with least experience) on the far left to company E (with most experience) on the far right, the tendency of using automation approach increases and the role of human decreases.

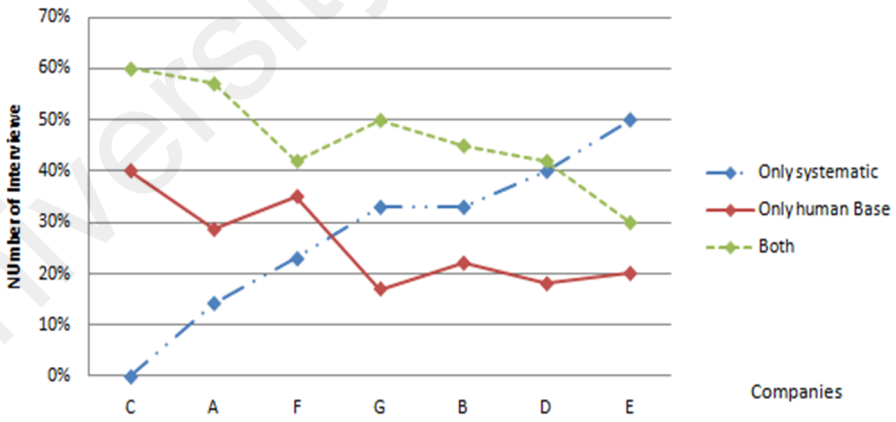


Figure 3-7: Tendency of companies to human or systematic approaches

From Figure 3-6, we can see that as companies' experience increases, use of "only human" resources decreases. It is shown with the solid line and this means the tendency toward the only human based approaches is low. On the other side, the dot-dash line that shows only systematic approaches correlates positively with the experience. The

dashed line, as it is observed in the Figure 3-6, is more common because it is a combination of human and systematic approaches.

3.3.6. Discussion on the findings

Since there is no unified comprehensive approach for software development planning, this study investigated planning approaches that are normally used by companies on new software releases. The studied companies had different levels of experience and their products were different in nature. The results of this study showed that companies with more experience focus more on improving their existing products and try to avoid taking new risks. These companies believe that their product reliability is quite sufficient to compete in the market. With increasing experience, companies prefer automation and avoid human resources. On the other hand, newer companies with not enough experience need more human involvement to achieve that goal. This study could be further expanded by increasing the number of companies under investigation. Further researches could also be carried out to inspect the challenges that software companies confront during the development processes. Results could eventually improve the process of release planning.

CHAPTER 4: SOFTWARE RELEASE MANAGEMENT IN INDUSTRY

4.1. Overview

Release management has an essential role in the success of large software projects and there is no doubt that correct and efficient management of producing a new release can help the quality of the product and the satisfaction of the customers to a great extent. But still there exist problems in many companies for generating a new release. Despite the presence of all necessary resources like human, financial, and time, sometimes a new release has a lower quality than the older ones. This can be because of the lack of management in release planning.

In this chapter, after a definition of release management, some possible challenges that can affect management of software release projects will be shown. In other words, we investigate what needs to be strengthened in management of a release to have a better release in software companies. This study is an exploration of software projects in small companies in Malaysia. This research was conducted to investigate and understand effective challenges in release management that have been identified by effective role players in release management within software domain.

4.2. Software release management

In simple words, a release is a set of features of a software application, implemented during a software development process. Release management is an important part of quality management, since it is concerned with the delivery of high quality software to users (Levin & Yadid, 1990). Release management consists of technical and management activities that are needed to take a release from a set of requirements to the delivery of a software application that implements these requirements.

The development of software applications is an incremental process, moving towards a series of sequenced and unknown goals. These goals are usually provided in the form of a release. Release management is about monitoring how changes flow into systems. Whenever these changes are updated, new features may be added in the next release. Release management is generally not just a management effort. It often includes a great deal of collaboration and automated arrangement of complex computer systems. If you want to generate a new release, you might look at release candidates before and perhaps investigate the ambiguities before going to full production.

Software Release Management is the process of ensuring releases that can be reliably planned, scheduled and successfully deployed to real environments. Figure 4-1 gives a clear overview of release management during a release planning process. As seen in Figure 4-1, there are three main steps in release planning process.

Figure 4-1: Overview of release planning process

The first step is release goals, which includes: requirements, updates, patches, and the main goals of releases, services and business policies of the organization. It can also be called the targets of releases. Release management plays a great role, too. It includes: standardization, control, approval for implementation of features and investigation of correctness of the new changes for a release. After changes and modifications are done and features are implemented, it is time to execute the release in real environment in order to get feedback from customers. In fact, release execution step is the behavior of new release. The main idea of Figure 4-2 is adapted from (Drapeau & Oudi, 2007).

This part is in fact an exploratory study to find the challenges of release management in software industry among small software companies in Malaysia. For the purpose of this study, we interviewed with developers and release managers of different projects individually to identify release management challenges that affect release planning.

Once the basic release of software is delivered to the market, a number of new features and enhancements can be identified. Release management is necessary to investigate and evaluate these new demands. Although release management is the main part of release planning process for improving the quality of a product, few surveys have been conducted in the area of release management practices and challenges. Michlmayr (Michlmayr, Hunt, & Probert, 2007) finds problems and practices for release management in free software projects. An exploratory study was performed to get a better view of actual practices and problems associated with release management in free and open source software projects. In fact we explored release practices employed by volunteer free software projects and showed their problems. Hall et al. (Hoek, Hall, Heimbigner, & Wolf, 1997) discussed and identified the issues encountered in software release management, and presented an initial set of requirements for a software release management tool. They described a prototype of such a tool that supports both developers and users in the software release management process. On the other hand, (Hoek & Wolf, 2003) have discussed the problems of release management for component-based software and proposed SRM, a prototype software release management tool; it supported both developers and users during the software release management process. Mayuram S. Krishna in (Krishnan, 1994) with a business perspective introduced an economic model to capture the various trade-offs involved in software release decisions and further discussed methods to obtain optimal software release time.

4.3. Objective

One of the biggest problems faced by software projects is that, although enough experts or skilful human resources, sufficient budget and resources are available during software development, there are still chances of having problems in a new release and this may return to the management desk for further decisions(Levin & Yadid, 1990).

In this study, we studied the most probable challenges in software release management by interviewing developers, managers, and release managers. The main goal of this chapter is to find challenges that are related to management of a new release in small software companies in Malaysia. Every software company must face these challenges, understand these obstacles and try to resolve them. They are effective during release planning process and consequently the success of the release depends on them.

4.4. Research Method

In this study, we have presented exploratory results on software projects to find release management challenges in small software companies in Malaysia. Definition of a "small" business varies by industry (Fayad, Laitinen, & Ward, 2000). However, as our focus is on the size of organizations, we defined an organization's size based on the number of employees. Organizations with less than 100 employees are usually considered small. Exploratory studies are used when the "research looks for patterns, ideas, or hypotheses rather than research that tries to test or confirm hypotheses" (Vogt, 1993). The aim of this study is to identify the challenges that can affect and improve release management process in practice. This research was conducted as semi-structured based on interviews with eleven software organizations. We interviewed 41 practitioners, including release managers, project managers, analyzers and developers of various projects. These 41 people were selected from among those who had management positions in new releases. In some larger projects, a release manager, and in some others, a project manager was responsible for managing the release. In smaller

projects, the system analyzer managed the development of the new release for the market. Our selection was based on the management role people played in new release projects. Although the size of these companies was small, sensitive projects such as web-based systems, database systems and some tools for organizations. As Table 4-1 shows, the participants are mainly Project Managers (12.2%), Release managers (9.7%), Software developers (41.5%), and software analyzers (36.6%). The duration of each interview that included discussions about research question was 15-25 minutes.

Table 4-1: The responders' numbers

Role of respondent	Number
Project managers	5
Release managers	4
Developers	17
Analysers	15

4.4.1. Research question

In addition to the interview, we had discussions with some practitioners for clarifying purposes. We asked, for instance, the following question: "what challenges, in your opinion, can improve or affect release management?"

Since the intent of this research was to investigate the effective challenges that could affect release management, we had a pertinent research question:

Q: *Which challenges are significant in release management?*

This question deals with the investigation of significant challenges of release management in small software companies. The question was operationalized by asking the participants of the study to find challenges.

4.5. Challenges in industry

After interviews and observation with practitioners, we found out their ideas of what can be effective in release management of small software companies. Although some of them were mentioned in the journal articles or past researches, there were some new helpful ones, which can improve release management to a great extent. In the following section, we organize our findings based on data collected from small software companies in Malaysia. The main challenges are:

4.5.1. Categorization of releases

One of the most important challenges that were discussed in interviews and the practitioners tried to focus on, was type of releases. It is clear and obvious that management of different types of releases is different. As mentioned in (Michlmayr et al., 2007), general term of release management is used to refer to three different types of releases: Development release that is aimed at developers, who are interested in working with cutting-edge technology. It means developers try to move towards latest technology in their release. A major release usually introduces new capabilities and functions or some new and significant changes, and a minor release incorporates a number of fixes for known problems into the baseline, updated to the existing release. When we talked to members about release management, the first thing that they told us was about the types of releases and they agreed that management for developing a new release based on some minor changes is easier.

As one can see, the type of release we are going to deliver is important. For instance, a release which is developed after one year and is getting feedback from various customers is different from a release with some minor defects only. Obviously, the management of these releases cannot be the same. Sometimes a release is planned for special features and takes time and effort, but sometimes it is only a different language. These projects are mostly based on the requests of the customers.

From forty-one participants coming from eleven companies who we met, thirty-seven of them strongly agreed that the type of release has a great role in release management, and categorization of changes for a new release in order to identify types of release can be effective in release management. Therefore, one of the most important challenges that has value and has always been discussed in release managements is type of release.

4.5.2. The need for some support tool in release management

Automation enables you to do tasks without dependency to human resources, and standardization of the process ensures that your automation works well and the results are consistent.

In this study, nine out of eleven companies liked to have a systematic approach for their release management. Even one of the project managers said he preferred to have at least two new releases per year, because maintenance is easier and new demands will not accumulate.

In the participants' opinion, automation for release planning means having standard cycles for releases. They do not like to face unexpected events, although they may occur. After several years of experience, release mangers try to develop a standard process for release delivery. Their goal is to have systematic rules for managing their releases.

One of the issues that was discussed with the practitioners, was the ability to manage releases automatically in a standard cycle. However, it can be very hard for release managers to achieve this. In fact, (Carlshamre, 2002) classified release planning as a "wicked problem", which means it is not predictable. As one can see in Figure 4-2, eleven out of forty-one respondents agreed to have automated process for release management, and twenty-two of them liked ad-hoc approach for release management. Ad-hoc signifies a solution designed for a specific problem in release management.

They think that release management problems are not foreseeable, and based on customers' demands they use specific solutions. Nine of participants told that their release management depends on the release type.

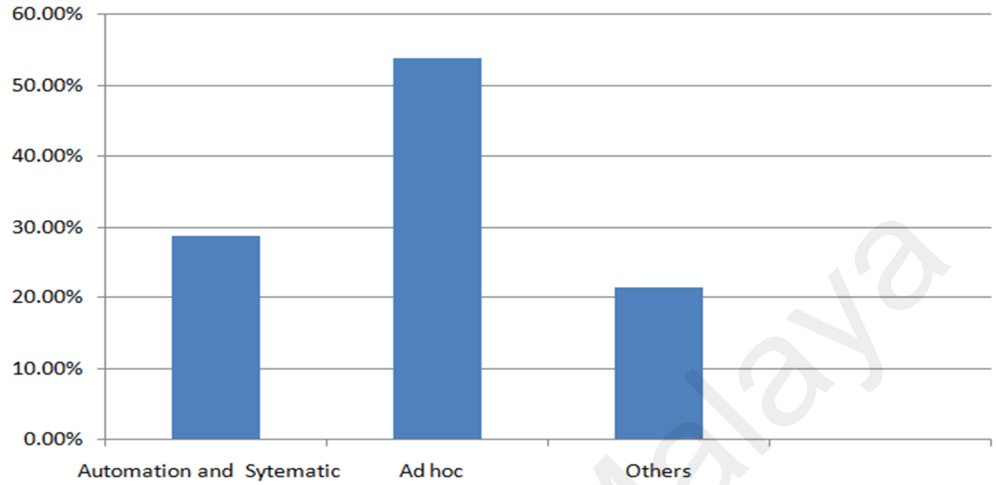


Figure 4-2: Respondents

4.5.3. Appropriate tools

Surprisingly, release management is lacking tools that would help automate the process. Using a proper tool for release management in a software company means an acceptable approach for release management. So it can be said that the right tool, to some extent, is related to automation of release management. Despite the importance of release management for delivering a good quality software release, responders are still wondering how to find proper tool(s) for new releases.

Therefore, one of the major challenges that practitioners were talking about was having a suitable tool for release managers to plan a new release, and this is very difficult as each release has different conditions and problems.

For example, one release may need minor changes, but another release may even need a change in functions. Therefore, the need for a suitable tool that can satisfy all conditions is strongly felt.

Among investigated companies, only one company used specific tools for release management and that is because they have a systematic approach for new releases. The rest, to some extent, depended on release managers but they are trying to have a suitable tool and they admit that it is very hard.

4.5.4. Foreseeing a new release before real execution

Release managers have the duty to predict the proper execution of a new release in the real world, and one of their worries is to predict it correctly.

Sometimes new releases have lower qualities than the previous releases. Therefore, it is the responsibility of release managers to determine whether or not the new release will outperform the previous ones. They must be able to predict the future of the release, before it gets to the market.

Participants in this survey agreed that testing their product, at least once, before releasing it to the market is very important for predicting the quality of their products. They emphasized that this prediction is necessary for software companies to determine the future of their products.

So all of the investigated companies try to have solutions for their products in a real environment, and sometimes they try to predict the level of satisfaction with some tools before sending the product to the market.

4.5.5. Release manager's role

The role of the release manager is diverse and demanding, because they have to interact with different people, understand technical issues, and also know how to plan and coordinate (Michlmayr et al., 2007).

In a small project, a release manager usually has an administrative role, which involves preparation of the release in different formats to be distributed. He/she is also

responsible for creating release notes and actual distribution of the software (Michlmayr et al., 2007).

One of the most important issues, that has a great role in both release planning process and release management, is having qualified release managers. Michlmayr (Michlmayr et al., 2007) states that having different skills such as community building, strong vision, discipline, judgement, good communication, and management skills are very important.

Participants in the survey acknowledged that having good project managers plays an important role in the quality of the products. It is desirable for managers to say something once and everyone understands it, but this does not always happen in reality.

The role of the release manager is diverse and demanding, because they have to interact with a large number of different people, understand technical issues and also know how to plan and coordinate (Michlmayr et al., 2007). In small projects that we were targeting on, some properties of release managers were also discussed. For example, they must have discipline, need a lot of experience in software development, and sometimes have to know the details. They are responsible for transferring new demands to developers and team managers, and in a new release they must adjust the new release and the requested requirements.

Based on the data collected from practitioners, 65% of responders agree that the most important factor in delivering a successful release is the release manager.

4.5.6. Proper understating of Request for Changes (RFC)

Request for changes (RFC) is a change request that captures the details of a change that is needed to be made to existing releases based on customer demands. The reason for generating a new release is implementing a series of these changes and modifying the behavior of a release due to normal business values because there are some problems in

the current release. One of the important challenges that companies are worried about and want to improve is the true understanding of proper changes. In release management, this is very sensitive and must be done carefully because misunderstanding of new requirements may lead to low quality and less satisfaction.

4.5.7. Release policy

Release policies are high-level statements of how releases are to be managed, organized, and performed in the environment. Policies include management goals, objectives, beliefs, and responsibilities. One of the main topics that we discussed with practitioners was release policies and, although these policies are different in each company and depend on many factors, recognizing and understating them can help us to have a better release management. For instance, in some companies that are related to the government, it is helpful to know the government policies like specific goals or rules for a software release.

Based on the interviews, although every company obeys some policies and they are important to the release planning process, sometime these polices lead into limitations and restrictions for release management.

In eleven investigated companies, there are only 4 companies developing software for some governmental organizations and have to accept their policies. In the other 7 companies, policies are based on the managers.

4.6. Threats to validity

Validity of the results needs to be planned by gauging proper counter measures. We have followed the recommendations by Yin (Yin, 2003) where he chooses four possible ways for the validation. The first one which is referred to as construct validity is a test or a measurement tool that is established by demonstrating its ability to identify or measure the variables or constructs that it proposes to identify or measure. For this

research, we have used two main researchers for interviews in order to reduce the risk of misunderstanding the information gained from the interviewees. One of the researchers is the first author and the second one is a master student.

The second important measurement is trying to demonstrate the internal validity of the study. Internal validity checks the "true" causes of the outcomes extracted in the research. After investigating and finding the challenges in release management, we send our findings to the same participants (41 people) in those companies to approve and determine the level of importance. The third validity measure is to show the reliability of the study. To increase the reliability of our study, all collected data and derivations are stored in a database accessible only to the researchers in the study.

The last validity measure is external validity that is related to generalizing. As mentioned before, our findings are based on small software companies in Malaysia. Hence, they may not be generalized to medium and large companies. We think, however, some of the main points of our findings might be valuable for software companies of all sizes.

After investigating and finding our challenges, as already mentioned before, we send the findings to the same participants in those companies to understand their importance. As you can see in appendix A Table 4-2 we have prepared a questionnaire to check their ideas about our findings.

Table 4-2: A Questioner sample

1=extremely important, 2=Very important, 3= moderately important, 4=Not that important, 5=Unsure					
Challenges	1	2	3	4	5
Types of releases					
Automation of release management					
Proper tools					
foreseen new release before real execution					
Release manager					
Proper Understating of RFC					
Release policy					
Comments:					

In this step, we sent the questionnaires to the participants by email and asked them to fill the forms and send them back to us. We gave them reminders by telephone to fill up the questionnaires. The main goal of this step was to internally validate our result and know the level of importance of our findings.

After analyzing the data collected from the same practitioners, as Figure 4-3 illustrates the level of importance of the challenges, type of releases and understating RFC are shown to be the most important challenges and investigation of these challenges is necessary for release management process. Based on Figure 4-3, some other challenges that are important have less importance in release management.

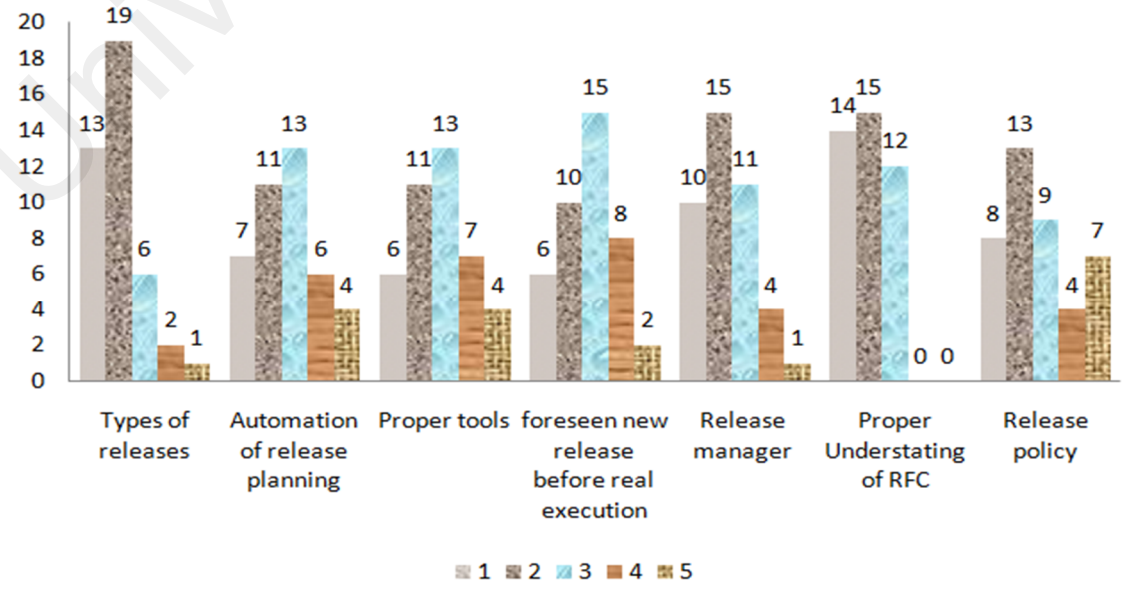


Figure 4-3: Level of importance

Release management plays a great role in delivering a successful release. There are several issues and challenges in release management that knowing about and improving on them can affect management of a new release. We found some possible challenges that can affect the management in software release projects. This study was an exploration of small software projects in Malaysia and was conducted in order to understand critical challenges in release management. These challenges were identified by key role players in release management in software domain. We found some challenges that are important in release management and their improvement and investigation can be of great value. In fact, success of the release management depends on them. The findings of this study can offer several important components which can be a good basis for future analysis. Future research will extend the number of participants and further investigate medium and large size companies to find a general framework for release management process.

4.7. Summary and conclusion

This chapter discusses about challenges and problems of release planning somehow ignored in today's systematic methods. The challenges and problems present in both applied and managerial aspects of release planning are always focused on lack of integrity and confidence in systematic methods. Highlighting and analysing these challenges highly depends on the fact that systematic methods cannot help in software development without considering problems specific to every certain project. Therefore, it is possible to achieve a general view of present systematic methods through a comprehensive examination of them. Through this general comprehensive process, one can define patterns and consequently achieve pattern-based release planning approach which, indeed, involves all present methods.

CHAPTER 5: DESCRIPTION OF THE PATTERN-BASED RELEASE PLANNING METHODOLOGY

5.1. Introduction

In this section, the pattern-based release planning methodology is explained and customization of the release planning process model is described to clarify activities involved and the quality of pattern formation in this methodology. The customization can be used for new patterns developed in every activity of the process model. Of course, more patterns are going to be presented later in other parts of the thesis.

5.2. The process model of release planning

A great deal of research has been accomplished in the field of release planning and management in most of which release planning is considered an issue in decision making and seeking optimized solutions (Bagnall et al., 2001; Durillo et al., 2011; Ruhe, Eberlein, & Pfahl, 2003). During the software development process many technical documents are produced. Such documents and their evolution history contain rich information about the development process (Junji & Ruhe, 2013). There are two points here that need to be emphasized. First, there are various ambiguous and uncertain parameters that influence the solutions. Second, there is no single solution for any problem. Various solutions can be found that differ in their performance (e.g., time performance, complexity performance, etc.).

Parameter variation in the field of release planning originates from the nature of software engineering and its activities. Although there are companies which work only in a certain field of software development, but still various parameters influence their release planning projects (Seyed Danesh & Ahmad, 2012). Effective parameters on release planning adopt certain values in a project but, as a result of ambiguity, they may take different values in a similar project or company (J. Li & Ruhe, 2003). This

originates from differences in release planning projects which, per se, refers to the nature of software engineering. Although most parameters have been identified in this field, due to its nature, eventually a certain parameter (or more) influences a specific project of release planning.

Research on release planning can be divided into two main groups. First are those studies that try to present or extend the parametric atmosphere of planning challenges and problems in order to propose points, guidelines and methods to figure one (or more) parameter out. (Al-Emran, Jadallah, Paikari, Pfahl, & Ruhe, 2010; Al-Emran, Kapur, et al., 2010; Al-Emran, Pfahl, & Ruhe, 2010; A. Jadallah, Galster, Moussavi, & Ruhe, 2009; Mc Elroy & Ruhe, 2010; Michlmayr et al., 2007; Ngo-The & Ruhe, 2009; Seyed Danesh & Ahmad, 2012) are typical among these investigations. The second group involves studies focused on developing a methodology, a generalized framework or a certain objective for release planning in order to improve it. Among these studies, (AlBourae, 2007; AlBourae et al., 2006; Colares et al., 2009; Lindgren, Land, Norström, & Wall, 2008; Przepiora et al., 2012; Ruhe & Saliu, 2005a; Slooten, 2012) are remarkable. From another viewpoint, the first and second groups can be called singular-to-general and general-to-singular methods respectively. Examining the first group results in a creation set of effective parameters on planning, while investigating the second group leads to formation of common steps in release planning development. These results are used in building steps of pattern-based release planning methodology. In other words, the first step (conducting the process model of release planning) employs results of investigating planning methodologies and the second step, in which developing a pattern (or more) for each stage of the process model is considered, results of the first group of studies are mostly used.

The main objective of release planning is selecting a set of requirements or properties to be included in a product. Hence, the set of received requirements must first be

prioritized, and estimations and forces related to various resources must be identified. Finally, high priority requirements or properties with the evaluating estimations and forces are selected to be developed in the release (Carlshamre, 2002). These three steps can be considered as the basis for all release planning methods. It must be noted that requirement reception is prior to release planning. Various methods only differ in the order of these three steps, details of each step, inputs and outputs of each step, being general or specific, and the procedure of each activity within the steps. Table 5-1 summarizes the steps of the most known methods of release planning. They are selected with regards to the relationship between the release planning methods presented in (Svahnberg et al., 2010). Other than method 1 in which tasks and steps are not clearly specified, the three steps mentioned above are underlined in all the other release planning methods. In methods 4 and 5, the prioritization and selection of requirements or properties are integrated with higher priorities which evaluate estimations and forces. Moreover, the "Release Planner" software is used in methods 3 and 4 to accomplish these steps. Most release planning methods in Table 5-1use the stakeholders' viewpoints to perform a part of the planning activities, particularly in specifying the primary priority of requirements and their properties which method 4 does not use.

Table 5-1: Steps of implementing release planning methodologies

Order	Methodology	Implementation steps
1	Ad-Hoc (Seyed Danesh, 2011)	Steps are not clearly specified.
2	Quantitative WinWin (Ruhe et al., 2003)	1. Primary evaluation of requirements feasibility using primary constraints 2. Per each repetition 2-1. Determining candidate requirements per repetition 2-2. Determining requirement priority by stakeholders 2-3. Prioritization among and inside requirement groups using AHP method and based on stakeholders' view 2-4. Prioritization of all selected requirements using steps 2 and 3 2-5. Reviewing selected requirements

Order	Methodology	Implementation steps
		2-6. Examining feasibility of requirements 3. Evaluating all repetitions to determine their consistency with specified forces
3	Quality Improvement Paradigm (Amandeep, Ruhe, & Stanford, 2004)	1. Introducing project environment and characteristics including main qualitative features. 2. Problem definition 2-1. Determining stakeholders 2-2. Determining requirements 2-3. Determining the importance of each stakeholder 2-4. Determining various forces 2-5. Attributing priority to requirements by stakeholders 3. Planning and completing primary information 4. Running the software Release Planner 5. Result analysis 6. Recording results for next release planning
4	Release Planning under Fuzzy Effort Constraints(Ngo & Saliu, 2005)	1. Determining costs and effort required to implement requirements 2. Specifying structural dependency constraints among requirements (Coupling, Precedence) 3. Determining costs and effort required for each repetition 4. Specifying the precision threshold for structural dependency among requirements 5. Running Release Planning Software with various parameters 6. Selecting a proper plan respecting evaluation curve of dependency and meeting goals
5	Planning Game (Seyed Danesh, 2011)	1. Estimating required resources by developers 2. Selecting the most prior requirement by stakeholders until estimations and forces are met
6	Optimization-Based Techniques (Bagnall et al., 2001)	1. Determining the dependency between requirements 2. Calculating implementation costs for each requirement 3. Specifying implementation priority by stakeholders 4. Attributing weight to requirements based on stakeholders' priorities, dependency and implementation costs 5. Selecting the most prior requirements with optimizer algorithm
7	Hybrid Intelligence Approach (EVOLVE Family) (Saleem & Shafique, 2008) ,	1. Modeling 1-1. Grouping properties and requirements 1-2. Determining the relationship between properties and requirements

Order	Methodology	Implementation steps
	(Greer & Ruhe, 2004)	1-3. Specifying resource constraints 1-4. Estimating required resources 1-5. Determining priorities by stakeholders 2. Identification (detection) 2-1. Producing various release plans using algorithm 3. Integration 3-1. Evaluating various release plans using algorithm 3-2. Producing various scenarios for release re-planning 3-3. Selecting the best release plan to implement
8	Bi-Objective Release Planning (Saliu & Ruhe, 2007)	1. Determining the importance, necessity and value of requirements by stakeholders 2. Specifying the relationship between requirements 3. Calculating SD-Coupling between requirements 4. Determining Trade-off parameters (satisfying SD-Coupling) 5. Evaluating the systematic value of requirements 6. Implementing Bio-objective Release Planning Model 6-1. Implementing the business-based value function 6-2. Implementing a function based on requirements' counter effect (Synergy) 7. Producing different release plans 8. Evaluating different release plans based on Trade-off parameters and selecting the best plan

In investigating the release planning methods listed in Table 5-1, it was discovered that most of the methods have emphasized implementing the three steps mentioned earlier. This finding can be considered from two aspects. Firstly, it was noticed that every release planning method consists of these three common steps, which vary from one method to another based on the focus of the method. Secondly, these steps need to receive information or parameters from users in order to produce the results and parameters required, such as the primary constraints of the requirements and the requirements' priority from the stakeholders' viewpoint. In fact, without this information, release planning can only be performed with low precision or may produce improper results.

A more detailed look at the release planning methods indicates that they have other common specifications which can be used for a better understanding of the release planning properties. The followings are the common properties of release planning:

- Most release planning methods make use of requirement priorities determined by stakeholders as input.
- Most methods produce more than one release plan or scenario and will consider the analysis and selection of these plans as one of the steps toward final implementation.
- Most methods consider the interdependency of requirements and use them for release planning.

Similar to the three mentioned steps, these can also be regarded as common points in release planning methodologies but, indeed, they cannot be referred to as key steps and hence some methodologies can be found that do not respect these or acquire and produce their required data in a different manner. However, considering these points and previously mentioned common activities, the following (approximately common) activities can be considered for accomplishment of a release planning process model:

- Requirements prioritization
- Resource estimation, i.e. estimating the required resources for implementation of requirements
- Pre-release planning
- Trade-off analysis of plans

An important point to be considered here is that whether a process model of release planning can be obtained by putting these steps together or not. As shown earlier, the steps and points are common in almost all release planning methods; hence, putting them together may result in the main steps of a process model of release planning,

regardless of how they are implemented. To answer this question, three things must be specified: 1) the objective of each step's inputs and outputs, 2) whether the mentioned steps provide required inputs for release planning, and 3) whether the considered output of release planning is produced, regardless of the performance of the outputs.

To answer the first question, each common step must be explained and its objective, inputs and outputs should be clear. Answering the second question requires a comparison between inputs of these common general steps with those of the release planning methodologies. The comparison, on one hand, allows for investigation of input deficiencies and, on the other hand, helps in clarifying whether the deficiencies are resulted from incomplete steps or whether they can be produced with primary inputs through other processes. Response to this question confirms the inclusion and generalizability of common steps. Answering the third question ensures production of outputs for the sake of release planning goals by the common steps. Putting all the answers together demonstrates that the common steps can involve a process model of release planning and customization of this process leads to different release planning methodologies. Now, we first explain common steps, their objectives, inputs and outputs and then present the general release planning process.

5.2.1. Requirements prioritization

Often, the primary priority of requirements is taken from the stakeholders' viewpoint. The prioritization parameters are identified in this step based on which requirement priorities are determined. From the software developer's viewpoint, some stakeholders are more significant than the others and this is specified by the weight given to them. Stakeholders' weight can be determined based on certain rules and principles. If stakeholders are not prioritized, from the developer organization's viewpoint, all of them are considered to have the same weight. Set C contains n stakeholders and set W involves weight of each stakeholder and w_n shows the weight of stakeholder n.

$C = \{c_1, c_2, c_3, \dots, c_n\}$
$W = \{w_1, w_2, w_3, \dots, w_n\}$

As such, set R contains m requirements and m_{ij} shows the requirement i related to stakeholder j.

$$R = \{(r_{1,1}, r_{1,2}, \dots, r_{1,m}), (r_{2,1}, r_{2,2}, \dots, r_{2,m}), (r_{3,1}, r_{3,2}, \dots, r_{3,m}), (r_{n,1}, r_{n,2}, \dots, r_{n,m})\}$$

And set P contains priorities of each requirement from every stakeholder's perspective.

$$P = \{(p_{1,1}, p_{1,2}, \dots, p_{1,m}), (p_{2,1}, p_{2,2}, \dots, p_{2,m}), (p_{3,1}, p_{3,2}, \dots, p_{3,m}), \dots, (p_{n,1}, p_{n,2}, \dots, p_{n,m})\}$$

Priorities determined by each stakeholder are prioritized again (based on requirement prioritization parameters) and finally every requirement's priority is specified. Release planning methodologies have different parameters for requirement prioritization since the parameters are dependent on such issues as software type, number of involved stakeholders, developer organization size, team experience, organizational development strategy, and many other aspects. These parameters, in general, can be classified into three categories: parameters related to the project, to the developer organization, and to external parameters. The first class includes those parameters which contribute to a certain project and, hence, can vary in every certain project of the organization. Team experience and size can be mentioned as examples of this class (Akker, Brinkkemper, Diepen, & Versendaal, 2008). The second class involves some parameters over the first class. They are highly effective on company's software projects and are more permanent than the previous ones. Examples of such parameters are company's specialty and number of projects (Seyed Danesh & Ahmad, 2012). External or environmental parameters, as the name shows, originate from outside the developer organization and influence release planning. The most important parameters in this class are competition in presenting releases and environmental complexity, particularly in web-based software (J. Li & Ruhe, 2003).

5.2.2. Resource estimation

For every determined requirement, the required resources to implement it must be estimated. This is not always respected since some resources can be calculated using several parameters or other resources. For instance, a project's required costs can be estimated by determining the effort required to accomplish it. Although most methodologies try to limit resources to time and budget (Svahnberg et al., 2010), it must be noticed that other resources can also have a significant effect on release planning and bring about serious harms.

Set E is employed to show estimations of resources needed to implement a requirement in which $e_{n,k}$ shows the k^{th} estimation for n^{th} requirement. In fact, it is assumed that for every certain resource a corresponding estimation of resources exists. For example, if time and costs are being considered, the required time and cost is perceived as estimation for every requirement.

$$E = \{(e_{1,1}, e_{1,2}, \dots, e_{1,k}), (e_{2,1}, e_{2,2}, \dots, e_{2,k}), (e_{3,1}, e_{3,2}, \dots, e_{3,k}), \dots, (e_{n,1}, e_{n,2}, \dots, e_{n,k})\}$$

Having perceived estimations of every certain requirement, it is also necessary to perceive constraints of each requirement; a task which is performed in next step. Perceived estimations can be classified using estimation parameters. These parameters are not considered in many release planning methodologies and only one estimation for every certain requirement has sufficed already. Moreover, there are other parameters affecting resource estimation that depend, indeed, on decisions of the developer organization and project characteristics. In general, like effective parameters on requirement prioritization, parameters effective on estimation are divided into three groups with the same classification. In other words it can be said that estimation parameters are regarded similar to requirement prioritization parameters but with a particular look at resource estimation. For example, team experience which is effective on requirement prioritization as a project-related parameter impacts resource estimation

too, but in prioritization it results in faster or slower implementation of some specific requirements. In resource estimation, it leads to earlier or later allocation of specific resources and determines the amount of resources allocated to a certain requirement. Moreover, some effective parameters on requirement prioritization cannot influence constraint prioritization or resource estimation, and vice versa. For instance, the team size which is an effective parameter on requirement prioritization is mentioned as a resource constraint in resource estimation but not an estimation parameter. Therefore, resource estimation looks at this prioritization parameter in a different way.

Considering the priority of every requirement and other relevant inputs, a requirement can be stated based on the triplet set of *R*, *P* and *E*. in other words, every requirement includes requirement code, priority from different stakeholders' viewpoint and estimations of resources required for implementation.

$$Req = \{< R >, < P >, < E >\}$$

Up to this stage, it is necessary to find a set of release planning requirements that: 1) have higher priorities, 2) have less resource estimations, and 3) are selected by customers with more weight. It is necessary to search in the project environment to find and select such requirements. Most release planning methodologies try to optimally solve this problem and hence a variety of algorithms are presented.

5.2.3. Pre-release planning

Estimations and constraints are determinant parts of every release planning methodology. Constraints show the number of requirements that can be selected for every release. If a release is already planned, then a certain set of constraints are to be used. On the other hand, if a release is planned gradually and repeatedly like most software methodologies, new constraints should be identified for every software development iteration. It must be noted though, that every release needs re-planning as a

result of a probable change in users' constraints, requirements and demands in different iterations. However, if the constraints are assumed to be stable, they can be determined for each iteration. Budget and time are the most important constraints for every release development but other constraints such as technical and human resources are also influential (Saleem & Shafique, 2008).

Set Z is used to display requirement implementation constraints and Z_k is the k^{th} constraint.

$$Z = \{z_1, z_2, z_3, \dots, z_k\}$$

Once resource constraints have been determined, an algorithm is used to plan the release in this step. A review of the literature on release planning methods shows that determining the dependency between requirements should be considered prior to the implementation of the release plan(Svahnberg et al., 2010). This dependency is determined by the developer team and shows which requirements are dependent on each other. If the requirement R_i is dependent on requirement R_j, it is shown as $Req_i \rightarrow Req_j$ or (Req_i, Req_j). Therefore, set D which holds a series of all requirement interdependencies can be defined as:

$$D = \{\forall\, i,j \in m | Req_i \rightarrow Req_j\}$$

Respecting requirements' interdependencies, for every certain requirement *i* there must be a set of requirements developed on which *i* is dependent. This completes as well as complicates the release planning issue mentioned above.

The algorithm used in release planning receives the set *Req* and requirements inter-relationships to find a set of pre-release plans meeting following goals:

- Having higher priorities

- Having lower resource estimations
- Their estimations being equal to (or lower than) resource constraints
- Being selected by customers with heavier (higher order) weights
- Relative requirements being developed maintaining the relationship.

The algorithm must present one or more release plans using inputs from previous steps. Of course, the algorithm is likely to receive no plans. If so, one must modify parameters of estimated resources or increase resource constraints.

$$P = \{\forall\, i \in m | Req_i\}$$

Similar to other mentioned steps, various parameters affect the development of a good release plan. Some effective parameters on release planning algorithm include: type of planning in terms of repetition, type of algorithm, precision, and the way input data are used for computation. In fact, this step is the main difference between various release planning methodologies. Varying release planning algorithms result in different outputs of planning methodologies. No specific classification can be developed for effective parameters on release planning at this stage, but they can be partly recognized based on past experiences. Besides, most effective parameters on the two previous steps are no more effective here since they focused on algorithm inputs and data prioritization. Thus, they cannot be used in algorithm implementation unless they are being considered in a specific condition. In other words, while implementing an algorithm, parameters that are regarded the most are those which change the implementation manner not the algorithm implementation inputs. This will be discussed in details in the section on "describing effective parameters on release planning".

5.2.4. Analysis of pre-release plan and selecting the final release

Pre-release planning can be investigated and analysed from various perspectives. This is accomplished by the project manager in an Ad-Hoc. The most important ideas in analysing pre-release plans include: type of resources used in every planning, amount of resources used (particularly time and budget) and flexibility of each plan.

In addition to ideas from analysis of pre-release plan, management decisions also affect release plans. In fact, these decisions cannot be considered as ideas and are more a selection between final release plans likely to originate from ideas and management preferences of the developer organization; for instance, a selection between a plan requiring less time, a plan requiring less costs and a plan requiring less human labour. Although parameters can prioritize plans, the organization's manager or project manager makes the final decision on selecting the proper plan.

5.2.5. The process model of release planning

Based on the definition of each common step in release planning, it can be observed that there is a conceptual dependency between the outputs and inputs of these steps which will lead to the pre-planning step and the final release plan accordingly. Regardless of the release planning algorithm used in the third step and how each step is implemented, these steps can be called the "release planning process steps" which receive a set of inputs in various phases and produce a release plan as the output. Explaining how each step is implemented and describing the release planning algorithm will convert release planning process model to a precise or customized methodology.

To complement this perspective in which presented steps can be used as a release planning process and to answer the second and third questions asked earlier, we need to examine outputs and inputs of various methodologies and compare them with those of the present method. Svahnberg et al. (Svahnberg et al., 2010) studied different release

planning methods and classified a series of their required input parameters (Figure 5-1). From the figure, it is observed that "requirements dependencies" is one of the most important parameters used in about 75% of the presented release planning methods. This is followed by “effort constraints” (50%), “value factors” (37.5%), “resource constraints” (33.3%), “stakeholders' influence factors” (29.2%) and “budget and cost constraints” (29.1%). These inputs, which are considered the most typical inputs of release planning methods, are received and determined in steps 1 and 2 of the process model.

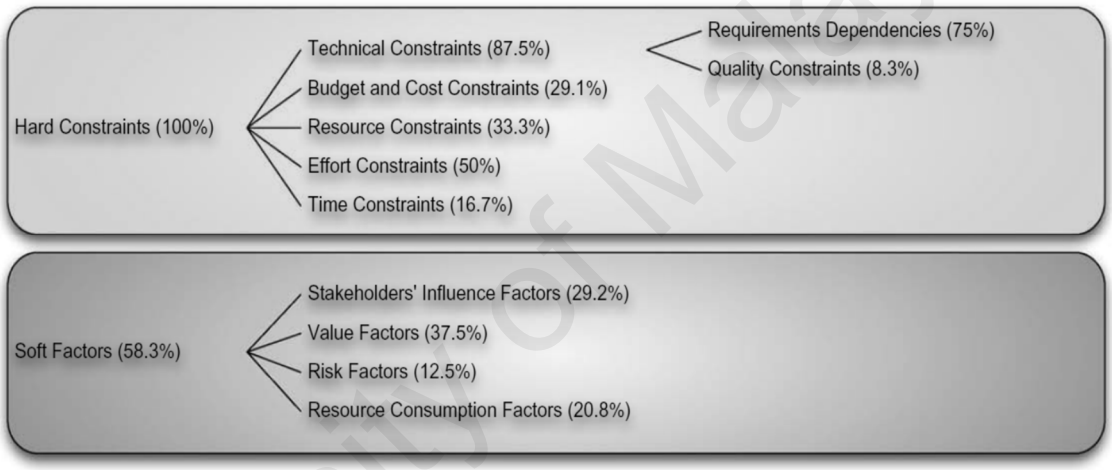


Figure 5-1: Taxonomy of requirements selection factors (Svahnberg et al., 2010)

In addition, Shafique and Saleem (Saleem & Shafique, 2008) examined the parameters of release planning methods. They concluded that requirements dependencies, resource consumption factors, effort constraints and stakeholders' influence factors, respectively, were the most important common parameters in the methods they studied. These were followed by other parameters such as budget and time. Since the number and type of constraints and estimations vary in the common steps, they do not encounter any difficulties in receiving other inputs and are considered "perfect" in terms of receiving inputs. With regard to the output, most of the releases planning methods produce one or more primary plans where the best one is then selected, and this is also supported by the common steps.

Therefore, the common steps of release planning methods can be used as the process model of release planning considering the fact that they receive various inputs and produce the expected outputs. Moreover, it has to be noted that these steps must be customized and their implementation has to be precise to achieve the proper plan. In other words, the process model explains a series of required steps to achieve a release planning but every step and its implementation mode must be described carefully. Therefore, it is necessary to determine every effective parameter in customization and explain their effects.

Figure 5-2 presents common steps in the process model of release planning, along with their inputs and outputs. Note that some common activities which solely receive inputs are not considered as an independent step and their key input is left to the appropriate key common steps. In addition, the figure shows all the input data that are significant in making the best decisions on requirements for an effective release.

This type of presentation separates the input data from the effective parameters in every certain step. The parameters in each step help the proper and precise implementation of the step. Moreover, they can be used to customize the planning process for a certain project or company. To do this, they must have specific values. As mentioned earlier, the parameter value can be identified and classified using past experiences in release planning and reviewing release planning literature. The next section describes how this is done in every step.

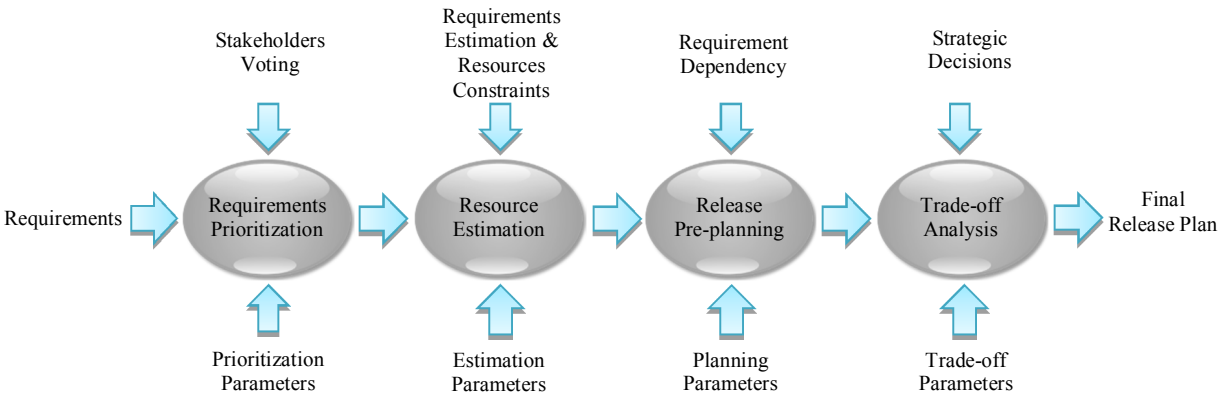


Figure 5-2: Inputs, outputs and activities in the release planning process model

5.3. Release planning process model customization

The defined process model of release planning is the result of a series of common steps in release planning methodologies. It contains a definition of every certain step and its inputs and outputs, but lacks any explanation on how every step is implemented. The process shows the tasks needed to implement a premium release planning methodology. Effort has been made to define its specifications and parameters so that it can cover most methodologies by simply altering inputs and outputs. Covering various release planning tasks highly depends on the customization of release planning process model. This customization means to correctly value effective parameters on every step in order to develop a precise guide for that step. To do this, the effective parameters must be identified, described and valued and it is necessary to determine their effect on method selection. Below, we will first describe parameters effective on every step of release planning process model and then will examine their influence on method selection in all steps. Finally, we will examine the influence of parameters' inter-relationship.

5.3.1. Customization of requirements prioritization

Requirements prioritization is the first step in regulating requirements, the objective of which is to facilitate requirement analysis in the next steps. Various steps of this task are presented below.

5.3.1.1. Effective parameters on requirement prioritization

In its simplest form, this task is accomplished in an ad-hoc manner regardless of any given parameters. In its current form, which is adopted in most release planning methods, priority is given by the stakeholders and is then integrated with the developers' vote, hence, requirement priority is determined (Svahnberg et al., 2010). The most popular requirement prioritization method used in release planning methodologies is "Analytical Hierarchy Process" (AHP) (Berander & Andrews, 2005; Slooten, 2012), which was introduced by Thomas Saaty in 1970s. The process in this method is designed in a way that it is consistent and can be associated with human intellect and nature. Technically, AHP is one of the most comprehensive systems designed for decision-making with multiple measures since it allows the formulation of the problem in a hierarchical manner and enables one to consider various quantitative and qualitative measures (Saaty, 1980; Slooten, 2012). In addition, there are other methods for requirements prioritization, some of which are mentioned below (Berander & Andrews, 2005; Durillo et al., 2011):

- Cumulative Voting or the 100-Dollar test
- Numerical Assignment (Grouping)
- Ranking
- Top-Ten Requirements
- Quality Function Development (QFD)
- Cost-Value Approach

These are not the only existing methods, of course; other prioritization techniques are B-Tree based methods, Quality-based methods, genetic Algorithm or Value-based methods (Chatzipetrou, Angelis, Rovegard, & Wohlin, 2010; Iqbal, Zaidi, & Murtaza,

2010; Marjaie & Kulkarni, 2010; Ninaus, 2012; Otero, Dell, Qureshi, & Otero, 2010; Perini et al., 2012; Racheva, Daneva, Herrmann, & Wieringa, 2010; Tonella, Susi, & Palma, 2010). These methods try to increase the requirements prioritization quality by decreasing the number of comparisons, emphasizing certain specifications in prioritization, and lessening the complexity. Nevertheless, many software development teams are wondering how to select the proper method to prioritize their software. Most teams seek a simpler prioritization method and go for methods such as ad-hoc and Numerical Assignment (Svensson et al., 2011). Finding the answer to the above question can clarify some requirements prioritization parameters, which are also related to method selection.

Aasem, M. et al. (Aasem, Ramzan, & Jaffar, 2010) compared existing requirements prioritization methods and suggested measures to evaluate them. Some of these measures, including scale and granularity, are considered parameters for prioritization and implementation type. Table 5-2 shows this classification, which helps to find best-fitted requirements prioritization methods based upon specific parameters. Requirements manager can determine the value of each parameter for his/her project and find out the proper method quickly.

According to these measures, every prioritization method is only suitable for certain cases. For example, since AHP, B-Tree and 100-Dollar Test are complicated methods, small-size companies with a limited number of stakeholders are not expected to be able to use them. Therefore, the number of stakeholders involved in requirements prioritization and the size of the software development team or company can partly determine or limit a certain prioritization method. This is also true about granularity which shows the precision of every output, e.g. fine, medium, coarse or extremely coarse. If the requirements are to be prioritized carefully, the methods with a fine

granularity are preferred; but when precision is less important, the methods with a coarse granularity can be employed.

The identified parameters can be used as a classifier for requirements prioritization methods. In fact, finding such specifications that can narrow down the prioritization methods and identify the parameters to be used in prioritization will help customize the requirements prioritization step and therefore, can determine various customization parameters. In addition, similar to Aasem, M. et al. (Aasem et al., 2010) in which every parameter was classified and methods were placed in these classes, primary classifications must be made for every requirement prioritization parameter.

Table 5-2: Classification of prioritization methods (Aasem et al., 2010)

Technique	Scale	Granularity	Sophistication	Aspect	Perspective	Type
AHP	Ratio	Fine	Very Complex	Strategic importance, Penalty	Product Manager	Algorithmic
B-Tree	--	Fine	Complex	-	-	Algorithmic
100-Dollars Test	Ratio	Fine	Complex	Customer importance	Customers	Manual
Ranking	Ordinal	Medium	Easy	Volatility	Requirements Specialist	Manual
Numerical Assignment	Ordinal	Coarse	Very Easy	Time, Risk	Project Manager, Requirements Specialist	Manual
Top 10	---	Extremely Coarse	Extremely Easy	Customer importance	Customers	Manual

In two independent studies, Kashif Ahmed Khan et al. (Berander, Khan, & Lehtola, 2006; Khan, 2006) examined requirements prioritization methods and presented parameters for comparison. These parameters are presented in Table 5-3. As can be seen in the table, methods are studied from various aspects.

Table 5-3: Comparison parameters of requirements prioritization methods

Variable Type	Variable	Sub-Variables
Independent	Qualitative Process Description	
	Goal	
	Hierarchy Level	
	Input	Lower-level approaches used
		Aspects taken into account
	Output	
Dependent	Time	
	Accuracy	
	Ease of Learning	
	Ease of Use	
	Fault Tolerance	
	Scalability	
	Understandability of Results	
	Attractiveness	
Context	Environment	Type of market
		Process model
		Phase of prioritization
		Size of the project and organization
		Application domain
	Study Setup	Prioritization tools
		Work mode
		Location/Amount of control
		Duration of study
		Selection strategy for prioritization approach
		Role of the researcher
	Subjects	Roles/Perspectives
		Commitment
		Experience
		View on software development
		Gender and Age
	Requirement	Number of requirements
		Type of requirements
		Abstraction level
		Structure

Out of these studies and other corresponding literature (Herrmann & Paech, 2008; Ma, 2009; Marjaie & Kulkarni, 2010), a list of specifications can be obtained in which the best prioritization methods are described. To achieve this goal, these specifications and their correlated objectives were studied and a set of such specifications with their allowed values was obtained. Table 5-4 shows these parameters along with their instances. An instance of every parameter represents allowed values for that parameter and can be added later to expand the method.

Table 5-4: Parameters of requirements prioritization

Parameter	Allowed Value	Description
Market type (MT)	Customized (MT_1)	The software is designed and developed for a certain costumer.
	Limited customer (MT_2)	The number of customers is limited.
	Unlimited customer (MT_3)	The number of customers is unlimited.
Development methodology (DM)	Waterfall (DM_1)	Requirements are perceived at the beginning of the project.
	Agile (DM_2)	Requirements are perceived and revised at the beginning of each iteration.
	RUP (DM_3)	Requirements are perceived and revised at the beginning of each iteration.
	RAD (DM_4)	Requirements are perceived at the beginning and after producing a protoype.
Team size (TS)	Low (TS_1)	Just one individual votes on requirements prioritization.
	Medium (TS_2)	1-3 individuals vote on requirements prioritization.
	High(TS_3)	More than 3 individuals vote on requirements prioritization.
Requirements number (RN)	Low (RN_1)	Less than 20 requirements exist.
	Medium (RN_2)	Between 21 and 50 requirements exist.
	High (RN_3)	More than 51 requirements exist.
Requirements granularity	Fine (RG_1)	Proposed requirements are in the technical level and do not need to be broken up.
	Medium (RG_2)	Proposed requirements must be divided into 2 or 3 fine requirements.
	Coarse (RG_3)	Proposed requirements must be divided into 2 or more medium requirements.
Number of prioritization inputs (PI)	Low (PI_1)	Only one prioritization input (usually requirement value factor) is valued by stakeholders or members of the development team.
	Medium (PI_2)	2 or 3 prioritization inputs are valued by stakeholders or members of the development team.
	High (PI_3)	More than 3 prioritization inputs are valued by stakeholders or members of the development team.
Team experiences (TE)	Experienced (TE_1)	The team has implemented more than 3 software projects in the considered field.
	Half experienced (TE_2)	The team has implemented less than 3 software projects in the considered field but had implemented projects in similar and relevant fields.

Parameter	Allowed Value	Description
	Inexperienced (TE_3)	The team has no experience of project implementation in the considered field or relevant scope.
Development environment (DE)	Web-based (DE_1)	The software is developed as a web-based one and users access it through the web.
	Client-server (DE_2)	The software has a server and set of clients.
	Desktop (DE_3)	The software is installed on a personal computer.

These mentioned parameters of prioritization methods have resulted from studies on release planning. But naturally they are not perfect and it is possible to add new parameters; this enables expansion of the method. Moreover, since this kind of methodology did not exist in previous studies and those trying to classify different release planning methods did it solely by comparison and without using these parameters to select the proper release planning methodology, the parameters can be considered the first group of release planning parameters which are developed through a pattern-based methodology. One point to be considered is the ability to combine this with the next step. Although there are some release planning methodologies in which requirement prioritization and resource estimation are done simultaneously (Berander & Andrews, 2005), the two are entirely separate in the process model and requirements are prioritized regardless of resource estimations. In other words, prioritization in this step is performed based on parameters (such as stakeholders' priorities, risk, developer team's priorities, etc.) which do not need resource estimation. Parameters such as costs, time and labor (requiring resource estimation) are considered in the next step and selection is accomplished in release planning based on these inputs. Separating these parameters can be result in more precise estimation and improved efficiency of the method.

5.3.1.2. Effect of parameters on requirements prioritization method

The effect of every parameter on requirements prioritization method can be determined based on parameters themselves and their instances. Table 5-5 examines the effect of every parameter instance independently. It is observed that some parameter instances can directly and strongly determine the requirements prioritization method but all instances must be considered to be able to specify the method precisely.

Table 5-5: Effect of instances on the requirements prioritization method

Parameter	Instance	Description
Market type	Customized	Every method can be used. Therefore, method selection is dependent on other parameters and no method can be deduced directly.
	Limited customer	Every method can be used. Therefore, method selection is dependent on other parameters and no method can be deduced directly.
	Unlimited customer	Methods which can manage high volume of requirements number are used here. Method selection is also dependent on other parameters and no method can be deduced directly.
Development methodology	Waterfall	Iterative methods are less considered and requirements prioritization method must preferably be based on methodology's specifications. Method selection is also dependent on other parameters and no method can be deduced directly.
	Agile	Iterative methods are more considered. Methods such as "Planning Game" are specifically designed for these methodologies. Method selection is also dependent on other parameters and no method can be deduced directly.
	RUP	Iterative methods are more considered. Method selection is also dependent on other parameters and no method can be deduced directly.
	RAD	Iterative methods and frequent changes of supporting requirements are more considered. Method selection is also dependent on other parameters and no method can be deduced directly.
Team size	Small	Lighter and simpler methods (such as Top 10 and Numerical Assignment) are often used. Method selection is also dependent on other parameters and no method can be deduced directly.
	Medium	Every method can be used. Therefore, method selection is dependent on other parameters and no method can be deduced directly.
	Big	More complicated and stronger methods (such as AHP and B-Tree) are often used. Method selection is also dependent on other parameters and no method can be deduced directly.

Parameter	Instance	Description
Requirements number	Small	Methods such as Top 10 and Numerical Assignment can be used. Method selection is also dependent on other parameters and no method can be deduced directly.
	Medium	Most methods can be used. Method selection is also dependent on other parameters and no method can be deduced directly.
	Big	Methods such as Top 10 which are performed manually (cannot be made automatic) are too demanding and are not suitable. Method selection is also dependent on other parameters and no method can be deduced directly.
Team experiences	Experienced	Every method can be used. Method selection is dependent on other parameters and no method can be deduced directly.
	Half experienced	Every method can be used. Method selection is dependent on other parameters and no method can be deduced directly.
	Inexperienced	Methods emphasizing requirement value and stakeholders' inputs are more considered since the team is often not able to comment on the requirements value. Method selection is also dependent on other parameters and no method can be deduced directly.
Development environment	Web-based	Methods emphasizing stakeholders' vote are more considered since individuals' opinions are important in this type of software. Method selection is also dependent on other parameters and no method can be deduced directly.
	Client-server	Every method can be used. Method selection is dependent on other parameters and no method can be deduced directly.
	Desktop	Every method can be used. Method selection is dependent on other parameters and no method can be deduced directly.
Requirements level	Fine	In a research (Aasem et al., 2010) investigating requirements level in prioritization methods, methodologies are classified according to the requirements level. It is demonstrated in this research that methods such as AHP, B-Tree and 100-Dolars test are suitable for this level of requirements. Method selection is also dependent on other parameters and no method can be deduced directly.
	Medium	In a research (Aasem et al., 2010) investigating requirements level in prioritization methods, methodologies are classified according to the requirements level. It is demonstrated in this research that Ranking method is suitable for this level of requirements. Method selection is also dependent on other parameters and no method can be deduced directly.
	Coarse	In a research (Aasem et al., 2010) investigating requirements level in prioritization methods, methodologies are classified according to the requirements level. It is demonstrated in this research that Numerical Assignment and Top 10 methods are suitable for this level of requirements. Method selection is also dependent on other parameters and no method can be deduced directly.

Parameter	Instance	Description
Number of prioritization inputs	Small numbers	In a research (Aasem et al., 2010) investigating requirements level in prioritization methods, methodologies are classified according to the requirements level. Following methods are proposed for 1 prioritization input: • Fine: 100-Dolars test • Medium: 100-Dolars test, Ranking • Coarse: Top 10 Precise method selection is dependent on other parameters.
	Medium numbers	In a research (Aasem et al., 2010) investigating requirements level in prioritization methods, methodologies are classified according to the requirements level. Following methods are proposed for 1 prioritization input: • Fine: AHP, B-Tree • Medium: AHP, Ranking • Coarse: Ranking, Numerical Assignment Precise method selection is dependent on other parameters.
	Big numbers	In a research (Aasem et al., 2010) investigating requirements level in prioritization methods, methodologies are classified according to requirements level. Following methods are proposed for 1 prioritization input: • Fine: AHP, B-Tree • Medium: AHP, Ranking • Coarse: Ranking, Numerical Assignment Precise method selection is dependent on other parameters.

Although customization can be accomplished using requirements parameters and their instances, the relationship between parameters and instances must be specified clearly in order to determine proper method(s) for every set of parameters.

5.3.1.3. The relationship between effective parameters on requirements prioritization

It is essential to determine the relationship between parameters and their instances in order to make use of them in selecting prioritization method. Two parameters can have direct, reverse or no relationship. In a direct relationship, every parameter can affect the other and both parameters are necessary for the method selection. In a reverse relationship, presence of one parameter makes the other insignificant. Also, the parameters can have no relationship and have no effect on each other. The relationship between every parameter instance and all other parameters is examined in order to explain every certain relationship.

Table 5-6: Relations between "market type" instances and other parameters

Parameter	Instance	Description
Development methodology	Waterfall	This method cannot be applied in limited or unlimited customer cases since requirements in these cases change and they cannot be perceived at the beginning of the project. However, this is possible for customized software.
	Agile	This can be used in all three market types.
	RUP	This is not efficient in unlimited customer cases. Those cases require agility which is absent in RUP. However, it can be used for customized and limited customer cases.
	RAD	This can be used in all three market types.
Team size	Small	Small teams can be used in customized market. But it is not responsive in limited or unlimited customer cases.
	Medium	This can be used in all three market types.
	Big	This can be used in all three market types.
Requirements number	Small	The number of requirements is 20 in most limited and unlimited customer software; hence, the small number of requirements is only possible in customized software.
	Medium	The requirements number can be medium in all three market types.
	Big	The requirements number can be big in all three market types.
Requirements level	Fine	The requirements level can be fine in all three market types.
	Medium	The requirements level can be medium in all three market types.
	Coarse	The requirements level can be coarse in all three market types.
Number of prioritization inputs	Small numbers	The number of prioritization inputs can be 1 in all market types.
	Medium numbers	The number of prioritization inputs can be 2 or 3 in customized and limited customer software.
	Big numbers	More than 3 prioritization inputs can exist in customized software, but for limited or unlimited customer cases the number progressively increase and a large number of activities need to be done on every requirement.
Team experience	Experienced	Experienced teams can enter all market types.
	Half-experienced	Half-experienced teams can enter all market types.

Parameter	Instance	Description
	Inexperienced	Inexperienced teams cannot be used in customized and limited customer software, since it is necessary for the team to recognize customers' concerns. However, they can be used in unlimited customer market.
Development environment	Web-based	It is mostly used in limited and unlimited customer software.
	Client-server	It is mostly used in limited customer and customized software.
	Desktop	It is only used in customized software.

As observed in the above table, some instances contradict market type instances and are less likely (or unlikely) to occur for two parameters. A set of ordered pairs can be defined as follows for those instances of two parameters which coincide:

$$K_{MTi} = \{x \in MT, y \in \{DM, TS, RN, RG, PI, TE, DE\} | (x, y)\}, \quad for\ all\ \ i \in \{DM, TS, RN, RG, PI, TE, DE\}$$

According to this definition, members of every *K* set are as follows:

$K_{MTDM} = \{(\mathrm{MT_1}, \mathrm{DM_1}), (\mathrm{MT_1}, \mathrm{DM_2}), (\mathrm{MT_1}, \mathrm{DM_3}), (\mathrm{MT_1}, \mathrm{DM_4}), (\mathrm{MT_2}, \mathrm{DM_2}), (\mathrm{MT_2}, \mathrm{DM_3}), (\mathrm{MT_2}, \mathrm{DM_4}), (\mathrm{MT_3}, \mathrm{DM_2}), (\mathrm{MT_3}, \mathrm{DM_4})\}$

$K_{MTTS} = \{(\mathrm{MT_1}, \mathrm{TS_1}), (\mathrm{MT_1}, \mathrm{TS_2}), (\mathrm{MT_1}, \mathrm{TS_3}), (\mathrm{MT_2}, \mathrm{TS_2}), (\mathrm{MT_2}, \mathrm{TS_3}), (\mathrm{MT_3}, \mathrm{TS_2}), (\mathrm{MT_3}, \mathrm{TS_3})\}$

$K_{MTRN} = \{(\mathrm{MT_1}, \mathrm{RN_1}), (\mathrm{MT_1}, \mathrm{RN_2}), (\mathrm{MT_1}, \mathrm{RN_3}), (\mathrm{MT_2}, \mathrm{RN_2}), (\mathrm{MT_2}, \mathrm{RN_3}), (\mathrm{MT_3}, \mathrm{RN_2}), (\mathrm{MT_3}, \mathrm{RN_3})\}$

$K_{MTRG} = \{(\mathrm{MT_1}, \mathrm{RG_1}), (\mathrm{MT_1}, \mathrm{RG_2}), (\mathrm{MT_1}, \mathrm{RG_3}), (\mathrm{MT_2}, \mathrm{RG_1}), (\mathrm{MT_2}, \mathrm{RG_2}), (\mathrm{MT_2}, \mathrm{RG_3}), (\mathrm{MT_3}, \mathrm{RG_1}), (\mathrm{MT_3}, \mathrm{RG_2}), (\mathrm{MT_3}, \mathrm{RG_3})\}$

$K_{MTPI} = \{(\mathrm{MT_1}, \mathrm{PI_1}), (\mathrm{MT_1}, \mathrm{PI_2}), (\mathrm{MT_1}, \mathrm{PI_3}), (\mathrm{MT_2}, \mathrm{PI_1}), (\mathrm{MT_2}, \mathrm{PI_2}), (\mathrm{MT_3}, \mathrm{PI_1})\}$

$K_{MTTE} = \{(\mathrm{MT_1}, \mathrm{TE_1}), (\mathrm{MT_1}, \mathrm{TE_2}), (\mathrm{MT_2}, \mathrm{TE_1}), (\mathrm{MT_2}, \mathrm{TE_2}), (\mathrm{MT_3}, \mathrm{TE_1}), (\mathrm{MT_3}, \mathrm{TE_2}), (\mathrm{MT_3}, \mathrm{TE_3})\}$

$K_{MTDE} = \{(\mathrm{MT_1}, \mathrm{DE_2}), (\mathrm{MT_1}, \mathrm{DE_3}), (\mathrm{MT_2}, \mathrm{DE_1}), (\mathrm{MT_2}, \mathrm{DE_2}), (\mathrm{MT_3}, \mathrm{DE_1})\}$

The sets include 50 ordered pairs that each of which represents the relationship between two instances of two parameters. Every pair can be considered as a combination likely to lead to a prioritization method. The likelihood becomes absolute when the pair creates a combination of all possible states along with other parameters. Other states absent in this pair, as mentioned earlier, are either invalid or unlikely and they can be ignored. The probability of every state originates from the instance analysis of each parameter and the best experiences recorded for that parameter instance. For example,

avoiding RUP for unlimited customer market type is due to the fact that the methodology is heavy weighted, requires certain documentations, and it is a predictor method while this market type requires an agile and rapid method to respond to customers' demands not already determined. This does not make software developing companies to neglect RUP in unlimited customer projects, but the best practice is always to consider the characteristics of the two instances.

Table 5-7 shows the relationship between "development methodology" and other requirements prioritization parameters.

Table 5-7: Relations between "development methodology" and other parameters

Parameter	Instance	Description
Team size	Small	Agile, RAD and Waterfall methodologies can be used in small teams.
	Medium	All methodologies can be used.
	Big	RUP and Waterfall methodologies can be used.
Requirements number	Small	All methodologies can be used.
	Medium	Agile, RUP and Waterfall methodologies can be used but RAD cannot support medium number of requirements because of the implementation nature.
	Big	RUP and Waterfall methodologies can be used but RAD and Agile methods cannot support big number of requirements because of the implementation nature.
Requirements level	Fine	All methodologies can be used.
	Medium	All methodologies can be used.
	Coarse	RUP and Waterfall methodologies can manage coarse requirements level considering their structure and documentations.
Number of prioritization inputs	Small numbers	All methodologies can be used.
	Medium numbers	All methodologies can be used.
	Big numbers	All methodologies can be used.

Parameter	Instance	Description
Team experience	Experienced	All methodologies can be used.
	Half-experienced	All methodologies can be used.
	Inexperienced	All methodologies can be used.
Development environment	Web-based	Agile and RUP methodologies can be used.
	Client-server	All methodologies can be used.
	Desktop	All methodologies can be used.

According to Table 5-7, some instances of development methodologies contradict other parameter instances. A set of ordered pair is defined as follows for those coinciding parameter instances:

$$K_{DMi} = \{x \in DM, y \in \{TS, RN, RG, PI, TE, DE\} | (x, y)\}, \qquad for\ all\ \ i \in \{TS, RN, RG, PI, TE, DE\}$$

According to this definition, members of every *K* set are as follows:

$K_{DMTS} = \{(DM_1, TS_1), (DM_1, TS_2), (DM_1, TS_3), (DM_2, TS_1), (DM_2, TS_2), (DM_3, TS_2), (DM_3, TS_3), (DM_4, TS_1), (DM_4, TS_2)\}$

$K_{DMRN} = \{(DM_1, RN_1), (DM_1, RN_2), (DM_1, RN_3), (DM_2, RN_1), (DM_2, RN_2), (DM_3, RN_1), (DM_3, RN_2), (DM_3, RN_3), (DM_4, RN_1)\}$

$K_{DMRG} = \{(DM_1, RG_1), (DM_1, RG_2), (DM_1, RG_3), (DM_2, RG_1), (DM_2, RG_2), (DM_3, RG_1), (DM_3, RG_2), (DM_3, RG_3), (DM_4, RG_1), (DM_4, RG_2)\}$

$K_{DMPI} = \{(DM_1, PI_1), (DM_1, PI_2), (DM_1, PI_3), (DM_2, PI_1), (DM_2, PI_2), (DM_2, PI_3), (DM_3, PI_1), (DM_3, PI_2), (DM_3, PI_3), (DM_4, PI_1), (DM_4, PI_2), (DM_4, PI_3)\}$

$K_{DMTE} = \{(DM_1, TE_1), (DM_1, TE_2), (DM_1, TE_3), (DM_2, TE_1), (DM_2, TE_2), (DM_2, TE_3), (DM_3, TE_1), (DM_3, TE_2), (DM_3, TE_3), (DM_4, TE_1), (DM_4, TE_2), (DM_4, TE_3)\}$

$K_{DMDE} = \{(DM_1, DE_2), (DM_1, DE_3), (DM_2, DE_1), (DM_2, DE_2), (DM_2, DE_3), (DM_3, DE_2), (DM_3, DE_3), (DM_4, DE_1), (DM_4, DE_2), (DM_4, DE_3)\}$

These sets contain 62 ordered pairs each of which represents the relationship between two instances of two parameters and forms a certain state.

Table 5-8 shows the relationship between "team size" and other parameters.

Table 5-8: Relations between "team size" and other parameters

Parameter	Instance	Description
Requirements number	Small	Whatever the team size is, it can manage small number of requirements.
	Medium	A small team cannot manage medium number of requirements in a proper time. Thus, team size must increase.
	Big	A big team can manage large number of requirements.
Requirements level	Fine	Whatever the team size is, it can manage this level of requirements.
	Medium	Whatever the team size is, it can manage this level of requirements.
	Coarse	Whatever the team size is, it can manage this level of requirements.
Number of prioritization inputs	Small numbers	Whatever the team size is, it can manage 1 input for requirements prioritization.
	Medium numbers	Whatever the team size is, it can manage 2 or 3 inputs for requirements prioritization.
	Big numbers	Whatever the team size is, it can manage more than 3 inputs for requirements prioritization.
Team experiences	Experienced	Whatever the team size is, it can be experienced.
	Half experienced	Whatever the team size is, it can be half-experienced.
	Inexperienced	Whatever the team size is, it can be inexperienced.
Development environment	Web-based	Whatever the team size is, it can develop web-based software.
	Client-server	Whatever the team size is, it can develop client-server software.
	Desktop	Whatever the team size is, it can develop desktop software.

The following ordered pair is defined for two coinciding parameter instances:

$$K_{TSi} = \{x \in TS, y \in \{RN, RG, PI, TE, DE\} | (x, y)\} \ , \qquad for\ all\ \ i \in \{RN, RG, PI, TE, DE\}$$

According to this definition, members of every *K* set are as follows:

$K_{TSRN} = \{(\mathrm{TS_1}, \mathrm{RN_1}), (\mathrm{TS_2}, \mathrm{RN_1}), (\mathrm{TS_2}, \mathrm{RN_2}), (\mathrm{TS_3}, \mathrm{RN_1}), (\mathrm{TS_3}, \mathrm{RN_2}), (\mathrm{TS_3}, \mathrm{RN_3})\}$

$K_{TSRG} = \{(\mathrm{TS_1}, \mathrm{RG_1}), (\mathrm{TS_1}, \mathrm{RG_2}), (\mathrm{TS_1}, \mathrm{RG_3}), (\mathrm{TS_2}, \mathrm{RG_1}), (\mathrm{TS_2}, \mathrm{RG_2}), (\mathrm{TS_2}, \mathrm{RG_3}), (\mathrm{TS_3}, \mathrm{RG_1}), (\mathrm{TS_3}, \mathrm{RG_2}), (\mathrm{TS_3}, \mathrm{RG_3})\}$

$K_{TSPI} = \{(\mathrm{TS_1}, \mathrm{PI_1}), (\mathrm{TS_1}, \mathrm{PI_2}), (\mathrm{TS_1}, \mathrm{PI_3}), (\mathrm{TS_2}, \mathrm{PI_1}), (\mathrm{TS_2}, \mathrm{PI_2}), (\mathrm{TS_2}, \mathrm{PI_3}), (\mathrm{TS_3}, \mathrm{PI_1}), (\mathrm{TS_3}, \mathrm{PI_2}), (\mathrm{TS_3}, \mathrm{PI_3})\}$

$K_{TSTE} = \{(TS_1, TE_1), (TS_1, TE_2), (TS_1, TE_3), (TS_2, TE_1), (TS_2, TE_2), (TS_2, TE_3), (TS_3, TE_1), (TS_3, TE_2), (TS_3, TE_3)\}$

$K_{TSDE} = \{(TS_1, DE_1), (TS_1, DE_2), (TS_1, DE_3), (TS_2, DE_1), (TS_2, DE_2), (TS_2, DE_3), (TS_3, DE_1), (TS_3, DE_2), (TS_3, DE_3)\}$

These sets contain 42 ordered pairs, each of which represents the relationship between two instances of two parameters and forms a certain case.

Table 5-9 shows the relationship between "requirements number" and other parameters.

Table 5-9: Relations between "requirements number" and other parameters

Parameter	Instance	Description
Requirements level	Fine	This level can contain every number of requirements.
	Medium	This level can contain every number of requirements.
	Coarse	This level can have small or medium number of requirements. If the requirements number is big, requirements level declines and approaches fine level. Hence, the coarse level is not seen if requirements number is big.
Number of prioritization inputs	Small numbers	Whatever the requirements number is, number of prioritization inputs can be 1.
	Medium numbers	Whatever the requirements number is, number of prioritization inputs can be 2 or 3.
	Big numbers	An increase in prioritization inputs enhances the activities to be done on every requirement. Hence, if there are more than 3 inputs, requirements number must be small or medium.
Team experience	Experienced	An experienced team can manage every number of requirements.
	Half-experienced	A half-experienced team can manage every number of requirements.
	Inexperienced	An inexperienced team can manage every number of requirements.
Development environment	Web-based	Whatever the requirements number is, it can be presented in web-based software.
	Client-server	Whatever the requirements number is, it can be presented in client-server software.
	Desktop	Whatever the requirements number is, it can be presented in desktop software.

The following ordered pair is defined for two coinciding parameter instances:

$$K_{RNi} = \{x \in RN, y \in \{RG, PI, TE, DE\} | (x, y)\}, \qquad for\ all\ i \in \{RG, PI, TE, DE\}$$

According to this definition, members of every *K* set are as follows:

$K_{RNRG} = \{(\mathrm{RN_1}, \mathrm{RG_1}), (\mathrm{RN_1}, \mathrm{RG_2}), (\mathrm{RN_1}, \mathrm{RG_3}), (\mathrm{RN_2}, \mathrm{RG_1}), (\mathrm{RN_2}, \mathrm{RG_2}), (\mathrm{RN_2}, \mathrm{RG_3}), (\mathrm{RN_3}, \mathrm{RG_1}), (\mathrm{RN_3}, \mathrm{RG_2})\}$

$K_{RNPI} = \{(\mathrm{RN_1}, \mathrm{PI_1}), (\mathrm{RN_1}, \mathrm{PI_2}), (\mathrm{RN_1}, \mathrm{PI_3}), (\mathrm{RN_2}, \mathrm{PI_1}), (\mathrm{RN_2}, \mathrm{PI_2}), (\mathrm{RN_2}, \mathrm{PI_3}), (\mathrm{RN_3}, \mathrm{PI_1}), (\mathrm{RN_3}, \mathrm{PI_2})\}$

$K_{RNTE} = \{(\mathrm{RN_1}, \mathrm{TE_1}), (\mathrm{RN_1}, \mathrm{TE_2}), (\mathrm{RN_1}, \mathrm{TE_3}), (\mathrm{RN_2}, \mathrm{TE_1}), (\mathrm{RN_2}, \mathrm{TE_2}), (\mathrm{RN_2}, \mathrm{TE_3}), (\mathrm{RN_3}, \mathrm{TE_1}), (\mathrm{RN_3}, \mathrm{TE_2}), (\mathrm{RN_3}, \mathrm{TE_3})\}$

$K_{RNDE} = \{(\mathrm{RN_1}, \mathrm{DE_1}), (\mathrm{RN_1}, \mathrm{DE_2}), (\mathrm{RN_1}, \mathrm{DE_3}), (\mathrm{RN_2}, \mathrm{DE_1}), (\mathrm{RN_2}, \mathrm{DE_2}), (\mathrm{RN_2}, \mathrm{DE_3}), (\mathrm{RN_3}, \mathrm{DE_1}), (\mathrm{RN_3}, \mathrm{DE_2}), (\mathrm{RN_3}, \mathrm{DE_3})\}$

These sets contain 34 ordered pairs, each of which represents the relationship between two instances of two parameters and forms a certain state.

Table 5-10shows the relationship between "requirements level" and other parameters.

Table 5-10: Relations between "requirements level" and other parameters

Parameter	Instance	Description
Number of prioritization inputs	Small numbers	Whatever the requirements level is, number of prioritization inputs can be 1.
	Medium numbers	Whatever the requirements level is, number of prioritization inputs can be 2 or 3.
	Big numbers	More precise requirements level (closer to Fine) can cause an increase in prioritization inputs number. Because of lack of precision in Coarse level, only a limited number of inputs can be predicted (to be considered for input prioritization). Therefore, Fine and Medium levels only can have more than 3 inputs.
Team experiences	Experienced	An experienced team can manage all requirements levels.
	Half-experienced	A half-experienced team can manage all requirements levels.
	Inexperienced	An inexperienced team can manage all requirements levels.
Development environment	Web-based	All requirements levels can be presented in web-based software.
	Client-server	All requirements levels can be presented in client-server software.
	Desktop	All requirements levels can be presented in desktop software.

The following ordered pair is defined for two coinciding parameter instances:

$$K_{RGi} = \{x \in RG, y \in \{PI, TE, DE\} | (x, y)\} \ , \qquad for\ all\ \ i \in \{PI, TE, DE\}$$

According to this definition, members of every *K* set are as follows:

$K_{RGPI} = \{(\mathrm{RG_1}, \mathrm{PI_1}), (\mathrm{RG_1}, \mathrm{PI_2}), (\mathrm{RG_1}, \mathrm{PI_3}), (\mathrm{RG_2}, \mathrm{PI_1}), (\mathrm{RG_2}, \mathrm{PI_2}), (\mathrm{RG_2}, \mathrm{PI_3}), (\mathrm{RG_3}, \mathrm{PI_1}), (\mathrm{RG_3}, \mathrm{PI_2})\}$

$K_{RGTE} = \{(\mathrm{RG_1}, \mathrm{TE_1}), (\mathrm{RG_1}, \mathrm{TE_2}), (\mathrm{RG_1}, \mathrm{TE_3}), (\mathrm{RG_2}, \mathrm{TE_1}), (\mathrm{RG_2}, \mathrm{TE_2}), (\mathrm{RG_2}, \mathrm{TE_3}), (\mathrm{RG_3}, \mathrm{TE_1}), (\mathrm{RG_3}, \mathrm{TE_2}), (\mathrm{RG_3}, \mathrm{TE_3})\}$

$K_{RGDE} = \{(\mathrm{RG_1}, \mathrm{DE_1}), (\mathrm{RG_1}, \mathrm{DE_2}), (\mathrm{RG_1}, \mathrm{DE_3}), (\mathrm{RG_2}, \mathrm{DE_1}), (\mathrm{RG_2}, \mathrm{DE_2}), (\mathrm{RG_2}, \mathrm{DE_3}), (\mathrm{RG_3}, \mathrm{DE_1}), (\mathrm{RG_3}, \mathrm{DE_2}), (\mathrm{RG_3}, \mathrm{DE_3})\}$

These sets contain 26 ordered pairs, each of which represents the relationship between two instances of two parameters and forms a certain state.

Table 5-11shows the relationship between "number of prioritization inputs" and other parameters.

Table 5-11: Relations between "number of prioritization inputs" and other parameters

Parameter	Instance	Description
Team experiences	Experienced	An experienced team can manage every number of prioritization inputs.
	Half-experienced	A half-experienced team can manage every number of prioritization inputs.
	Inexperienced	An inexperienced team can manage every number of prioritization inputs.
Development environment	Web-based	Every number of prioritization inputs can be considered in web-based software.
	Client-server	Every number of prioritization inputs can be considered in client-server software.
	Desktop	Every number of prioritization inputs can be considered in desktop software.

The following ordered pair is defined for two coinciding parameter instances:

$$K_{PIi} = \{x \in PI, y \in \{TE, DE\} | (x, y)\} \ , \qquad for\ all\ \ i \in \{TE, DE\}$$

According to this definition, members of every *K* set are as follows:

$K_{PITE} = \{(PI1, TE1), (PI1, TE2), (PI1, TE3), (PI2, TE1), (PI2, TE2), (PI2, TE3), (PI3, TE1), (PI3, TE2), (PI3, TE3)\}$

$K_{PIDE} = \{(PI1, DE1), (PI1, DE2), (PI1, DE3), (PI2, DE1), (PI2, DE2), (PI2, DE3), (PI3, DE1), (PI3, DE2), (PI3, DE3)\}$

These sets contain 18 ordered pairs, each of which represents the relationship between two instances of two parameters and forms a certain state.

Table 5-12 shows the relationship between "team experience" and other parameters.

Table 5-12: Relations between "team experience" and other parameters

Parameter	Instance	Description
Development environment	Web-based	Whatever a team's experience level is, it can develop web-based software.
	Client-server	Whatever a team's experience level is, it can develop client-server software.
	Desktop	Whatever a team's experience level is, it can develop desktop software.

The following ordered pair is defined for two coinciding parameter instances:

$$K_{TEDE} = \{x \in TE, y \in \{DE\} | (x, y)\}$$

According to this definition, members of every *K* set are as follows:

$K_{TEDE} = \{(TE_1, DE_1), (TE_1, DE_2), (TE_1, DE_3), (TE_2, DE_1), (TE_2, DE_2), (TE_2, DE_3), (TE_3, DE_1), (TE_3, DE_2), (TE_3, DE_3)\}$

This set contains 9 ordered pairs, each of which represents the relationship between two instances of two parameters and forms a certain state.

Every set of ordered pairs is considered a definite state. Each set is combined with other sets of ordered pairs with which it has a common point and forms a set of common ternaries. Every common ternary represents a combination of an instance of three parameters, the accuracy of which must be determined like the relationship between two instances. Moreover, it is necessary to omit improbable or less likely states which can

be neglected. It must be noted that integrating all states will generate other new common states with all three parameters of an instance, which must be omitted. Around 1350 primary states are created for the three parameters upon which decisions are made and evaluation is performed. Having the states for three parameters generated, it is necessary to combine them to achieve a four-parameter state and this is repeated until an *N*-parameter state is obtained. A tool is developed to generate and record these different states, to perceive their inter-relationships and to determine common and uncommon states. This tool is capable of perceiving every parameter's instances and can generate the relationship status between instances of two, three or more parameters. Having identified the inter-relationships between parameters' instances in every stage, the tool automatically generates new multi-parameter relationship states and allows users to recognize the likelihood of new relationships.

Figure 5-3 displays the relationship between instances of requirements prioritization parameters in Pattern Release Planning tool.

	DE1	DE2	DE3	DM1	DM2	DM3	DM4	MT1	MT2	MT3	PI1	PI2	PI3	RG1	RG2	RG3	RN1	RN2	RN3	TE1	TE2	TE3	TS1	TS2	TS3
DE1					X		X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
DE2				X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
DE3				X	X	X	X	X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
DM1		X	X					X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
DM2	X	X	X					X	X	X	X	X	X	X	X		X	X		X	X	X	X	X	
DM3		X	X					X	X		X	X	X	X	X	X	X	X	X	X	X	X		X	X
DM4	X	X	X					X	X	X	X	X	X	X	X		X			X	X	X	X	X	
MT1		X	X	X	X	X	X				X	X	X	X	X	X	X	X	X	X	X		X	X	X
MT2	X	X			X	X	X				X	X		X	X	X		X	X	X	X			X	X
MT3	X				X		X				X			X	X	X		X	X	X	X	X		X	X
PI1	X	X	X	X	X	X	X	X	X	X				X	X	X	X	X	X	X	X	X	X	X	X
PI2	X	X	X	X	X	X	X	X	X					X	X	X	X	X	X	X	X	X	X	X	X
PI3	X	X	X	X	X	X	X	X						X	X		X	X		X	X	X	X	X	X
RG1	X	X	X	X	X	X	X	X	X	X	X	X	X				X	X	X	X	X	X	X	X	X
RG2	X	X	X	X	X	X	X	X	X	X	X	X	X				X	X	X	X	X	X	X	X	X
RG3	X	X	X	X		X		X	X	X	X	X					X	X		X	X	X	X	X	X
RN1	X	X	X	X	X	X	X	X			X	X	X	X	X	X				X	X	X	X	X	X
RN2	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X				X	X	X		X	X
RN3	X	X	X	X		X		X	X	X	X	X		X	X					X	X	X			X
TE1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				X	X	X
TE2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				X	X	X
TE3	X	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X	X				X	X	X
TS1	X	X	X	X	X		X	X			X	X	X	X	X	X	X			X	X	X			
TS2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X			
TS3	X	X	X	X		X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			

Figure 5-3: Relations between requirements prioritization parameters instances in tool

An increase in the number of parameters and their instances causes a significant enhancement in the number of states. It is absolutely difficult to examine all these

states, considering the fact that these only belong to requirements prioritization step and other step also have their own states. Although the tool accelerates the operation and state discovery, increase in the number of the states makes it difficult.

5.3.2. Customizing resource estimation

Estimating the required resources plays an important role in software development in terms of time and costs. In order to customize resource estimation, we first characterize the effective parameters in the resource estimation method, and then demonstrate their effects on the method, before finally discussing their relationships.

5.3.2.1. Effective parameters on resource estimation

Resource estimation is one of the most important tasks in all release planning methods. Most of release planning methods identify resource constraints in addition to making necessary estimations of the resources for each requirement or specification, thus resources are estimated for every requirement. As a result, the generated release plan is based on a set of estimated resources for the requirements which are usually less than their constraints. This simple procedure is usually present in most release planning methods. It should be noted that the most important point here is the estimation method used for every definite requirement or capability since it significantly influences the release planning.

Most release planning methods do not focus on the circumstances of resource estimation, because they suppose the users will enter estimation values for every requirement. One of the release planning methods claiming to plan a release or prioritize requirements based on resource estimation is that of Karlsson and Ryan (J. Karlsson & Ryan, 1997). This method first determines the importance of every requirement and estimates its implementation costs. It, then, compares the requirement pairs based on AHP. In reality, no estimation method is used here. Further research done by Karlsson

and Regnell (L. Karlsson & Regnell, 2005) led to the introduction of a costs-based method in which the implementation costs are divided into three classes. Resource estimation is also done through XP and Planning Game method (Beck, 1999). Another research on this topic is done by Kulkarni and Marjaie (Marjaie & Kulkarni, 2010) who studied the parameters hidden in various release planning methods and suggested that the available methods pay little attention to the parameters capable of making judgments between requirements. A small number of studies have also focused on the classification of resource estimation methods, among which are (Galorath & Evans, 2006; Jorgensen & Shepperd, 2007; Khatibi Bardsiri & Norhayati Abang Jawawi, 2011; Pfleeger, Wu, & Lewis, 2005; Suwanjang & Nakornthip, 2012). Galorath and Evans (Galorath & Evans, 2006), for instance, classified different resource estimation methods based on their characteristics.

Classification of resource estimation methods can be used in selecting the right method for specific projects. This can be done by identifying the properties of the methods. Table 5-13 presents the classification of resource estimation methods and their advantages and constraints. As the table shows, every method has some advantages and disadvantages that make it suitable for some projects only. For example, the analogy method can be used only when a similar project has already been accomplished. This can be determined with the "team experience" parameter. Suwanjang and Nakornthip (Suwanjang & Nakornthip, 2012) suggested an analogy framework in which the set of effective parameters on resource estimation resulted from the team and software specifications. Some of these parameters included: team experience in the specific area, team experience in programming and planning, and software specifications such as number of pages, behavior of pages, number of accessed tables, type of software round up and number of inputs and outputs. Although some of these require design

knowledge, the rest (such as number of inputs and outputs) can be precisely determined in the primary steps.

Table 5-13: Classification of different resource estimation methods (Galorath & Evans, 2006)

Method	Objective	Advantage	Constraint
Analogy	Comparing with previous projects	Reality-based estimation	There must be a completely similar project
Expert judgment	Consulting one or more experts	Previous data is partly required; suitable for new or unique projects	Experts may prejudge; knowledge of experts is a matter of question; it may not be consistent
Top-to-down estimation	Hierarchical separation of the system to smaller components in order to estimate software size	Estimates resources based on requirements and common libraries are generated in low levels	Needs valid requirements; difficult architecture follow up; hierarchy type can cause false estimations
Down-to-top estimation	Separate evaluation of every component; estimations are summed to result in main estimation	Precise estimation is possible	It is time-consuming; data detail may not be available; round up costs may be ignored; hierarchy type can cause false estimations
Algorithmic models	Using designing parameters and algorithms	Can be rapidly and simply used; useful in primary steps; objective and replicable	Result in false estimations if not implemented correctly; false estimation of project size may result in false estimation of resources; being optimistic may result in false estimations

In addition to above-mentioned studies, Pfleeger et al. (Pfleeger et al., 2005) examined resource estimation methods and tried to find some specifications for their selection. As Table 5-14 shows, only the algorithmic method can be accomplished without similar information and knowledge of previous projects while other methods require background knowledge. Moreover, when the algorithmic method is selected, it is necessary to have the project characteristics well-identified in order to implement configurations specific to the algorithmic methods.

Table 5-14: Characteristics of resource estimation method

Method	Characteristics
Analogy	• It cannot be used if the project is novel. • It can be used for high level estimations.
Expert judgment	• It is used to complement judgments of other methods. • Previous knowledge and experience is available. • It cannot be used in new projects. • It can be used only in initial steps.
Top-to-down estimation	• It can be used if experts with proper knowledge, experience and data are present. • It is hard to implement in initial project steps.
Down-to-top estimation	• It can be used if experts with proper knowledge, experience and data are present. • Requires a long time period. • It is hard to implement in initial project steps.
Algorithmic models	• It can be used if project's environment and characteristics are well-identified.

Combining the characteristics mentioned in Tables 5-13 and 5-14, they can now be classified into two groups: characteristics related to the project and those related to the developer organization. *Team experience* is one of the parameters related to the developer organization, while *project environment* and *access to data of previous projects* are among parameters concerning the project. These will determine the most appropriate method of resource estimation.

Unlike parameters effective on requirements prioritization which were clearly presented before and can be used for an almost precise selection of priorities, effective parameters on resource estimation are not transparent and precise; hence, the estimation method cannot be exactly determined. This is especially true in determining a certain algorithmic method, since algorithmic models have various sub-methods applicable in different projects.

Effort required to perform every task is the main resource to be estimated. This estimation makes it possible to calculate the time and budget needed for every requirement. However, it is not feasible in certain states. For example, in order to design

Real-time or Embedded systems which require expertise, it is necessary to specify the type of expertise along with the required effort. Therefore, it must be taken into consideration that the estimated effort required may be greater than its exact figure. Moreover, other specifications have to be considered which depend on the input and output details of the resource estimation method. Methods such as analogy and expert judgment are capable of being well-customized but this is not true about algorithmic models. Considering what was mentioned above and using studies by (Galorath & Evans, 2006; Khatibi Bardsiri & Norhayati Abang Jawawi, 2011; Pfleeger et al., 2005; Suwanjang & Nakornthip, 2012), the effective parameters on resource estimation as well as their instances can be presented as in Table 5-15.

Table 5-15: Effective parameters on resource estimation

Parameter	Instance	Description
Development methodology	Waterfall	Requirements are perceived at the beginning of the project.
	Agile	Requirements are perceived and revised at the beginning of every repetition.
	RUP	Requirements are perceived and revised at the beginning of every repetition.
	RAD	Requirements are perceived at the beginning and after producing a sample.
Requirements number	Small	Less than 20 requirements exist.
	Medium	Between 21 and 50 requirements exist.
	Big	More than 51 requirements exist.
Team experience	Experienced	The team has accomplished more than 3 software projects in the field.
	Half-experienced	The team has accomplished less than 3 software projects in the considered field but has accomplished projects in similar fields.
	Inexperienced	The Team has not accomplished any projects in the considered field and lacks experience in related ones.
Input-output amount	Small	Sum of software perceived inputs and outputs is less than 30.
	Medium	Sum of software perceived inputs and outputs is between 30 to 100.

Parameter	Instance	Description
	Big	Sum of software perceived inputs and outputs is more than 100.

5.3.2.2. The effect of parameters on resource estimation method

The effect of every parameter on selecting resource estimation methods can be determined by the parameters and their instances. Table 5-16 shows the effect of every parameter instance independently. As can be observed, some parameter instances can directly (and to a significant extent) determine resource estimation methods but all instances should be considered in relation to the others.

Table 5-16: Effect of parameters and instances on resource estimation method

Parameter	Instance	Description
Development methodology	Waterfall	Steps in this method are well-known and can be used with every resource estimation method. Selecting the precise method depends on other parameters.
	Agile	Steps in this method are well-known and can be used with every resource estimation method. Selecting the precise method depends on other parameters.
	RUP	Steps in this method are well-known and can be used with every resource estimation method. Selecting the precise method depends on other parameters.
	RAD	Steps in this method are not known and can have a wide range. Hence, the algorithmic method is mostly not applicable in this methodology. Analogy and expert judgment are mostly used here and it is less feasible to employ other methods.
Requirements number	Small	Every method can be used for this number of requirements. Decision-making is dependent on other parameters.
	Medium	Every method can be used for this number of requirements. Decision-making is dependent on other parameters.
	Big	Analogy and expert judgment cannot be used because of risk increase. Selecting alternative methods depends on other parameters.
Team experience	Experienced	An experienced team can select every method, and precise selection is dependent on other parameters.
	Half-experienced	A half-experienced team can select every method, and precise selection is dependent on other parameters.
	Inexperienced	Obviously, analogy cannot be used. Employing other methods depends on other project parameters.
Input-output amount	Small	Every method can be used in this case and decision-making depends on other parameters.

Parameter	Instance	Description
	Medium	Every method can be used in this case and decision-making depends on other parameters.
	Big	Every method can be used in this case and decision-making depends on other parameters.

Parameters presented for resource estimation are obtained from limited research on the subject and are obviously not perfect. It is possible, though, to add new parameters to those available and improve the method. Furthermore, as this approach cannot be found in studies conducted so far, the parameters can be considered as the first resource estimation parameters in release planning which are developed according to a pattern-based methodology.

5.3.2.3. Relationship between effective parameters on resource estimation

It is important and necessary to clarify the relationship between parameters and their instances in order to use them in selecting a resource estimation method. Similar to parameters effective on requirements prioritization, two parameters can have direct, reverse or no relationship here. The relationship between every parameter instance and all other parameters is examined independently in order to explain every certain relationship.

Table 5-17 shows the relationship between "development methodology" instances and other effective parameters on resource estimation.

Table 5-17: Relations between "development methodology" and other parameters

Parameter	Instance	Description
Requirements number	Small	All methodologies can be used.
	Medium	Agile, Waterfall and RUP methodologies can be used but RAD cannot be employed because of its implementation nature.
	Big	Waterfall and RUP methodologies can be used but Agile and RAD cannot be employed because of their implementation nature.

Parameter	Instance	Description
Team experience	Experienced	All methodologies can be used.
	Half-experienced	All methodologies can be used.
	Inexperienced	All methodologies can be used.
Input-output amount	Small	All methodologies can be used.
	Medium	All methodologies can be used.
	Big	RAD cannot be used in this case since great number of inputs and outputs requires documentation and structure which RAD is not capable to support.

As observed in the above table, some instances contradict development methodologies instances and are less likely (or unlikely) to occur compared to the two parameters. A set of ordered pair can be defined as follows for those instances of the two parameters which coincide:

$$L_{DMi} = \{x \in DM, y \in \{RN, TE, IO\} | (x, y)\}, \quad for\ all\ \ i \in \{RN, TE, IO\}$$

According to this definition, members of every L set are as follows:

$L_{DMRN} = \{(\mathrm{DM_1}, \mathrm{RN_1}), (\mathrm{DM_1}, \mathrm{RN_2}), (\mathrm{DM_1}, \mathrm{RN_3}), (\mathrm{DM_2}, \mathrm{RN_1}), (\mathrm{DM_2}, \mathrm{RN_2}), (\mathrm{DM_3}, \mathrm{RN_1}), (\mathrm{DM_3}, \mathrm{RN_2}), (\mathrm{DM_3}, \mathrm{RN_3}), (\mathrm{DM_4}, \mathrm{RN_1})\}$

$L_{DMTE} = \{(\mathrm{DM_1}, \mathrm{TE_1}), (\mathrm{DM_1}, \mathrm{TE_2}), (\mathrm{DM_1}, \mathrm{TE_3}), (\mathrm{DM_2}, \mathrm{TE_1}), (\mathrm{DM_2}, \mathrm{TE_2}), (\mathrm{DM_2}, \mathrm{TE_3}), (\mathrm{DM_3}, \mathrm{TE_1}), (\mathrm{DM_3}, \mathrm{TE_2}), (\mathrm{DM_3}, \mathrm{TE_3}), (\mathrm{DM_4}, \mathrm{TE_1}), (\mathrm{DM_4}, \mathrm{TE_2}), (\mathrm{DM_4}, \mathrm{TE_3})\}$

$L_{DMIO} = \{(\mathrm{DM_1}, \mathrm{IO_1}), (\mathrm{DM_1}, \mathrm{IO_2}), (\mathrm{DM_1}, \mathrm{IO_3}), (\mathrm{DM_2}, \mathrm{IO_1}), (\mathrm{DM_2}, \mathrm{IO_2}), (\mathrm{DM_2}, \mathrm{IO_3}), (\mathrm{DM_3}, \mathrm{IO_1}), (\mathrm{DM_3}, \mathrm{IO_2}), (\mathrm{DM_3}, \mathrm{IO_3}), (\mathrm{DM_4}, \mathrm{IO_1}), (\mathrm{DM_4}, \mathrm{IO_2})\}$

The sets include 32 ordered pairs, each of which represents the relationship between two instances of two parameters. Every pair can be considered a combination likely to lead to a resource estimation method. The likelihood becomes absolute when the pair creates a combination of all valid states along with other parameters. Other states absent in this pair, as mentioned earlier, are either invalid or unlikely and can be ignored.

Table 5-18 shows the relationship between "requirements number" instances and other effective parameters on resource estimation.

Table 5-18: Relations between "requirements number" and other parameters

Parameter	Instance	Description
Team experience	Experienced	This team can manage every number of requirements.
	Half-experienced	This team can manage every number of requirements.
	Inexperienced	This team can manage every number of requirements.
Input-output amount	Small	In this case, the number of requirements can be small or medium.
	Medium	In this case, the number of requirements can be small or medium.
	Big	In this case, the number of requirements can be medium or big.

The following ordered pair is defined for two coinciding parameter instances:

$$L_{RNi} = \{x \in RN, y \in \{TE, IO\} | (x, y)\}, \quad for\ all\ \ i \in \{TE, IO\}$$

According to this definition, members of every L set are as follows:

$L_{RNTE} = \{(\mathrm{RN_1}, \mathrm{TE_1}), (\mathrm{RN_1}, \mathrm{TE_2}), (\mathrm{RN_1}, \mathrm{TE_3}), (\mathrm{RN_2}, \mathrm{TE_1}), (\mathrm{RN_2}, \mathrm{TE_2}), (\mathrm{RN_2}, \mathrm{TE_3}), (\mathrm{RN_3}, \mathrm{TE_1}), (\mathrm{RN_3}, \mathrm{TE_2}), (\mathrm{RN_3}, \mathrm{TE_3})\}$

$L_{RNIO} = \{(\mathrm{RN_1}, \mathrm{IO_1}), (\mathrm{RN_1}, \mathrm{IO_2}), (\mathrm{RN_2}, \mathrm{IO_1}), (\mathrm{RN_2}, \mathrm{IO_2}), (\mathrm{RN_2}, \mathrm{IO_3}), (\mathrm{RN_3}, \mathrm{IO_2}), (\mathrm{RN_3}, \mathrm{IO_3})\}$

The sets include 16 ordered pairs, each of which represent the relationship between two instances of two parameters and forms a certain state.

Table 5-19 shows the relationship between "team experience" instances and other effective parameters on resource estimation.

Table 5-19: Relations between "team experience" and other parameters

Parameter	Instance	Description
Input-output amount	Small	A team with any level of experience can manage this number of inputs and outputs.
	Medium	A team with any level of experience can manage this number of inputs and outputs.
	Big	A team with any level of experience can manage this number of inputs and outputs.

The following ordered pair is defined for two coinciding parameter instances:

$$L_{TEi} = \{x \in TE, y \in \{IO\} | (x, y)\}, \quad for\ all\ \ i \in \{IO\}$$

According to this definition, members of every L set are as follows:

$L_{TEIO} = \{(TE_1, IO_1), (TE_1, IO_2), (TE_1, IO_3), (TE_2, IO_1), (TE_2, IO_2), (TE_2, IO_3), (TE_3, IO_1), (TE_3, IO_2), (TE_3, IO_3)\}$

The sets include 9 ordered pairs, each of which represents the relationship between two instances of two parameters and forms a certain state.

Every set of ordered pairs can be considered as a definite state that is combined with other sets of ordered pairs with which it has a common point and forms a set of common ternaries. Every common ternary represents a combination of an instance of three parameters, the accuracy of which must be determined like the relationship between two instances. Moreover, it is necessary to omit improbable or less likely states which can be neglected. Of course, it must be noted that integrating all states will generate new common states with all three parameters of an instance and these must be omitted. Data related to various states between pairs is entered into the software designed for this purpose. Figure 5-4 displays the relationship between resource estimation parameters instances in Pattern Release Planning tool.

	DM1	DM2	DM3	DM4	IO1	IO2	IO3	RN1	RN2	RN3	TE1	TE2	TE3
▸ DM1					X	X	X	X	X	X	X	X	X
DM2					X	X	X	X	X		X	X	X
DM3					X	X	X	X	X	X	X	X	X
DM4					X	X		X			X	X	X
IO1	X	X	X	X				X	X		X	X	X
IO2	X	X	X	X				X	X	X	X	X	X
IO3	X	X	X						X	X	X	X	X
RN1	X	X	X	X	X	X					X	X	X
RN2	X	X	X		X	X	X				X	X	X
RN3	X		X			X	X				X	X	X
TE1	X	X	X	X	X	X	X	X	X	X			
TE2	X	X	X	X	X	X	X	X	X	X			
TE3	X	X	X	X	X	X	X	X	X	X			

Figure 5-4: Relations between resource estimation parameters instances in tool

Similar to requirements prioritization, an increase in the number of parameters and their instances significantly enhances the number of states to be examined.

5.3.3. Customization of pre-release plan

Although the other release planning steps are just as important, the pre-planning step is regarded the main task because it produces the release plan. Customization of this step is more difficult than that of previous ones for three reasons:

- Most of the available release planning methods and algorithms are dependent on the input data of previous steps.
- Some algorithms contain steps which must be performed in a certain order and cannot be separated from previous steps.
- There is a large number of release planning methods with a wide range of diversity (Seyed Danesh, 2011).

In customizing the pre-planning process, it is necessary to select a proper method to determine the release plans of the company or organization. This sounds easy because method selection only affects the first step, and the required parameters have already been identified in previous steps. However, considering the existing obstacles, it is not possible to select the appropriate release planning method in the same manner as in previous steps and a change is needed in how we look at method selection. This means that we need to select a method group based on the planning technique in this step instead of selecting a single method. This was also partly true in resource estimation where it was impossible to determine the precise algorithmic methods based on the parameters, hence, method selection only ended in algorithm selection and other sub-methods could not be chosen. The same can be done in this step through defining various classes of release planning.

According to (Seyed Danesh, 2011) which has tried to review all of the release planning methods, release planning classes can be defined as follows based on the characteristics and capabilities of the planning methods:

- **Class of Ad-hoc methods:** Ad-hoc or plan deficient class is the most basic and simplest class of the release planning methods. No definite method or algorithm is used to plan a release in this class and its previous step (resource estimation) is accomplished through a set of simple parameters. These simple parameters can result in different release plans independently or in a combined manner. Business rules, demand-based customers' needs, and anticipated time and costs are among these parameters. In fact, this class of release planning exerts no systematic method on estimated and prioritized requirements but tries to generate different release plans based on proposed parameters. One of the generated plans will then be used in the next step. Although this method has low level of efficiency and reliability, it is suitable for small projects where there are no constraints (Seyed Danesh, 2011). One of the most important characteristics of this method is that it relies on individuals. It must also be noted that requirements inter-relationships may be neglected here.

- **Class of single-variable methods:** This is the most basic class of systematic methods. Unlike the previous class which was individual-based, this class is implemented in an algorithmic and regular manner on outputs of the previous steps and generates a set of different release planning methods. Game Planning is one of the most important methods of this class in which required inputs are first perceived and then a release plan is generated based on the delivery date or priorities. Although many characteristics are ignored in this method and it lacks efficiency in large projects with inter-related parameters, it is still suitable for small and especially agile projects (Seyed Danesh, 2011). It is noteworthy that

this method is dependent on a certain parameter. Although information on time, costs and other characteristics may be perceived at the beginning and various estimations are made, plans are generated based solely on a single parameter.

- **Class of multi-variable methods:** This class generates different release plans based on several parameters regardless of their inter-relationships. In fact, this method plans a release according to a single parameter at first and then the results are optimized based on other parameters. For example, two parameters are considered in selecting the most valuable requirements with the lowest costs. Optimization-based methods are amongst the members of this class (Seyed Danesh, 2011). Some of these methods use primary techniques for classification or categorization of requirements in order to select suitable requirements in the best way. Ignoring the inter-relationships of release planning parameters is one of the main disadvantages of this class, but it is still more efficient than the previous classes in big projects (Seyed Danesh, 2011). Moreover, it must be noted that similar to previous classes, an increase in the number of requirements and their inter-relationships will decrease the efficiency of the class.

- **Class of intellectual methods:** This class includes a wide range of complicated and complex methods. The most typical characteristics of the class is paying attention to different release planning parameters and considering parameters' inter-relationship and their inter-dependence. Most intellectual methods try to generate a graph of requirements inter-relationships by perceiving primary data and requirement estimations and constraints, and are usually based on optimized searching methods (Seyed Danesh, 2011). Some of the most well-known methods of this class include EVOLVE, Light Weight Re-Planning and methods based on genetic algorithms. Most intellectual methods generate a default set of release plans from which the best one is selected. Many of the problems and

disadvantages of the previous classes do not exist here, but this class is highly complicated and requires more primary data than previous ones. That is why they are not usually efficient in small projects. Furthermore, this class cannot be implemented manually or by using primary tools as it requires proper tools such as Release Planner. Therefore, if an organization is willing to use this class, it should be able to provide the required tools.

5.3.3.1. Effective parameters on pre-release planning

Similar to previous steps, a certain class of release planning methods should be selected in order to act as the best method to generate the organization's pre-release plans. Parameter creation for release planning has been discussed to some extent. One of studies in this area is (Slooten, 2012), which has tried to come up with a mature model for evaluating release planning methods. By definition, the proposed model is a set of activities and tasks to be accomplished in release planning and has not tried to determine parameters for selecting or customizing release planning methods. In addition, (Mohebzada, 2012) aimed to provide a guide in order to optimize release plans generated through Release Planner. The guide makes suggestions for release planning optimization through acquiring characteristics of different projects. Other studies have also been conducted on selecting a certain method for a specific scope or environment such as Web-based Development (J. Li & Ruhe, 2003) or Agile Development (M. Li, Huang, Shu, & Li, 2006) in which method customization or parametric state have been neglected. Although these studies have tried to customize release planning in some ways and optimize release plans, no research has been done on customizing different release planning methods based on characteristics or parameters. Thus, required parameters for customizing this step of release planning are being presented for the first time.

Some characteristics of different classes of release planning methods, described by (Seyed Danesh, 2011), are presented in Table 5-20. As can be observed, more systematic and precise methods require more input data and result in more accurate outputs. Besides, an increase in the number of requirements and stakeholders and an extension of the project size necessitate more accurate methods. Notice that accuracy and precision of the selected release planning method depends on precise requirements prioritization and resource estimation.

Table 5-20: Characteristics of release planning methods (Seyed Danesh, 2011)

Method	Characteristics
Ad-hoc	• Has easy implementation. • Requires short time periods. • Is suitable for small projects. • Is less reliable. • Requirements interdependencies are ignored.
Single-variable	• Has easy implementation. • Requires short time periods. • Is suitable for small projects. • Requirements interdependencies are ignored.
Multi-variable	• Requires precise data for every requirement. • Requires data on various constraints. • Is highly reliable. • Requirements interdependencies are mostly ignored. • Its manual implementation is usually difficult.
Intellectual	• Requires precise data for every requirement. • Requires data on various constraints. • Is highly reliable. • Requirements interdependencies are mostly ignored. • Requires suitable implementation tools.

A set of effective parameters on selecting pre-release planning methods can be defined based on characteristics of the methods. Table 5-21 demonstrates effective parameters on pre-release planning and their instances. Every parameter represents its allowed virtual values.

Table 5-21: Effective parameters on pre-release planning

Parameter	Instance	Description
Market type	Customized	The software is designed and developed for a certain costumer.
	Limited customer	The number of customers is limited and every customer can have different views about each requirement.
	Unlimited customer	The number of customers is unlimited and unlimited number of views are available for each requirement.
Development methodology	Waterfall	Requirements are perceived at the beginning of the project.
	Agile	Requirements are perceived and revised at the beginning of every repetition.
	RUP	Requirements are perceived and revised at the beginning of every repetition.
	RAD	Requirements are perceived at the beginning and after producing a sample.
Project size	Very small	Only 1 or 2 individuals are involved in the project.
	Small	3 to 7 individuals are involved in the project.
	Medium	8 to 15 individuals are involved in the project.
	Big	More than 15 individuals are involved in the project.
Requirements number	Small	Less than 20 requirements exist.
	Medium	21 and 50 requirements exist.
	Big	More than 51 requirements exist.
Number of plan generation parameters	No parameters	No certain parameter is considered for generating release plans.
	Small	Only one certain parameter is considered for generating release plans.
	Medium	2 or 3 certain parameters are considered for generating release plans.
	Big	More than 3 certain parameters are considered for generating release plans.
Team experience	Experienced	The team has implemented more than 3 software projects in the particular field.
	Half-experienced	The team has implemented less than 3 software projects in the particular field but has had projects in similar and relevant fields.
	inexperienced	The team has no experience of project implementation in the particular field or relevant ones.

Similar to parameters of previous steps, those of pre-release planning are not perfect and it is possible to add new ones; this will provide for method expansion in terms of depth (selecting the method besides choosing the method class) and surface (selecting a more precise planning method).

5.3.3.2. Effect of parameters on pre-release planning

According to parameters and instances mentioned in previous step, it is possible to determine their effect on pre-release planning. Table 5-22 shows the effect of every parameter instance independently. It is observed that some instances can directly (and to a considerable extent) determine the pre-release plan but most instances must be considered respecting other parameters in order to characterize the precise method.

Table 5-22: Effect of parameters and instances on pre-release planning method

Parameter	Instance	Description
Market type	Customized	All classes can be used. Method selection depends on other parameters.
	Limited customer	All classes can be used. Method selection depends on other parameters.
	Unlimited customer	Ad-hoc class with its manual nature cannot support an unlimited number of customers. Hence, unsystematic methods cannot be used but a systematic one.
Development methodology	Waterfall	All classes can be used. Method selection depends on other parameters.
	Agile	All classes can be used. Method selection depends on other parameters.
	RUP	All classes can be used. Method selection depends on other parameters.
	RAD	Rapid nature and lack of documentation in RAD make it possible to use simple methods. Intellectual methods cannot be used because they are complicated and require precise and well-documented data.
Project size	Very small	The small project size leads to the tendency to make use of simple and rapid methods. Multi-variable and intellectual classes are not employed since they are complicated and require heavy workloads.
	Small	The small project size leads to the tendency to make use of simple and rapid methods. Simple methods are mostly used because they are economic in terms of human resources. Intellectual class is not employed since it is complicated and requires certain human resources and specific tools.

Parameter	Instance	Description
	Medium	It is necessary to employ structured and systematic methods considering individual inter-relationships and the need to documentation for every step and task. Ad-hoc class cannot be employed since it is not reliable enough.
	Big	Big projects require highly reliable methods. Hence, systematic and reliable methods are mostly used. Ad-hoc and single-variable classes are not applied as they are weak in proving and satisfying the stakeholders' needs.
Requirements number	Small	For small number of requirements, it is not economic (in terms of time and human resources) to use complicated methods. Therefore, Ad-hoc or single-variable classes are mostly employed.
	Medium	For medium number of requirements, it is not possible to use Ad-hoc class because it is not reliable. Intellectual class cannot be used either because it requires time, certain tools and human resources which are not economic with this number of requirements. Hence, single- and multi-variable classes are usually used.
	Big	Large number of requirements makes simple and manual methods unreliable and unusable. Therefore, multi-variable and intellectual classes are often used.
Number of plan generation parameters	No parameters	If there is no certain parameter, only Ad-hoc class is used regardless of instances of other parameters.
	Small	With a small number of parameters, it is possible to perform the planning manually or through systematic methods. Hence, Ad-hoc and single-variable classes are often used, and it is not economic (in terms of time and human resources) to employ complicated methods.
	Medium	At this level, it is not possible to use manual methods because of their unreliability. On the other hand, complicated methods are not economic enough to use. Thus, single- and multi-variable classes are usually used.
	Big	This level requires reliable and documented methods. Therefore, multi-variable and intellectual classes are used.
Team experience	Experienced	All classes can be used by experienced teams. Method selection depends on other parameters.
	Half-experienced	All classes can be used by half-experienced teams. Method selection depends on other parameters.
	Inexperienced	Intellectual and multi-variable classes cannot be used by this team, since they require precise knowledge and data. Therefore, this team can use Ad-hoc and single-variable classes.

If a project does not need pre-release plan parameters, only Ad-hoc class can be used since the project is absolutely small, a small numbers of individuals are involved and all other parameters are in their lowest levels.

Although customization can be performed using requirements prioritization parameters and their instances, the relationship between parameters and instances must be well clarified in order to select the method(s) for every set of parameters and instances.

5.3.3.3. Relationships between effective parameters on pre-release planning

Identifying the relationship between different parameter instances helps in selecting the precise class of pre-release planning method. Similar to previous steps, two certain parameters can have direct, reverse or no relationship. The relationship between every parameter instance and all other parameters is examined independently in order to explain every certain relationship.

Table 5-23shows the relationship between "market type" instances and other effective parameters on pre-release planning.

Table 5-23: Relations between "market type" and other parameters

Parameter	Instance	Description
Development methodology	Waterfall	This method cannot be applied in limited or unlimited customer cases since requirements change in these cases and cannot be perceived at the project beginning. This is possible for customized software, though.
	Agile	This method can be used in all three market types.
	RUP	This is not efficient in unlimited customer cases; these require agility which RUP does not have. This method can be used for customized and limited customer cases, though.
	RAD	This can be used in all three market types.
Project size	Very small	All market types can have very small projects.
	Small	All market types can have small projects.
	Medium	All market types can have medium projects.
	Big	All market types can have big projects.
Requirements number	Small	The number of requirements is 20 in most limited and unlimited customer software. Small number of requirements is only possible in customized software.

Parameter	Instance	Description
	Medium	The requirements number can be medium in all three market types.
	Big	The requirements number can be big in all three market types.
Number of plan generation parameters	No parameters	Only Ad-hoc class can be used in every case.
	Small	All market types can have a small number of plan generation parameters.
	Medium	All market types can have a medium number of plan generation parameters.
	Big	All market types can have a big number of plan generation parameters.
Team experience	Experienced	Experienced teams can enter all market types.
	Half-experienced	Half-experienced teams can enter all market types.
	Inexperienced	This team cannot be used in customized and limited customer software since it is necessary for the team to recognize customers' concerns. However, it can be used in unlimited customer market.

The following ordered pair is defined for two coinciding parameter instances:

$$M_{MTi} = \{x \in MT, y \in \{DM, PS, RN, PP, TE\} | (x, y)\}, \quad for\ all\ \ i \in \{DM, PS, RN, PP, TE\}$$

According to this definition, members of every M set are as follows:

$M_{MTDM} = \{(\mathrm{MT_1}, \mathrm{DM_1}), (\mathrm{MT_1}, \mathrm{DM_2}), (\mathrm{MT_1}, \mathrm{DM_3}), (\mathrm{MT_1}, \mathrm{DM_4}), (\mathrm{MT_2}, \mathrm{DM_2}), (\mathrm{MT_2}, \mathrm{DM_3}), (\mathrm{MT_2}, \mathrm{DM_4}), (\mathrm{MT_3}, \mathrm{DM_2}), (\mathrm{MT_3}, \mathrm{DM_4})\}$

$M_{MTPS} = \{(\mathrm{MT_1}, \mathrm{PS_1}), (\mathrm{MT_1}, \mathrm{PS_2}), (\mathrm{MT_1}, \mathrm{PS_3}), (\mathrm{MT_1}, \mathrm{PS_4}), (\mathrm{MT_2}, \mathrm{PS_1}), (\mathrm{MT_2}, \mathrm{PS_2}), (\mathrm{MT_2}, \mathrm{PS_3}), (\mathrm{MT_2}, \mathrm{PS_4}), (\mathrm{MT_3}, \mathrm{PS_1}), (\mathrm{MT_3}, \mathrm{PS_2}), (\mathrm{MT_3}, \mathrm{PS_3}), (\mathrm{MT_3}, \mathrm{PS_4})\}$

$M_{MTRN} = \{(\mathrm{MT_1}, \mathrm{RN_1}), (\mathrm{MT_1}, \mathrm{RN_2}), (\mathrm{MT_1}, \mathrm{RN_3}), (\mathrm{MT_2}, \mathrm{RN_2}), (\mathrm{MT_2}, \mathrm{RN_3}), (\mathrm{MT_3}, \mathrm{RN_2}), (\mathrm{MT_3}, \mathrm{RN_3})\}$

$M_{MTPP} = \{(\mathrm{MT_1}, \mathrm{PP_2}), (\mathrm{MT_1}, \mathrm{PP_3}), (\mathrm{MT_1}, \mathrm{PP_4}), (\mathrm{MT_2}, \mathrm{PP_2}), (\mathrm{MT_2}, \mathrm{PP_3}), (\mathrm{MT_2}, \mathrm{PP_4}), (\mathrm{MT_3}, \mathrm{PP_2}), (\mathrm{MT_3}, \mathrm{PP_3}), (\mathrm{MT_3}, \mathrm{PP_4})\}$

$M_{MTTE} = \{(\mathrm{MT_1}, \mathrm{TE_1}), (\mathrm{MT_1}, \mathrm{TE_2}), (\mathrm{MT_2}, \mathrm{TE_1}), (\mathrm{MT_2}, \mathrm{TE_2}), (\mathrm{MT_3}, \mathrm{TE_1}), (\mathrm{MT_3}, \mathrm{TE_2}), (\mathrm{MT_3}, \mathrm{TE_3})\}$

The sets include 44 ordered pairs, each of which represents the relationship between two instances of two parameters. Every pair can be considered as a combination likely to lead to a method. The likelihood becomes absolute when the pair creates a combination

of all valid states along with other parameters. Other states absent in this pair, as mentioned earlier, are either invalid or unlikely so they can be ignored.

Table 5-24 shows the relationship between "development methodology" instances and other effective parameters on pre-release planning.

Table 5-24: Relations between "development methodology" and other parameters

Parameter	Instance	Description
Project size	Very small	Agile and RAD methods can be used in very small projects due to their simple and rapid nature.
	Small	All methodologies can be used.
	Medium	Considering the required documentation, RUP, Agile and Waterfall methods are used.
	Big	Considering the required documentation and communication among teams, it is not possible to use Agile and RAD methods. Hence, RUP and Waterfall methods are employed.
Requirements number	Small	All methodologies can be used.
	Medium	Agile, Waterfall and RUP methods can be used, but RAD cannot support medium number of requirements (because of its implementation nature).
	Big	Waterfall and RUP methods can be used but RAD and Agile cannot support large number of requirements (due to their implementation nature).
Number of plan generation parameters	No parameters	Only Ad-hoc class is used in such cases.
	Small	All methodologies can be used.
	Medium	All methodologies can be used.
	Big	Methods which require documentation (such as RAD) cannot be used. Hence, RUP, Agile and Waterfall methods are employed.
Team experience	Experienced	All methodologies can be used.
	Half-experienced	All methodologies can be used.
	Inexperienced	All methodologies can be used.

The following ordered pair is defined for two coinciding parameter instances:

$$M_{DMi} = \{x \in DM, y \in \{PS, RN, PP, TE\} | (x, y)\}, \qquad for\ all\ \ i \in \{PS, RN, PP, TE\}$$

According to this definition, members of every M set are as follows:

$M_{DMPS} = \{(DM_1, PS_2), (DM_1, PS_3), (DM_1, PS_4), (DM_2, PS_1), (DM_2, PS_2), (DM_2, PS_3), (DM_3, PS_2), (DM_3, PS_3), (DM_3, PS_4), (DM_4, PS_1), (DM_4, PS_2)\}$

$M_{DMRN} = \{(DM_1, RN_1), (DM_1, RN_2), (DM_1, RN_3), (DM_2, RN_1), (DM_2, RN_2), (DM_3, RN_1), (DM_3, RN_2), (DM_3, RN_3), (DM_4, RN_1)\}$

$M_{DMPP} = \{(DM_1, PP_2), (DM_1, PP_3), (DM_1, PP_4), (DM_2, PP_2), (DM_2, PP_3), (DM_2, PP_4), (DM_3, PP_2), (DM_3, PP_3), (DM_3, PP_4), (DM_4, PP_2), (DM_4, PP_3)\}$

$M_{DMTE} = \{(DM_1, TE_1), (DM_1, TE_2), (DM_1, TE_3), (DM_2, TE_1), (DM_2, TE_2), (DM_2, TE_3), (DM_3, TE_1), (DM_3, TE_2), (DM_3, TE_3), (DM_4, TE_1), (DM_4, TE_2), (DM_4, TE_3)\}$

The sets include 43 ordered pairs, each of which represents the relationship between two instances of two parameters and forms a state.

Table 5-25 shows the relationship between "project size" instances and other effective parameters on pre-release planning.

Table 5-25: Relations between "project size" and other parameters

Parameter	Instance	Description
Requirements number	Small	Most very small and small projects have a small number of requirements.
	Medium	Most small and medium-sized projects have a medium number of requirements.
	Big	Most medium and big projects have a big number of requirements.
Number of plan generation parameters	No parameters	Only Ad-hoc class is used in such cases.
	Small	Whatever a team's size is, it can manage a small number of plan parameters.
	Medium	Small, medium and big teams can manage a medium number of plan parameters.
	Big	Medium and big teams can manage large number of plan parameters.
Team experience	Experienced	Whatever the project size is, it can be accomplished by an experienced team.
	Half-experienced	Whatever the project size is, it can be accomplished by a half-experienced team.

Parameter	Instance	Description
	Inexperienced	Inexperienced teams can't accomplish big projects.

The following ordered pair is defined for two coinciding parameter instances:

$$M_{PSi} = \{x \in PS, y \in \{RN, PP, TE\} | (x, y)\} \ , \qquad for\ all\ \ i \in \{RN, PP, TE\}$$

According to this definition, members of every *M* set are as follows:

$M_{PSRN} = \{(PS_1, RN_1), (PS_2, RN_1), (PS_2, RN_2), (PS_3, RN_2), (PS_3, RN_3), (PS_4, RN_3)\}$
$M_{PSPP} = \{(PS_1, PP_2), (PS_2, PP_2), (PS_2, PP_3), (PS_3, PP_2), (PS_3, PP_3), (PS_3, PP_4), (PS_4, PP_2), (PS_4, PP_3), (PS_4, PP_4)\}$
$M_{PSTE} = \{(PS_1, TE_1), (PS_1, TE_2), (PS_1, TE_3), (PS_2, TE_1), (PS_2, TE_2), (PS_2, TE_3), (PS_3, TE_1), (PS_3, TE_2), (PS_3, TE_3), (PS_4, TE_1), (PS_4, TE_2)\}$

The sets include 26 ordered pairs, each of which represents the relationship between two instances of two parameters and forms a state.

Table 5-26 shows the relationship between "requirements number" instances and other effective parameters on pre-release planning.

Table 5-26: Relations between "requirements number" and other parameters

Parameter	Instance	Description
Number of plan generation parameters	No parameters	Only Ad-hoc class is used in such cases.
	Small	An increase in requirements' number demands a corresponding increase in the number of plan parameters to select better ones. Hence, if a small number of plan parameters are present, then the requirements' number is small or medium.
	Medium	Unlimited number of requirements can be applied with medium number of parameters.
	Big	It is not economic to use large number of parameters for a small number of requirements. Hence, requirements' number must be medium or large.
Team experience	Experienced	Experienced teams can manage unlimited number of requirements.
	Half-experienced	Half-experienced teams can manage unlimited number of requirements.
	Inexperienced	Inexperienced teams can manage unlimited number of requirements.

The following ordered pair is defined for two coinciding parameter instances:

$$M_{RNi} = \{x \in RN, y \in \{PP, TE\} | (x, y)\}, \qquad for\ all\ \ i \in \{PP, TE\}$$

According to this definition, members of every *M* set are as follows:

$M_{RNPP} = \{(\mathrm{RN_1}, \mathrm{PP_2}), (\mathrm{RN_1}, \mathrm{PP_3}), (\mathrm{RN_2}, \mathrm{PP_2}), (\mathrm{RN_2}, \mathrm{PP_3}), (\mathrm{RN_2}, \mathrm{PP_4}), (\mathrm{RN_3}, \mathrm{PP_3}), (\mathrm{RN_3}, \mathrm{PP_4})\}$

$M_{RNTE} = \{(\mathrm{RN_1}, \mathrm{TE_1}), (\mathrm{RN_1}, \mathrm{TE_2}), (\mathrm{RN_1}, \mathrm{TE_3}), (\mathrm{RN_2}, \mathrm{TE_1}), (\mathrm{RN_2}, \mathrm{TE_2}), (\mathrm{RN_2}, \mathrm{TE_3}), (\mathrm{RN_3}, \mathrm{TE_1}), (\mathrm{RN_3}, \mathrm{TE_2}), (\mathrm{RN_3}, \mathrm{TE_3})\}$

The sets include 16 ordered pairs, each of which represents the relationship between two instances of two parameters and forms a state.

Table 5-27 shows the relationship between "number of plan generation parameters" instances and other effective parameters on pre-release planning.

Table 5-27: Relations between "number of plan generation parameters" and other parameters

Parameter	Instance	Description
Team experience	Experienced	Experienced teams can manage unlimited number of plan generation parameters.
	Half-experienced	Half-experienced teams can manage unlimited number of plan generation parameters.
	Inexperienced	An increase in the number of plan parameters requires recruiting experienced or half-experienced individuals. Hence, an inexperienced team can only manage a small or medium number of plan parameters.

The following ordered pair is defined for two coinciding parameter instances:

$$M_{PPi} = \{x \in PP, y \in \{TE\} | (x, y)\}, \qquad for\ all\ \ i \in \{TE\}$$

According to this definition, members of every *M* set are as follows:

$M_{PPTE} = \{(\mathrm{PP_2}, \mathrm{TE_1}), (\mathrm{PP_2}, \mathrm{TE_2}), (\mathrm{PP_2}, \mathrm{TE_3}), (\mathrm{PP_3}, \mathrm{TE_1}), (\mathrm{PP_3}, \mathrm{TE_2}), (\mathrm{PP_3}, \mathrm{TE_3}), (\mathrm{PP_4}, \mathrm{TE_1}), (\mathrm{PP_4}, \mathrm{TE_2})\}$

The sets include 8 ordered pairs, each of which represents the relationship between two instances of two parameters and forms a state.

Similar to the previous steps, every set of ordered pairs can be considered a certain state which is combined with the other sets with which it has a common point and forms a set of common ternaries. This combination continues until there are as many states as parameters in pre-release planning step. Data related to various states between pairs is entered into the software designed for this purpose. Figure 5-5 displays the relationship between pre-release planning parameter instances in Pattern Release Planning tool.

	DM1	DM2	DM3	DM4	MT1	MT2	MT3	PP1	PP2	PP3	PP4	PS1	PS2	PS3	PS4	RN1	RN2	RN3	TE1	TE2	TE3
DM1					X				X	X	X		X	X	X	X	X	X	X	X	X
DM2					X	X	X		X	X	X	X	X	X		X	X		X	X	X
DM3					X	X			X	X	X		X	X	X	X	X	X	X	X	X
DM4					X	X	X		X	X		X	X			X			X	X	X
MT1	X	X	X	X					X	X	X	X	X	X	X	X	X	X	X	X	
MT2		X	X	X					X	X	X	X	X	X	X		X	X	X	X	
MT3		X		X					X	X	X	X	X	X	X		X	X	X	X	X
PP1																					
PP2	X	X	X	X	X	X	X					X	X	X	X	X	X		X	X	X
PP3	X	X	X	X	X	X	X						X	X	X	X	X	X	X	X	X
PP4	X	X	X		X	X	X							X	X		X	X	X	X	
PS1		X		X	X	X	X		X							X			X	X	X
PS2	X	X	X	X	X	X	X		X	X						X	X		X	X	X
PS3	X	X	X		X	X	X		X	X	X						X	X	X	X	X
PS4	X		X		X	X	X		X	X	X							X	X	X	
RN1	X	X	X	X	X				X	X		X	X						X	X	X
RN2	X	X	X		X	X	X		X	X	X		X	X					X	X	X
RN3	X		X		X	X	X			X	X			X	X				X	X	X
TE1	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X			
TE2	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X			
TE3	X	X	X	X			X		X	X		X	X	X		X	X	X			

Figure 5-5: Relations between pre-release planning parameters instances in tool

5.3.3.4. Customization of trade-off analysis and selecting the final release plan

Unlike the previous steps which could be customized through parameters and their instances, this step involves management whereby only a project manager or authorized individuals can select the primary release plan. In order to select a release plan and present it to the development team, the primary release plans generated at this stage are compared to one another but from the management point of view. Usually, requirements and characteristics absent in the selected plan are decided in the next step. This should be done before the implementation of the final release plan.

5.4. Pattern-based release planning methodology

Research on release planning has shown that similar to most engineering tasks, this activity is highly influenced by past experiences. In the general release planning process, past experiences are incorporated in the form of effective parameters on each step and the influence of every parameter on the method or algorithm used in the step independently. Besides, PRP Software is designed and run to enter and record all states and to trace the inter-relationship of parameters and instances. The software provides the release planner with the possibility to add new parameters and instances and allows a simple relationship between them.

Every step of the general release planning process model contains a set of inputs, outputs and parameters, and a suitable algorithm or method is selected using these parameters. In addition, selecting a proper algorithm for every step determines the precise inputs to be used and outputs to be generated. Although most algorithms or methods have some common inputs in every step, determining precise inputs is highly dependent on the selected method. Figure 5-6 illustrates the relationship between parameters, instances and inputs of the selected method in every step.

In its simplest form, a suitable method to perform every certain step can be achieved by developing a mapping table between different parameter instances and the related methods and algorithms. This table examines all the different parameter instances that are extracted from the project specification and relates them to one or more suitable algorithm or method in each step. Using the table, the software team can specify the parameter instances in order to identify a suitable method or algorithm to perform in the specific step of the process model based on past experiences. As observed in customization of every step of release planning process model, such a mapping requires determining the effect of parameters and instances on each other and needs making use of past experiences in order to select the best method for every stage. One of the tasks

performed using PRP software is to generate a mapping of all available states. As Figure 5-6 displays, steps can have common parameter instances which affect the whole release planning procedure.

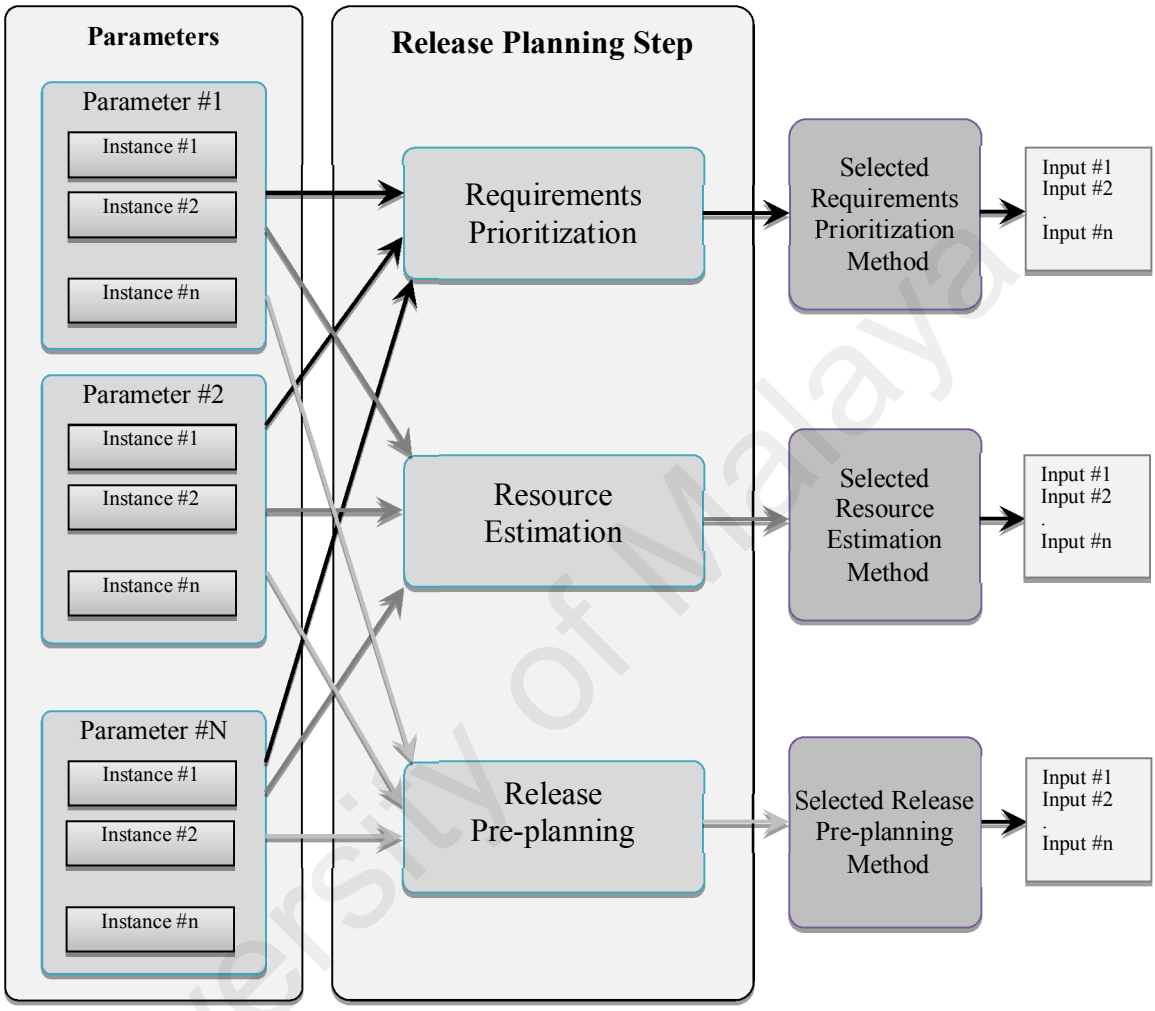


Figure 5-6: Relationships between release planning process model and the method

One of the most important advantages of this mapping is the quick and simple customization of the release planning process. Moreover, the planner is benefited from past experiences used in the mapping without previous knowledge. Past experiences determine the relationship between different parameters in the customization which result in the mapping table. Therefore, one must map all parameter instances to others and specify the related method or algorithm to use in each step. Every set of parameter instances can be considered as one state and a method or algorithm is assigned to each state.

This mapping table may lead to generation of numerous states by adding a certain parameter or instance. Furthermore, many of the generated states may be practically impossible; hence, these states should be identified and excluded by the planner. Therefore, the customization process has to be altered or optimized so that the planner can achieve the most suitable methods in the shortest time possible.

The concept of "pattern", which is a well-known notion in software development, is used in release planning to optimize planning customization and remove the main weakness of this method. In fact, not all states are generated for parameter instances in release planning but only successful experiences in using a certain method are documented based on the parameter instances. This eliminates the necessity to examine all of the states whereby the planner can now directly enter his (her) own experiences. Using past experiences in software development is tightly tied with the concept of "pattern". There are different patterns in software development such as design patterns and architectural patterns where the main idea is to make rapid use of past experiences in order to accomplish software development. Figure 5-7 shows how to use the pattern in release planning.

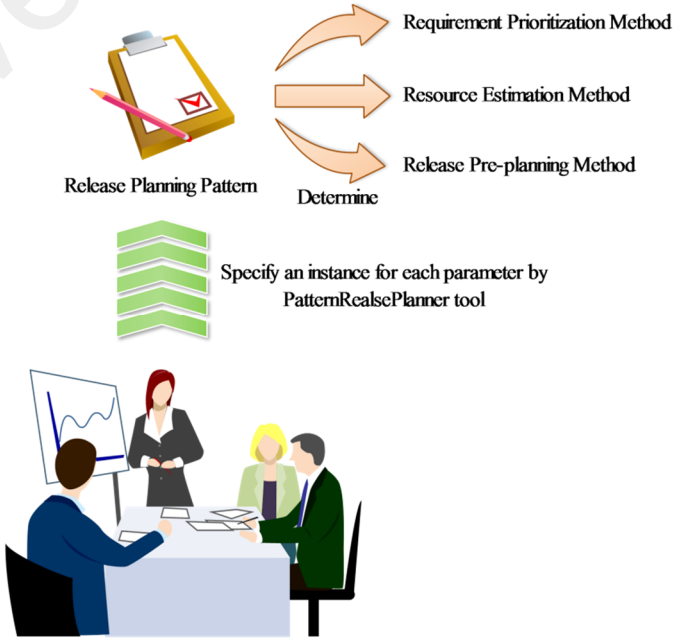


Figure 5-7: Using pattern in release planning

5.4.1. Definition of release planning patterns

Similar to other software developing patterns, release planning pattern should be well-defined to be used in release planning. Patterns employed in release planning are divided into two groups based on their effect: patterns effective on all release planning steps and those effective on a certain step. The former group is "release planning patterns" and the latter is "release planning steps' patterns". Moreover, as mentioned in release planning customization, some parameters are common among multiple steps and some others only influence a certain one. Putting all parameters together makes it possible to obtain a set of effective parameters on all release planning steps which are called "effective parameters on release planning".

According to what mentioned above, best experience in release planning is achieved when the release planning pattern:

- Is obtained by valuing every effective parameter on release planning,
- Contains value of one or more parameter,
- Determines running mode of every step in the process model, and
- Is determined by instances of effective parameters on release planning.

By definition, a release planning pattern can result from the influence of one or more successfully implemented effective parameter which has been used in similar projects with similar parameter instances for several times. Moreover, best experience in release planning is achieved when the release planning pattern:

- Is obtained by valuing every parameter effective on the same step of release planning process model,
- Contains value of one or more effective parameters on the same step of release planning process model,
- Determines running mode of the same step of release planning process model,

- Is determined by instances of effective parameters on the same step of release planning process model.

Using this definition, the pattern of release planning steps can be divided into three areas: requirement prioritization, resource estimation and pre-release planning, in each of which patterns specific to that certain step are identified and described. The patterns are developed using experiences in resource estimation, requirement prioritization and release planning tasks.

5.4.2. The structure of release planning patterns

Release planning patterns are proposed in order to improve planning quality by making use of successful experiences in this field. Regarding the need to employ a well-defined structure for such patterns and to provide a framework to describe them, a set of characteristics is proposed to show features of software development in the structure of release planning patterns. The framework contains specifications required for distinguishing release planning patterns. It includes:

- **Pattern name**

This is the selected name for release planning pattern. Every pattern has a main name distinguishing it from other patterns.

- **Side name**

Every pattern may have one or more side names besides its main name. The side name may be used for the application of pattern in certain fields.

- **Pattern problem**

It describes specifications of a problem or complexity in which the pattern can be applied. Pattern objective is usually presented in the pattern problem in an explicit

manner. The objective specifies the pattern application. The problem is different in every certain step.

- **Context**

This determines a certain filed in which the pattern is employed. It can be release planning or a step of release planning process model.

- **Constraints**

It describes different parameters and instances or various steps of release planning. One or more parameters are considered in this pattern and an instance is mentioned for each one. Constraints are also presented by parameters. Pattern problem is solved respecting these parameter instances.

- **Solution**

This defines the selected method to solve the problem. The method is proposed considering the problem (pattern objective), parameters (constraints) and all instances.

- **Resulting context**

It describes possible states and situations after pattern implementation, including: (good or bad) results of running the pattern, and other problems and patterns likely to occur in the new context.

- **Rationale**

It describes reasons for selecting the proposed problem solution. In fact, this part explains how a release planning pattern really works, why it works and why it is good. The rationale must specify how every constraint's parameter instance is met by the proposed solution.

- **Known use**

Previous and known uses of the pattern are presented in this section.

- **Related sub-patterns**

A release planning pattern can include patterns of every stage. Therefore, this characteristic presents the names of patterns related to every certain stage.

The structure can be used for every pattern of various steps of release planning.

5.4.3. Development method of release planning patterns

Development of various release planning patterns is one of the most important factors in this area. Similar to software development patterns obtained from previous experiences, the most basic method to develop a release planning pattern is to make use of best experiences in release planning area. Since this is being proposed for the first time, no documentation is available on release planning experiences especially on release planning parameters.

In the first methods, as observed in customization of steps of release planning process model, only some limited investigations have been done to categorize effective parameters on requirements prioritization and other steps lack such explorations. Hence, in the development of requirements prioritization patterns, it has been tried to make best use of these investigations.

In the second method, different parameters and states they generate are implemented and experienced in several projects and in different software development companies. Thus, case studies are selected in a way they cover different states and instances of parameters appropriately. Patterns developed in this way are generated based on executive realities and as an example of method implementation. Moreover, it is possible to develop more patterns.

5.4.4. Algorithms of using release planning patterns

Making use of release planning patterns in every step of the release planning process requires information on the needed instances and then identifying and employing the

best suited pattern for that instance. The process for the requirement prioritization, resource estimation and release pre-planning steps is explained below. Algorithm 1 shows how the best suited pattern for the requirement prioritization step is chosen where the characteristics of parameters, instances and the list of the requirements prioritization patterns are perceived as input and the proper patterns are printed as output.

Algorithm 1: Finding best suited pattern for 'Requirement Prioritization'

Input: A set *P*of parameter= {*MT*, *DM*, *TS*, *RN*, *RG*, *PI*, *TE*,DE}
A set *I* of instance of each parameter
List *L* of requirement prioritization patterns

Output: Appropriate pattern for requirements prioritization

```
Begin
  Z =0
  For each parameters in P
    Begin
      /* Get and assign priority value to each parameter */
      /* Q[i] =0 means that the parameter should not incorporate in pattern selection
      Q[i] = Priority of i^th parameter in P
      If Q[i] > 0 thenZ= Z+1
    End
  Sort P based on Q
  K =0
  For each parameters in P from top priority where Q[i] >0
    For each pattern in L
      Begin
        /* Select instance of parameter P[i] */
        If instance I[i] exist in L[j]then
          Begin
            /*Add L[j] to Select Pattern List */
            K = K +1
            SPL [K] = L[j]
            If L[j] did not added beforethen
              SPLCounter [K] =1
            Else
              SPLCounter [K] =SPLCounter [K] + 1
            End
          End
      End
  For all patterns in SPL
    If SPLCounter = Zthen
```

Print SPL [k] as selected pattern

End

The algorithm first receives a priority for every input parameter. The priority is used to sort the parameters and exclude those which do not affect the pattern selection. In other words, the algorithm may ignore parameters with the value of zero. Having the parameters prioritization accomplished, a list of possible patterns is explored based on the nonzero parameters. From the patterns list, those with parameter instances are added to the list of selected patterns and the pattern counter is also added. The counter identifies the patterns that include all possible instances. When the search is finished, patterns with the same frequency as nonzero parameters are printed as the selected pattern.

The algorithms for selecting resource estimation and pre-release planning patterns are similar to that of requirement prioritization but they differ in the list of parameters, instances and selected patterns. While identifying the best suited pattern for each step, it is possible to deactivate every definite parameter by assigning it with a 0 value. This is important as some parameters can be ignored in release planning for different projects or the search may be performed based on a specific priority. This means that the patterns that are sought are for more important parameters rather than a predetermined range.

Release planning pattern is a defined combination of patterns for different steps of release planning. Similar to the pattern of every definite step which cannot be obtained by combining all arbitrary parameter instances, release planning pattern also cannot be obtained by generating every combination of patterns for different steps. In other words, combining different patterns to generate various states is ignored in identifying the planning pattern but all parameters and their instances are considered to determine the

best experiences for every step so that an integrated combination of methods can be generated in all steps of the general release planning process.

Finding best suited patterns for release planning requires a combination of patterns concerning different steps of release planning. Algorithm 2 is presented for this purpose. This algorithm receives all parameters of release planning and those of every stage, their instances, as well as the list of every stage's patterns and release planning patterns as inputs and prints release planning patterns fitting the parameter instances as output.

Similar to Algorithm 1, the following one receives a priority for every input parameter to exclude those ineffective on determining the pattern. Then, parameter sets of every stage (*TR, TE and TP*) are sorted based on the priority. Afterwards, fitting input parameter patterns are found for each of the three steps.

Algorithm 2: Finding best suited pattern for 'Release Planning'

Input: A set *P* of requirements prioritization parameter, *P*= {*MT*, *DM*, *TS*, *RN*, *RG*, *PI*, *TE*, *DE*, *IO*, *PS*, *PP*}
A set *TR* of requirements prioritization parameter, *TR*= {*MT*, *DM*, *TS*, *RN*, *RG*, *PI*, *TE*, *DE*}
A set *TE* of resource estimation parameter, *TE*= {*DM*, *RN*, *TE*, *IO*}
A set *TP*of pre-release pattern parameter, *TP*= {*MT*, *DM*, *PS*, *RN*, *PP*, *TE*}
A set *I* of instance of each parameter
List *LR, LE, LP, and L* are requirements prioritization, resource estimation, pre-release, and release planning patterns sequentially
Output: Appropriate pattern for release planning
Begin
Z =0
For *each parameters* **in** *P*
Begin
/ Get and assign priority value to each parameter */*
/ Q[i] =0 means that the parameter should not incorporate in pattern selection*
Q[i] = Priority of i^{th} parameter in P
End
Sort TR, TE, and TP based on Q

```
K =0
Z [1] = 0
While parameters in TRwhere Q[i] >0and from top priority
    Begin
        Z [1] = Z [1] + 1
        For eachpattern in LR
        Begin
           /* Select instance of parameter TR[i] */
           If instance I[i] exist in LR[j]then
              Begin
                  /*Add LR[j] to Select Pattern List */
                  K = K +1
                  SPLR [K] = LR [j]
                  If LR[j] did not added before then
                     SPLCounterR [K] =1
                  Else
                     SPLCounterR [K] =SPLCounterR [K] + 1
                  End
              End
        End
    End
K =0
Z [2] = 0
While parameters in TEwhere Q[i] >0and from top priority
    Begin
           Z [2] = Z [2] + 1
           For eachpattern in LE
           Begin
            /* Select instance of parameter TE[i] */
            If instance I[i] exist in LE [j]then
               Begin
                  /*Add LE [j] to Select Pattern List */
                  K = K +1
                  SPLE [K] = LE [j]
                  If LE [j] did not added beforethen
                     SPLCounterE [K] =1
                  Else
                     SPLCounterE [K] =SPLCounterE [K] + 1
                  End
               End
           End
    End
K =0
Z [3] = 0
```

```
While parameters in TPwhere Q[i] >0and from top priority
    Begin
            Z [3] = Z [3] + 1
            For eachpattern in LP
            Begin
              /* Select instance of parameter TP[i] */
              If instance I[i] exist in LP [j]then
                 Begin
                   /*Add LP [j] to Select Pattern List */
                   K = K +1
                   SPLP [K] = LP [j]
                   If LP [j] did not added beforethen
                        SPLCounterP [K] =1
                   Else
                        SPLCounterP [K] =SPLCounterP [K] + 1
                   End
                End
            End
    End
For all patterns in L
    PreviousStagePatternSelected = false
     If L[i][1] is in SPLR then
       Begin
            Index= Index ofL[i][1] in SPLR
     IfSPLCounterR [Index] = Z [1] Then
            PreviousStagePatternSelected = true
        End
     IfPreviousStagePatternSelectedand L[i][2] is in SPLE then
       Begin
            Index= Index ofL[i][2] in SPLE
     IfSPLCounterE [Index] = Z [2] Then
            PreviousStagePatternSelected = true
            Else
            PreviousStagePatternSelected = false
        End
     IfPreviousStagePatternSelectedand L[i][3] is in SPLP then
       Begin
            Index= Index ofL[i][3] in SPLP
     IfSPLCounterP [Index] = Z [3] Then
            PreviousStagePatternSelected = true
            Else
            PreviousStagePatternSelected = false
        End
     IfPreviousStagePatternSelectedthen
```

Print L[i] as selected release planning pattern

End

In this algorithm, the Do-Loop "*while*" runs for every parameter available in the considered stage (with priority value of greater than 0) in order to find related patterns to specified parameter instances. It is necessary to involve parameters with priority values greater than a certain range and 0 is replaced by the value in consideration. While the loop is running parameters, *Z* counts the frequency of running. In contrast to Algorithm 1, since the number of variables greater than the priority in each stage is not specified, the counting must be performed to select those patterns supporting such parameters. For every certain stage, patterns suiting the entered instances are determined as input and are recorded in the list of selected patterns of that stage. Moreover, the counter of pattern application frequency increases for every selected pattern.

With all the three steps accomplished, the final "*for*" loop runs to identify release planning patters. To meet this goal, it is determined for every release planning pattern whether or not a sub-pattern exists in the list of proposed patterns. If the proposed sub-pattern is present in the list, its selection frequency is examined to ensure that it supports all prior parameters. If the answer is positive, the value for the variable "*Previous Stage Pattern Selected*" changes to "*True*" to show that sub-patterns related to the pattern in consideration are present on the list of selected patterns for every stage. The examination is performed for all three steps. At the end of the three examinations if the variable "*Previous Stage Pattern Selected*" has the value "*True*", the pattern is among selected ones for release planning and can be printed.

Algorithm 2 relies on finding and showing a pattern best suiting parameters and instances from available patterns. The algorithm is implemented in PRP Software and enables finding patterns best suiting the characteristics determined by project manager

or release planner. In addition, it is possible in this software to use filters to find patterns fitting specific states. It also allows for recording experienced patterns to determine parameters and their instances. The software can recognize frequent or repetitive patterns and warn if necessary.

5.5. Release planning patterns

Considering the presented structure for various release planning patterns, a set of patterns developed by described methods are presented separately for every stage of release planning.

5.5.1. Requirements prioritization patterns

Requirements prioritization is the first stage of release planning process model. It receives inputs from stakeholders or the team and gives a list of prioritized requirements as the output. Some requirements prioritization patterns are explained below.

5.5.1.1. Pattern of requirements prioritization for large projects

- **Pattern name**

  Requirements prioritization pattern for large projects

- **Side name**

  Pattern of requirements prioritization using AHP method

- **Problem**

  Requirements prioritization for release planning in large projects in which time and costs are of great importance for stakeholders needs a method which can, on one hand, involve all stakeholders in requirements selection and, on the other hand, be a systematic and regulated one. Furthermore, proper tools must be introduced to enable developers to manage large number of requirements.

- **Context**

  In requirement prioritization of release planning, it is always necessary to prioritize and sort primary requirements in order to facilitate selection of most prior requirements for implementation (and sending them to the next stage).

- **Constraints**
  - The market must be of limited- or unlimited-customer type.
  - Requirements must be in "Fine" level.
  - Number of inputs must be more than three.
  - Number of requirements must be large.
  - The team must be medium or large-sized.
  - RUP method must be used.
  - The team must be half-experienced or experienced.
  - Development environment must be Client-Server or Web-based.

- **Solution**

  Using AHP method for requirements prioritization

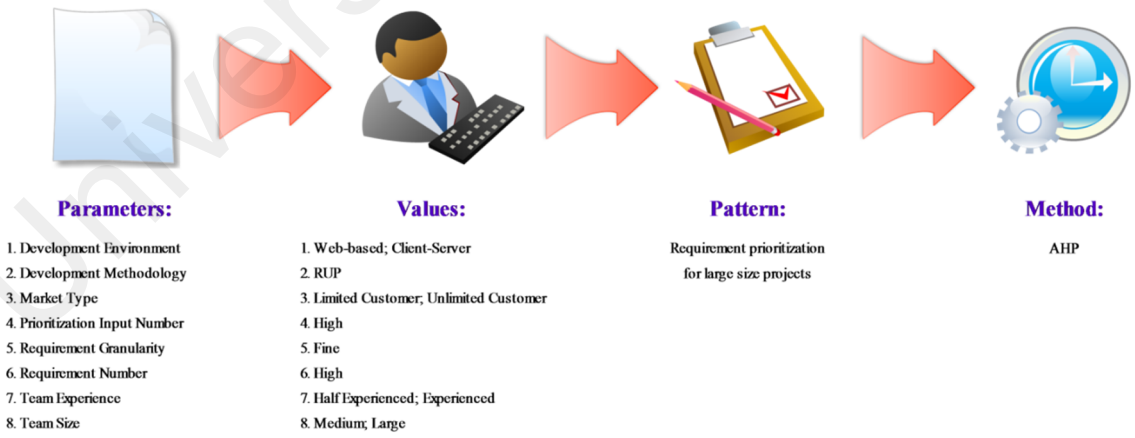


Figure 5-8: Requirements prioritization pattern for large projects

- **Resulting context**

In this method, all requirements enter the tool as inputs and stakeholders' vote is received for every requirement. By implementing AHP method, the tool provides priorities of various requirements.

- **Rationale**

  By receiving different inputs for every requirement, the AHP method regulates prioritization process by paired comparison analysis. The paired comparison and AHP decision-making rationale can overcome medium or big size of the team and heavy weight of RUP methodology. Besides, the method can be used with "fine" requirements level although the number of comparisons increases (a case in which improved AHP can be used). Moreover, the method possesses proper tools for requirements prioritization which can be used by the team.

- **Known use**

  Most big software developing companies dealing with complicated requirements employ AHP-supporting tools. For instance, "Rational Focal Point" (by IBM) is one of the most important tools which employs AHP and is used for requirements prioritization.

- **Related sub-pattern**

  Does not exist.

5.5.1.2. Pattern of requirement prioritization with medium level of requirements

- **Pattern name**

  The pattern of requirement prioritization with medium level of requirements

- **Side name**

  Pattern of requirements prioritization by Ranking method

- **Problem**

  In software developing projects with coarse or medium level of requirements (which require being broken down to fine levels) requirements must be prioritized in a way that the time spent by the team is reduced. The method must be a systematic one with discussable results. Moreover, it must be capable of receiving stakeholders' votes and engage them in prioritization.

- **Context**

  In requirement prioritization of release planning, it is always necessary to prioritize and sort primary requirements in order to facilitate selection of most prior requirements for implementation (and sending them to the next stage).

- **Constraints**

  - For all market types.
  - Requirements level must be coarse or medium.
  - Number of requirements prioritization (classification) must be less than three.
  - Number of requirements must be small or medium.
  - RUP or Waterfall methodologies must be used.
  - The team must be half-experienced or experienced.
  - For every development environment.

- **Solution**

  Using Ranking method for requirements prioritization

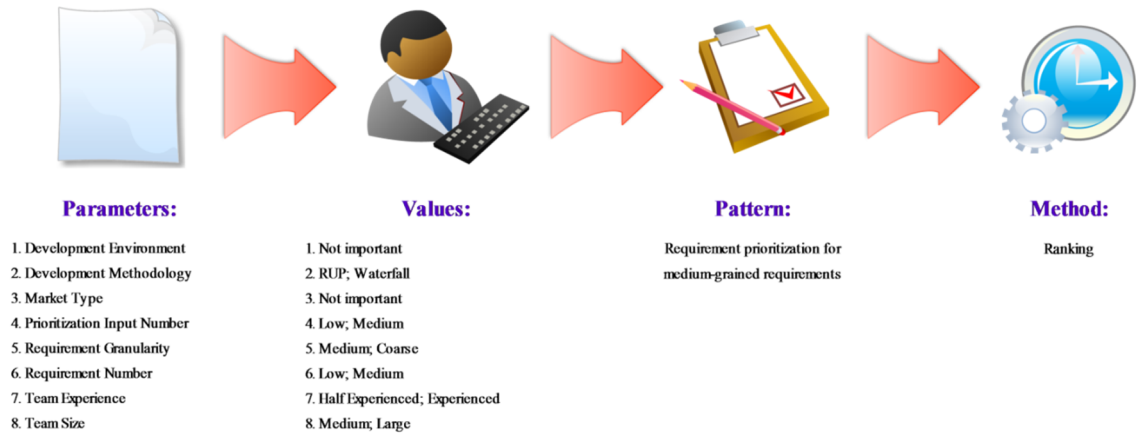


Figure 5-9: Requirement prioritization pattern with medium level of requirements

- **Resulting context**

  In this method, all requirements are considered as inputs and stakeholders' vote is received for every requirement. Each requirement is prioritized by running the method.

- **Rationale**

  Ranking method can be used for coarse or medium levels of requirements and supports less than three inputs. Although the method is performed manually and is time-consuming for large number of requirements, it can be accomplished by different individuals.

  Since RUP is a role-based method, it helps individuals adopt the role of requirements specialist to solve large number of requirements. Moreover, the method can be implemented using simple tools and this accelerates team activity.

- **Known use**

  Ranking method is mostly implemented with simple parameters in most software developing companies, and most tools can support sorting based on one or more parameters. Some tools can be installed on the team's website and enable stakeholders to score tasks.

- **Related sub-pattern**

  Does not exist.

5.5.1.3. Pattern of requirements prioritization with huge number of customers

- **Pattern name**

  The pattern of requirements prioritization with huge (unlimited) number of customers

- **Side name**

  The pattern of requirements prioritization using Top 10 method

- **Problem**

  In software developing projects with limited and unlimited customers where specific software usually has many customers, their vote and opinion is of great importance. In these cases, customers have many demands and addressing all of them is very time-consuming for the team. Moreover, reviewing and classifying these demands require considerable time and cost. In such projects with a high rate of demand, requirements prioritization requires an integrated method (implemented through a web-based background) to enable customers to enter and present their demands and comments.

- **Context**

  In requirement prioritization of release planning, it is always necessary to prioritize and sort primary requirements in order to facilitate selection of the most prior requirements for implementation (and sending them to the next stage).

- **Constraints**

  - The market must be of limited- or unlimited-customer type.
  - Requirements level must be Coarse.

- Number of requirements prioritization (classification) must be 1.
- Number of requirements must be large.
- Team must be small or medium-sized.
- Agile or RUP methodologies must be used.
- Team can have every level of experience.
- Development environment must be Web-based or Client-Server.

- **Solution**

  Using Top 10 method for requirements prioritization

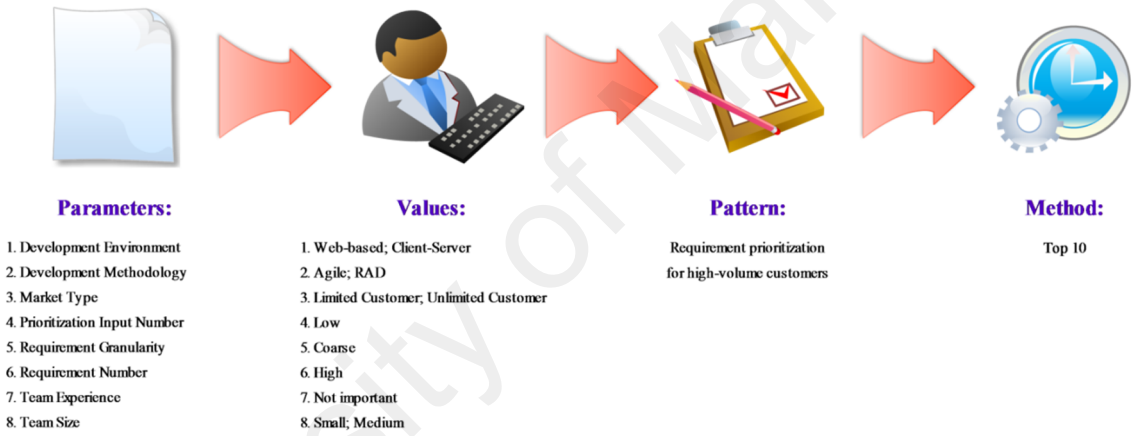


Figure 5-10: Requirements prioritization pattern with huge number of customers

- **Resulting context**

  In this method, a developer team first breaks down the project requirements to different groups to enable customers to classify their requirements. The breakdown must be performed in a proper manner. Too many groups will make customers confused and very few groups leads to significant differences among requirements while developing. Every requirement and customer demand is written in its related group and reviewed by project manager or one of the developers. The most prior group is the one with most customer demands.

- **Rationale**

  "Top 10" methodology can be used for coarse level of requirements. It is also employed when a certain software has unlimited number of customers with different needs and demands. Since comments on requirements are only entered by customers and reviewed by project manager, it does not impose a load of work on the team to select and sort. Moreover, "Agile" method is consistent with this method and frequencies can be determined based on customer needs. Besides, the requirement groups enable the method to support great number of requirements.

- **Known use**

  Most software developing companies employing "Agile" method use Issue Tracking, Bug Tracking and Request Tracking tools which allow customers to enter their demands. Then, the project manager reviews the demands and sends them to assigned individuals. The tools usually allow for demand classification.

- **Related sub-pattern**

  Does not exist.

5.5.1.4. Pattern of requirements prioritization for small projects

- **Pattern name**

  Requirements prioritization pattern for small projects

- **Side name**

  Requirements prioritization by Numerical Assignment method

- **Problem**

  Developer team is small in customized software developing projects and requirements must be prioritized respecting stakeholders' votes with low costs and

time. Besides, the method should be deducible and capable of altering during implementation to cover requirement changes.

- **Context**

  In requirement prioritization of release planning, it is always necessary to prioritize and sort primary requirements in order to facilitate selection of the most prior requirements for implementation (and sending them to the next stage).

- **Constraints**
  - The market must be of limited-customer or customized type.
  - For arbitrary level of requirements.
  - Number of requirements prioritization (classification) must be less than three.
  - Number of requirements must be small.
  - Team must be small-sized.
  - Agile or RUP methods must be used.
  - Team must be inexperienced or half-experienced.
  - For every development environment.
- **Solution**

  Using Numerical Assignment method for requirements prioritization

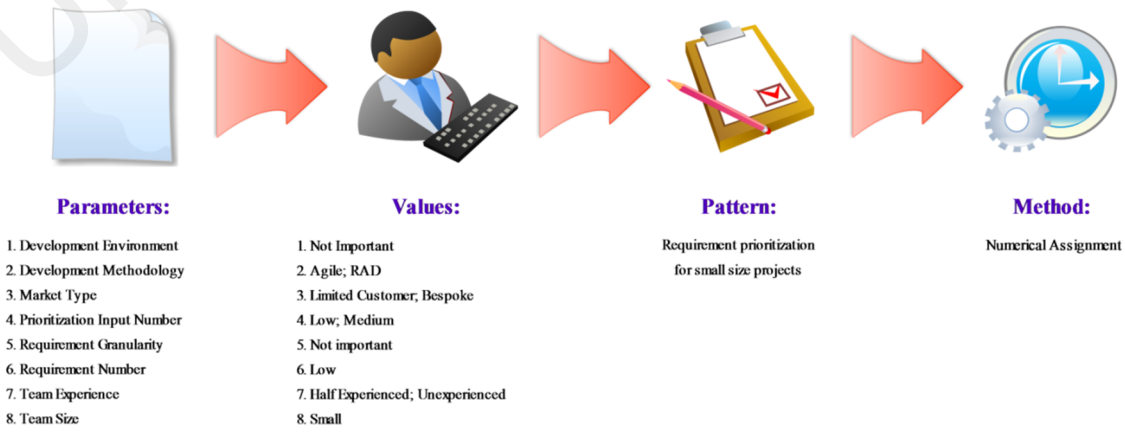


Figure 5-11: Requirements prioritization pattern for small projects

- **Resulting context**

  In this method, requirements are divided into several main groups based on their importance. These groups usually include critical, important, medium, less important and least important. Based on their view of each of their demands, stakeholders place the requirements in one of the groups and the project manager reviews and confirms them.

- **Rationale**

  Numerical Assignment method can be used for small customized projects with small teams. Such projects usually employ Agile or RAD methodologies, have small or medium number of requirements and, hence, must develop based on the customers' interests. Furthermore, since the team is inexperienced or half-experienced and customers have no expertise in software development, requirements cannot be classified like in Ranking method. This method helps in discovering important requirements according to customers and the team can focus on them specifically.

- **Known use**

  The method can be employed in most small projects. Issue Tracking, Bug Tracking or Request Tracking tools enable users to enter the importance of every certain requirement. The importance is used to recognize requirements of different groups.

5.5.2. Resource estimation patterns

Estimating required resources is the second stage of release planning process model which receives requirements (sorted based on priority) from the previous stage and the required work for every requirement from users, and gives an ordered list of

requirements with resource estimations as output. Some resource estimation patterns are described below.

5.5.2.1. Pattern of resource estimation in large projects

- **Pattern name**

  Pattern of resource estimation in large projects

- **Side name**

  Pattern of resources estimation by algorithmic models

- **Problem**

  In enormous software developing projects with a large number of requirements and high rates of predicted inputs and outputs a suitable, systematic method to estimate resources should be used which is feasible in terms of time and costs and is appropriately flexible to receive different parameters and perform required estimations. In addition, a proper tool should be employed to minimize the estimated time.

- **Context**

  In resource estimation stage of release planning, it is always necessary to estimate required work for every certain input requirement in order to select the best suited requirements for implementation and to send to the next stage.

- **Constraints**

  - RUP or Waterfall methods must be used.
  - Requirements number must be medium or large.
  - The team must be half-experienced or experienced.
  - Rate of inputs and outputs predicted for the software must be high.

- **Solution**

  Using algorithmic method for resource estimation

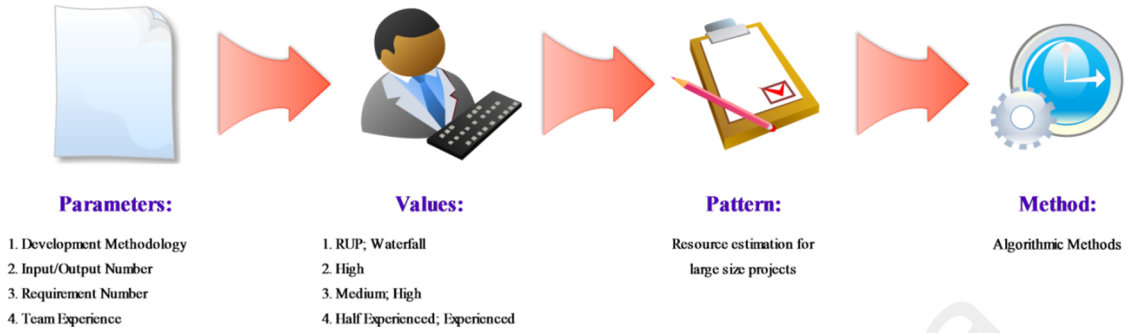


Figure 5-12: Resource estimation pattern in large projects

- **Resulting context**

  In this method, all requirements are considered as inputs and the effort required for implementing each one is estimated. Other parameters needed for the algorithmic method are introduced into the selected tool based on market type. The tool provides different estimations after receiving inputs.

- **Rationale**

  Algorithmic models are best suited for those projects in which the characteristics and environment are well-known and an experienced or half-experienced team can acquire the knowledge based on experiences of previous projects. Besides, considering the volume of requirements and amount of software inputs and outputs (which influence tasks required for resource estimation), making use of algorithmic models and their different tools can significantly reduce time and costs. It must be remembered that, in algorithmic models, the tool or the team can propose an accurate estimation method and hence a suitable tool must be selected.

- **Known use**

  Most software developing companies use a variety of tools with algorithmic methods for resource estimation.

- **Related sub-pattern**

  Does not exist.

5.5.2.2. Pattern of resource estimation for projects with unlimited customers

- **Pattern name**

  Pattern of resource estimation for projects with unlimited customers

- **Side name**

  Pattern of resources estimation by experts' judgment

- **Problem**

  In software developing projects with large number of requirements in which the team uses Agile or RAD methods, making use of more rapid methods must be considered. It must be noticed that in such cases less information is available on estimation parameters and accurate methods cannot be used. Hence, an approximate and primary estimation of every requirement is needed.

- **Context**

  In resource estimation stage of release planning, it is always necessary to estimate required work for every certain input requirement in order to select the best suited requirements for implementation and send them to the next stage.

- **Constraints**
  - RAD or Agile methods must be used.
  - Requirements number must be medium or large.
  - The team must be half-experienced or experienced.
  - Rate of inputs and outputs predicted for the software must be high.

- **Solution**

  Using experts' judgment for resource estimation

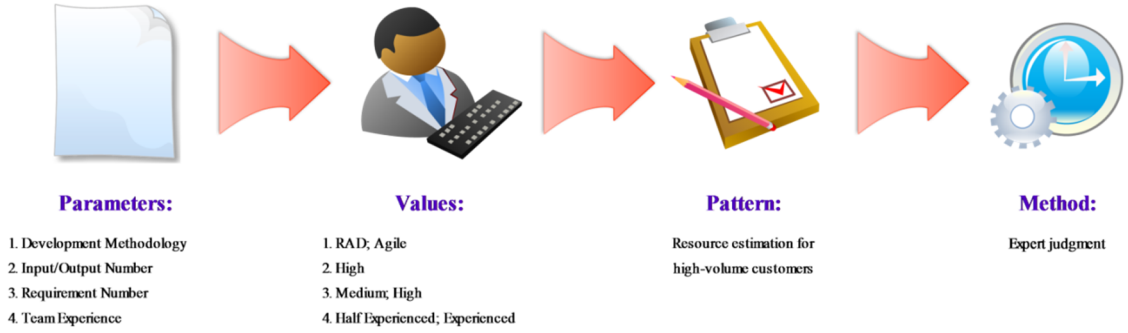


Figure 5-13: Resource estimation pattern for projects with unlimited customers

- **Resulting context**

  In this method, all requirements are considered as inputs and the effort required for implementing each one is estimated and introduced into the tool by experts. Then, the estimation is reviewed by the project manager or a group of specialists.

- **Rationale**

  Experts' judgment is best suited for those projects in which the characteristics and environment are known by experts and the team has limited number of members since RAD or Agile methods are used. In such projects, considerable amount of requirements and software's inputs and outputs (which influence tasks required for resource estimation) increase the time and costs of the algorithmic method due to their required initial data. Therefore, it is rational that the project manager or an experienced developer enters time, costs and labor estimations for every requirement to accelerate the task. Because of Agile methodology, it is possible to correct and manage predictions.

- **Known use**

  The method can be used in most projects with unlimited customers, and project management tools (Issue Tracking, Bug Tracking and Request Tracking) enable users to enter resource estimations.

- **Related sub-pattern**

  Does not exist.

5.5.2.3. Pattern of resource estimation in small projects

- **Pattern name**

  Pattern of resource estimation in small projects

- **Side name**

  Pattern of resources estimation by Down-to-Top estimation method

- **Problem**

  In software developing projects with small number of requirements and small or medium number of inputs and outputs predicted for the software, algorithmic models are not often used because of the absence of estimation parameters. Small number of requirements necessitates almost accurate estimations for development, based on which decisions are made for the release.

- **Context**

  In resource estimation stage of release planning, it is always necessary to estimate required work for every certain input requirement in order to select the best suited requirements for implementation and send them to next stage.

- **Constraints**
  - RAD or Agile methods must be used.
  - Requirements number must be medium or large.
  - The team must be half-experienced or experienced.
  - Rate of inputs and outputs predicted for the software must be small or medium.

- **Solution**

  Using Down-to-Top estimation method for resource estimation

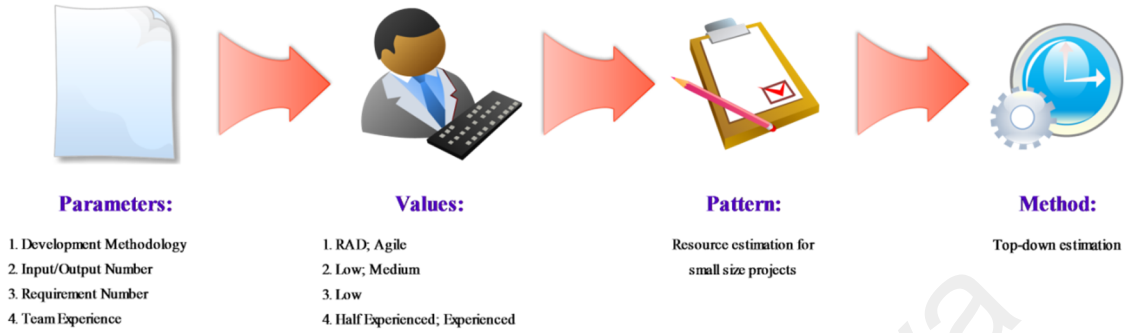


Figure 5-14: Resource estimation pattern in small projects

- **Resulting context**

  In this method, all requirements are considered as inputs and every requirement is divided into smaller tasks or requirements and this goes on until the task can be estimated. Then, estimations are made and entered into the tool for every smaller division.

- **Rationale**

  Down-to-Top estimation methods is usually used in projects employing Agile or RAD methods which have limited number of team members and low requirements volume and rate of inputs and outputs (which influence tasks required for resource estimation). In such cases, requirements are rapidly analyzed and divided into smaller requirements or tasks. Then, the workload, time and costs for every certain requirement or task are estimated and entered into the tool.

- **Known use**

  The method can be used in most small projects, and project management tools (Issue Tracking, Bug Tracking and Request Tracking) enable users to enter resource estimations.

- **Related sub-pattern**

  Does not exist.

5.5.3. Patterns of pre-release planning

Pre-release planning is the third stage of release planning process model. It receives sorted (based on priority) and estimated requirements from previous stage and uses pre-release planning algorithm to generate a set of release plans to be selected by a project manager or authority in the next stage. Some pre-release planning patterns are presented below.

5.5.3.1. Pattern of release planning in large projects

- **Pattern name**

  Pattern of release planning in large projects

- **Side name**

  Pattern of pre-release planning with intelligent methods

- **Problem**

  In software developing projects with large number of customers and limited or unlimited customer markets in which the number of requirements is very big, it is usually necessary to select a method which is capable of generating plans based on different parameters and the inter-relationship of various requirements. Regarding the number of individuals involved in the project, the method must be prone to implementation by valid tools.

- **Context**

  In pre-release planning, it is always necessary to generate a set of pre-release plans based on requirements priorities and received estimations, and send them to the next stage.

- **Constraints**
  - The market must be of limited- or unlimited-customer type.
  - RUP or Waterfall methods should be used.
  - The project must be big or medium-sized.
  - Number of requirements must be large.
  - More than three parameters must be used to generate every plan.
  - The team must be experienced or half-experienced.
- **Solution**

  Using intelligent methods for pre-release planning

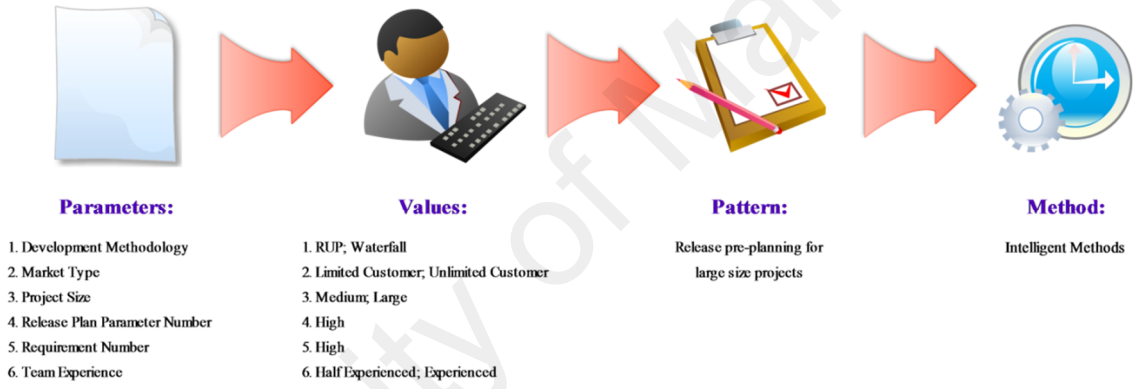


Figure 5-15: Release planning pattern in large projects

- **Resulting context**

  In this method, all requirements, their priorities and resource estimations are received as inputs from previous stage. Then, the project manager or release official enters resource constraints and requirements inter-relationships. Finally, different release plans are generated based upon more than three parameters.

- **Rationale**

  Intelligent methods can be efficient for big or medium-sized projects with limited or unlimited customers in which heavy methods such as RUP and Waterfall are used for development. Because there are many requirements which are mostly dependent, release plans must be generated accurately and considering all possible

aspects. Tools employed in these projects must be able to receive information on interdependencies and constraints of different resources and generate various release plans respecting different parameters.

- **Known use**

  The method can be used in most large projects. Release Planner is one of the most important tools employed.

- **Related sub-pattern**

  Does not exist.

5.5.3.2. Pattern of pre-release planning with large number of customers

- **Pattern name**

  Pattern of pre-release planning with large number of customers

- **Side name**

  Pattern of pre-release planning with multi-variable methods

- **Problem**

  In software developing projects with large number of customers and limited or unlimited-customer markets, very big number of requirements but small or medium project size, it is usually necessary to select a method for generating release plans which is capable of producing plans based on different parameters. Regarding the number of individuals involved in the project, the method must be prone to implementation by simple tools. Besides, it must be systematic to be able to produce various release plans.

- **Context**

  In pre-release planning, it is always necessary to generate a set of pre-release plans based on requirements priorities and received estimations and send them to the next stage.

- **Constraints**

  - The market must be of limited- or unlimited-customer type.
  - RAD or Agile methods should be used.
  - The project must be small or medium-sized.
  - Number of requirements must be large.
  - Two or three parameters must be used to generate every plan.
  - The team must be experienced or half-experienced.

- **Solution**

  Using multi-variable methods for pre-release planning

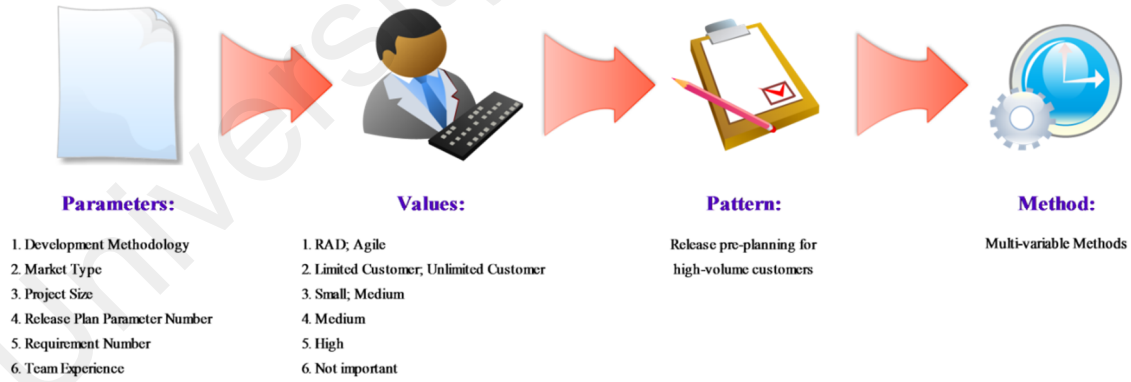


Figure 5-16: Pre-release planning pattern with large number of customers

- **Resulting context**

  In this method, all requirements, their priorities and resource estimations are received as inputs from previous stage. Then, the project manager or release official enters resource constraints, and different release plans are generated based

upon two or three parameters (including requirements priorities, time, costs or plan's activity). Besides, requirements interdependency can be one of the parameters.

- **Rationale**

  Multi-variable methods can be efficient for small or medium-sized projects with limited or unlimited customers in which Agile or RAD are used for development. In these projects, the team cannot spend considerable time on generating release plans but it needs systematic methods which can be documented and are implemented by simple tools. Use of requirements interdependencies as a parameter in generating release plans addresses this issue properly for such projects.

- **Known use**

  The method can be used in most small projects. Besides, it can be implemented using simple tools.

- **Related sub-pattern**

  Does not exist.

5.5.3.3. Pattern of pre-release planning in small projects

- **Pattern name**

  Pattern of pre-release planning in small projects

- **Side name**

  Pattern of pre-release planning with single-variable methods

- **Problem**

  In software developing projects with very small or small project size and small number of requirements, it is often necessary to use a method for generating

release plans which is capable of producing plans based on different parameters. Regarding the number of individuals involved in the project, the method must be prone to implementation by simple tools. Besides, it must be systematic to be able to produce various release plans.

- **Context**

  In pre-release planning, it is always necessary to generate a set of pre-release plans based on requirements priorities and received estimations and send them to the next stage.

- **Constraints**
  - The market must be of limited-customer or customized type.
  - RUP or Agile methods should be used.
  - The project must be very small or small-sized.
  - Number of requirements must be small.
  - One parameter must be used to generate plans.
  - The team can be of any experience level.

- **Solution**

  Using single-variable methods for pre-release planning

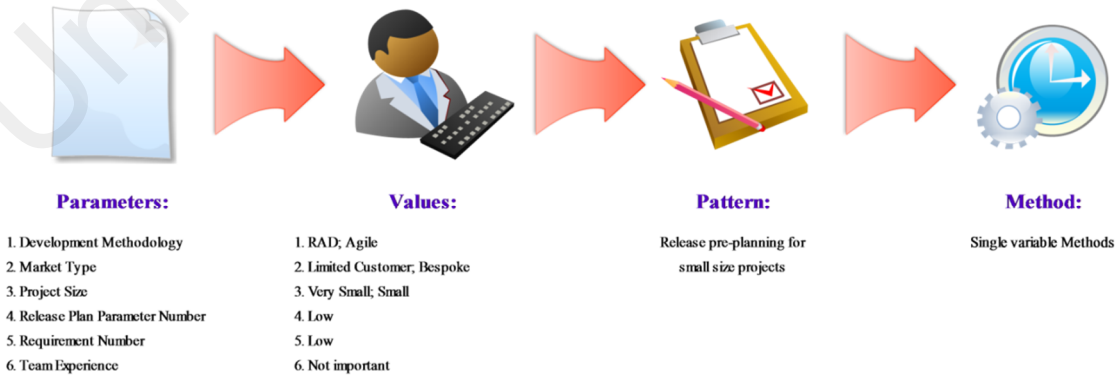


Figure 5-17: Pre-release planning pattern with single-variable methods

- **Resulting context**

  In this method, all requirements, their priorities and resource estimations are received as inputs from previous stage. Then, the project manager or release official enters resource constraints and requirements inter-relationships. Different release plans are generated based upon one parameter (including requirements priorities, time, costs or plan's activity).

- **Rationale**

  Single-variable methods can be efficient for very small or small-sized projects with limited-customer or customized markets in which RAD or Agile methods are used for development. In these projects, the team cannot spend considerable time on generating release plans but it needs systematic methods which can be documented and are implemented by simple tools.

- **Known use**

  The method can be used in most small projects. Besides, it can be implemented using simple tools.

- **Related sub-pattern**

  Does not exist.

5.5.4. Release planning patterns

Release planning patterns influence steps of release planning process model and determine a specific method for performing tasks in every stage. Some release planning patterns are presented below.

5.5.4.1. Pattern of release planning in large projects

- **Pattern name**

  Pattern of release planning in large projects

- **Side name**

  No side name

- **Problem**

  In software developing projects with large number of customers and limited or unlimited-customer markets in which the number of requirements is very big, it is usually necessary to select a method for generating release plans which is capable of prioritizing many requirements. The method must also be able to estimate required resources by receiving precise and proper data and to produce plans based on different parameters and the inter-relationship of various requirements. Regarding the number of individuals involved in the project, the method must be prone to implementation by valid tools.

- **Context**

  In pre-release planning, it is always necessary to generate a set of pre-release plans based on requirements priorities and received estimations. Then, project manager or release planner selects the best suited plan.

- **Constraints**

  - The market must be of limited or unlimited–customer type.
  - Requirements must be in "Fine" level.
  - Number or requirements prioritization (classification) inputs must be more than three.
  - Number of requirements must be medium or large.
  - The team must be big or medium-sized.
  - The team must be half-experienced or experienced.
  - Development environment must be Client-Server or Web-based.

- Number of inputs and outputs predicted for the software must be considerable.
- The project must be big or medium-sized.
- More than three parameters must be used to generate every plan.

- **Solution**
  - Using AHP method for requirements prioritization stage
  - Using Algorithmic models for resource estimation stage
  - Using Intelligent methods for pre-release planning

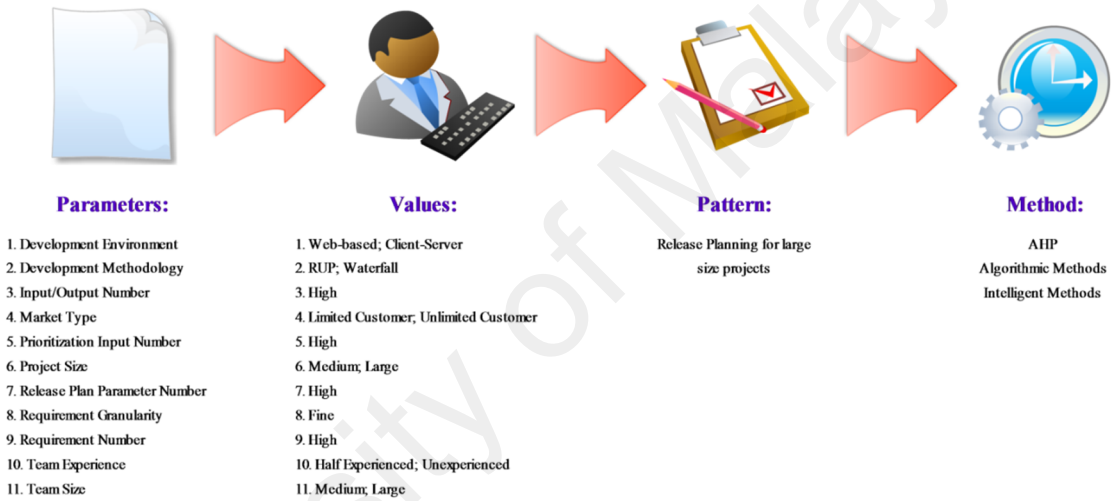


Figure 5-18: Release planning pattern in large projects

- **Resulting context**

  In this method, all requirements are introduced to the tool to be prioritized. Having prioritization accomplished, the required resources are estimated by the proper tool. Then, the project manager or release official enters resource constraints and requirements inter-relationships. Next, different release plans are generated based upon more than three parameters (including requirements priorities, time, costs or plan's activity). Finally, the project manager or release planner selects the best suited plan.

- **Rationale**

  In large project with large number of requirements and involved individuals, reliable methods should be employed for every stage. Rationale of every stage is described in each pattern according to the solution used.

- **Known use**

  The method can be used in most large projects. Release planner is one of the most important tools used in this method.

- **Related sub-pattern**

  - Pattern of requirements prioritization in large projects
  - Pattern of resource estimation in large projects
  - Pattern of pre-release planning in large projects

5.5.4.2. Pattern of release planning with large number of customers

- **Pattern name**

  Pattern of release planning with large number of customers

- **Side name**

  No side name

- **Problem**

  In software developing projects with a small-sized team but large number of customers and limited or unlimited-customer markets in which the number of requirements is medium or big, it is usually necessary to select a method for generating release plans which is capable of prioritizing many coarse requirements. The method must also be able to estimate required resources by receiving precise and proper data and to produce plans based on different parameters and the inter-relationship of various requirements. Regarding the

number of individuals involved in the project, the method must be prone to implementation by valid tools.

- **Context**

  In pre-release planning, it is always necessary to generate a set of pre-release plans based on requirements priorities and received estimations. Then, the project manager or release planner selects the best suited plan.

- **Constraints**
  - The market must be of limited or unlimited–customer type.
  - Requirements must be in coarse or very coarse levels.
  - Number of requirements prioritization (classification) inputs must be 1.
  - Number of requirements must be medium or large.
  - Agile or RAD methods must be used.
  - The team can have every level of experience.
  - Development environment must be Client-Server or Web-based.
  - Number of inputs and outputs predicted for the software must be considerable.
  - The project must be big or medium-sized.
  - Two or three parameters must be used to generate every plan.
- **Solution**
  - Using Top 10 method for requirements prioritization stage
  - Using experts' judgment for resource estimation stage
  - Using multi-variable methods for pre-release planning

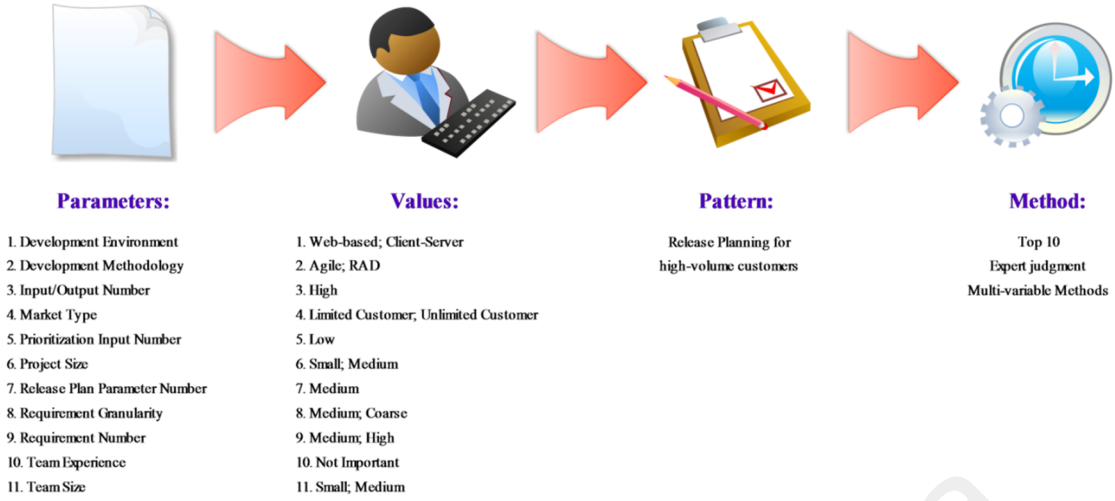


Figure 5-19: Release planning pattern with large number of customers

- **Resulting context**

  In this method, all requirements are introduced to the tool to be prioritized. Having prioritization accomplished, the required resources are estimated by the proper tool. Then, the project manager or release official enters resource constraints. Requirements inter-relationships can be considered as an independent parameter. Next, different release plans are generated based on two or three parameters (including requirements priorities, time, costs or plan's activity). Finally, the project manager or release planner selects the best suited plan.

- **Rationale**

  In large project with large number of requirements and involved individuals, reliable methods should be employed for every stage. Rationale of every stage is described in each pattern based on the solution used.

- **Known use**

  No known use

- **Related sub-pattern**

  - Pattern of requirements prioritization with large number of customers
  - Pattern of resource estimation with large number of customers

- Pattern of pre-release planning with large number of customers

5.5.4.3. Pattern of release planning in small projects

- **Pattern name**

  Pattern of release planning in small projects

- **Side name**

  No side name

- **Problem**

  In software developing projects with a small-sized team and small number of customers and customized or limited-customer markets in which the number of requirements is insignificant, it is usually necessary to select a method for generating release plans which is capable of prioritizing many coarse requirements. The method must also be able to estimate required resources by receiving proper and sometimes deficient data and to produce plans based on different parameters. Regarding the number of individuals involved in the project, the method must possibly be easy to handle and prone to implementation by simple tools.

- **Context**

  In pre-release planning, it is always necessary to generate a set of pre-release plans based on requirements priorities and received estimations. Then, project manager or release planner selects the best suited plan.

- **Constraints**

  - The market must be of limited-customer or customized type.
  - Requirements level is arbitrary.

- Number or requirements prioritization (classification) inputs must be less than three.
- Number of requirements must be small.
- The team must be small-sized.
- Agile or RAD methods must be used.
- The team can have every level of experience.
- Every development environment can be used.
- Number of inputs and outputs predicted for the software must be small or medium.
- The project must be very small or small-sized.
- Only one parameter must be used to generate every plan.

- **Solution**
  - Using Numerical Assignment method for requirements prioritization stage
  - Using Down-to-Top estimation method for resource estimation stage
  - Using single-variable methods for pre-release planning

Figure 5-20: Release planning pattern in small projects

- **Resulting context**

  In this method, all requirements are introduced to the tool to be prioritized. Having prioritization accomplished, the required effort is also estimated by the same tool. Then, the project manager or release official enters resource constraints. Next, different release plans are generated based on one parameter (including requirements priorities, time, costs or plan's activity). Finally, the project manager or release planner selects the best suited plan.

- **Rationale**

  In small projects with insignificant number of requirements and involved individuals, reliable rapid methods should be employed for every stage. Rationale of every stage is described in each pattern based on the solution used.

- **Known use**

  No known use

- **Related sub-pattern**

  - Pattern of requirements prioritization in small projects
  - Pattern of resource estimation in small projects
  - Pattern of pre-release planning in small projects

5.6. Release planning anti-patterns

The idea of anti-patterns results from the fact that most common tasks in software engineering focus on constructive and effective solutions. Anti-patterns focus on negative and unsuccessful solutions and describe a set of common solutions (for a certain problem) which definitely generate negative and unsuccessful results. Similar to the necessity of knowing different patterns available for a certain field, recognizing anti-patterns helps in better development of software. Indeed, many software engineers are willing to know whether the solution they adopt leads to failure. The answer to this

question plays an important role in project implementation. Advances in anti-pattern concept lead to a stronger relationship between them and patterns. In fact, when a problem background changes, patterns change to anti-patterns and this causes the pattern's proposed solutions fail to cope with. In such a case, anti-pattern emerges and shows that using previous pattern leads to failure.

Anti-patterns were first proposed by Brown (Brown, 1998) and different works were done to identify and record various anti-patterns afterwards. Similar to patterns, anti-patterns have several classifications, the most prominent of which is the one proposed by Brown (Brown, 1998) which divides them into three groups: management, architecture and development anti-patterns. Management anti-patterns focus on management concepts of a project. Development anti-patterns are used in software development and architecture ones are applied in the architecture of the software.

Anti-patterns are present in release planning as well as other software engineering fields and include the items that must not be done during different steps of release planning. Similar to release planning patterns, development of anti-patterns requires experiences resulting from recorded failures in release planning. Using Ad-hoc method here has proved to have lower reliability and to lead to failure in slightly complicated projects. Besides, Planning Game method cannot balance the demanded requirements on which stakeholders disagree (Seyed Danesh, 2011). This is the main weakness of the method, and means that the method loses its efficiency when the number of requirements prioritization parameters increases.

A set of anti-patterns are identified and classified in developing release planning patterns. These are presented using a structure similar to that of release planning patterns. The proposed structure is as follows:

- **Anti-pattern name**

  This shows the selected name for release planning anti-pattern. Every anti-pattern has a main name distinguishing it from other anti-patterns.

- **Side name**

  Every anti-pattern may have one or more side names besides its main name. The side name may be used for anti-pattern application in specific fields.

- **Problem**

  Anti-pattern problem describes the problems and troubles always present in a certain field that lead to failure in release planning tasks or its different steps.

- **Cause**

  Explains causes of a problem. Usually, parameters and instances are used to describe or explain every cause and it is stated based on various parameters.

- **Solution**

  Describes proposed ways to solve a problem. The selected method is presented according to proposed problems (the goal of anti-pattern) and mentioned causes.

- **Related anti-pattern**

  The names of other related anti-patterns are mentioned in this feature.

5.6.1. Anti-patterns of requirements prioritization

Anti-patterns of requirements prioritization mostly focus on problems and troubles in requirements prioritization stage of release planning. Some common anti-patterns of requirements prioritization are described below.

5.6.1.1. Anti-pattern of requirements prioritization in large projects

- **Anti-pattern name**

  Anti-pattern of requirements prioritization in large projects

- **Side name**

  No side name

- **Problem**

  In large projects with significant number of customers and Web-based development environment, most companies employ web-based methods and tools to receive requirements of many customers. Such methods usually receive only one parameter for prioritization. Top 10, Ranking and Numerical Assignment methods are mostly used in these companies, since many tools readily support them. This results in troubles in requirements prioritization and failure in identification of the most prior requirements.

- **Cause**

  The main cause of employing these methods is their simplicity, massive requirements, lack of knowledge on their obstacles and failure of purchased tools in supporting more complicated and stronger methods.

- **Solution**

  Making use of requirements prioritization pattern resolves this obstacle.

- **Related anti-pattern**

  Does not exist.

5.6.1.2. Anti-pattern of requirements prioritization in small projects

- **Anti-pattern name**

  Anti-pattern of requirements prioritization in small projects

- **Side name**

  No side name

- **Problem**

  In small projects with insignificant number of customers and Web-based or customized development environment, most companies employ Ad-hoc methods and tools to prioritize customers' requirements which are mostly individual-based and not reliable. These companies assign a team member to perform the task disregarding the fact that requirements prioritization must be principle-based. The member accomplishes the assigned task according to his/her experience. As a result, customers' requirements are developed based on experiences only and this usually results in separation of development process and recruiting specialists, especially when requirements are in higher levels and require a breakdown.

- **Cause**

  The main cause of employing these methods is their strong reliance on experiences, simplicity of using Ad-hoc, lack of knowledge on method's obstacles and neglecting principles of requirements prioritization.

- **Solution**

  Making use of requirements prioritization pattern for small projects resolves this obstacle.

- **Related anti-pattern**

  Does not exist.

5.6.2. Anti-patterns of resource estimation

Anti-patterns of resource estimation mostly focus on problems and troubles in resource estimation stage of release planning. Some common anti-patterns of resource estimation are described below.

5.6.2.1. Anti-pattern of resource estimation in large projects

- **Anti-pattern name**

  Anti-pattern of resource estimation in large projects

- **Side name**

  No side name

- **Problem**

  In large projects with significant number of requirements and customers, most companies tend to use web-based methods and tools to record resource estimations and this is usually done according to individual judgment. The companies employ a project manager or specialist to estimate required effort and resources since most available tools readily support them. This causes many problems in resource estimation challenges such as component accumulation and consistency of estimated resources with different implementation parameters (e.g. programming language) and, hence, resources are estimated with a high error rate.

- **Cause**

  The main cause of employing these methods is their strong reliance on experiences, simplicity of use, lack of knowledge on method's obstacles and deficiency in using proper tools.

- **Solution**

  Making use of resource estimation pattern for large projects resolves this obstacle.

- **Related anti-pattern**

  Does not exist.

5.6.2.2. Anti-pattern or resource estimation in small projects

- **Anti-pattern name**

  Anti-pattern of resource estimation in small projects

- **Side name**

  No side name

- **Problem**

  In small projects with insignificant number of customers and customized type of development, most companies tend to estimate required resources according to a developer's initial guess, regardless of many primary parameters. This guessing in many cases delays the project. But the method (without any parameters) is increasingly being used. In such small projects, resources are estimated in macro levels regardless of components, number and level of requirements and many implementation parameters.

- **Cause**

  The main cause of employing these methods is their strong reliance on experiences, dismissing resource estimation task and disrespecting resource estimation standards.

- **Solution**

  Making use of resource estimation pattern for small projects resolves this obstacle.

- **Related anti-pattern**

  Does not exist.

5.6.3. Anti-patterns of pre-release planning

Anti-patterns of pre-release planning mostly focus on problems and troubles of this stage of release planning. Some common anti-patterns of pre-release planning are described below.

5.6.3.1. Anti-pattern of pre-release planning in large projects

- **Anti-pattern name**

  Anti-pattern of pre-release planning in large projects

- **Side name**

  No side name

- **Problem**

  In large projects with significant number of requirements and customers, most companies tend to use multi-variable methods. Although these methods can consider several parameters, they fail to include requirements' interdependencies and relationship complexities. Most companies perform pre-release planning regardless of the interdependencies, which can be of great importance in many release plans. This causes many problems in implementation of generated plans, such as: increase in time and costs of development, deficiency in some requirements, etc.

- **Cause**

  The main cause of employing these methods is lack of knowledge on advanced methods and tools of release planning and problems of multi-variable methods.

- **Solution**

  Making use of pre-release planning pattern for large projects resolves this obstacle.

- **Related anti-pattern**

  Does not exist.

5.6.3.2. Anti-pattern of pre-release planning in small projects

- **Anti-pattern name**

  Anti-pattern of pre-release planning in small projects

- **Side name**

  No side name

- **Problem**

  In small projects with insignificant number of requirements and customers and customized development environment, most companies do not plan a release and, in fact, no release plan is generated but a set of requirements are selected by Ad-hoc method for the next stage. This set cannot be considered a plan, since it lacks structure, integration and logic of a plan and it is only a list of requirements to be developed. This postpones the project to a considerable extent; a new requirement is added to (or removed from) the list while developing the requirements, and a drastic change occurs in resource consumption. In many cases, this can lead to project failure.

- **Cause**

  The main cause of employing these methods is their reliance on personal knowledge and experience, neglecting pre-release planning task and lack of knowledge on different release planning methods.

- **Solution**

  Making use of pre-release planning pattern in small projects resolves this obstacle.

- **Related anti-pattern**

  Does not exist.

5.7. Summary

Pattern-based release planning methodology is a new methodology to release planning based upon shared tasks in release planning. The methodology tries to use those shared tasks to generate the process model of planning. In every step, the process receives a set of input parameters and generates a set of outputs through a selected method. Several parameters that influence determining the method are identified and conducted through different studies in order to achieve a precise and suitable method for every certain step. These parameters enable customization of release planning process model for different companies and projects.

Customization of the process model can be performed using various parameters but the important point is the number of customization states generated by combining the parameters. In order to reduce the number of these states, a set of relationships are found and specified between the parameters. Some of these relationships can well determine the selected method for the next step. Although the number of states reduces by determining their inter-relationships, it is not possible to determine the method for all available states. Therefore, PRP tool has been developed to facilitate the task.

In order to optimize parameters, the concept of pattern in software development was redefined and used for release planning. Patterns of release planning and its different steps are suggested as a new methodology to release planning. Patterns originate from previous experiences in release planning and are generated based on different parameters effective on every step of the planning process. Release planning patterns can be specified by determining effective parameters on release planning.

CHAPTER 6: EVALUATION OF THE METHODOLOGY

6.1. Introduction

The main objective of evaluating the pattern-based release planning methodology is to assess the applicability of the methodology in real world. In this section, first, the objective of evaluation and its expected results are identified. Then, the circumstances of the evaluation and the step by step process of evaluation are described. To understand the characteristics of case studies, evaluation of effective parameters on the pattern is also presented. These parameters affect the pattern selection in each step of release planning process model.

6.2. Evaluation Objectives

Proving the applicability of the methodology is the main objective of the evaluation. The applicability conditions are very important and can influence the results; hence, they should be clearly identified. These conditions refer to situations and requirements that should be considered and provided for measuring applicability. The applicability conditions considered for evaluating the pattern-based release planning methodology are as follows:

- *Real world experiences*

  The methodology should be applicable in the real world and industrial projects.

- *Variety of experiences*

  The methodology should be applicable in various project types and companies.

- *Complete usage*

  The methodology should be applicable in the true project lifecycle and if a project has more than one iteration, it should be used in all iterations.

The conditions should be considered as the default conditions for proving the applicability of the pattern-based methodology. Besides, secondary objectives are considered as follows:

- The applicability of each proposed pattern in the release planning
- The applicability of overall patterns of each release planning step
- The applicability of pattern-based release planning methodology

6.3. Evaluation method

Evaluation and demonstration of validity and applicability of the proposed methodology is one of the most important issues in presenting an empirical idea. The most important method to prove a methodology is operational and its results are confirmed is implementing various case studies. Industrial case studies that run in real world environment can adequately evaluate a release planning methodology and its weakness and strengths. Therefore, to evaluate the proposed methodology a variety of empirical studies in real environments and projects should be performed.

To evaluate the proposed methodology, two methods are used: 1) Empirical case studies 2) Experts review by questionnaires. To perform case studies, a software tool called "Pattern Release Planner (PRP)" was developed. The tool helps the release planner to search, identify and select patterns and gather experts' opinions on them. The PRP records steps, parameters, instances and patterns of release planning. Thus, it is used in performing case studies and its data is complemented with accomplished studies. In the later sections, details of each evaluation method are presented.

6.3.1. Case studies evaluation method

In order to evaluate the case studies, following tasks are performed for each case study:

- **Specifying characteristics of the project and choosing the right pattern**

  In the first step, characteristics of the case studies are specified and entered in the PRP tool. Then, using these characteristics, the PRP tool determines a list of proposed pattern(s). Characteristics of every parameter of each release planning step should be entered. In fact, the characteristics are the parameter instances that are already introduced into the tool and the release planner has to select. In most cases, parameter instances are fully entered but if the organization, company or project has a characteristic that is absent, the software enables the users to add it.

  The PRP tool searches for related patterns in its database and then, based on the recorded experiences "proposes" one of the available patterns. If the tool does not have recorded experiences, it does not propose any patterns. This is also true when no characteristics are available and the release planner allows recording the pattern result or selecting an existing pattern. In other words, the tool improves its suggestions based on preceding experiences and is limited only to searches and listing patterns if no experience is recorded. The result of this step is the best fitted pattern for projects based on the entered characteristics.

  We introduced 13 basic patterns in the PRP tool that can be used by the release planner of each project. More patterns can be added up later. The properties of each pattern are based on the structure of the release planning patterns presented before. The patterns are "initial pattern" and were not tested before and resulted based on our experiences and studies. The case studies can validate and measure the effectiveness of the patterns as well as the pattern-base release planning methodology. The pattern structure is described to all release planners of each

project in the case studies and they can either use the pattern or define their own patterns.

- **Accomplishing release planning through proposed patterns**

  The pattern proposed by the tool for each step of the process model should be employed by the development team of each project. The objective of this step is to make actual use of the proposed pattern in actual software development. Depending on the type of pattern, its implementation can include employing a certain method or a combination of different methods; therefore, it may require several attempts until implementation is completed. All events and the results are documented during pattern usage. Besides, challenges, ideas and problems faced during the implementation are recorded to be used to improve decision-making.

  The actual use of the proposed patterns is the heart of evaluation and only patterns that are used in various projects can be evaluated. Other patterns cannot be evaluated simply because there are no ideas about their actual usage. Therefore, choosing the pattern by release planners and applying the method proposed for each step of release planning process model can lead to certain results that should be logged by the planners.

- **Recording and analyzing the pattern usage results**

  In the third step, the results of pattern usage are recorded into the tool and analyzed. The tool allows the release planner to enter the general points and details of the usage results. The PRP tool decides based on the overall results, to either place or not to place the pattern on the selected list for the next pattern search. Moreover, the tool allows for entering details of the results based on which selection is to be done among several patterns. Finally, it is possible to enter and record additional information about the pattern if it is reselected.

Figure 6-1 shows the method of performing case studies. All information on accomplishing case studies and employing patterns are recorded in the PRP to be used in the next steps.

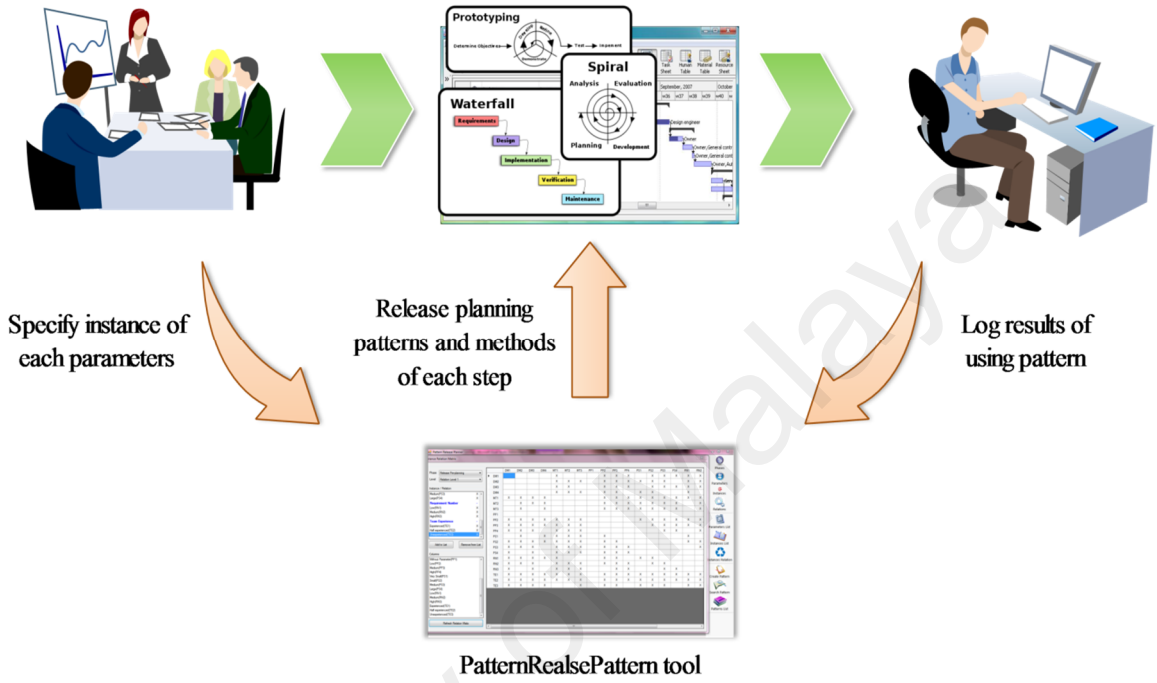


Figure 6-1: Method of peforming case studies

6.3.2. Experts review evaluation method

After executing empirical studies and using patterns and the pattern-based methodology in case studies, to fully evaluate the methodology, three questionnaires (about each pattern, the patterns of specific steps in the process model and the overall pattern-based release planning methodology) are filled by the experts that was project manager or release planner. These help to enhance the empirical study for all patterns and the methodology.

The project manager or release planner who used the patterns and release planning pattern in their projects as an expert filled the questionnaires for all patterns. Questionnaires are presented in Tables A, B and C of Appendix B.

6.4. Evaluation Success Factors

Evaluation of the proposed methodology can be performed by the recorded results of different patterns' usage in various projects in the PRP tool as well as the questionnaires filled by project manager or release planner. Evaluation is done on the whole pattern-based release planning methodology and each release planning pattern. To evaluate each pattern, it is necessary to analyze PRP tool information on the pattern and examine the recorded descriptions. Naturally, results of evaluating every pattern fall into one of the following categories:

- **The proposed pattern fits the project**

  If the number of pattern usage is adequate and the rate of satisfaction is high, it can be claimed that the pattern suits for release planning. On the other hand, the results show the applicability of a pattern in release planning when the rate of satisfaction is high and the proposed pattern by the tool fits the projects.

- **The proposed pattern fits in some cases and does not in others**

  If the number of pattern usage is adequate and the rates of satisfaction and failure are moderate, it cannot be claimed that a pattern suits or fails in different cases. In fact, this is considered a moderate result which fails to prove that the proposed pattern can be implemented.

- **The proposed pattern fails to fit the project**

  If the number of pattern usage is adequate and the rate of failure is high, it can be claimed that the pattern fails. The fail result demonstrates that the proposed pattern fails to prove it can be implemented and the pattern is rejected.

In addition to what mentioned above, it must be noted that some other problems can influence the implementation of pattern-based release planning, including its complexity and the time required to understand the methodology. These are identified

as evaluation parameters of pattern-based release planning and must be recorded during implementation of planning methodology in order to be used when comparing the approach with others. Although the proposed methodology possesses the proper tool to suggest patterns, the parameters are also of great importance in selecting the pattern-based release planning methodology.

In order to reduce the influence of unknown parameters on parameters in case studies, it is essential to focus on understanding and assimilating parameters and methods and describe the methodology properly to users. This is done by the tool and users are able to observe different states of the proposed patterns to act well in valuing parameters and selecting parameter instances. It must be noted that some exceptions are inherent to case studies, but can be neglected if their frequency is insignificant.

6.5. Case studies Selection

Several case studies are used to validate the methodology in this study, regarding numerous parameter instances in release planning process model and likely states, and considering the fact that release planning patterns must be based upon executive experiences. In order to select case studies, three parameters were considered:

- *Number of projects*: number of active projects that company performs
- *Number of teams*: number of active teams that are independent from others
- *Team size*: number of people involved in a team

Companies with several projects and various teams were selected, since every team could be considered as an independent unit and document its experiences of a certain pattern. Team diversity in such companies causes various states in case studies. Moreover, manifoldness of projects generates same parameter instances in projects with the same characteristics. The instances are likely to result in the same patterns and this leads to implementation of a certain pattern in two different teams (two different experiences).

Selecting companies with such features and convincing them to participate in studies is difficult, since most of them use simple methods for release planning. Hence, PRP is developed in a way to facilitate the task. All selected companies have numerous projects in different fields and meet the goal of generating various states. Here, all companies have at least 7 projects and small, medium and big-sized software developing teams, except for one which has two projects only and two separate teams. This enables us to consider several states; nevertheless, claiming full implementation of all possible states is not the focus of this research. PRP was taught to teams which have release plans and interact with users and customers. Besides, the process of determining team and project characteristics was explained to them in order to help in achieving the release planning patterns. Moreover, every release planning method as well as its steps and procedures were described to the project manager or release planner in order to develop a unified perspective of all methods.

6.5.1. Company A description

Company A has been developing banking and insurance software for 7 years and is known as one of the pioneers in national level. The company is in charge of developing and maintaining "Core Banking" software in two state banks and one private bank in Iran, and is considered as one of the main contributors to the comprehensive financial guidelines of the country. Moreover, it is a public contractor in information and communication technology with the capacity and power to implement large-scale projects. It breaks large projects down to smaller ones to be accomplished by small companies and manages the projects itself. Main products and services of the company include:

- Banking comprehensive guidelines
- Insurance comprehensive guidelines
- Capital market strategies

- Transportation guidelines

Now, the company has more than 250 IT specialists and more than 10 separate software developing teams which are working continuously. In addition to these teams, new ones are formed for newer projects. Teams are managed through project management hierarchy, and they employ a simple tool (Issue Tracking) to collect customers' requirements and their comments and suggestions. Regular procedures are mediated and conducted to accomplish tasks in higher level teams, but the procedures are not considered as teams' internal methodologies and every team uses its own methodology based on its size. Although "Agile" methodology is the most common, RUP method is efficient in some certain projects. Teams are not allowed to use methods without documentation. They usually have an expert, but sometimes there are teams whose members are not experienced at all. Most of the company projects are Web-based software.

In this company, six projects are accomplished using pattern-based release planning methodology for at least three releases. Characteristics of every project are summarized in Table 6-1. According to the table, most of its projects have common characteristics. Additionally, some projects have completely identical characteristics.

Table 6-1: Characteristics of projects in company A

Project / Parameters	A1	A2	A3	A4	A5	A6
Development Environment	Web-based	Web-based	Web-based	Web-based	Web-based	Web-based
Development Methodology	Agile	Agile	Agile	Agile	Agile	Agile
Input/Output Number	Low	Medium	High	High	High	Low
Market Type	Bespoke	Bespoke	Unlimited Customer	Unlimited Customer	Unlimited Customer	Bespoke
Prioritization Input Number	Low	Low	Low	Low	Low	Low

Project / Parameters	A1	A2	A3	A4	A5	A6
Project Size	Small	Small	Small	Medium	Medium	Small
Release Plan Parameter Number	Low	Low	Medium	Medium	Medium	Low
Requirement Granularity	Fine	Medium	Medium	Medium	Coarse	Fine

6.5.2. Company B description

This company has been working on software development for more than 25 years. It started working on financial software exactly when PCs were just entering the country and were not much common yet. This is the first software company in Iran which supplied Windows-based systems as an integrated one (based on technology transformation and customers' demands) in 1997. Nowadays, with more than 9500 customers in big, medium and small businesses and more than 1100 employees, the company is one of the biggest software developing companies in Iran and is almost dominant in the field of financial software. In recent years, it has turned its focus to accelerating service delivery and increasing product quality. Most important activities of this company in software developing include:

- Presenting software solutions for businesses based on operational processes of big or medium-sized organizations in mother, manufacturing, service and trading industries.
- Presenting software strategies for small businesses
- Presenting specific IT guidelines to state organizations and institutions
- Serving in educating, establishing and maintaining software strategies

At the present time, the company has more than 15 different software developing teams in financial and accounting fields, most of which have at least five members. It mostly uses an Agile-based customized method for software development and all development and support procedures are produced and implemented by quality assurance. A "Request Tracking" tool that developed in a customized manner is used to receive demands and new requirements as well as software problems. Besides customers, requirements are entered by different support teams in various cities. Every team is composed of experts, half-experienced and sometimes inexperienced individuals and the company's labor development procedures forms gradually.

In this company, nine projects are accomplished using pattern-based release planning methodology for at least two releases. Features of every project are summarized in Table 6-2. According to the table, most of its projects have common characteristics. Additionally, some projects possess completely identical characteristics.

Table 6-2: Characteristics of projects in Company B

Project / Parameters	B1	B2	B3	B4	B5
Development Environment	Client-Server	Web-based	Client-Server	Client-Server	Client-Server
Development Methodology	Agile	Agile	Agile	Agile	Agile
Input/output Number	High	High	Low	High	High
Market Type	Unlimited Customer	Unlimited Customer	Limited Customer	Unlimited Customer	Unlimited Customer
Prioritization Input Number	Low	Low	Low	Low	Low
Project Size	Medium	Medium	Small	Medium	Medium
Release Plan Parameter Number	Medium	Medium	Low	Medium	Medium
Requirement Granularity	Medium	Medium	Medium	Medium	Medium

Project / Parameters	B6	B7	B8	B9
Development Environment	Client-Server	Client-Server	Client-Server	Client-Server
Development Methodology	Agile	Agile	Agile	Agile
Input/output Number	Medium	Low	High	High
Market Type	Limited Customer	Limited Customer	Limited Customer	Limited Customer
Prioritization Input Number	Low	Low	Low	Low
Project Size	Small	Small	Medium	Medium
Release Plan Parameter Number	Low	Low	Medium	Medium
Requirement Granularity	Coarse	Medium	Coarse	Medium

6.5.3. Company C description

This company was founded in 2005 by a combination of reputable IT companies and support and investment of active companies in the capital market with the goal of presenting the first "total online guideline" in the field of capital market. With less than 8 years of activity background, the company hosts 51 agents in the Stock Exchange, 41 agents in Goods Exchange and 62 investment funds with more than 1500 branches in Iran. It also organizes operations of 2600 users for 500,000 customers and serves more than half of Iranian exchange companies.

By establishing a "Dedicated Data Center" inside Iran and presenting its solution in the form of "Software as a Service", the company tries to attract trust and increase productivity of IT industry. Hosting its "Data Intensive Total Solution" in this database, the company saves its customers from spending on hardware and struggling with IT specialists and lets them focus on their own businesses. It has solved the problem of "Data Fragmentation" through the Data Intensive Total and has made it possible to control operations, provide reports and process data without any technical complications

and human errors. The company has overcome the obstacle of geographical distance, which is the main problem facing suppliers of IT solutions, through the "Dedicated Data Center" which is used to maintain hardware equipments and provide software services. Without worrying about this problem, company experts can support customers in the shortest time (without any need to be present at the customer's location). Most important activities of this company, which are mostly in the field of stock exchange, include:

- Management system for agents of the Stock Exchange
- Management system for agents of Goods Exchange
- Trade Work Station (TWS)
- Online trade
- Common investment fund
- Exchange fund
- Stock future

At the present, 8 different software developing teams are working in this company besides the support teams. Considering massive demands and customers' new requirements, a Request Tracking tool (developed by the company) is used and customers' needs are assigned to different teams for follow up. Furthermore, demands for new software and requirements of Securities and Exchange Organization (according to Exchange rules and principles) lead to formation of new teams to develop related software. Regarding the nature of Exchange tasks which are mainly based upon regulations of Securities and Exchange Organization of Iran, customers are less likely to propose new requirements and are more tended to ask for slight changes.

In this company, three projects are accomplished using pattern-based release planning methodology for at least two releases. Features of every project are summarized in Table 6-3.

Table 6-3: Characteristics of projects in Company C

Project / Parameters	C1	C2	C3
Development Environment	Web-based	Web-based	Web-based
Development Methodology	Agile	Agile	Agile
Input/Output Number	Medium	Medium	Medium
Market Type	Limited Customer	Unlimited Customer	Limited Customer
Prioritization Input Number	Low	Low	Low
Project Size	Small	Small	Small
Release Plan Parameter Number	Low	Low	Low
Requirement Granularity	Fine	Medium	Fine

6.5.4. Company D description

This company was founded in 1997 to design and plan IT integrated guidelines. After 15 years of activity, thousands of users in the form hundreds of firms use this company's software services. It has also implemented most successful software projects in Iran. Activities of this company mostly include providing financial, administrative, trade and factory management software and software strategies for business process management in medium and large-sized organizations. Benefiting from around 150 IT, industries and management specialists, the company has founded many branches in different cities for sales, training and after-sales services of its products. The company has been awarded various prizes of software development in Iran for innovation, product diversity, quality assurance and management, office automation systems and employing younger IT workers. This is evidence on its significant growth in the past 10 years.

Main activities of this company include:

- Designing and implementing integrated operational and management information systems

- Designing and implementing Enterprise Resource Planning systems
- Designing and producing customer order software
- Designing and implementing Business Processes Management Systems

Now, the company supplies more than 35 different products in financial, trading, office automation, engineering production and human resources fields and provides more than 1000 large, medium and small organizations and institutions with software services. This company has become a symbol of reliability in customers' viewpoint because of multiplicity of its users in different companies, rapid customization of software based on customer needs, and online responsibility. In addition, its newest and latest product, Business Process Management Software, is well-adopted especially by medium and small-sized companies.

Various software developing teams are working for this company which can use customized RUP, Scrum or customized RAD depending on the project. All outputs, products, documentations and activities of different teams are well defined through quality assurance management and, hence, the company owns a well-engineered software development cycle. It uses its own customized tool for requirements engineering and management which is able to record all online conversations and send them to different teams. Recording conversations eliminates the need to writing them down and reduces errors in thought transfer. This way, it is also possible to talk to every project analyzer (if necessary) to optimize demands and requirements.

In this company, eleven projects are accomplished using pattern-based release planning methodology for at least two releases. Characteristics of every project are summarized in Table 6-4.

Table 6-4: Characteristics of projects in Company D

Project / Parameters	D1	D2	D3	D4	D5	D6
Development Environment	Client-Server	Client-Server	Client-Server	Client-Server	Web-based	Web-based
Development Methodology	RUP	Agile	Agile	Agile	Agile	Agile
Input/Output Number	High	High	Low	Medium	High	Medium
Market Type	Limited Customer	Unlimited Customer	Limited Customer	Limited Customer	Unlimited Customer	Bespoke
Prioritization Input Number	High	Low	Low	Low	Low	Low
Project Size	Medium	Medium	Small	Small	Medium	Small
Release Plan Parameter Number	High	Medium	Low	Low	Medium	Low
Requirement Granularity	Fine	Medium	Medium	Coarse	Medium	Medium

Project / Parameters	D7	D8	D9	D10	D11
Development Environment	Client-Server	Web-based	Client-Server	Web-based	Client-Server
Development Methodology	Agile	Agile	RUP	Agile	RUP
Input/Output Number	High	Low	High	Medium	High
Market Type	Limited Customer	Bespoke	Limited Customer	Limited Customer	Limited Customer
Prioritization Input Number	Low	Low	High	Low	High
Project Size	Medium	Small	Medium	Small	Large
Release Plan Parameter Number	Medium	Low	High	Low	High
Requirement Granularity	Medium	Fine	Fine	Medium	Fine

6.5.5. Company E description

This company works in the field of electronics and manufactures electronic and telecommunication parts. It is the biggest electronics development company in Iran which develops safety-critical and embedded system software based on required capacities of electronics and telecommunication industries. More than 25 years of experience in electronics and 7 years in safety-critical and embedded systems have led the company to work only for a certain group of customers. It is the only company in Iran which has invested a considerable amount of capital in this field. All software and electronic developments of the company are customized and based upon customers' needs. As a result, software developing teams in safety-critical and embedded systems are experts and have special consultants for every certain customized system.

Now, the company is working on developing Railway Interlocking software which includes a series of different plans and software. Two main software teams are committed to perform this project and every team consists of four side teams. Each side team accomplishes all software engineering tasks from requirements to testing and delivery based on documented procedures. All software developing teams in this company use IBM Rational Door and IBM Rational Focal Point tools for requirements management and release planning, respectively. These tools are linked together and enable release re-planning (if necessary). Every new requirement is recorded, confirmed by team manager, and planned for various releases (if any).

In this company, two projects are accomplished using pattern-based release planning methodology for at least one release. Characteristics of every project are summarized in Table 6-5.

Table 6-5: Characteristics of projects in Company E

Project / Parameters	E1	E2
Development Environment	Desktop	Desktop
Development Methodology	Waterfall	Waterfall
Input/output Number	High	High
Market Type	Bespoke	Bespoke
Prioritization Input Number	High	High
Project Size	Large	Medium
Release Plan Parameter Number	High	High
Requirement Granularity	Fine	Fine

6.6. Effective parameters in case studies

Data pertaining to every effective parameter on pattern determination in 31 projects implemented using pattern-based release planning indicate parameters' distribution and authenticate the results. Figure 6-2 shows development environments for the studied projects. Widely-used environments are Web-based and Client-Server ones and only two projects are based upon Desktop environment. Although Web-based approach is the prominent one in most modern generation systems, it must be noted that many organizations prefer Client-Server-based software for organizational reasons, and consequently software developing companies follow this demand.

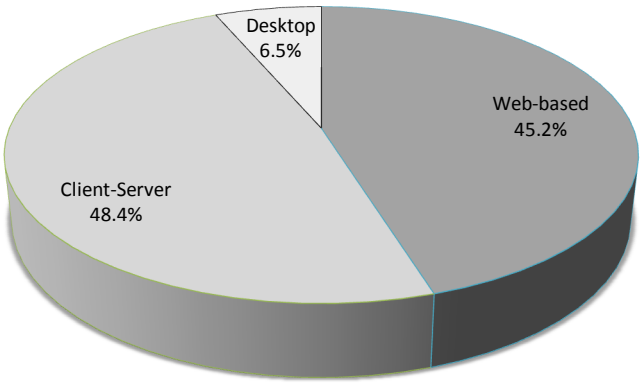


Figure 6-2: Development environment in studied projects

Figure 6-3 shows development methodologies used in the studied projects. As expected, most projects and software developing companies employ "Agile" methodology and this influences their release planning. Two "Safety Critical" projects use "Waterfall" methodology because of their nature. Since RAD method lacks documentations by nature, it is not employed in software developing companies.

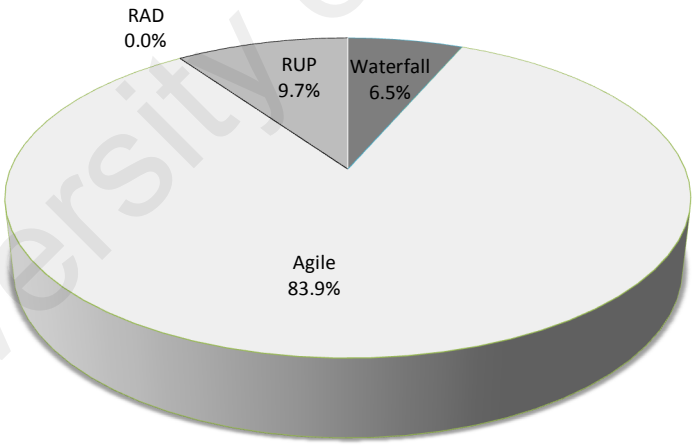


Figure 6-3: Development methodology in studied projects

Figure 6-4 shows the input and output amount of the studied projects. The input and output amount is a parameter affecting resource estimation. More than 80% of the studied projects have medium or high input and output volume. This indicates that a team is mostly faced with a heavy workload in project implementation and the release plan needs to establish a proper balance between different releases.

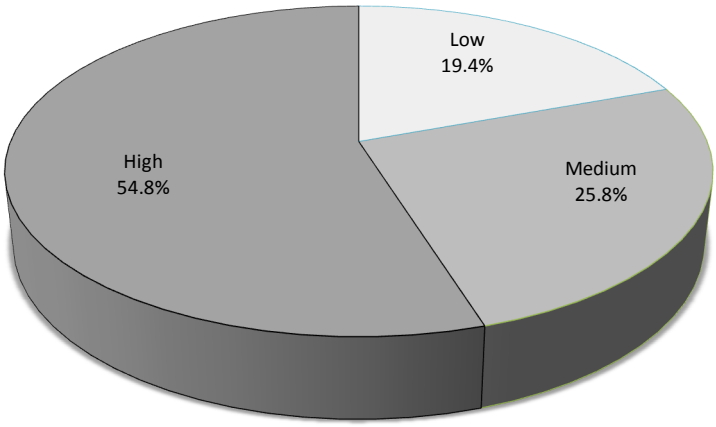


Figure 6-4: Input and output volume in studied projects

Figure 6-5 presents market types of the studied projects. Approximately, 45% of the projects have limited customers and this means that the plan must predict releases in a way that satisfies more customers and considers more key functions. This can be achieved by those release planning methods which make decisions through receiving more input parameters.

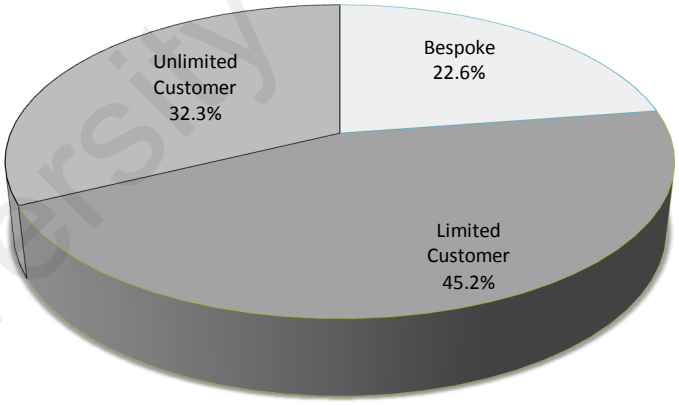


Figure 6-5: Market type in studied projects

Figure 6-6 shows the number of requirements prioritization inputs in the studied projects. Most software companies increasingly tend to exclude customers from making decisions on a release. This is more evident in projects using "Agile" methodology. Hence, companies are more and more interested in considering fewer inputs for requirements prioritization. One of the main reasons is the speed of such projects.

Moreover, it must be noted that teams tend to implement preplanned requirements first, so they try to exclude customers, especially from pre-releases, to be able to follow the plans. This is usually achieved by reducing input number and users' preferences; customers solely comment.

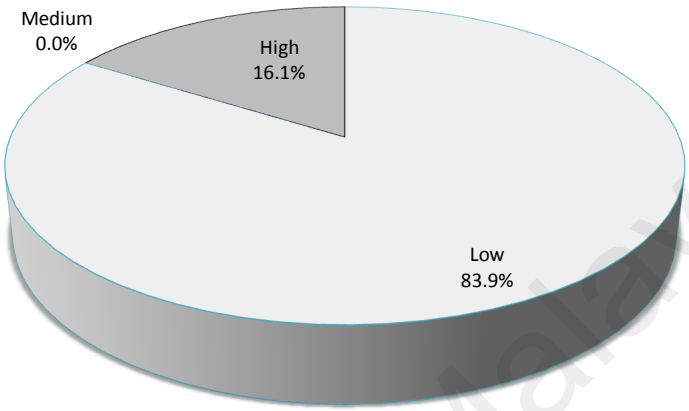


Figure 6-6: Requirements prioritization input number in studied projects

Figure 6-7 displays the size of different teams in the studied projects. More than 93% of the teams are medium or small-sized and this is directly related to the employed methodology. It must be noted that release planning tasks are usually considered as a side, not main, job by small teams and the project manager is in charge. Enthusiasm of small teams to speed up and achieve the final product reduces their tendency to plan a release. This is the main reason, as mentioned earlier, for their tendency to perform preplanned tasks.

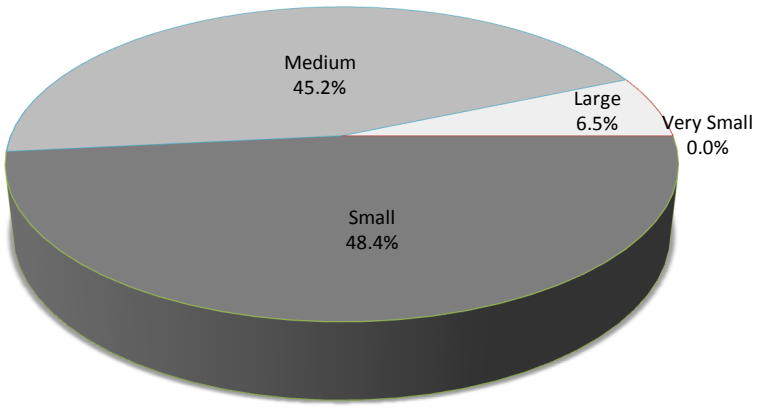


Figure 6-7: Team size in studied projects

Figure 6-8 shows the number of release planning parameters in the studied projects. Most programming teams with varying team seizes usually consider at least one parameter for the release plan and develop at least two plans to be able to compare them. Although teams consider release planning as a side task, they do not risk producing only one plan. They try, mostly, to develop various plans, compare them and select the best suited one. This indicates that Ad-hoc method is not used in any company or project. Besides, an enhancement in project and release planning importance follows an increase in the number of parameters.

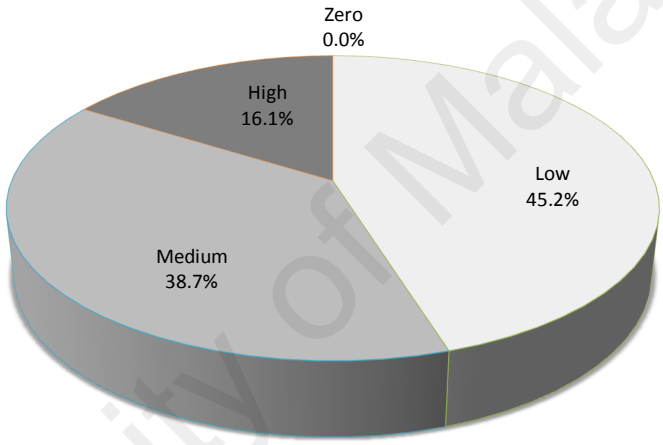


Figure 6-8: Number of release planning parameters in studied projects

Figure 6-9 shows requirements level in the studied projects. This is the typical and dominant requirements level. This means that requirements in a project may be in different levels, but the considered level is determined by averaging all requirement levels based on the experience of project managers and requirement authorities. In most projects, requirements break to a lower level (medium level of requirements). Requirements level has a direct influence on the requirement prioritization method. The higher is the requirements level, the more difficult to use accurate methods to prioritize requirements.

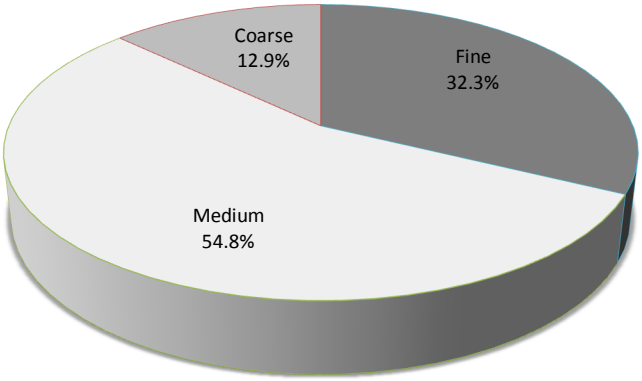


Figure 6-9: Requirements level in studied projects

Figure 6-10 presents the number of requirements in the studied projects. Considering the diversity of these projects, a huge variety of requirements are employed. Majority of requirements have small numbers and this is due to the project size and making use of "Agile" methodology. A type of logical relationship seems to be present between the three parameters but it is not always true and may be neglected in some projects. Of course, it must be noted that the requirements level is also effective on this relationship.

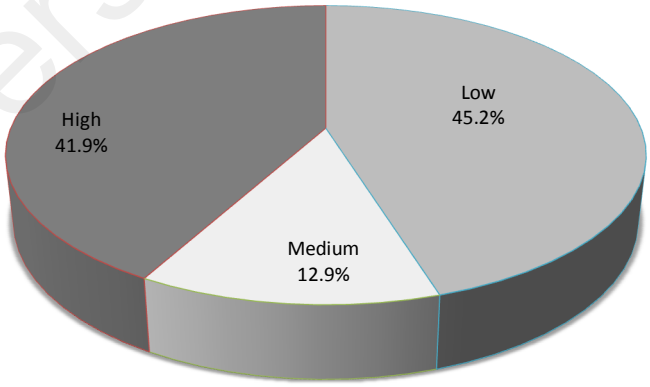


Figure 6-10: Requirements number in studied projects

Figure 6-11displays team experience in the studied projects. Inexperienced teams are not employed in any project. Although some inexperienced or half-experienced

individuals are recruited but the dominant average of teams is considered here. Pattern-based release planning is useful for inexperienced teams but it has been tried in case studies to use experienced or at least half-experienced teams. No inexperienced teams were observed in the studied companies.

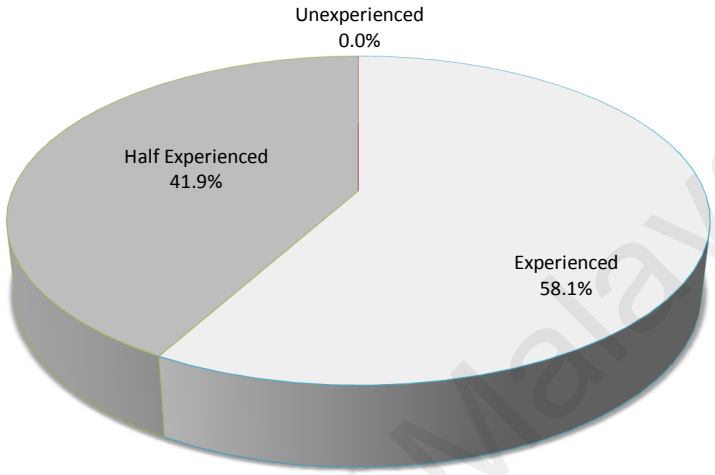


Figure 6-11: Team experience in studied projects

Figure 6-12 illustrates team size or the number of individuals voting on requirements prioritization. In most companies and projects, only one individual is involved in requirements prioritization and this is also true for release planning. Although this is directly related to project size, it is not always the case since employing efficient tools in large projects limits the number of individuals voting on release planning. In large projects with high sensitivity of time and costs, team size increases and more individuals vote for release planning.

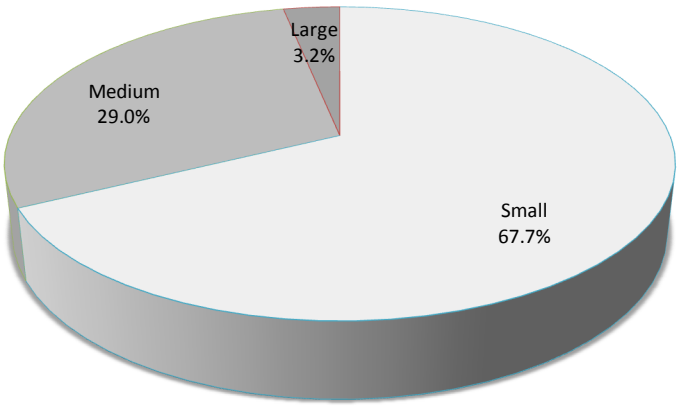


Figure 6-12: Team size in studied projects

6.7. Case studies patterns usage

In the studied projects, a variety of patterns are employed to perform release planning and accomplish its different steps. Since, in case studies, only one pattern is presented for every project with any number of releases, all various releases of a project are considered only once. It must be noted that different patterns may be used for planning a release in each case, but in the case studies only one pattern is employed in each project. To simplify, a certain code is given to each pattern which are specified in Table 6-6.

Table 6-6: Pattern codes

Step	Pattern name	Pattern code
Requirements prioritization	Pattern of requirements prioritization for large projects	PR1
	Pattern of requirements prioritization with large (unlimited) number of customers	PR2
	Pattern of requirements prioritization for small projects	PR3
	Pattern of requirement prioritization with medium level of requirements	PR4
Resource estimation	Pattern of resource estimation in large projects	PE1
	Pattern of resource estimation for projects with unlimited customers	PE2

Step	Pattern name	Pattern code
	Pattern of resource estimation in small projects	PE3
Pre-release planning	Pattern of release planning in large projects	PP1
	Pattern of pre-release planning with large number of customers	PP2
	Pattern of pre-release planning in small projects	PP3
Release planning	Pattern of release planning in large projects	P1
	Pattern of release planning with large number of customers	P2
	Pattern of release planning in small projects	P3

To what percent every pattern is used is displayed in Figure 6-13. Since every step's pattern is selected by choosing the release planning pattern, only the chart pertaining to release planning patterns is presented here. The most used pattern is P3, indicating that most studied projects had the characteristics required for making use of this pattern. The least used pattern, on the other hand, is P1. P2 is also one of the common patterns, since most big or medium-sized companies have limited- or unlimited-customer software and their customers are more than a certain limit but they use small and agile teams.

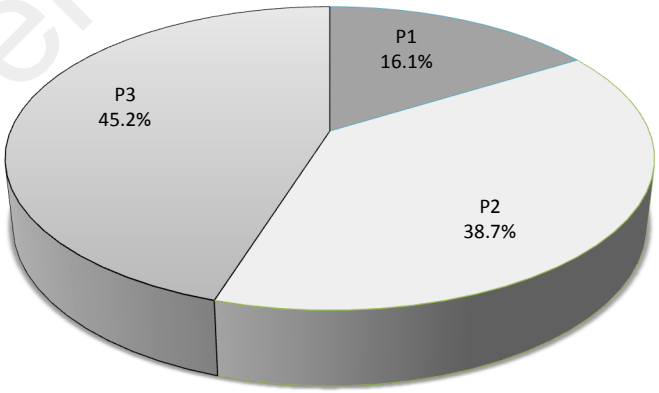


Figure 6-13: Usage of different patterns in the studied projects

6.8. Experts Demography

Validating the method by questionnaires performed after the implementing the case studies and 13 project managers or release planners are selected to fill the 3 questionnaires about pattern, patterns of each phase and pattern-based release planning (Appendix B). All of the selected experts have performed more than 4 successful projects in last 5 years of their software development experiences and have minimum 11 years experiences in software development. Table 6-7 presents the expert's demographic data.

Table 6-7: Expert demographic data

#	Role	Age	Experience	Number of Projects	Project Size*		
					Small	Medium	Large
1	Project Manager	44	17	15	5	8	2
2	Project Manager	43	14	8	2	5	1
3	Project Manager	47	14	7	0	5	2
4	Release Planner	38	12	13	10	3	0
5	Release Planner	35	11	15	9	6	0
6	Project Manager	39	13	9	3	5	1
7	Project Manager	39	11	8	5	3	0
8	Project Manager	43	18	7	0	2	5
9	Project Manager	36	11	10	8	2	0
10	Project Manager	40	15	17	13	4	0
11	Project Manager	39	14	9	4	5	0
12	Release Planner	34	11	7	3	3	1
13	Project Manager	39	12	14	11	3	0

* *Small*: 3 to 7; *Medium*: 8 to 15; *Large*: More than 15 individuals

6.9. Summary

Five companies with 31 projects are selected for case studies. The case studies are selected based on the number of projects, number of teams, and team size. All companies have at least 7 projects and small, medium and large-sized software developing teams, except for one which has two projects and two separate teams only. Also 13 experts are selected to fill the 3 questionnaires.

CHAPTER 7: EVALUATION RESULTS

7.1. Introduction

In the studied projects, a variety of patterns are employed to perform release planning. The results for using patterns and pattern-based methodology for release planning in the empirical studies for each step of the process model are calculated separately. In this section, the results of the empirical case studies and the questionnaires evaluation are presented. The results contain the evaluation for each pattern separately, the patterns in the specified release planning process step and the overall proposed methodology.

7.2. Requirements prioritization pattern evaluation

Figure 7-1 shows the result of requirements prioritization patterns usage in the case studies. The patterns failed only in two projects and were employed successfully in the others. This indicates that patterns proposed by PRP suited the teams and they could use them in practice. Therefore, parameters employed to determine the requirements prioritization patterns were also efficient.

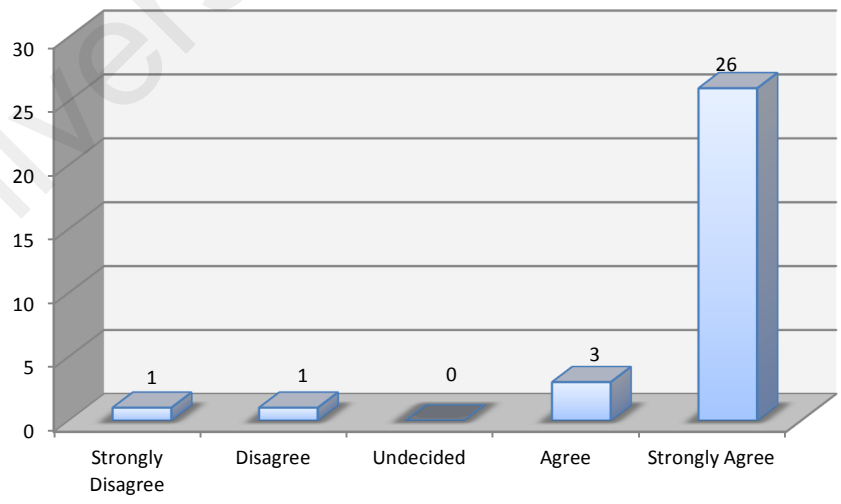


Figure 7-1: Results of using requirements prioritization patterns in the projects

Figure 7-2 shows the experts review results for the requirements prioritization patterns. As the figure shows, 29.6% of release planners noted that reducing time of activity is

the most important strength of the patterns. Also, 28.9% of release planners believed that the patterns can improve precision of the result. Other strengths specified by the release planners are "reduce/remove ambiguity", "reduce dependability to a specific person" and "create adoptability for various projects". The "un-usability", "other problems" and "inefficiency" are the most noted problems of the prioritization pattern, respectively with 2.5%, 2.5% and 1.4%. The "other problems" contains "ambiguity in choosing the exact technique" and "inadequacy of requirement prioritization technique". On the whole, 92.4% of release planners believed that the patterns have some strengths or added values and 7.6% believed that some shortcomings and problems exist and the patterns cannot be used practically.

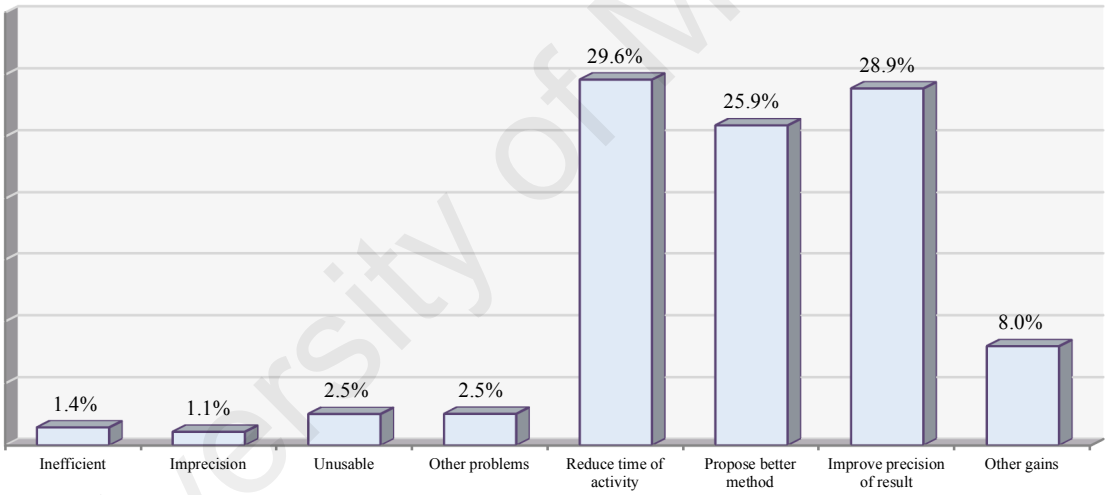


Figure 7-2: Experts' reviews for the requirements prioritization patterns

Figure 7-3 illustrates results of employing PR1 in different studied projects. The pattern was used in five projects and scored best by project managers or release planners. This demonstrates that the pattern can perfectly suit requirements prioritization in large projects.

Figure 7-4 shows the experts review results for the PR1 pattern. As the figure shows, 31.1% of release planners that noted that reducing time of activity is the biggest strength of PR1 pattern. Also, 30.0% of release planners believed that the PR1 can improve

precision of the result. The "other problems" is the most noted problem with the pattern that contains "Pattern is not truly adequate for large projects" and "The proposed technique of the pattern is not clearly defined". On the whole, 94.4% of release planners believed that the pattern has some strengths or added values and 5.6% believed that some shortcomings and problems exist and the pattern cannot be used practically.

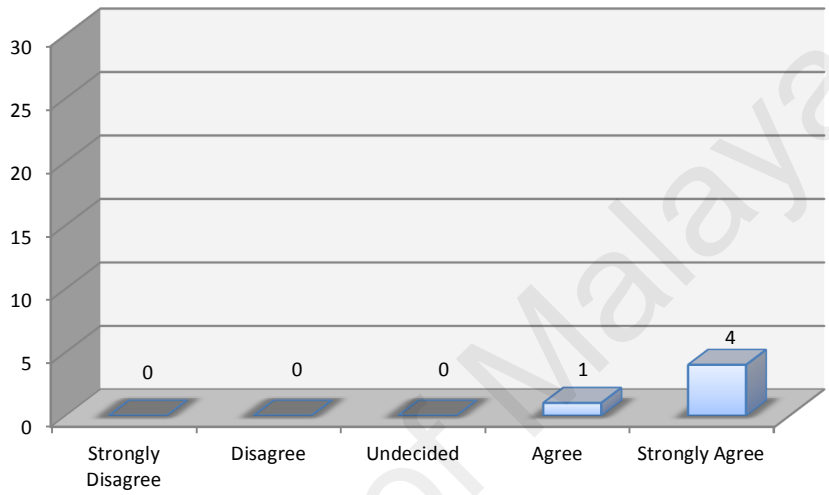


Figure 7-3: Results of using PR1 in the projects

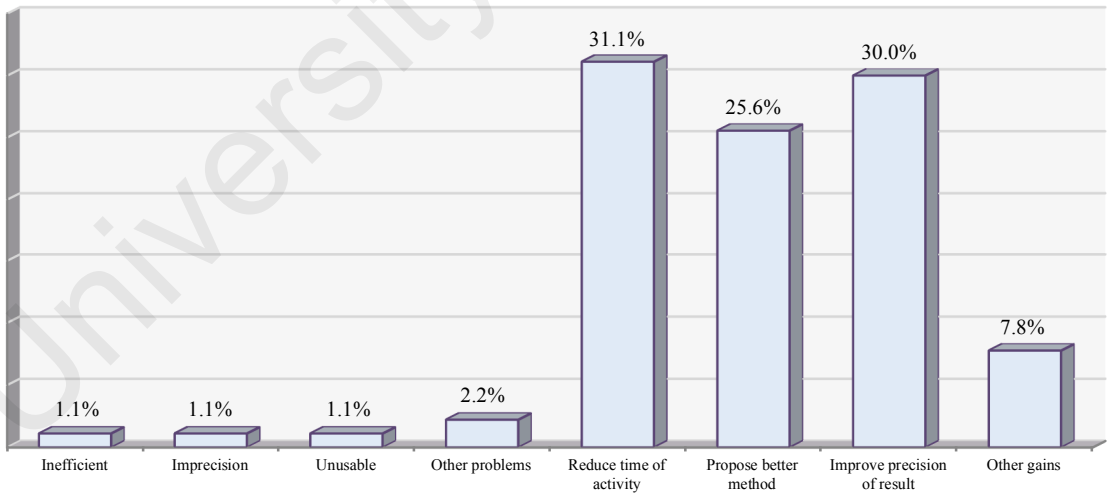


Figure 7-4: Experts' reviews for PR1 pattern

Figure 7-5 shows the results of using PR2 in different studied projects. It was employed in 12 projects, in one of which it was voted to be unsuitable because of lack of preference in scored requirements by stakeholders (in project manager's viewpoint). Hence, it was not used in that project.

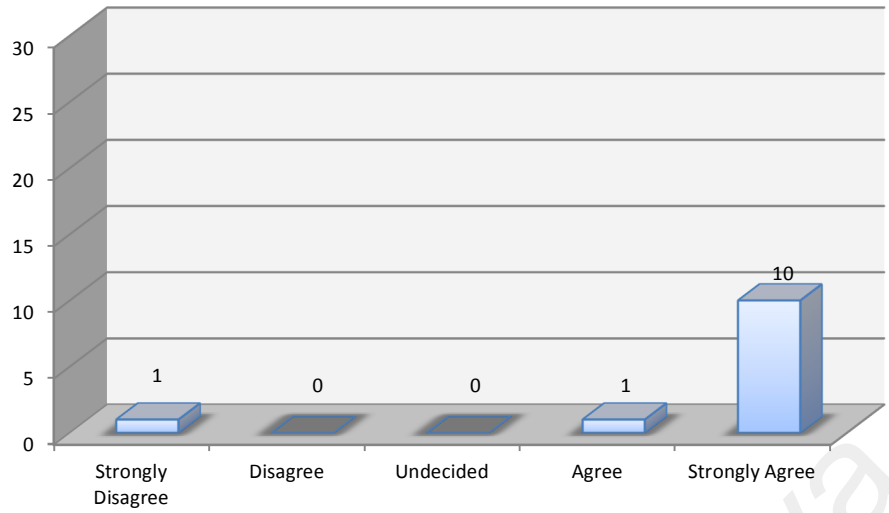


Figure 7-5: Results of using PR2 in the projects

Figure 7-6 shows the experts review results for the PR2 pattern. As the figure shows, 27.7% of release planners noted that "reduce time of activity" is the biggest strength of the pattern. Also, 26.6% of release planners believed that the pattern can propose better methods. The "un-usability" is the most noted problem of the PR2 pattern by 4.3%. In fact, 89.4% of release planners believed that the pattern has some strengths or added values and 10.6% believed that some shortcomings and problems exist and the pattern cannot be used practically.

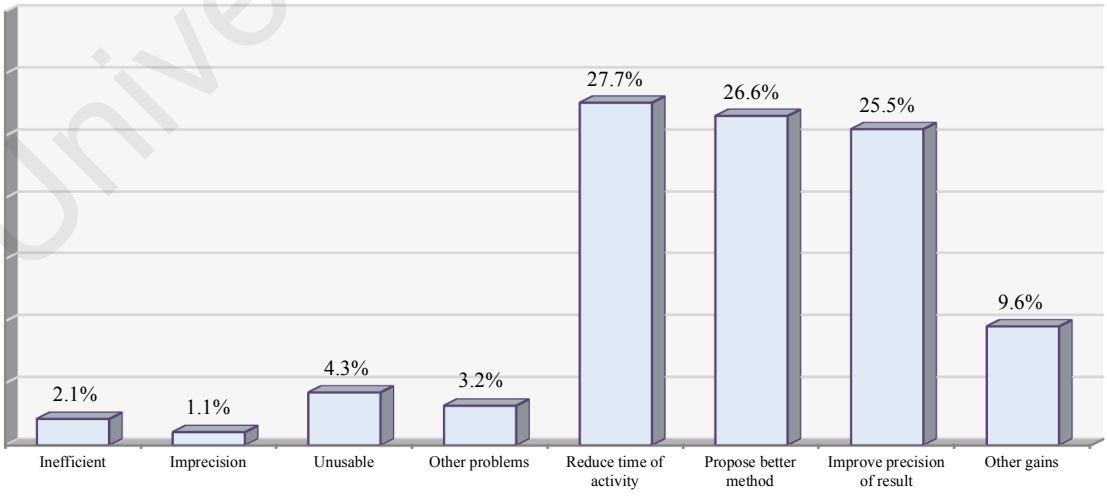


Figure 7-6: Experts' reviews for PR2 pattern

Figure 7-7 displays the results of using PR3 in different studied projects. The pattern was employed in 14 projects, in one of which it was identified as unsuitable because the project manager considered the outputs of "Numerical Assignment" method as improper. Hence, another pattern was used to prioritize requirements. A manager can consider an output as improper for many reasons, some of which are: inconsistency with teams' working procedures, longer time than expected, high costs, etc.

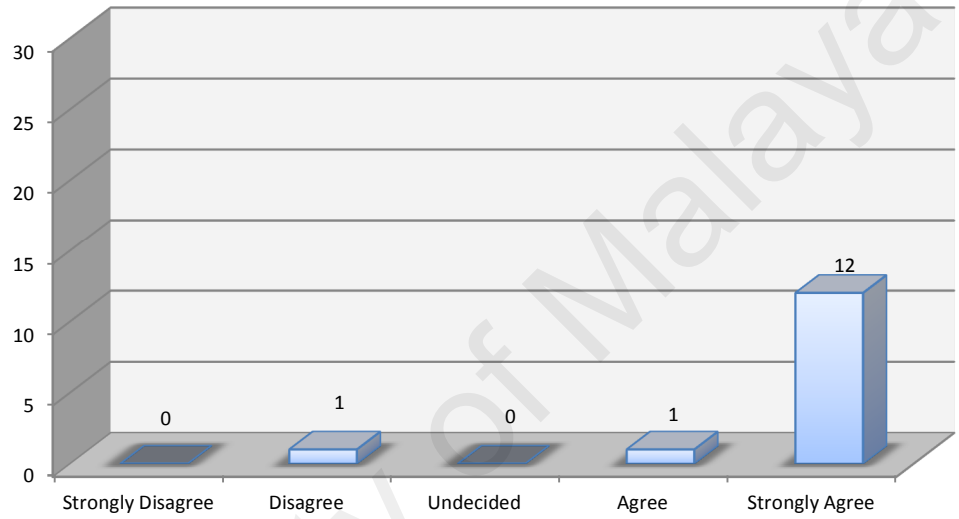


Figure 7-7: Results of using PR3 in the projects

Figure 7-8 shows the experts review results for the PR3 pattern. As the figure shows, 31.1% of release planners noted that "improve precision of result" is the biggest strength of the pattern. Also, 30.0% of release planners believed that the pattern can reduce time of activity. The "un-usability" is the most noted problem of the PR3 pattern by 2.2%. On the whole, 93.3% of release planners believed that the pattern has some strengths or added values and 6.7% believed that some shortcomings and problems exist and the pattern cannot be used practically.

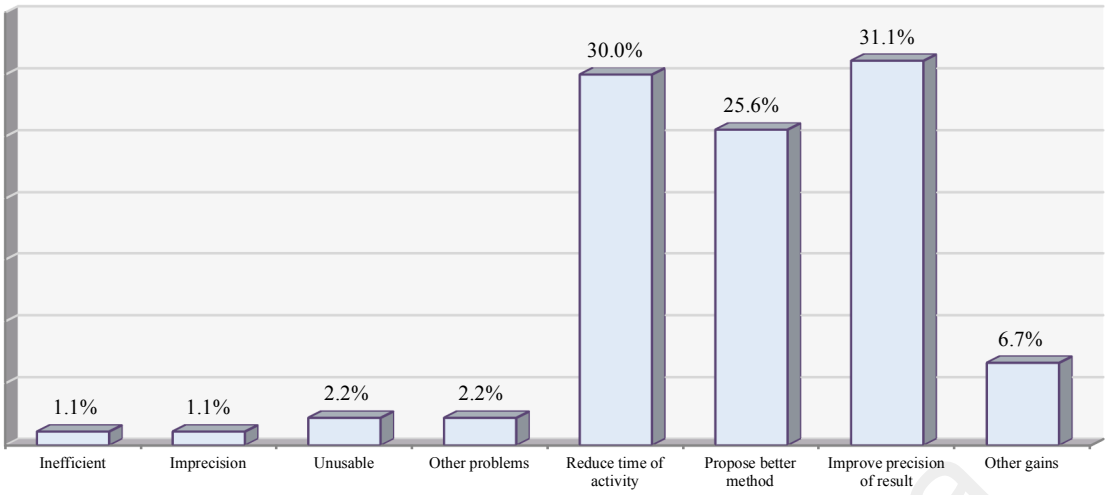


Figure 7-8: Experts' reviews for PR3 pattern

7.3. Resource estimation pattern evaluation

Figure 7-9 illustrates the results of employing resource estimation patterns. The patterns, as mentioned earlier, do not specify the method precisely and do not receive full agreement scores, similar to requirements prioritization patterns. Although teams are tried to be assisted in specifying the exact resource estimation method, most teams wish the pattern bears the method's name. Resource estimation patterns failed only in two projects, gained moderate votes in 5 projects and were successful in the rest. In other words, the patterns suited 24 projects and this indicates fitness of the proposed estimation pattern to the team and demonstrates that teams could use the patterns in practice. Therefore, parameters used to estimate resources have been efficient.

Figure 7-10 shows the experts review results for the resource estimation patterns. As the figure shows, 24.6% of release planners noted that "reduce time of activity" is the biggest strength of the patterns. 21.5% of release planners believed that the patterns can develop better result, while 12.3% of them believed that the patterns have precision problem. The precision problem means that the method proposed by the pattern does not have sufficient details and cannot be directly used in the process model step. Often, these methods are a class of methods from which the release planner should select the best one. The "other problems" is the next problem that contains "ambiguity in choosing

the exact method" and "estimation technique is not specified exactly". In fact, 71.8% of release planners believed that the patterns have some strengths or added values and 28.2% believed that some shortcomings and problems exist and the patterns cannot be used practically.

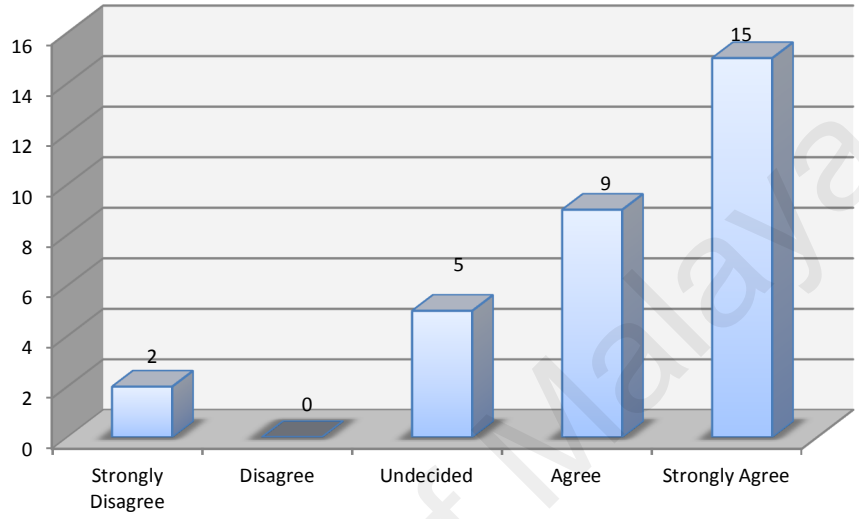


Figure 7-9: Results of using resource estimation patterns in the projects

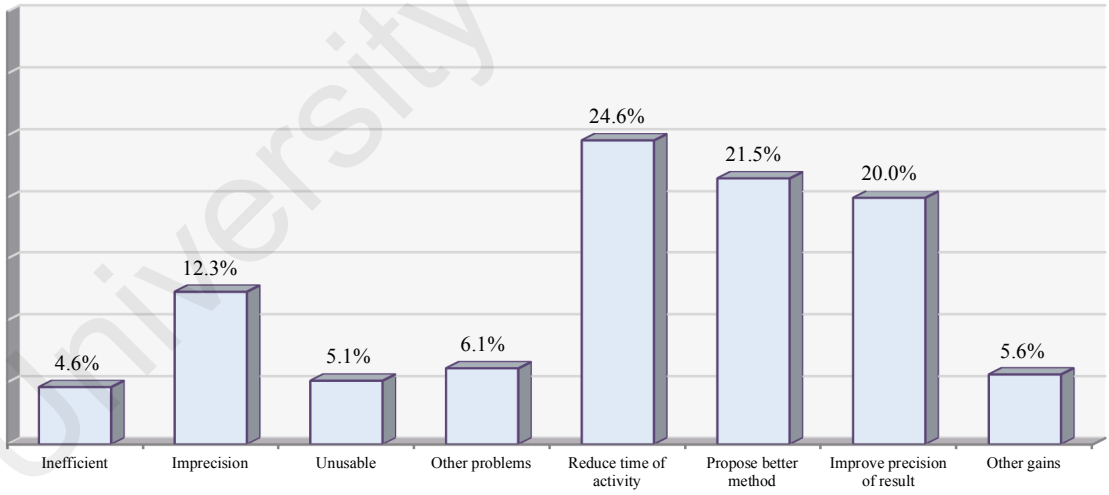


Figure 7-10: Experts' reviews for the resource estimation patterns

Figure 7-11shows the results of employing PE1 in different studied projects. The pattern was used in five projects, in three of which it was given a good score by team members and in the two remaining one it gained medium scores. This shows that the pattern is perfectly suitable for resource estimation in large projects.

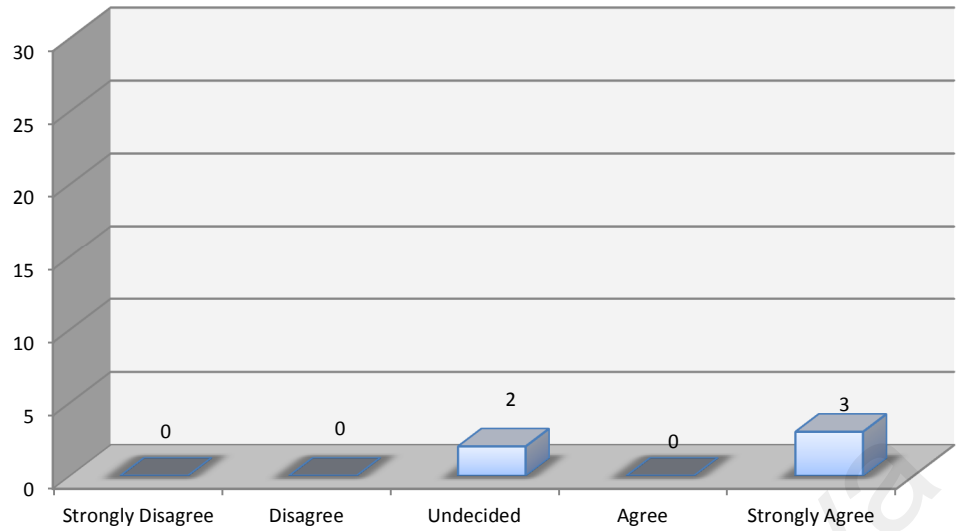


Figure 7-11: Results of employing PE1 in the projects

Figure 7-12 shows the experts review results for the PE1 pattern. As the figure shows, 25.8% of release planners noted that "reduce time of activity" is the biggest strength of the pattern, while 10.6% of them believed that the pattern has precision problem. The "other problems" is the next problem that contains "ambiguity in choosing the exact method" and "estimation technique is not specified exactly". In fact, 75.8% of release planners believed that the pattern has some strengths or added values and 24.2% believed that some shortcomings and problems exist and the pattern cannot be used practically.

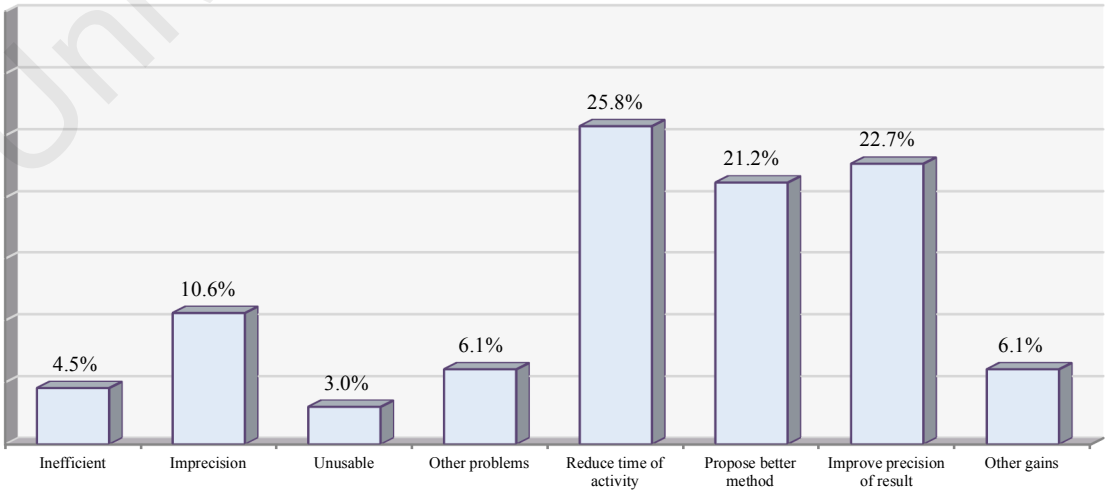


Figure 7-12: Experts' reviews for PE1 pattern

Figure 7-13 illustrates results of employing PE2 in different studied projects. The pattern was employed in 12 projects, in 11 of which it was given good scores by team members and was considered unsuitable in one project. This indicates that the pattern is totally suitable for resource estimation in unlimited-customer projects. The project manager considered the pattern unsuitable because of the inconsistency in results of resource estimation by "experts' judgment" method. Thus, another method was used to estimate resources.

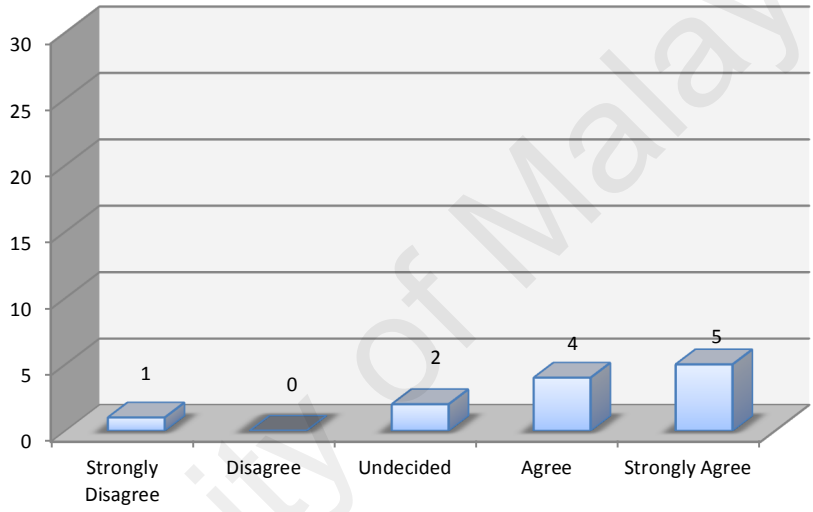


Figure 7-13: Results of employing PE2 in the projects

Figure 7-14 shows the experts review results for the PE2 pattern. As the figure shows, 23.1% of release planners noted that "reduce time of activity" is the biggest strength of the pattern, while 13.8% of them believed that the pattern has precision problem. The "other problems" is the next problem that contains "ambiguity in choosing the exact method" and "estimation technique is not specified exactly". On the whole, 66.2% of release planners believed that the pattern has some strengths or added values and 33.8% believed that some shortcomings and problems exist and the pattern cannot be used practically.

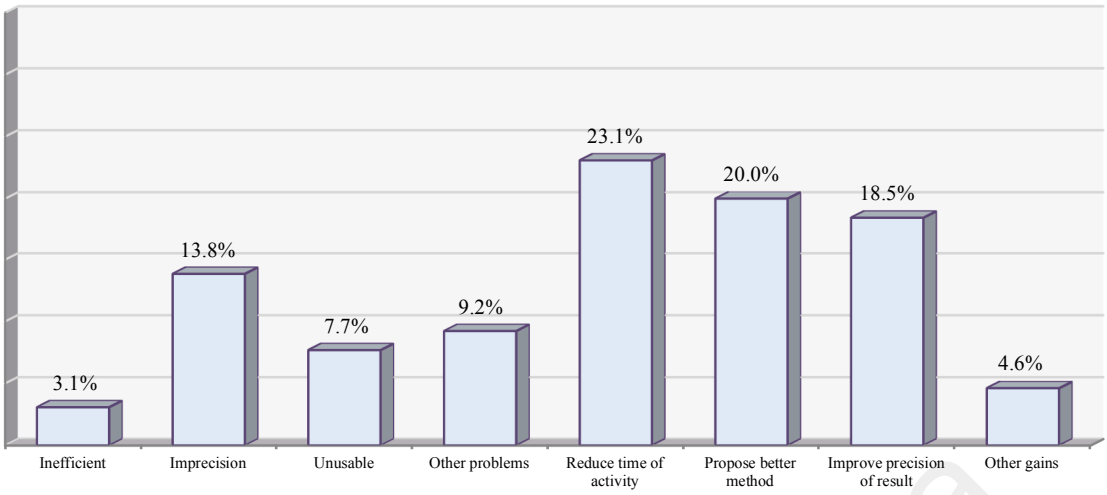


Figure 7-14: Experts' reviews for PE2 pattern

Figure 7-15 shows the results of employing PE3 in different studied projects. The pattern was employed in 14 projects, in 13 of which it was given good scores by team members. This indicates that the pattern is totally suitable for resource estimation in small projects. In the single project in which the pattern was considered unsuitable, the project manager used an identical previous estimation and hence the pattern's proposed method was not employed.

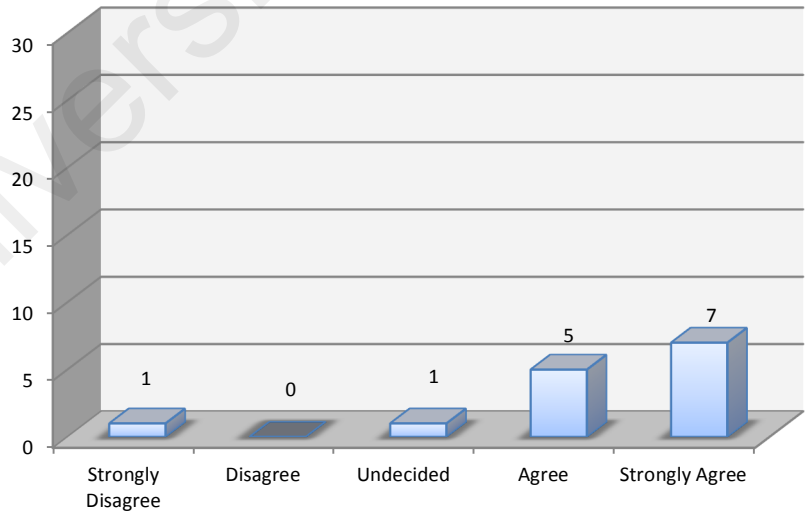


Figure 7-15: Results of employing PE3 in the projects

Figure 7-16 shows the experts review results for the PE3 pattern. As the figure shows, 25.0% of release planners noted that "reduce time of activity" is the biggest strength of the pattern, while 12.5% of them believed that the pattern has precision problem. Also,

23.4% of release planners believed that the pattern proposes better methods. On the whole, 73.4% of release planners believed that the pattern has some strengths or added values and 26.6% believed that some shortcomings and problems exist and the pattern cannot be used practically.

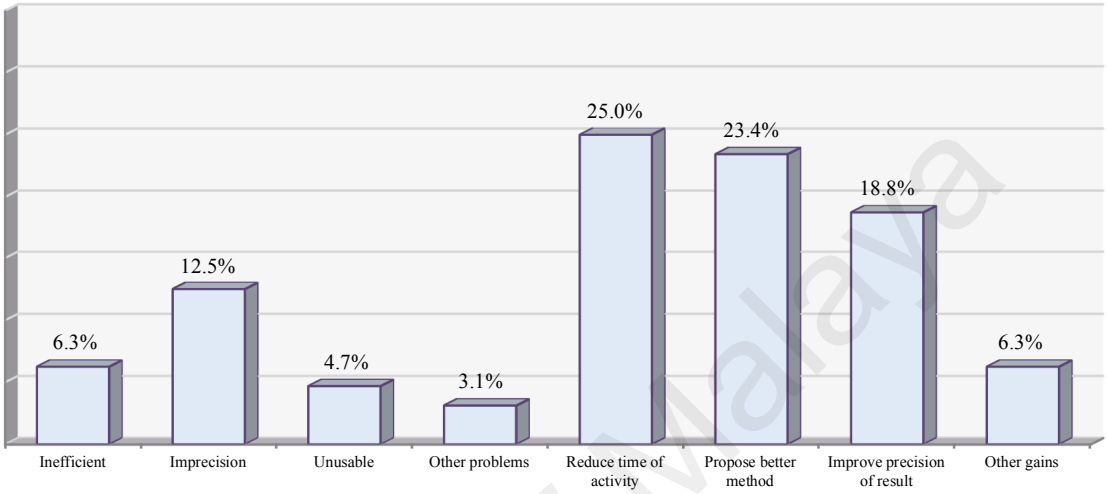


Figure 7-16: Experts' reviews for PE3 pattern

7.4. Pre-release planning pattern evaluation

Figure 7-17 illustrates the results of employing pre-release planning patterns. The patterns, as mentioned earlier, do not specify the method precisely and because of that they do not receive full agreement scores, similar to requirements prioritization patterns. Teams are tried to be assisted in specifying the exact pre-release planning method and finding the most appropriate tools. Pre-release planning patterns failed only in 2 projects, gained moderate votes in 2 other projects and were successful in the rest. In other words, the patterns suited 27 projects and this indicates fitness of PRP proposed pre-release planning pattern to the team. Therefore, parameters used to plan pre-releases have been efficient.

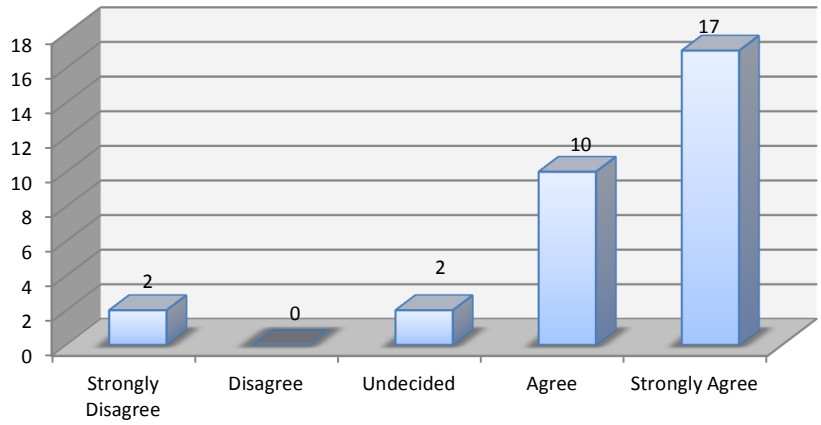


Figure 7-17: Results of using pre-release planning patterns in the projects

Figure 7-18 shows the experts review results for the pre-release planning patterns. As the figure shows, 21.4% of release planners noted that "improve precision result" is the biggest strength of the patterns, while 11.1% of them did not believe so. The precision problem means the method proposed by the pattern does not have sufficient details and cannot be directly used in the process model step. Often, these methods are a class of methods from which the release planner should select the best one. Also, 19.4% of release planners believed that the patterns reduce time of activity. On the whole, 65.6% of release planners believed that the patterns have some strengths or added values and 34.5% believed that some shortcomings and problems exist and the patterns cannot be used practically.

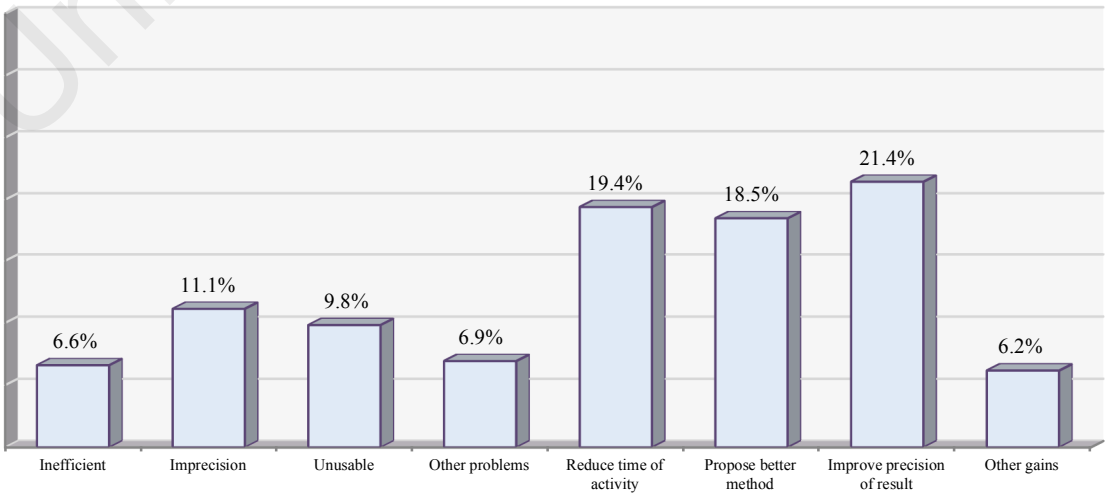


Figure 7-18: Experts' reviews for the pre-release planning patterns

Figure 7-19 shows the results of employing PP1 in different studied projects. The pattern was employed in five projects and was given proper scores by team members in all of them. This indicates appropriateness of the pattern for pre-release planning in large projects.

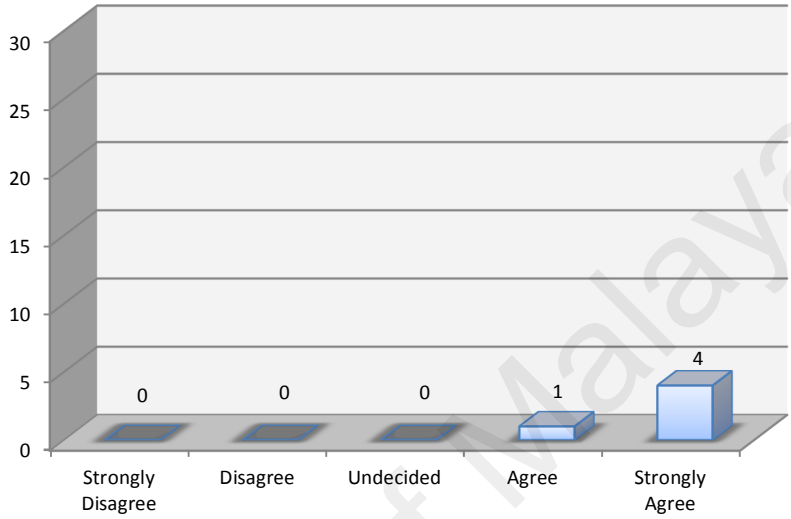


Figure 7-19: Results of employing PP1 in the projects

Figure 7-20 shows the experts review results for the PP1 pattern. As the figure shows, 23.9% of release planners noted that "improve precision result" is the biggest strength of the pattern, while 9.0% of them believed that the pattern is unusable. Also, 22.4% of release planners believed that the pattern can reduce time of planning activity. On the whole, 73.1% of release planners believed that the pattern has some strengths or added values and 26.9% believed that some shortcomings and problems exist and the pattern cannot be used practically.

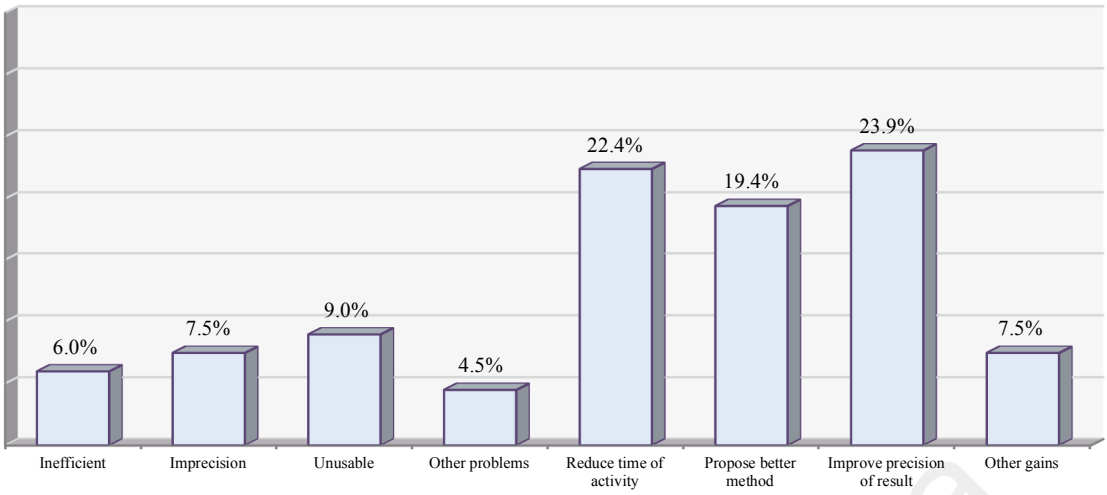


Figure 7-20: Experts' reviews for PP1 pattern

Figure 7-21 illustrates the results of employing PP2 in various studied projects. The pattern was employed in 12 projects, in 11 of which it gained good scores from team members and was considered unsuitable in one project only. This indicates that the pattern is totally suitable for pre-release planning in unlimited-customer projects. Method complexity was mentioned as the reason for inappropriateness of the pattern in the one project.

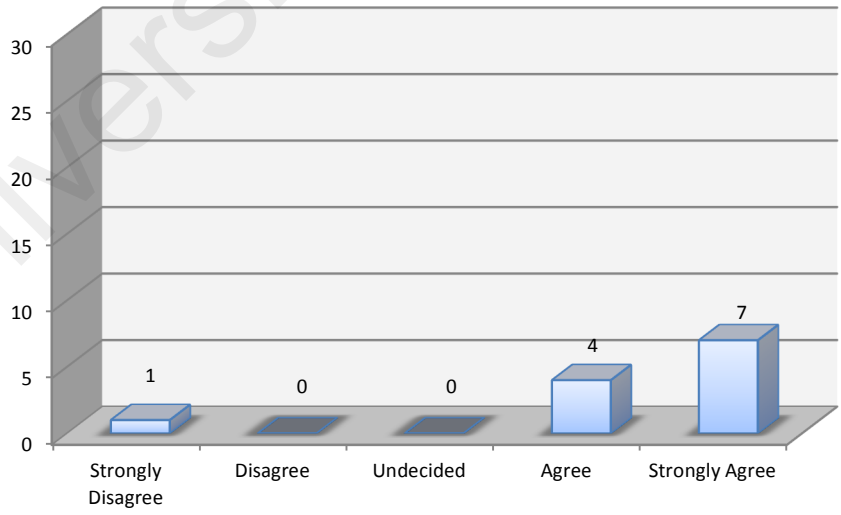


Figure 7-21: Results of employing PP2 in the projects

Figure 7-22 shows the experts review results for the PP2 pattern. As the figure shows, 24.6% of release planners noted that "improve precision result" is the biggest strength of the pattern, while 10.1% of them did not believe so. The precision problem means the

method proposed by the pattern does not have sufficient details and cannot be directly used in the process model step. Often, these methods are a class of methods from which the release planner should select the best one. Also, 20.3% of release planners believed that the pattern proposes better methods than before. On the whole, 68.1% of release planners believed that the pattern has some strengths or added values and 31.9% believed that some shortcomings and problems exist and the pattern cannot be used practically.

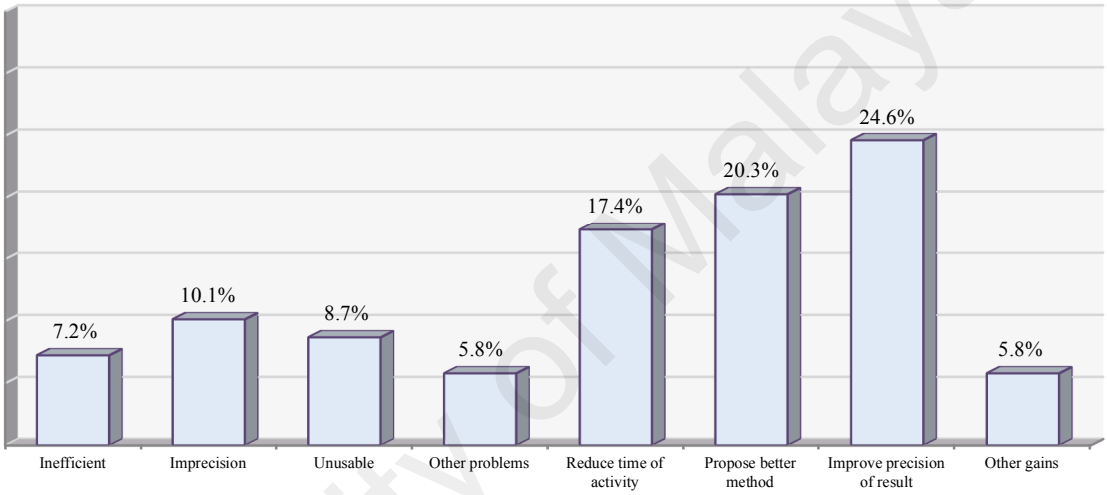


Figure 7-22: Experts' reviews for PP2 pattern

Figure 7-23 displays the results of employing PP3 in various studied projects. The pattern was employed in 14 projects, in 11 of which it was given good scores by team members and was considered unsuitable in one project only. This indicates that the pattern is totally suitable for pre-release planning in small projects. The project managers or release planners of the project in which the pattern was considered unsuitable mentioned the need to generate more plans for decision-making as the main reason for inappropriateness of the pattern (regarding nature of the project).

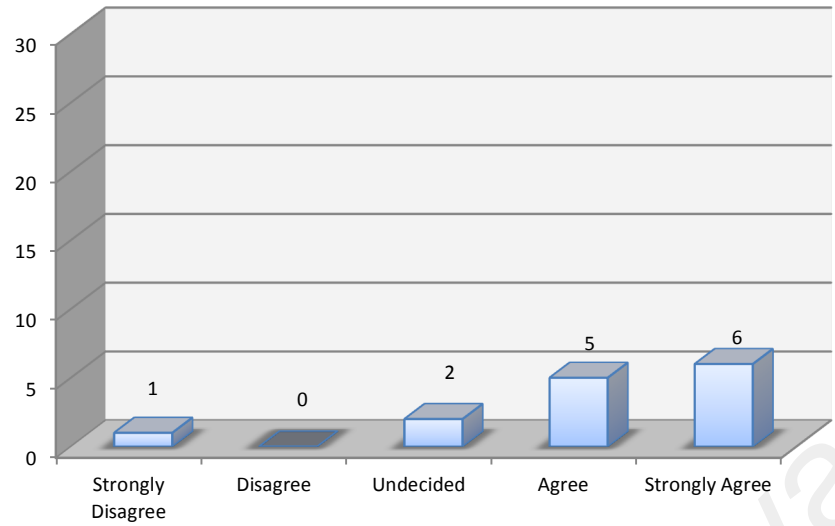


Figure 7-23: Results of employing PP3 in the projects

Figure 7-24 shows the experts review results for the PP3 pattern. As the figure shows, 18.4% of release planners noted that "reduce time of activity" is the biggest strength of the pattern, while 15.8% of them believed that the pattern has precision problem. On the whole, 55.3% of release planners believed that the pattern has some strengths or added values and 44.7% believed that some shortcomings and problems exist and the pattern cannot be used practically.

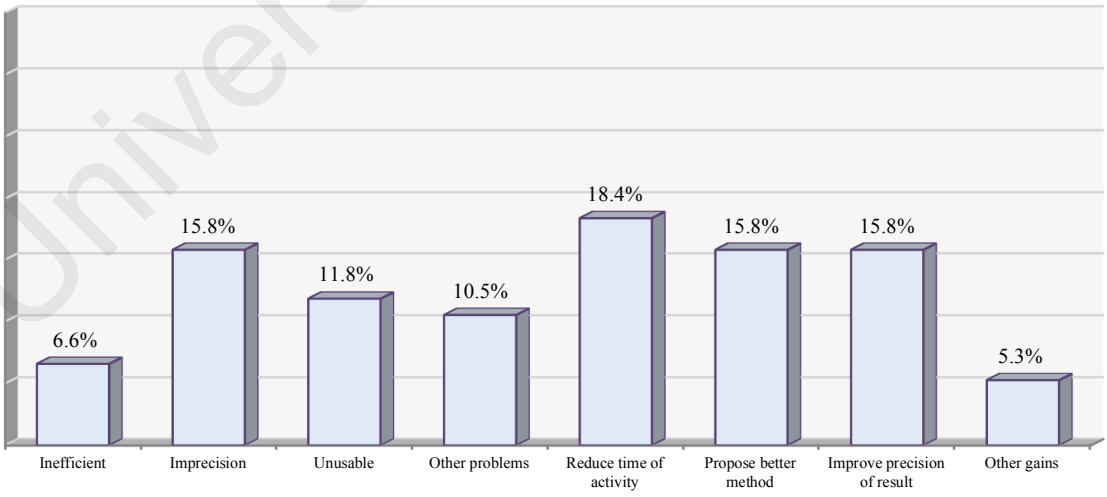


Figure 7-24: Experts' reviews for PP3 pattern

7.5. Release planning pattern evaluation

Figure 7-25 shows the results of employing release planning patterns in the studied projects. The patterns, as mentioned earlier, specify every stage of release planning. Releases planning patterns failed in two projects and were successful in the rest. In other words, the patterns suited 29 projects and this indicates fitness of PRP proposed release planning patterns to the teams. Therefore, parameters used to plan releases have been efficient. The reason for the failure of the pattern in those two projects was the proposed sub-patterns (patterns of release planning process model steps) discussed in previous sections.

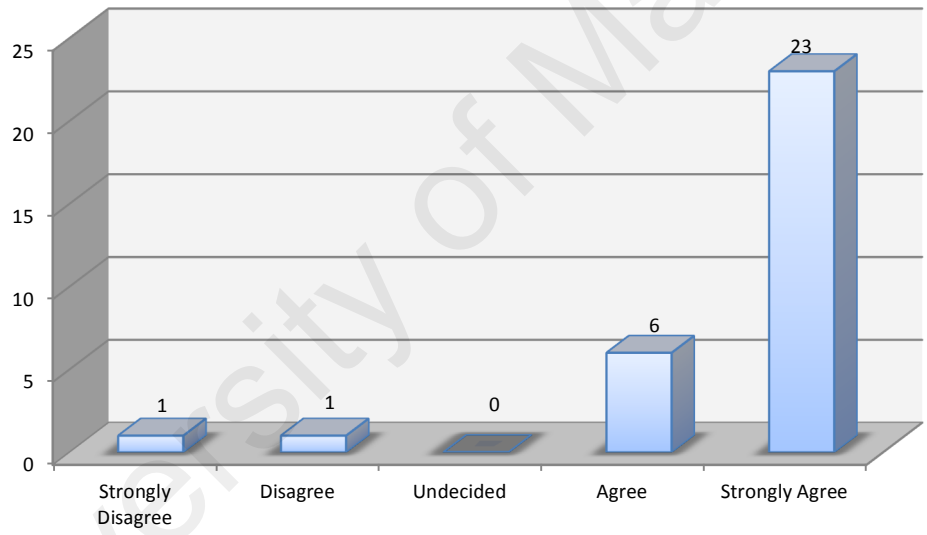


Figure 7-25: Results of using release planning patterns in the projects

Figure 7-26 shows the experts review results for the release planning patterns. As the figure shows, 25.0% of release planners noted that "propose better methods" is the biggest strength of the patterns, while 5.5% of release planners believed that the patterns are inefficient and unusable. Also, 24.9% of release planners believed that the patterns reduce time of activity. On the whole, 78.7% of release planners believed that the patterns have some strengths or added values and 21.3% believed that some shortcomings and problems exist and the patterns cannot be used practically.

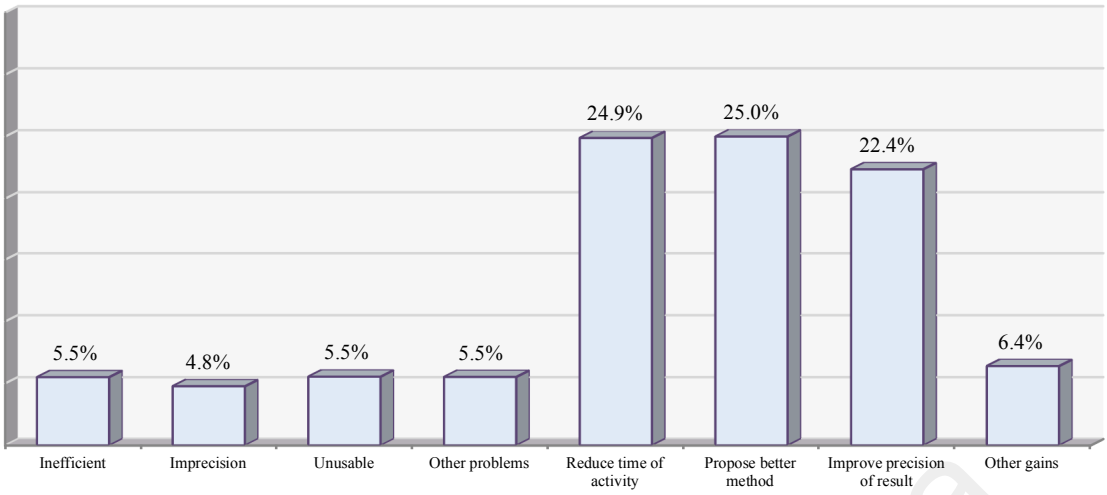


Figure 7-26: Experts' reviews for the release planning patterns

Figure 7-27 displays the results of employing P1 in different studied projects. The pattern was employed in five projects and gained proper scores from team members in all of them. This indicates the appropriateness of the pattern for release planning in large projects.

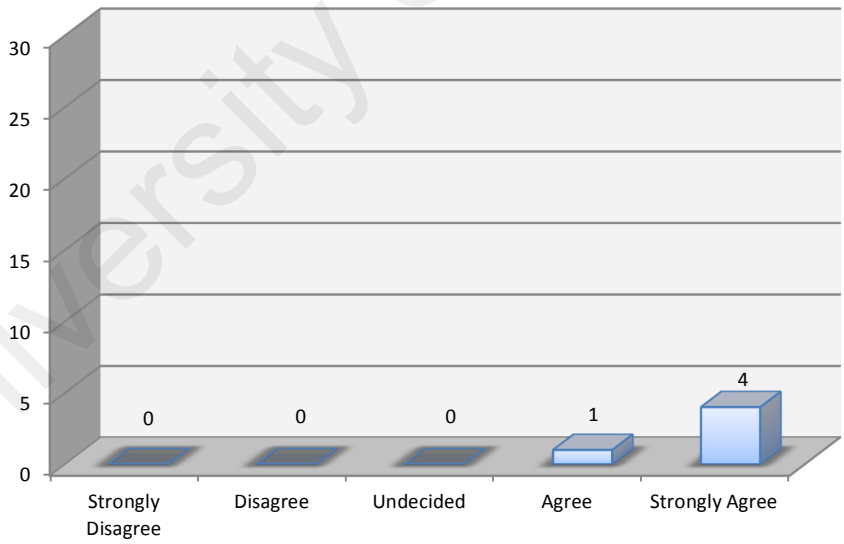


Figure 7-27: Results of using P1 in the projects

Figure 7-28 shows the experts review results for P1pattern. As the figure shows, 25.0% of release planners noted that "propose better method" and the "reduce time of activity" are the biggest strengths of the pattern, while 6.6% of them believed that the pattern unusable. On the whole, 78.9% of release planners believed that the pattern has some

strengths or added values and 21.1% believed that some shortcomings and problems exist and the pattern cannot be used practically.

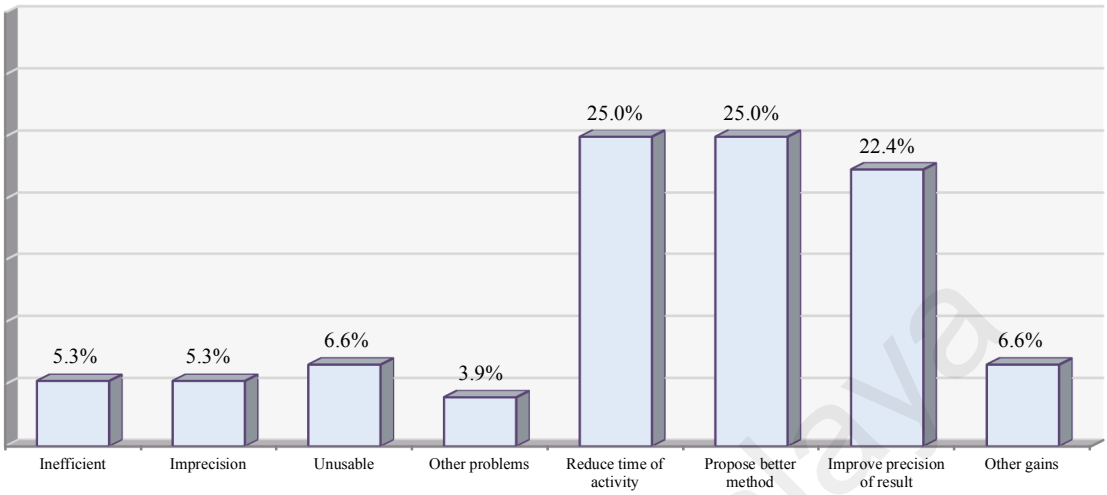


Figure 7-28: Experts' reviews for P1 pattern

Figure 7-29 shows the results of employing P2 in different studied projects. The pattern was employed in 12 projects, in 11 of which it was given good scores by team members and was considered unsuitable in one project. This indicates that the pattern is totally suitable for release planning in unlimited-customer projects. Problems with the proposed sub-patterns were mentioned as the reasons why the pattern was called inappropriate in that one project.

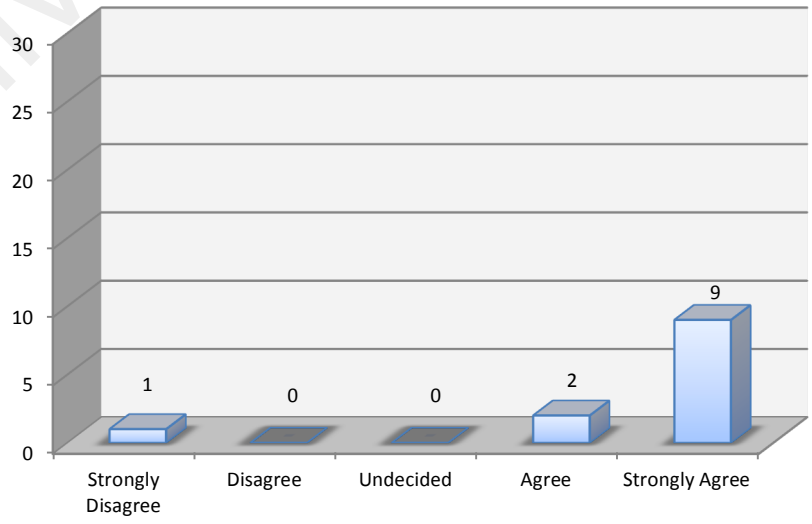


Figure 7-29: Results of using P2 in the projects

Figure 7-30 shows the experts review results for P2 pattern. As the figure shows, 25.0% of release planners noted that "reduce time of activity" is the biggest strength of the pattern, while 7.1% of them believed that the pattern has inefficiency problem. The "other problems" is the next problem with P2 pattern that contains "ambiguity in choosing the exact method" and "techniques are not specified exactly". On the whole, 76.2% of release planners believed that the pattern has some strengths or added values and 23.8% believed that some shortcomings and problems exist and the pattern cannot be used practically.

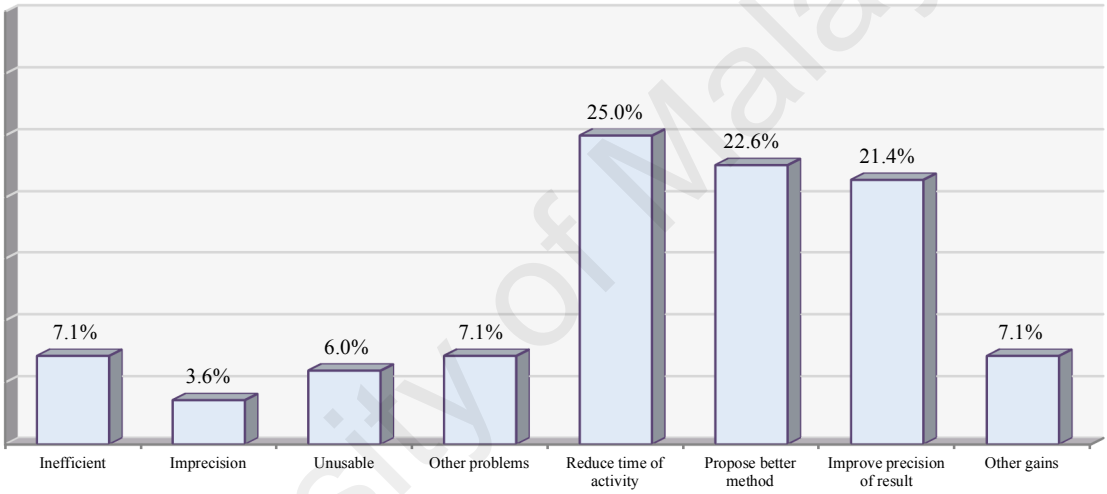


Figure 7-30: Experts' reviews for P2 pattern

Figure 7-31 illustrates the results of employing P3 in different studied projects. The pattern was employed in 14 projects, in 13 of which it was given good scores by team members and was considered unsuitable in one project. This indicates that the pattern is totally suitable for release planning in small projects. Problems with proposed sub-patterns were mentioned as the reasons why the pattern was thought inappropriate in that one project.

Figure 7-32 shows the experts review results for P3 pattern. As the figure shows, 27.4% of release planners noted that "propose better method" is the biggest strength of the pattern, while 5.5% of them believed that the pattern is not precise. They also

mentioned "other problems" that contains "ambiguity in choosing the exact method" and "techniques are not specified exactly". On the whole, 80.8% of release planners believed that the pattern has some strengths or added values and 19.2% believed that some shortcomings and problems exist and the pattern cannot be used practically.

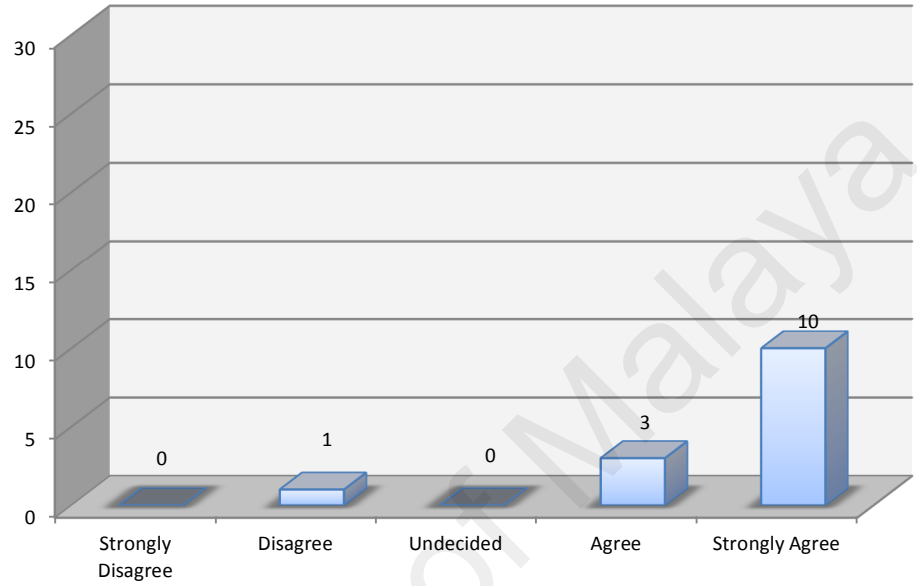


Figure 7-31: Results of employing P3 in the projects

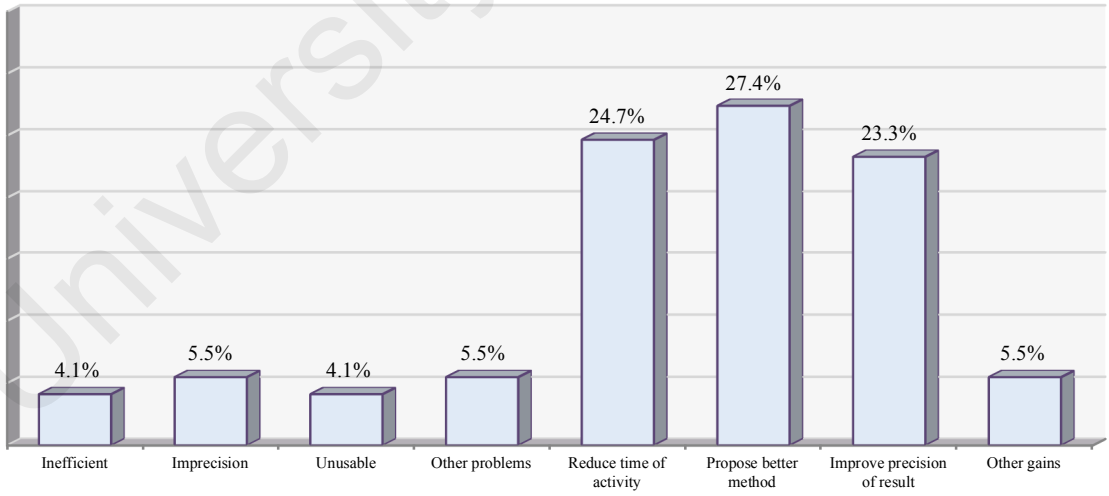


Figure 7-32: Experts' reviews for P3 pattern

7.6. Overall evaluation results

Overall result of using the pattern in the studied projects is presented in Figure 7-33. As the figure shows, 65.3% of release planners/project managers who used the pattern in their projects empirically strongly agreed to use the pattern in their projects, and in more than 87% of cases the pattern gained acceptance from release planners/project managers.

Figure 7-34 shows the experts review results for overall patterns of release planning. As the figure shows, 24.6% of release planners believed that the patterns can "reduce time of activity", while 7.3% of them believed that the patterns are not precise. In fact, 77.1% of release planners believed that the patterns have some strengths or added values and 22.9% believed that some shortcomings and problems exist and the patterns cannot be used practically.

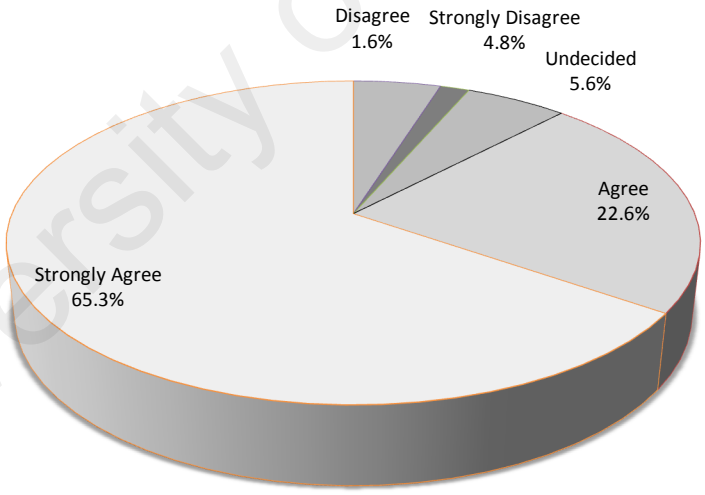


Figure 7-33: Results of employing patterns in the projects

Figure 7-35 shows the experts review results for pattern-based release planning methodology. As the figure shows, 29.8% of release planners believed that "Improve precision of result" is the most important gain of the methodology, while 3.2% of them believed that the methodology has precision problem. Improving decision making and reducing time of release planning activity are the next major benefits of the methodology specified by the release planners of case study projects. Generally, 90.4%

of release planners believed that the methodology has some gains or added values and 9.6% believed that some shortcomings and problems exist and the methodology cannot be used practically.

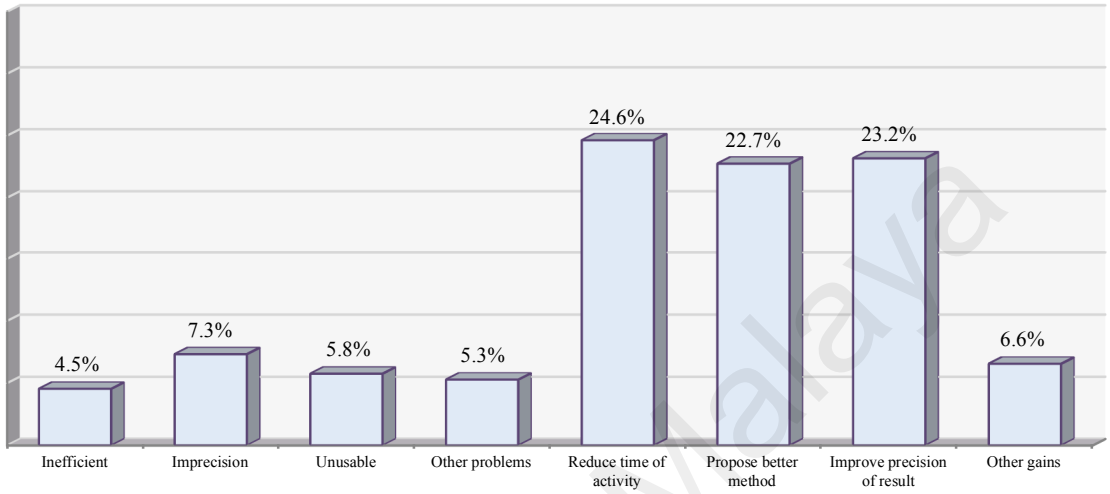


Figure 7-34: Experts' reviews for the patterns

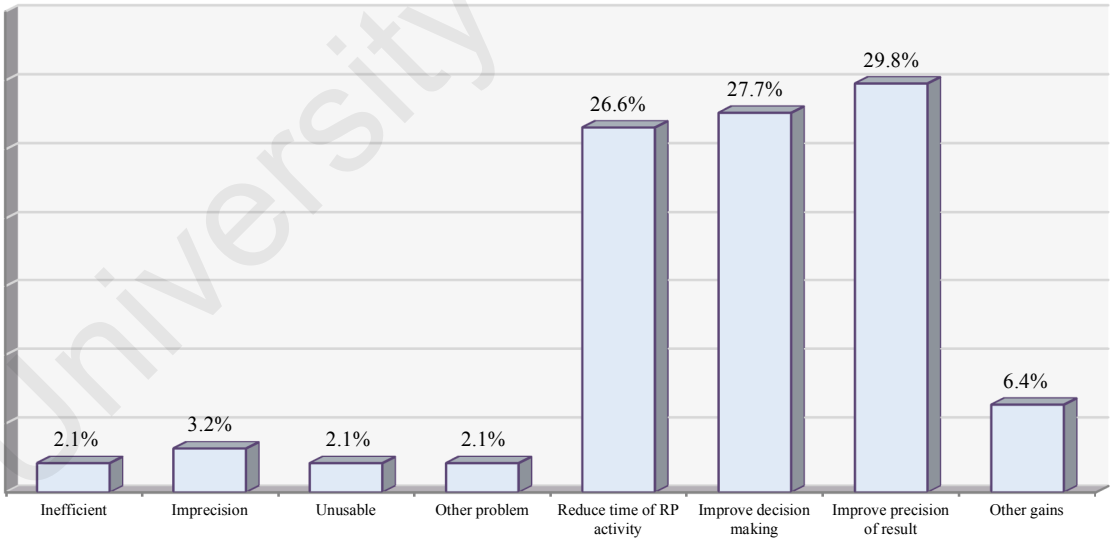


Figure 7-35: Experts' reviews for the methodology

7.7. Summary of evolution results

In the studied projects, a variety of patterns are employed to perform release planning. The results of the empirical case studies and the questionnaires evaluation are presented in this section. Based on empirical case studies performed, the pattern-based release planning methodology has been recognized as an efficient methodology to achieve different tasks of release planning process model. Making use of this methodology in most cases leads to improvement in the release planning and the resulted release plans. Moreover, the methodology can be applied in different projects and can provide teams with project-suited results since it makes use of parameters based upon projects and development organization specifications.

CHAPTER 8: RESULTS AND FUTURE WORK

8.1. Achievement objectives

As it was observed before, pattern-base methodology in most cases has attempted to improve the new release with maximum optimization. This is exactly due to the nature of this methodology that suggests solutions based on the project characteristics. The solutions are often exclusive to a particular project and may not be the optimum in a different project. In this method, all the parameters and characteristics of the project, which may be ignored in other methodologies, are taken into consideration and the solution is proposed based on them.

If this does not apply to some cases, it is most probably because all the parameters of a project have not been available or released at all.

On the whole, using this methodology, all the parameters even those vague and unclear or sometimes accidental ones that are understood and examined by project members only will be identified and implemented in the new release. In current methodologies, most of these parameters are not defined and are usually disregarded because of the inflexibility of the methods.

8.2. Research findings and contribution

This thesis verifies and validates the feasibility of patterns in release planning, and proposes a methodology based on the pattern. The contributions of this thesis are:

- Developing a process model for release planning contains four steps that cover common tasks in current applied release planning methods. These steps focus on requirement prioritization, resource estimation, release pre-planning and trade-off analysis that are performed in the same order. The process model of release

planning helps to categorize and break down release planning problem to smaller activities that have known solutions.

- Customizing every step of the process model using parameters and their instances that are extracted from various researches. This customization helps adjust the release planning method to various projects. Each step of general release planning process can be adjusted by the project specifications. It makes the method highly adoptable to fulfil project's needs. Also, mapping project specifications to the methods that can be used, which is one of the achievements in this research, helps project managers to choose the best method of release planning based on the project specifications.

- Developing the concept of release planning pattern to customize the process model in order to facilitate achieving the desired method in every step. Release planning pattern defined and described in this research can enhance customization and speed the release planning process by using previous experiences in designing patterns. This concept is extended to each step of release planning and the patterns for release planning steps are also developed.

Pattern-based release planning methodology is generated by planning patterns based on characteristics and parameters effective on every stage of the process model. Running this methodology leads to a series of results and achievements which are described below.

- **Improvement in quality of release plans**

  Pattern-based release planning methodology enables making use of previous successful experiences in release planning and enhances the quality of the developed plans. Unlike other release planning methodologies which try to suggest a comprehensive and inflexible strategy for planning, the methodology presented in

this research uses best of past experiences to propose a guideline to determine the best suited method for different steps. Moreover, since selecting the best method to accomplish release planning fits the considered project, the resulted plans will consequently improve significantly.

- **Reduction in time spent on release planning**

  Selecting methods in release planning and its various steps is a time consuming process in most companies and, as a result, they tend to use simple methods or tools which may even be improper for the company or the project. In fact, identifying different parameters of the steps such as resource estimation or requirements prioritization require time and expertise, but these resources are not always available in projects (especially small ones). Pattern-based release planning methodology provides (simply and rapidly) the release planner or project manager with successful experiences of other projects considering a set of predetermined parameters and, then, he or she can employ different parameters to determine methods that suit his or her project better.

- **Release planning relative to project characteristics**

  No doubt, one of the most important issues in release planning is to select a method which best suits the project. Most release planning methodologies solely try to present a planning method regardless of its fitness to the project. Besides, in such methods the project manager or release planner has to spend much time or rely on his/her experience to select a method relative to project characteristics. However, this is usually challenged in different levels and the fitness is not ensured. But, pattern-based release planning methodology uses a project's specific features and the best of past experiences to present a method suiting the project. Thus, success rate of projects accomplished through a certain pattern ensures feasibility of the proposed method.

- **Making use of best experience in release planning**

  Transferring successful experiences is a main challenge in software development and release planning and this is inevitable considering the projects' diversity. Though, using pattern-based release planning and developing new release plans, it is possible to transfer experiences and knowledge of successful projects. Furthermore, this is also feasible about unsuccessful projects in developing anti-patterns. Therefore, an increase in the number of release planning patterns is expected to lead to more accomplishments in this field.

- **Omission of decisions made without technical support**

  Release planning is considered as an important but challenging task by most companies, especially small ones, since releases are of great significance to them and selecting a wrong release results in waste of time and delays in projects, something that small projects are highly vulnerable in. Consuming too much time for selection, impractical requirements or capabilities which are highly dependent on a wide range of other requirements make it important to choose a method that best suits the project. Pattern-based release planning helps select a planning method that fully fits the software developer company and is supported by past successful experiences. It also aids the project manager or release planner in achieving a better understanding of release planning and its different methods in the form of release planning patterns.

- **Recording successful experiences in release planning**

  By introducing some patterns in this research, the pattern-based release planning methodology makes it possible to present new patterns for planning. Indeed, benefiting from its successful experiences every company can generate and use a set of new patterns for release planning. In addition, the methodology enables determining new parameters for each stage of release planning, and companies may

employ their experiences to develop the required set of effective parameters on every stage and generate new patterns to record their experiences.

8.3. Research executive constraints

The main executive limitation of this research was the need to record successful experiences of release planning. Most companies plan their releases using simple methods and lack certain records of how the planning is performed. Therefore, no executive history was available for a project and a wide range of case studies were selected and performed to cover this deficiency in the present research. Moreover, many companies are only interested in executive outcomes of release planning and do not think much of selecting a correct method. This usually originates from stresses of developing a new release (due to customers' demands and expectations). In the case studies, it was observed that in some projects the manager abandoned the proposed method because of the pressures and used another method. This is inevitable.

8.4. Future works

Pattern-based release planning is presented for the first time in the field of release planning, and developing new release planner patterns and anti-patterns can be considered as one of the most important works of future. It is important that new patterns and anti-patterns result from implementing a set of projects and be experienced to a considerable extent. Besides, effective parameters on release planning steps can be developed, especially in resource estimation and pre-release planning steps, to make their patterns more precise and efficient.

REFERENCES

Aasem, M., Ramzan, M., & Jaffar, A. (2010, 14-16 June 2010). *Analysis and optimization of software requirements prioritization techniques.* Paper presented at the International Conference on Information and Emerging Technologies (ICIET).

Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1983). *Data Structures and Algorithms* (1 ed.): Addison-Wesley.

Akker, M. v. d., Brinkkemper, S., Diepen, G., & Versendaal, J. (2008). Software product release planning through optimization and what-if analysis. *Information and Software Technology, 50*(1-2), 101–111.

Al-Emran, A., Jadallah, A., Paikari, E., Pfahl, D., & Ruhe, G. (2010, 8-9 July 2010). *Application of Re-estimation in Re-planning of Software Product Releases.* Paper presented at the International Conference on New Modeling Concepts for Today's Software Processes, Paderborn, Germany.

Al-Emran, A., Kapur, P., Pfahl, D., & Ruhe, G. (2010). Studying the impact of uncertainty in operational release planning – An integrated method and its initial evaluation. *Information and Software Technology, 52*(4), 446-461.

Al-Emran, A., Pfahl, D., & Ruhe, G. (2010, September 27 - October 1, 2010). *Decision Support for Product Release Planning based on Robustness Analysis.* Paper presented at the 18th IEEE International Requirements Engineering Conference (RE), Sydney, Australia.

AlBourae, T. A. (2007). *Re-planning of Software Product Releases.* (Master Science Master Thesis), University of Calgary, Canada.

AlBourae, T. A., Ruhe, G., & Moussavi, M. (2006, 12 September 2006). *Lightweight Replanning of Software Product Releases.* Paper presented at the International Workshop on Software Product Management (IWSPM'06), Minneapolis, MN, USA.

Alexander, C., Ishikawa, S., & Sara Ishikawa, C. A. M. S. (1977). *A Pattern Language: Towns, Buildings, Construction*: Oxford University Press.

Amandeep, A., Ruhe, G., & Stanford, M. (2004, 5-8 April 2004). *Intelligent Support for Software Release Planning.* Paper presented at the 5th International Conference on Product Focused Software Process Improvement (Profes), Kansai Science City, Japan.

Anton, A. I. (2003). Successful software projects need requirements planning. *Software, IEEE, 20*(3), 44, 46. doi: 10.1109/MS.2003.1196319

Bagnall, A. J., Rayward-Smith, V. J., & Whittley, I. M. (2001). The Next Release Problem. *Information and Software Technology, 43*(14), 883-890.

Beck, K. (1999). *Extreme Programming Explained: Embrace Change*: Addison-Wesley Professional.

Beck, K. (2001). *Extreme Programming Explained* (2 ed.): Addison-Wesley.

Beck, K., & W., C. (1987). Using Pattern Languages for Object-Oriented Programs: Tektronix, Inc.

Berander, P., & Andrews, A. (2005). Requirements Prioritization. In A. Aurum & C. Wohlin (Eds.), *Engineering and Managing Software Requirements* (pp. 69-94): Springer Berlin Heidelberg.

Berander, P., Khan, K. A., & Lehtola, L. (2006). *Towards a Research Framework on Requirements Prioritization.* Paper presented at the Sixth Conference on Software Engineering Research and Practice in Sweden (SERPS'06), Umeå University, Sweden.

Boehm, B. (1981). *Software Engineering Economics*: Prentice Hall.

Briand, L. C., & Wieczorek, I. (2002). Resource Estimation in Software Engineering. In J. J. Marcinak (Ed.), *Encyclopedia of Software Engineering.* New York: John Wiley & Sons.

Brown, W. J. (1998). *AntiPatterns: refactoring software, architectures, and projects in crisis*: Wiley.

Carlshamre, P. (2002). Release Planning in Market-Driven Software Product Development: Provoking an Understanding. *Requirements Engineering, 7*(3), 139-151.

Chatzipetrou, P., Angelis, L., Rovegard, P., & Wohlin, C. (2010, 1-3 Sept. 2010). *Prioritization of Issues and Requirements by Cumulative Voting: A Compositional Data Analysis Framework.* Paper presented at the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA).

Chatzoglou, P. D. (1997). Factors affecting completion of the requirements capture stage of projects with different characteristics. *Information and Software Technology, 39*(9), 627-640. doi: 10.1016/S0950-5849(97)00020-7

Colares, F., Souza, J., Carmo, R., Padua, C., & Mateus, G. R. (2009, 5-9 October 2009). *A New Approach to the Software Release Planning*. Paper presented at the XXIII Brazilian Symposium on Software Engineering, Fortaleza-CE, Brazil.

Coplien, J. O. (1992). *Advanced C++ programming styles and idioms*: Addison-Wesley Pub. Co.

Davis, A. M. (1993). *Software requirements: Objects, functions, and states* (2 Ed.): Prentice Hall.

Denne, M., & Cleland-Huang, J. (2003). *Software by Numbers: Low-Risk, High-Return Development*: Prentice Hall.

Denne, M., & Cleland-Huang, J. (2004). The incremental funding method: data-driven software development. *Software, IEEE, 21*(3), 39-47. doi: 10.1109/MS.2004.1293071

Drapeau, M., & Oudi, S. (2007). Release Management: Where to Start? , 2013

Du, G. M., J. ; Ruhe, G. (2006, 12-14 June 2006). *Ad hoc versus Systematic Planning of Software Releases - A Three-Staged Experiment*. Paper presented at the 7th International Conference on Product Focused Software Process Improvement (PROFES), Amsterdam, Netherlands.

Durillo, J. J., Zhang, Y., Alba, E., Harman, M., & Nebro, A. J. (2011). A study of the bi-objective next release problem. *Empirical Software Engineering, 16*(1), 29-60.

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology, 50*(9–10), 833-859. doi: 10.1016/j.infsof.2008.01.006

Dyba, T., Prikladnicki, R., Ronkko, K., Seaman, C., & Sillito, J. (2011). Qualitative research in software engineering. *Empirical Softw. Engg., 16*(4), 425-429. doi: 10.1007/s10664-011-9163-y

Fayad, M. E., Laitinen, M., & Ward, R. P. (2000). Thinking objectively: software engineering in the small. *Commun. ACM, 43*(3), 115-118. doi: 10.1145/330534.330555

Felderer, M., Beer, A., Ho, J., & Ruhe, G. (2014). *Industrial evaluation of the impact of quality-driven release planning*. Paper presented at the Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Torino, Italy.

Firesmith, D. (2004). Prioritizing requirements. *Journal of Object Technology, 3*(8), 35-47.

Freitas, F. G., Coutinho, D. P., & Souza, J. T. (2011). Software Next Release Planning Approach through Exact Optimization. *International Journal of Computer Applications, 22*(8), 1-8.

Galorath, D. D., & Evans, M. W. (2006). *Software sizing, estimation, and risk management: when performance is measured performance improves* (1 ed.): Auerbach Publications.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*: Addison-Wesley.

Greer, D., & Ruhe, G. (2004). Software Release Planning: An Evolutionary and Iterative Approach. *Information and Software Technology, 46*(4), 243-253.

Hall, T., Beecham, S., & Rainer, A. (2002). Requirements problems in twelve software companies: an empirical analysis. *Software, IEE Proceedings -, 149*(5), 153-160. doi: 10.1049/ip-sen:20020694

Hatton, S. (2007). *Early prioritisation of goals*. Paper presented at the Proceedings of the 2007 conference on Advances in conceptual modeling: foundations and applications, Auckland, New Zealand.

Hatton, S. (2008). *Choosing the Right Prioritisation Method*. Paper presented at the Proceedings of the 19th Australian Conference on Software Engineering.

Herrmann, A., & Paech, B. (2008). Practical Challenges of Requirements Prioritization Based on Risk Estimation: Result of Two Student Experiments. Germany: Software Engineering Group, University of Heidelberg.

Ho-Won, J. (1998). Optimizing value and cost in requirements analysis. *Software, IEEE, 15*(4), 74-78. doi: 10.1109/52.687950

Ho, J., & Ruhe, G. (2013, 20-20 May 2013). *Releasing sooner or later: An optimization approach and its case study evaluation*. Paper presented at the Release Engineering (RELENG), 2013 1st International Workshop on.

Ho, J., Shahnewaz, S., & Ruhe, G. (2014). *A Prototype Tool Supporting When-to-release Decisions in Iterative Development.* Paper presented at the Proceedings of the Second International Workshop on Release Engineering.

Hoek, A. V. D., Hall, R. S., Heimbigner, D., & Wolf, A. L. (1997). *Software Release Management.* Paper presented at the 6th European Software Engineering Conference, Berlin.

Hoek, A. V. D., & Wolf, A. L. (2003). Software release management for component-based software. *Software—Practice & Experience, 33*(1), 77 - 98.

Hoepfl, M. C. (1997). Choosing Qualitative Research: A Primer for Technology Education Researchers. *Journal of Technology Education, 9*(1), 47-63.

Iqbal, M. A., Zaidi, A. M., & Murtaza, S. (2010, 9-10 Feb. 2010). *A New Requirement Prioritization Model for Market Driven Products Using Analytical Hierarchical Process.* Paper presented at the International Conference on Data Storage and Data Engineering (DSDE).

Jadallah, A., Al-Emran, A., Moussavi, M., & Ruhe, G. (2009). The How? When? and What? for the Process of Re-planning for Product Releases. In Q. Wang, V. Garousi, R. Madachy & D. Pfahl (Eds.), *Trustworthy Software Development Processes* (pp. 24-37). Vancouver, Canada: Springer Berlin Heidelberg.

Jadallah, A., Galster, M., Moussavi, M., & Ruhe, G. (2009, 20-26 September 2009). *Balancing Value and Modifiability when Planning for the Next Release.* Paper presented at the International Conference on Software Maintenance (ICSM'09), Edmonton, Canada.

Jadallah, A. G. (2010). *Proactive and Reactive Decision Support for Handling Change Requests in Software Release Planning.* (Master Scinece), University of Calgary, Calgary.

Jorgensen, M., & Shepperd, M. (2007). A Systematic Review of Software Development Cost Estimation Studies. *Software Engineering, IEEE Transactions on, 33*(1), 33-53. doi: 10.1109/TSE.2007.256943

Junji, Z., & Ruhe, G. (2013, 27-28 Sept. 2013). *DEVis: A tool for visualizing software document evolution.* Paper presented at the Software Visualization (VISSOFT), 2013 First IEEE Working Conference on.

Karlsson, J., & Ryan, K. (1997). A cost-value approach for prioritizing requirements. *Software, IEEE, 14*(5), 67-74. doi: 10.1109/52.605933

Karlsson, J., Wohlin, C., & Regnell, B. (1998). An evaluation of methods for prioritizing software requirements. *Information and Software Technology, 39*(14-15), 939-947.

Karlsson, L., Host, M., & Regnell, B. (2006). *Evaluating the practical use of different measurement scales in requirements prioritisation.* Paper presented at the Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, Rio de Janeiro, Brazil.

Karlsson, L., & Regnell, B. (2005). *Comparing Ordinal and Ratio Scale Data in Requirements Prioritisation.* Paper presented at the 3rd International Workshop on Comparative Evaluation in Requirements Engineering (CERE'05), Paris, France.

Khaari, M., & Ramsin, R. (2010, 22-26 March 2010). *Process Patterns for Aspect-Oriented Software Development.* Paper presented at the 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS).

Khan, K. A. (2006). *A Systematic Review of Software Requirements Prioritization.* (Master), Blekinge Institute of Technology, Ronneby, Sweden. (MSE-2006-18)

Khatibi Bardsiri, V., & Norhayati Abang Jawawi, D. (2011). Software Cost Estimation Methods: A Review. *Journal of Emerging Trends in Computing and Information Sciences, 2*, 21-29.

Krishnan, M. S. (1994). *Software release management: a business perspective.* Paper presented at the Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative Research, Toronto, Ontario, Canada.

Lausen, S. (2002). *Software Requirements: Styles and Techniques*: Addison-Wesley Professional.

Levin, K. D., & Yadid, O. (1990). Optimal release time of improved versions of software packages. *Inf. Softw. Technol., 32*(1), 65-70. doi: 10.1016/0950-5849(90)90048-v

Li, J., & Ruhe, G. (2003, 13 October 2003). *Web-Based Decision Support for Software Release Planning.* Paper presented at the Workshop on Applications, Products and Services of Web-based Support Systems (WSS03), Halifax, Canada.

Li, M., Huang, M., Shu, F., & Li, J. (2006). *A risk-driven method for eXtreme programming release planning.* Paper presented at the 28th International Conference on Software Engineering, Shanghai, China.

Lincoln, Y. S., & Guba, E. G. (1985). *Naturalistic Inquiry*. Beverly Hills: Sage Publications.

Lindgren, M., Land, R., Norstrom, C., & Wall, A. (2008, 25-28 March 2008). *Key Aspects of Software Release Planning in Industry*. Paper presented at the 19th Australian Conference on Software Engineering, Perth, WA, Australia.

Lindgren, M., Land, R., Norström, C., & Wall, A. (2008). Towards a Capability Model for the Software Release Planning Process: Based on a Multiple Industrial Case Study. In A. Jedlitschka & O. Salo (Eds.), *Product-Focused Software Process Improvement* (pp. 117-132). Monte Porzio Catone, Italy: Springer Berlin Heidelberg.

Lindgren, M., Norstrom, C., Wall, A., & Land, R. (2008, 18-21 Feb. 2008). *Importance of Software Architecture during Release Planning*. Paper presented at the Seventh Working IEEE/IFIP Conference on Software Architecture, (WICSA 2008).

Ma, Q. (2009). *The Effectiveness of Requirements Prioritization Techniques for a Medium to Large Number of Requirements: A Systematic Literature Review*. (Master Scinece), Auckland University of Technology.

Marjaie, S. A., & Kulkarni, V. (2010, 10-12 Dec. 2010). *Recognition of Hidden Factors in Requirements Prioritization Using Factor Analysis*. Paper presented at the International Conference on Computational Intelligence and Software Engineering (CiSE).

Maurice, S., Ruhe, G., Ngo-The, A., & Saliu, O. (2005). Decision Support for Value-based Software Release Planning. In S. Biffl, A. Aurum, B. Boehm, H. Erdogmus & P. Grünbacher (Eds.), *Value-based Software Engineering* (pp. 247-262): Springer Berlin Heidelberg.

Mc Elroy, J., & Ruhe, G. (2010). When-to-release decisions for features with time-dependent value functions. *Requirements Engineering, 15*(3), 337-358.

Michlmayr, M., Hunt, F., & Probert, D. (2007). Release Management in Free Software Projects: Practices and Problems. In J. Feller, B. Fitzgerald, W. Scacchi & A. Silitti (Eds.), *Open Source Development, Adoption and Innovation* (pp. 295-300). Limerick, Ireland: Springer US.

Mohebzada, J. G. (2012). *A Recommendation System for Planning Software Releases*. (Master Scinece), University of Calgary, Calgary. Retrieved from http://theses.ucalgary.ca/handle/11023/249.

Myers, M. D. (2009). *Qualitative Research in Business & Management*. London: Sage Publications.

Nejmeh, B. A., & Thomas, I. (2002). Business-driven product planning using feature vectors and increments. *Software, IEEE, 19*(6), 34-42. doi: 10.1109/MS.2002.1049385.

Ngo-The, A., & Ruhe, G. (2009). Optimized Resource Allocation for Software Release Planning. *IEEE Transactions on Software Engineering, 35*(1), 109-123.

Ngo, A., & Saliu, O. (2005, 22-25 May 2005). *Fuzzy Structural Dependency Constraints in Software Release Planning*. Paper presented at the The 2005 IEEE International Conference on Fuzzy Systems, Reno, NV, USA.

Ninaus, G. (2012). *Using group recommendation heuristics for the prioritization of requirements*. Paper presented at the Sixth ACM Conference on Recommender Systems, Dublin, Ireland.

Otero, C. E., Dell, E., Qureshi, A., & Otero, L. D. (2010, 26-28 May 2010). *A Quality-Based Requirement Prioritization Framework Using Binary Inputs*. Paper presented at the Fourth Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation (AMS).

Penny, D. A. (2002, 2002). *An estimation-based management framework for enhancive maintenance in commercial software products*. Paper presented at the Proceedings of International Conference on Software Maintenance.

Perini, A., Susi, A., & Avesani, P. (2012). A Machine Learning Approach to Software Requirements Prioritization. *IEEE Transactions on Software Engineering, PP*(99), 1-1. doi: 10.1109/TSE.2012.52

Perini, A., Susi, A., Ricca, F., & Bazzanella, C. (2007, 16-16 Oct. 2007). *An Empirical Study to Compare the Accuracy of AHP and CBRanking Techniques for Requirements Prioritization*. Paper presented at the Fifth International Workshop on Comparative Evaluation in Requirements Engineering (CERE '07).

Perry, D. E., Porter, A. A., & Votta, L. G. (2000). *Empirical studies of software engineering: a roadmap*. Paper presented at the Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland.

Pfleeger, S. L., Wu, F., & Lewis, R. (2005). *Software Cost Estimation and Sizing Methods, Issues, and Guidelines*: Rand Publishing.

Pressman, R. S. (2001). *Software Engineering: a practitioner's approach* (5 ed.): McGraw-Hill.

Przepiora, M., Karimpour, R., & Ruhe, G. (2012, 19-20 September 2012). *A hybrid release planning method and its empirical justification.* Paper presented at the International Symposium on Empirical Software Engineering and Measurement (ESEM'12), Lund, Sweden.

Racheva, Z., Daneva, M., Herrmann, A., & Wieringa, R. J. (2010, 19-21 May 2010). *A Conceptual Model and Process for Client-driven Agile Requirements Prioritization.* Paper presented at the Fourth International Conference onResearch Challenges in Information Science (RCIS).

Rittel, H., & Webber, M. (1984). Planning problems are wicked problems. In N. Cross (Ed.), *Developments in Design Methodology*. New York: John Wiley and Sons.

Robson, C. (2011). *Real World Research* (3 ed.): John Wiley and Sons.

Ruhe, G. (2005). Software Release Planning. In S. K. Chang (Ed.), *Handbook of Software Engineering and Knowledge Engineering - Vol. 3 - Recent Advances* (pp. 365-394). Singapore: World Scientific Publishing Co. Pte. Ltd.

Ruhe, G., Eberlein, A., & Pfahl, D. (2003). Trade-off Analysis for Requirements Selection. *International Journal of Software Engineering and Knowledge Engineering, 13*(4), 345-366.

Ruhe, G., & Ngo, A. (2004). Hybrid Intelligence in Software Release Planning. *Int. J. Hybrid Intell. Syst., 1*(1-2), 99-110.

Ruhe, G., & Ngo, A. (2004). Hybrid Intelligence in Software Release Planning. *International Journal of Hybrid Intelligent Systems, 1*(1-2), 99-110.

Ruhe, G., & Saliu, O. (2005a). The Art and Science of Software Release Planning. *IEEE Software, 22*(6), 47-53.

Ruhe, G., & Saliu, O. (2005b). The Science and Practice of Software Release Planning: University of Calgary.

Saaty, T. L. (1980). *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*: McGraw-Hill.

Saleem, S. B., & Shafique, M. U. (2008). *A Study on Strategic Release Planning Models of Academia and Industry.* (Master Science), Blekinge Institute of Technology, Ronneby. (MSE-2008-24)

Saliu, O., & Ruhe, G. (2005a). Software Release Planning for Evolving Systems. *Innovations in Systems and Software Engineering, 1*(2), 189–204.

Saliu, O., & Ruhe, G. (2005b, 7-7 April 2005). *Supporting Software Release Planning Decisions for Evolving Systems.* Paper presented at the 29th Annual IEEE/NASA Software Engineering Workshop.

Saliu, O., & Ruhe, G. (2007, 3-7 September 2007). *Bi-objective release planning for evolving software systems.* Paper presented at the The 6th joint meeting of the European Software Engineering Conference and The ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC-FSE '07), Dubrovnik, Croatia.

Seyed Danesh, A. (2011). A Survey of Release Planning Approaches in Incremental Software Development. In V. V. Das & N. Thankachan (Eds.), *Computational Intelligence and Information Technology* (pp. 687-692). Pune, India: Springer Berlin Heidelberg.

Seyed Danesh, A., & Ahmad, R. (2012). Software release planning challenges in software development: An empirical study. *African Journal of Business Management, 6*(3), 956-970.

Slooten, R. (2012). *Software release planning: Investigating the use of an advanced assessment instrument and evaluating a novel maturity framework.* (Master Scinece), Technische Universiteit Eindhoven, Eindhoven.

Sommerville, I. (2010). *Software Engineering* (9 ed.): Addison-Wesley.

Suwanjang , H., & Nakornthip, P. (2012). Framework for Developing a Software Cost Estimation Model for Software Modification Based on a Relational Matrix of Project Profile and Software Cost Using an Analogy Estimation Method. *International Journal of Computer and Communication Engineering, 1*(2), 6.

Svahnberg, M., Gorschek, T., Feldt, R., Torkar, R., Saleem, B. S., & Shafique, M. U. (2010). A systematic review on strategic release planning models. *Information and Software Technology, 52*(3), 237–248.

Svensson, R. B., Gorschek, T., Regnell, B., Torkar, R., Shahrokni, A., Feldt, R., & Aurum, A. (2011, Aug. 29 2011-Sept. 2 2011). *Prioritization of quality requirements: State of practice in eleven companies.* Paper presented at the 19th IEEE International Requirements Engineering Conference (RE).

Tonella, P., Susi, A., & Palma, F. (2010, 7-9 Sept. 2010). *Using Interactive GA for Requirements Prioritization.* Paper presented at the Second International Symposium on Search Based Software Engineering (SSBSE).

Tran, H. N., Coulette, B., & Bich Thuy, D. (2007, 25-31 Aug. 2007). *Modeling Process Patterns and Their Application.* Paper presented at the Software Engineering Advances, 2007. ICSEA 2007. International Conference on.

Vogt, D. W. P. (1993). *Dictionary of Statistics and Methodology: A Non-Technical Guide for the Social Sciences* California: Sage Publications.

Wohlin, C., & Aurum, A. (2005, 17-18 November 2005). *What is Important when Deciding to Include a Software Requirement in a Project or Release?* Paper presented at the International Symposium on Empirical Software Engineering, Noosa Heads, Australia.

Yin, R. K. (2003). *Case study research: Design and methods* (3 ed.): Sage Publications.

Zahran, S. (1998). *Software Process Improvement: Practical Guidelines for Business Success*: Addison-Wesley Professional.

Zhao, L. (2010, 7-8 Aug. 2010). *Research on Software Project Management Pattern What Based on Model-Driven.* Paper presented at the International Conference of Information Science and Management Engineering (ISME).

APPENDIX A: QUESTIONARIES

Table A: Summary of interview questions

Part 1:Introductory questions
1. What is your name?
2. What is your email address?
3. Give us a definition of your project?
4. What is your role in the project?
5. How many people attend this project?
6. What generally is the size of your project?
7. Are you satisfied with the project? If yes or no, why?
Part 2:Technical questions
1. Do your project managers conduct release planning?
2. What is the current release cycle?
3. What is your cooperation and interest in the project like?
4. What challenges and problems do you face when you want to release a new version?
5. How are the requirements generated? How are they tracked?
6. What are the ambiguous problems in your project?
7. When is the next release date? When was the last release date?
8. Are there any strategic changes in your project? If yes, what is their impact?
9. What kinds of stakeholders exist in the project? How are they involved in your project?
10. Who are your major customers or end users?
11. What is of most importance to the managers?
12. How are the new decisions made for a new release?
13. Do the mangers have any plan or prediction for the future of a product to ensure that the last release is the best one?
14. How do project managers evaluate the progress and completeness of the work?
15. What is the output of the release planning?
16. What kind of complexity you face? Does the complexity affect the process of your project?
17. What are the resource and technical constraints in your project?
18. What is your plan for release time? How is the time for next release determined?
19. Is this system related to other systems? If yes, which ones?
20. Are there any pressures on you for a new release?
21. How are the requirements prioritized in your projects?
22. Do you have any tools to support your process of release planning?

Table B: projects characterizes

	Damoon	**Saba**	**PKI /CA**	**EXIMBILLS**	**Islamic Loan**
Project Description	Internet shops	Internet banking systems		Trade Finance Systems	Islamic Loan systems
Number of employees	8	15	12	18	26
Customers/End-users	Melli, Saderat, Mine and Industry banks	Melli, Saderat, Mine and Industry, Export and Development banks	Central bank of Iran	Melli, Saderat banks	Melli, Saderat, Mine and Industry, Export and Development banks
Number of releases until now	2	6	2	2	9
Resources type and unit	Planning 3 man-months, Analysis 6 man-months, Design 9 man-months, Construction 12 man-months	Planning 6 man-months, Analysis 9 man-months, Design 12 man-months, Construction 20 man-months	Planning 2 man-months, Analysis 3 man-months, Design 3 man-months, Construction 5 man-months	Planning 5 man-months, Analysis 7 man-months, Design 9 man-months, Construction 12 man-months	Planning 10 man-months, Analysis 22 man-months, Design 25 man-months, Construction 35 man-months
Planning Criteria	Low	High	High	Low	Medium
Requirement groups	Yes	Yes	No	Yes	Yes
Requirement dependencies	Yes	yes	No	Yes	Yes
Role and responsibility	Project manager and a system analyst: The project manager was responsible for making sure the questions are on the right track and the analyst was responsible for getting the right answers (no existing system was available)	Project manager and a system analyst: The project manager was responsible for making sure the questions are on the right track and the analyst was responsible for getting the right answers	Project manager and a developer: The project manager was responsible for making sure the questions are on the right track and the developer was responsible for analyzing the required system	Project manager and a system analyst: The project manager was responsible for making sure the questions are on the right track and the analyst was responsible for getting the right answers and understanding the existing system	Project manager, and two system analysts: The project manager was responsible for making sure the questions are on the right track and the analysts were responsible for understanding the existing or old system
Project evaluation	Regularly and on weekly basis	Regularly and weekly the resources are modified according to project needs	Regularly and on monthly basis	Bi-weekly meetings	Regularly and on monthly basis

APPENDIX B: PATTERN-BASED RELEASE PLANNING METHODOLOGY QUESTIONARIES

Table A: Pattern Evaluation form

Pattern-based Release planning Evaluation Form				
Project Code:		Iteration:	Your Role:	
Pattern Evaluation				
1	Release planning phase	#1	#2	#3
2	Previous method used	#1	#2	#3
3	Pattern proposed by PRP	#1	#2	#3
4	Method proposed by pattern	#1	#2	#3
5	Method used in the phase	#1	#2	#3
6	If proposed method was successful, describe reasons	#1	#2	#3
7	If proposed method was failed, describe reasons	#1	#2	#3
8	Problems of engaging proposed method	#1	#2	#3
9	Problems of engaging proposed pattern	#1	#2	#3
10	Exact gains of using proposed method	#1	#2	#3
11	Exact gains of using proposed pattern	#1	#2	#3
12	Ranking of the method (Likert's scale)	#1	#2	#3
13	Ranking of the pattern (Likert's scale)	#1	#2	#3

Table B: Evaluation of patterns of specific phase from

Pattern-based Release planning Evaluation Form				
Project Code:			Your Role:	
Evaluation of Patterns of Specific Phase				
1	Release planning phase	#1	#2	#3
2	Number of patterns used	#1	#2	#3
3	Number of failed patterns	#1	#2	#3
4	Number of successful patterns	#1	#2	#3
5	Problems of engaging proposed methods	#1	#2	#3
6	Problems of engaging proposed patterns	#1	#2	#3
7	Exact gains of using proposed methods	#1	#2	#3
8	Exact gains of using proposed patterns	#1	#2	#3
9	Overall ranking of the methods (Likert)	#1	#2	#3
10	Overall ranking of the patterns (Likert)	#1	#2	#3

Table C: Pattern-based release planning methodology evaluation form

Pattern-based Release planning Evaluation Form		
Project Code:		Your Role:
Pattern-based Release planning methodology Evaluation		
1	Number of patterns used	
2	Number of failed patterns	
3	Number of successful patterns	
4	Problems of engaging methodology	
5	Exact gains of using methodology	
6	Ranking of the methodology (Likert)	

APPENDIX C: PATTERN RELEASE PLANNING (PRP) TOOL

The PRP software is developed and used to develop release planning patterns in different phases. The software is based mainly on the idea mentioned in the research and aims to define phases, parameters, instances and patterns of release planning, and to search for (and show) release patterns in different phases based on input parameters. It also makes decisions on best-suited developed patterns. The software supports and defines release planning patterns in two ways:

- In the first method, the software allows users to define different phases of release planning and to determine parameters suiting every phase. Having parameters defined, the user can enter parameter instances and establish a relationship between various instances. The software enables users to enter relationships in a level-by-level manner. In the first level, it is possible to enter the relationship between two instances of two separate parameters and the relationship between two instance pairs is allowed to form a ternary of instances. The ternaries are then related in the next level to form relationships with four parameters and the software develops levels as much as the number of parameters. In the last level where every parameter instances correlate, the software allows for defining a pattern for them. In this situation, the software deactivates in higher levels those instances without relationships. This means that where two instances of two different parameters are not correlated in the first level, they cannot be so in the next level and the software does this automatically. The method requires developing a set of states based upon relationships between parameter instances. In this method, all possible states of instance combinations are determined and, indeed, it is a comprehensive method for generating patterns for release planning.

- In the second method, the software allows defining a pattern for every step of release planning directly and based on parameter instances. In this method, the software enables users to develop, name, describe and fully document their patterns by selecting the best-suited parameter instances. The user is, also, allowed to select a certain set of parameters, a case in which the software considers all the states for other unselected parameters automatically. Thus, the user can develop a pattern through several specific parameters. This method enables defining release planning pattern in a rapid and need-based manner.

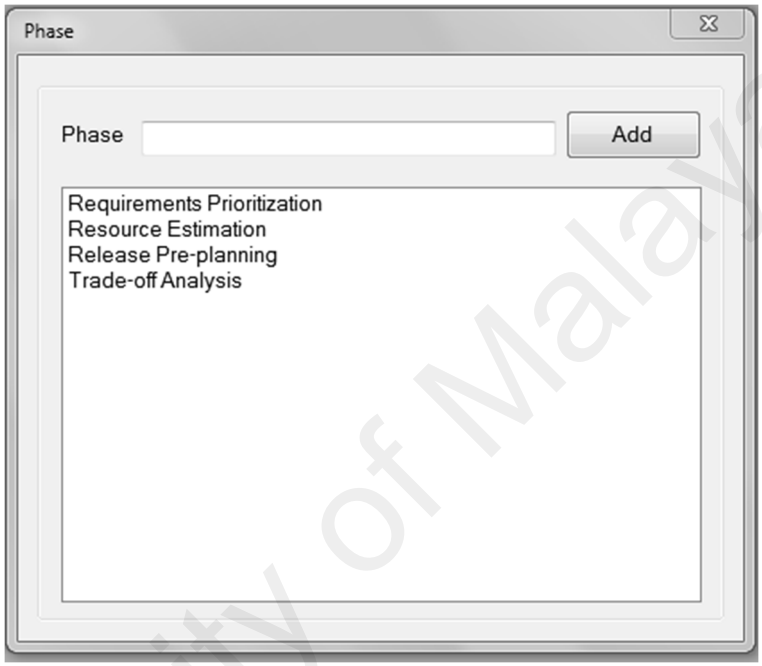
Using both methods, the software enables users to receive suggested patterns by entering the least information. Pattern selection is performed for different steps of release planning and every pattern is assigned to a certain phase or stage. Moreover, it is possible to keep a record of project-specific data in order to determine patterns during software development. Below is a brief description of different parts of the software.

1. The main page of the software

In the main page, icons pertaining to different tasks are located on top and the user can select them to perform the desired task.

2. Phase information

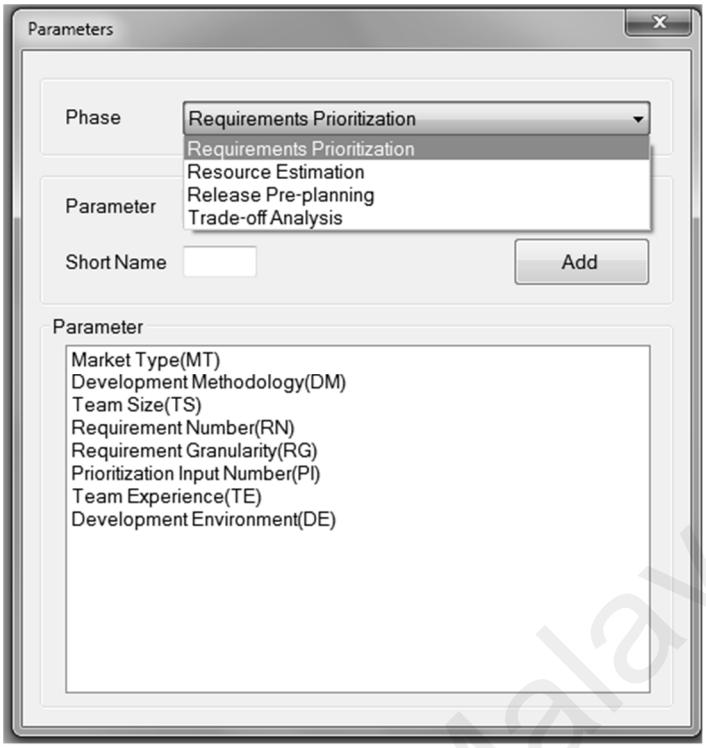
Since general phases are determined for release planning in the suggested method, it is also possible to record different steps of the project in the software. It enables the user to alter phases where parameters and instances are not present. The figure below shows recorded phases in the software.



3. Parameters information

Every certain step (phase) of release planning can have various parameters and hence the software enables entering phase parameter and its short name by selecting the phase to be used in different sections.

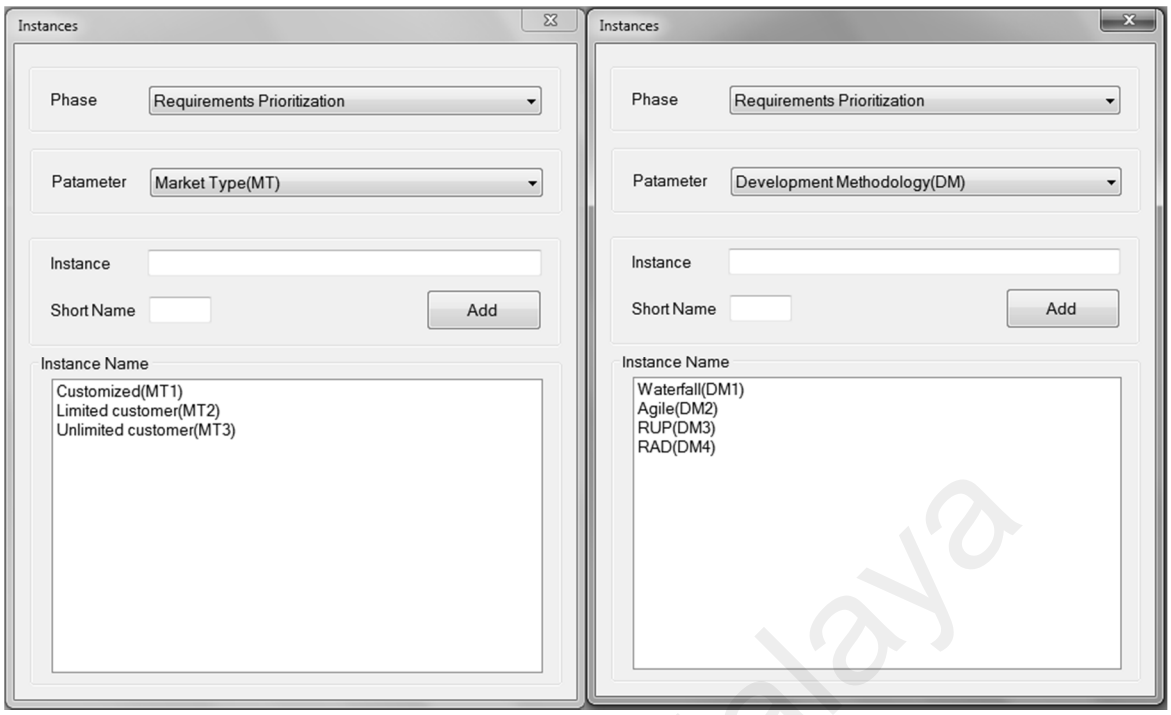
The figure below shows different parameters entered for "requirements prioritization". As the figure displays, a short name is entered for every parameter (showed in parentheses next to the parameter's name). The user can enter different parameters for evaluation and delete the parameter if no instances entered for it.



4. Instance information

Each parameter can have various instances. Moreover, parameters with the same names (in different phases) can have different instances. This is because an instance can have no influence on a certain phase or there may be new instances which affect a certain parameter in a phase. Considering the possibility of such cases, this is embedded in the software and the release planner can enter different instances freely.

Two parameter instances of requirements prioritization phase are presented in the following figure. The shortened name of every instance is also mentioned besides its main name to be used if necessary. Similar to previous stages, it is possible to add or remove instances and they can be removed if no relationship is recorded for them.



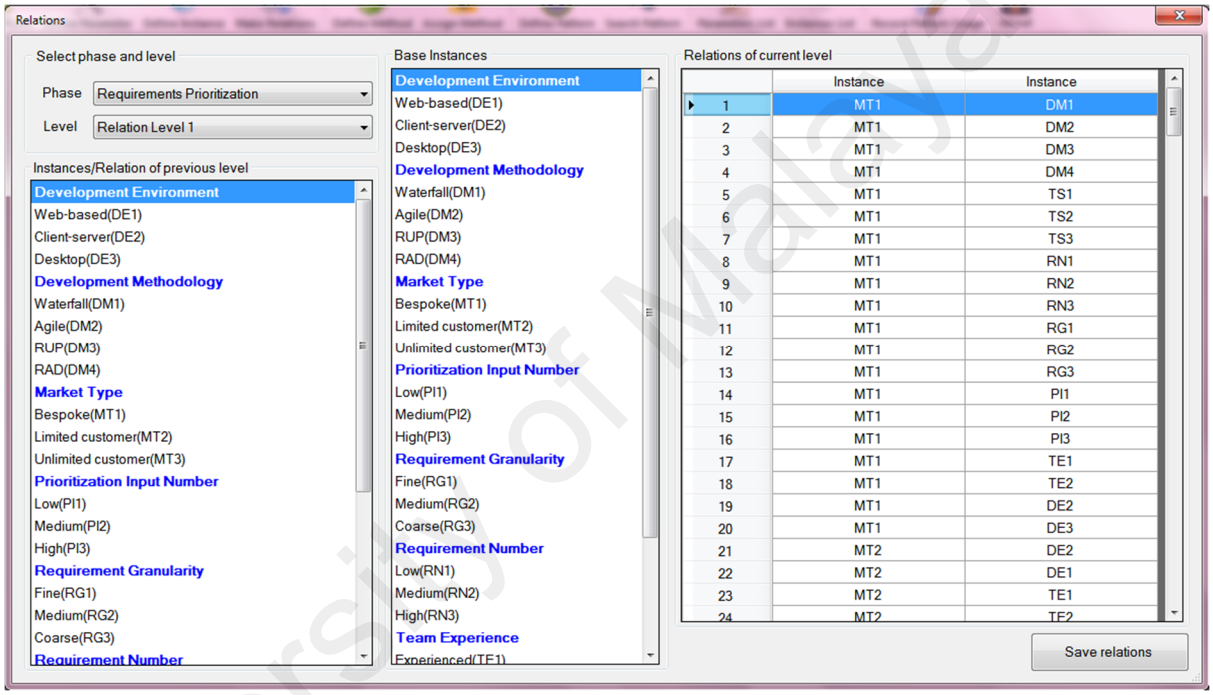
5. The relationship between instances

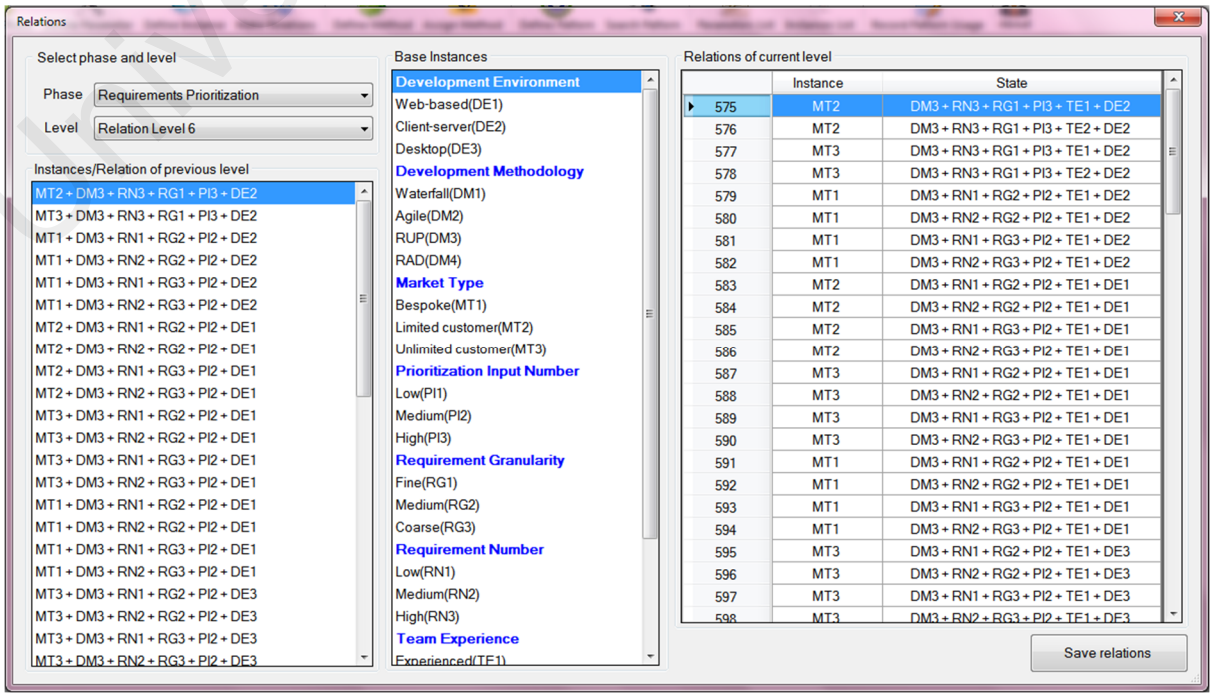
After recording primary data of release planning and steps and parameters of the methodology, the most important task is to establish a relationship between instances in order to generate required states. As mentioned earlier, the process can be accomplished in two ways. In this step, different states are generated in various levels establishing relationships between instances and this continues in a level-by-level manner until relationships are developed between all parameters of a phase.

Selecting the phase and level, the release planner can observe developed relations or instances (which form a state) and is able to use them to generate new relations. In the first step of this task (shown in the figure below), the user can observe different parameter instances by selecting the first level and can generate a new relation by choosing two instances of different parameters. The software automatically recognizes that the relation is established between different parameters and prevents repetitive relations. Having both instances selected, the software displays them with an "x",

generates the relation and assigns a certain number to it while the user clicks on "Save relations". This is done for every level and the highest level is determined based on the number of parameters in every step of the release planning process.

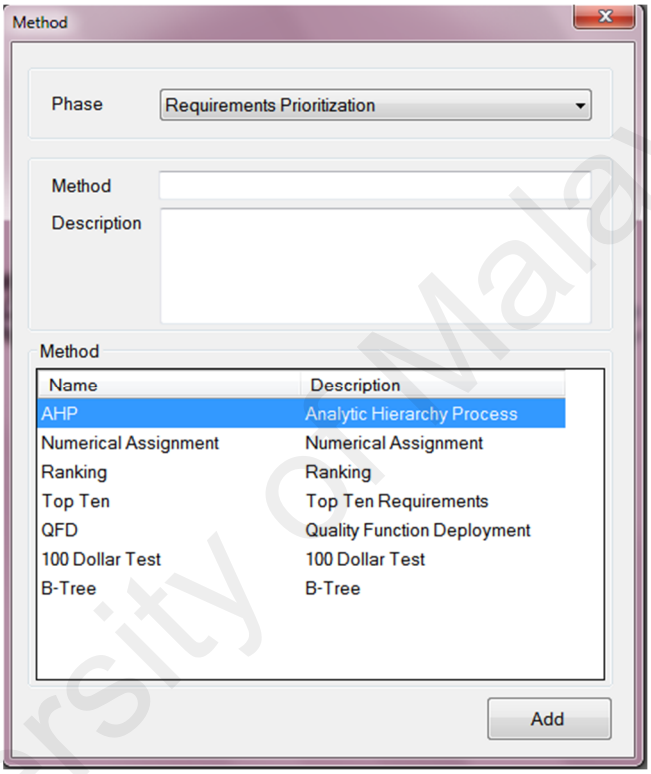
The software allows for removing a relation when the user has not been generating a higher level relation based upon it. It recognizes the number of parameters automatically and allows entering level by the same number.





6. Methods information

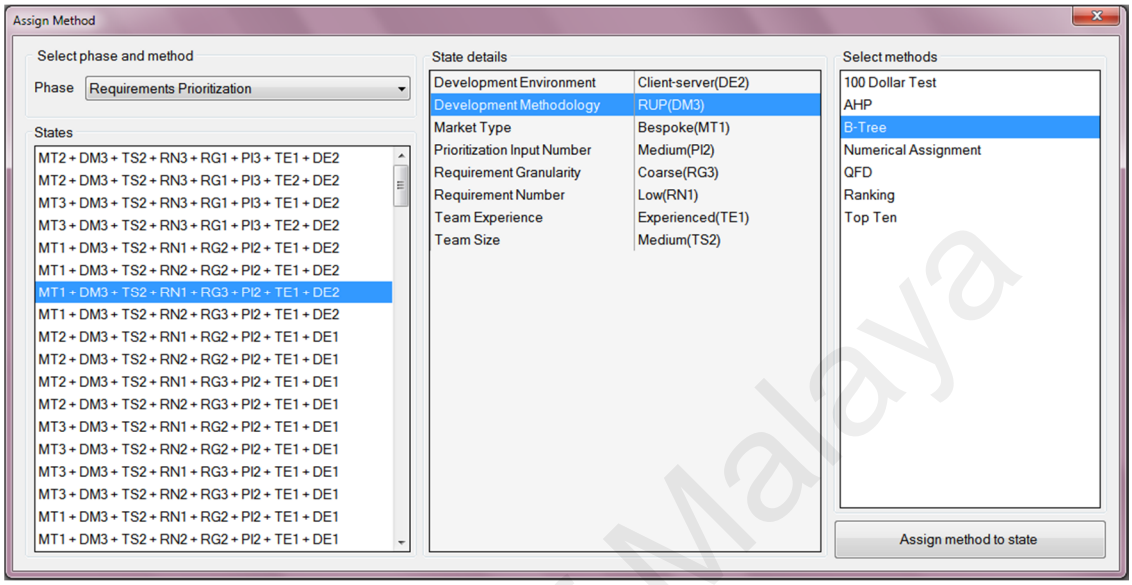
Having primary data recorded, the second step is accomplished using one of the methods mentioned earlier. Both methods are based upon the data pertaining to different methods used in every step of pattern-based release planning. The figure below shows methods used in requirements prioritization step.



7. Assigning the method to complementary states

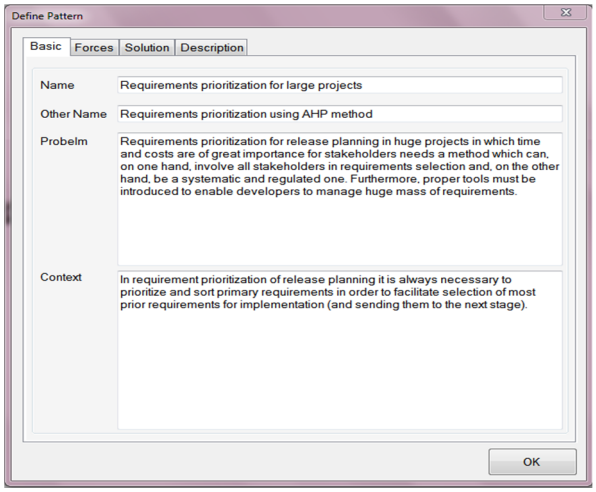
In order to perform release planning with the first method, one has to determine the relations between parameter instances of previous steps and then the software introduces all instance-containing states to the present step automatically and assigns them to a certain method. In fact, it automatically searches all complete states containing parameters from different steps and displays them to enable users to select implementation methods.

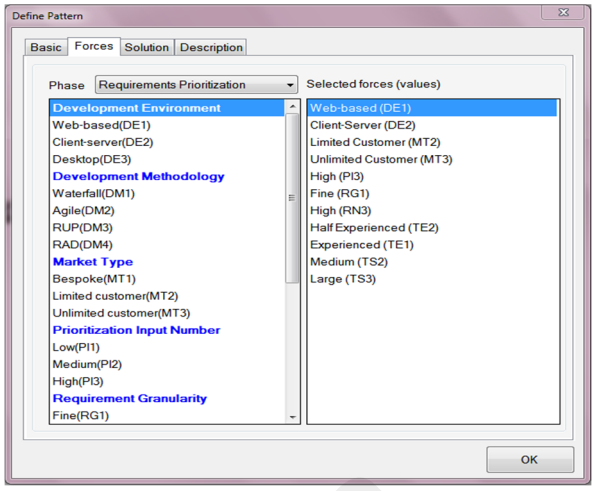
The user can assign more than one method to a state and this means that the release planner or project manager can choose each of the mentioned methods to accomplish a project with similar states. This is shown in the figure below.

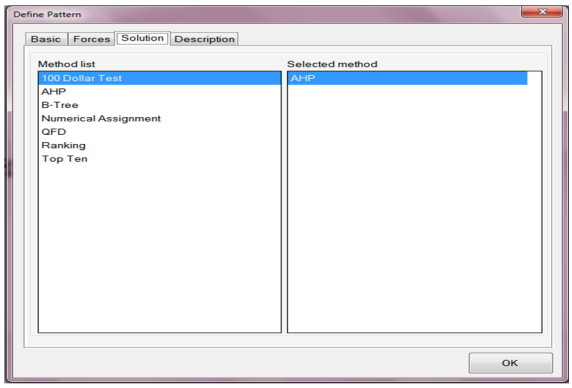


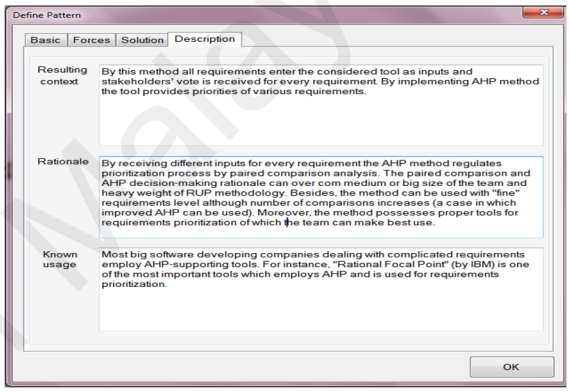
8. Pattern definition

Pattern definition section aims to describe a pattern's features based on the presented structure. This part includes four main sub-sections. The "base" subsection explains basic data on the pattern such as its name, side names, pattern problem and pattern field. In the "constraints" subsection, pattern constraints (parameter instances) can be defined for every step with defined patterns. The "solution" subsection specifies methods suggested by the pattern and the user can select one or more methods. This is consistent with the previous subsection. This means that one can observe, in "solution", methods related to a step already selected in "constraints". In "description" subsection, one can record explanations on the context, rationale and known uses of the pattern.



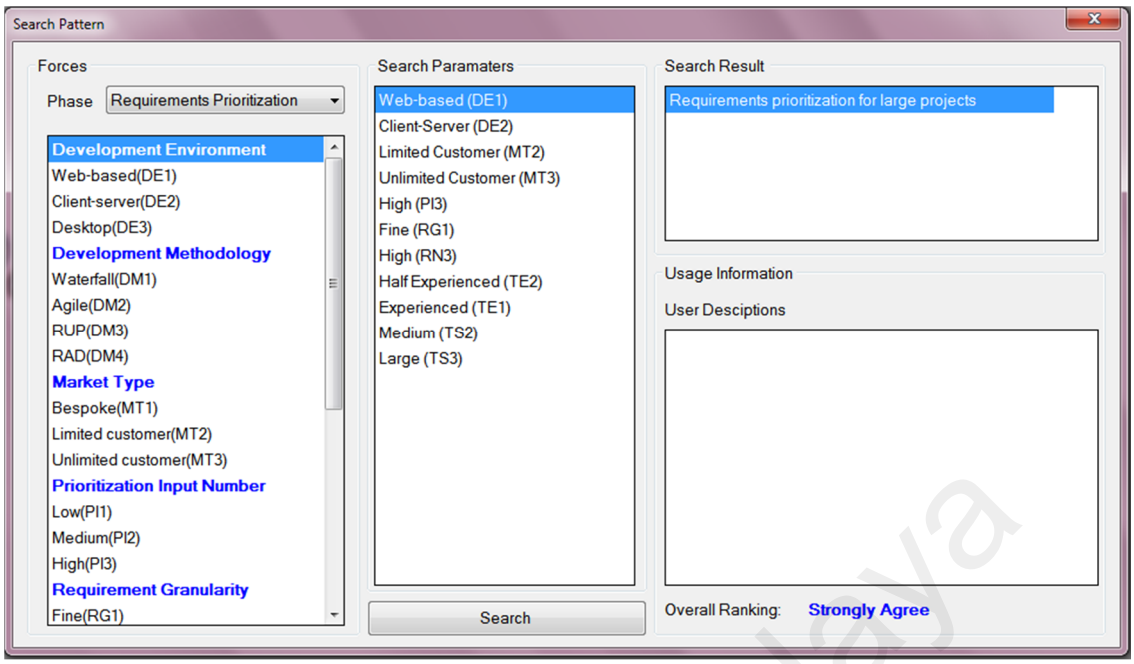






9. Pattern search

Pattern search helps finding the suitable pattern using certain features (constraints or instances). In this section, users can start the search by specifying the constraints. The software searches among available patterns and finds those with the considered constraints. Then, every found pattern is displayed along with data entered by users (after using the pattern) and the average scores given by them.



10. Recording the pattern usage

After the pattern is used, results can be recorded in this section. By selecting the considered pattern and signing up, the user can write about his experience of the pattern and score it based on Likert Scale. The data is shown to other users searching for patterns.