Chapter 1: Introduction

The current software industry is facing numerous difficulties. Statistics has shown that many software projects failed to deliver acceptable systems within schedule and budget. They were either aborted, never been released, released but never been used or released with low quality that always cause problems. As such, continuous effort has been made in finding ways to overcome these problems, with main focus on proper project management techniques. Among the solutions to these problems is the systematic implementation of software risk management in the project development.

Like many fields in their early stages, software projects have encountered their project disasters. Most post-mortems of these software disaster projects have indicated that their problems could have been avoided or reduced if the high-risk elements had been identified in advance and appropriate steps taken to resolve the problems.

1.1 What Is Software Risk Management?

Software Risk Management is a software engineering practice with processes, methods, and tools for managing risks in a project. It provides a disciplined environment for proactive decisions making to

- assess continuously what could possibly go wrong (risks);
- determine which risks are important(prioritize) and necessary to deal with;
- implement appropriate strategies to deal with these risks.

A single missed delivery or development delay can cause undesirable consequences, through impaired productivity, customer dissatisfactions, and deferred realization of revenue and cost increase. Risks, being opportunities for failure, are to be recognized and managed, not feared, ignored or left hidden until disaster strikes. The primary goal of risk management is to identify, confront, and reduce/eliminate risk items to a minimum with enough lead time to act on them before they become threats to successful software development or major sources of software rework.

1.2 Risk Definitions

There are a number of risk definitions and uses for the term risk, but there is no universally accepted definition. What all definitions have in common is an agreement that risk has two characteristics:

- Uncertainty – an event may or may not happen
- Loss – an event that leads to unwanted consequences or losses.

Examples of risk definitions:

Lawrence, William W.: *"Acceptable Risk" 1976* - ***Risk is the measure of the probability and severity of adverse effects.***

Webster's Third New International Dictionary 1981 – ***Risk is the probability of suffering loss, injury, disadvantages, or destruction.***

Kaplan and Garrick 1981 – ***Risk is "relative to the observer" and has to do with both uncertainty and damage.*** $R = \{(S_i, P_i, X_i)\}$, S_i is a scenario identification or description, P_i is the probability of that scenario and X_i is the consequence or evaluation measure of that scenario, i.e., the measure of damage.

Rowe, William D. [86]: *"An Anatomy of Risk" 1988* – ***Risk is the potential for realization of unwanted negative consequences of an event.***

Anon 1989 Dictionary – ***Exposure to the chance of injury or loss.***

Dean 1994 – ***Risk is the perceived extent of possible loss.*** Risk is individual to a person or organization because what is perceived by one as a major risk may be perceived by other as a minor risk. Perception is very much a factor.

Rowe 1994 – *The downside of a gamble which is described in terms of probability.*

Andrew 1995 – ***Exposure to harm or loss, includes the "possibility of risks and their impact".***

SEI risk definition [116] – ***Risk is the probability of suffering loss.*** The loss describes the impact to the project which could be in the form of diminished quality of the end product, increased costs, delayed completion, or failure.

David Gluch's: *"A Construct for Describing Software Development Risks"* – ***A risk is a combination of an abnormal event or failure and the consequences of that event or failure to a system's operators, users, or environment.*** A risk can range from catastrophic (loss of entire system, loss of life or permanent disability) to negligible (no system damage; no injury).

However, no definition of risk, no matter how complicated, will reduce risk one iota. Risk and opportunity go hand in hand. Many development projects strive to advance current capabilities and achieve something that has not been done before. The opportunity for advancement cannot be achieved without taking risk.

"Risk in itself is not bad; risk is essential to progress, and failure is often a key part of learning. But we must learn to balance the possible negative

consequences of risk against the potential benefits of its associated opportunity."[1][99]

Whether you identify a situation as a risk or as an opportunity depends on your point of view. Is the glass half full or half empty? Situations with high potential for failure often have the potential for high payback as well. Risk taking is essential to progress. Competitive pressures, the ability to recognize opportunities before the competitors do and the demands of modern society require that you take risks to be successful.

1.3 Benefits of Performing Risk Management

At first glance, software risk management might appear to just add complexity to an already complex undertaking. In reality, however, the activities make software projects less complex:

- Identification and prioritization of risks enables project managers and project staff to focus on the areas with the most impact on their project.
- Appropriate risk mitigation actions reduce overall project risk – which actually accelerates project completion.
- Projects that finish sooner cost less, plus risk mitigation actions can further reduce project cost.
- Project using software risk management have more predictable schedules, they experience fewer "surprises", since they have in advance identified (and in many cases, eliminated) possible risks before they can become problems.

[1] [Van Scoy, Roger L. Software Development Risk: Opportunity, Not Problem. Software Engineering Institute, CMU/SEI-92-Tr-30, ADA 258743, September 1992]

To sum it up, software risk management is critical to a project's success. It helps software organizations secure their customers' commitments and hence build up customer goodwill. Further, managers and project staff utilizing software risk management have a better overall understanding of their projects and make better business decisions.

1.4 Reasons Why Software Risk Management Is Not Performed

Software professionals have introduced several risk management approaches and a number of software risk management tools during the past decade, and the seriousness and consequences of failed risk management have also been published in several articles and books. But these efforts appear not to have awaken the general software organizations about the importance of software risk management. While some organizations have defined their own risk management approaches, most organizations do not manage their risks explicitly and systematically. When risk management methods are used, they are often simplistic and users have little confidence in their final risk analysis results. More commonly, the software development team does nothing about risks until something goes wrong. Then, the team flies into action in an attempt to correct the problem quickly. This is often called "fire fighting mode." When this fails, "crisis management" [82] takes over and the project is in real jeopardy. There are several reasons for these situations:

- Software Risk Management field is still in its infancy; there are very few literatures and guidelines available that offer a practical, step-by-step approach to managing risk.
- Risk is an abstract and fuzzy concept; users lack the necessary knowledge to define risk more accurately for deeper analysis.

- Most of the current risk management methods are formal methods. Software development organizations do not have their own risk management procedures. These organizations need to employ external risk consultants/experts to conduct risk management. It is generally very time consuming and expensive. Moreover, these software organizations are also worried about the confidentiality of their corporate information.
- Risks have different implications to different stakeholders. There are very few existing methods that provide support for dealing with these different stakeholders and their expectations.
- Each risk may affect a project in more than one way. Most existing risk management approaches focus on cost, schedule, or quality risks, yet other characteristics may be important factors that influence the real decision making process.
- Software development organizations suffer from the lack of a matured and reliable organization database. This fact had significantly impacted the effectiveness of the organization's risk management effort.
- There are very few automated tools available in the market due to the very little studies and researches done in this field. All currently available software risk management tools are actually risk analysis tools (i.e. Monte Carlo Simulation), or they are application add-ons, or databases of someone else's risk. Techniques employed in these tools involve complicated equations or require complex interactions with the software. These tools are generally very difficult to use and an understanding of their basic concept is not simple. A considerable amount of training is required.
- Software development organizations are not aware of the importance of risk management in the software project. Risk involves future happening; something that

has not happened is hard to be accepted as will happen. Software development personnel have very negative attitudes, such as "It is only software", "We'll fix it in the field later", "We'll do more things in parallel", "I don't need process, I have good people", "Risk management will only add complexity to my projects" and "I will have to spend more money if I perform risk management". Even though there are a number of million dollars runaway projects, there is no risk awareness until disasters strike.

Risk management based on intuition and individual initiative alone is seldom effective. There must be a systematic approach to manage risk and everyone on the project team must be committed to and participate in risk management activities.

1.5 The Current Problem

The whole software industry is full of examples of runaway projects, missed deadlines and low quality software (the boxes that follow provide some examples)[10]. The frequency of these software project disasters is a serious concern: A recent survey by Peat Marwick Mitchell & Co. of 600 client firms indicated that 35 percent of them had at least one runaway software project at the time. Another survey by The United States of America had also shown that about one half of all the software developed had never been released; and half of the released software were never used by the customers [32]. We can conclude that three quarters of all the software projects were unsuccessful! Furthermore, most of the software products completed were behind schedule, out of budget, performed unsatisfactorily, and required major modifications during implementation [83].

Software development today is a high-risk venture. At the same time, software has become an increasingly important element both in business processes and in products.

Figures from the United States of America government for the year 1980 [91] illustrated that approximately $57 billion was spent on computer systems. Out of this $57 billion, $32 billion was spent on the computer software. During the past two decades, there has also been a tremendous growth in the software needs, both in government and private sectors. It is predicted that the demand for software will continue to grow at a higher rate in the next decade. Information systems are central in managing the flow of information in a modern world.

The Sears, Roebuck & Co. subsidiary set out in 1982 to build the insurance industry's most sophisticated computer system, one that would make its competitors quake. The system was supposed to automate Allstate's offic operations and shorten the normal three-year period needed to introduce new types of policies to one month. Allstate hired Electronic Data Systems Corp., th systems-integration company, to develop the software and help install it on th firm's hardware. The targeted date for completion was December 1987; the targeted cost was $8 million. After spending $15 million later, Allstate had a ne project consultant, a new deadline of 11 years, and a new cost of $100 million [10].

In 1983, Blue Cross & Blue Shield United of Winsconsin hired EDS to build a $20 million system to coordinate all the services then being handled by five computers The system was completed on time – in just 18 months. But it didn't work. One example: because of an entry error, the computer sent out hundreds of checks t the fictitious hamlet of None, Wis. A month later the checks arrived back at Blue Cross for readdressing. During its first year of operation, the system disbursed $6 million in overpayments or duplicate checks. Before the runaway was stopped, Blue Cross lost 35,000 policyholders – a setback attributed by the computer problems [10].

In 1983, State of Oklahoma hired a Big Eight accounting firm to design a $500,000 system to handle explosive growth in workers' compensation claims. Two years and after spending more than $2 million, the system still didn't exist. It finally was finished in 1988 at a price of nearly $4 million [10].

In 1984, City of Richmond hired Arthur Young to develop a $1.2 million billing and information system for its water and gas utilities. Completion date: Marc 1987. After paying out about $1 million Richmond cancelled the contract, sayin no system had been delivered. Arthur Young filed a $2 million breach of contract suit against the city [10].

Business Men's Assurance: In 1985, the reinsurer began a one-year project t build a $500,000 system to help minimize the risk of buying insurance policies held by major insurers. The company spent nearly $2 million on the project and the project was completed only in early 1990 [10].

Between 1985 and 1987, at least two patients died as a consequence of severe overdoses of radiation delivered by the Therac-25 medical linear accelerato [63]. The cause was a fault in the control software [10].

During the 1991 Gulf War, a Scud missile penetrated the Patriot anti-missile shield and struck a barracks near Dhahran, Saudi Arabia. 28 Americans wer killed and 98 wounded. The software for the Patriot missile contained a cumulative timing fault. The Patriot was designed to operate for only a few hours at a time, after which the clock was reset. As a result, the fault never had significant effect and, therefore, was not detected. In the Gulf, however, the Dhahran Patriot missile battery ran continuously for over 100 hours. This caused the accumulated time discrepancy to become large enough to render the system inaccurate [10].

During the Gulf War, the United States shipped Patriot missiles to Israel fo protection against the Scuds. Israeli forces detected the timing problem afte only 8 hours and immediately reported it to the manufacturer in the United States. They corrected the fault as quickly as they could, but tragically, the new software arrived the day after the direct hit by the Scud [68].

1.6 Causes of Software Problems

Software development is often plagued with unanticipated problems that cause projects to miss deadlines, exceed budgets, or deliver less than satisfactory products. Most post-mortem studies [37], [85], [101] of these software project disasters revealed that they are usually due to one or more of the following reasons:

- Use of poor or undefined development methodologies;
- Lack of proper techniques, principles and tools in the management of the projects;

- Lack of a clear and precise visions of the complete software projects;
- Ignorance of the available techniques and tools for facilitating project development; but leading edge technologies were used to solve problems;
- Incomplete, imprecise, ambiguous and vague requirements specifications, that caused major changes at the later stages;
- Inaccurate resources, schedule and cost estimations;
- Failure to detect potential problems (risks) at the early or during software development project stages;
- Inappropriate personnel and technical skills employed during the different stages of the development;
- Lack of established standards, terminology and agreed upon notations for software project planning and designing;
- Designs were not matched with requirements specifications, and these were not discovered until during testing phase or after installation;
- Lack of necessary expertise in the programming language selection, that caused handicap in the implementation;
- Inappropriate or poor programming practices;
- Inadequate control and management supports during the project development;
- Inadequate testing or using the wrong test data or test metrics;
- Improper documentation.

F. P. Brooks, in his landmark article entitled 'No Silver Bullet" [Brooks, 1986], pointed out that "...building software will always be hard. There is inherently no silver bullet."[II] Software is more complex when compared to many other man made devices or constructions. Complexity is an inherent property of software. No matter how a nontrivial piece of software is designed, the pieces of the product will interact. For example, the states of a module will depend on the states of its arguments, and the states of global variables will also affect the state of the product as a whole. The consequence of this essential complexity of software is that a product becomes difficult to understand. In fact, it is often true that no one really understands a large product in its entirety. And if the product as a whole is not understood, then only the smaller, independent components can be fully understood. This leads to imperfect communication between team members that, in turn, results in the time and cost overruns. This essential complexity affects not only the software process itself, but also the management of the process. Unless a manager can obtain accurate information regarding the project process, it is very difficult to plan the succeeding stages of the project and to budget accurately. And if a project staff leaves, trying to train a replacement can be a nightmare.

Current approaches to the software process also make it too easy for software developers to make high-risk commitments that they will later regret:

- The sequential, document-driven waterfall model:

  Consider the following somewhat bizarre scenario. Jack and Joe decide to build a house. They consult an architect. Instead of showing them sketches, plans, and perhaps a model, the architect gives them a 30-page single-space typed document

[II] # "Silver bullet" in the title of Brooks's article refers to the recommended way of slaying werewolves, otherwise perfectly normal human beings who suddenly turn into wolves. Software usually appears to be innocent and straightforward, but like a werewolf, software can be transformed into something horrifying, in the shape of late deadlines, exceeded budgets, and residual specification and design faults not detected during testing.

describing the house in highly technical terms. Despite the facts that neither Jack nor Joe have any previous architectural experience and hardly understand the document, they enthusiastically sign it and say, "Go ahead, build the house!"

This scenario is highly unlikely. Nevertheless, that is precisely what happens when the waterfall model is used in software development. The process starts with the specifications. In general, specification documents are long, detailed, and frankly, boring to read. The client is usually inexperienced in the reading of software specification, and this difficulty is compounded by the fact that specification documents are usually written in a style that the client is unfamiliar with. Nevertheless, the client proceeds to sign off the specification document, whether he/she properly understood or not.

There is a considerable difference between the way a client understands a product as described in the specification document and the actual product. The specifications exist only on paper; the client therefore cannot really understand what the product itself will be like. The first time the client sees a working product is only after the entire product has been coded. This can lead to the construction of products that simply do not meet the clients' real needs.

- The open-ended incremental model:

  The incremental model can too easily degenerate into the build-and-fix approach. Control of the process as a whole can be lost, and the resulting product, instead of being open-ended, becomes a maintainer's nightmare. In a sense, the incremental model is a contradiction in terms, requiring the developer to view the product as a whole in order to begin with a design that will support the entire product, including future enhancements, but simultaneously, to view the product as a sequence of builds, each essentially independent of the others. Unless the developer is skilled enough to be able to handle this apparent contradiction, the incremental model may lead to an unsatisfactory product. Because the product is divided into many builds, this approach also incurs the real risk that the resulting builds will not fit together.

♦ The code-driven evolutionary development process model:

This model tempts people to say, "Here are some neat ideas I would like to put into this system. I will code them up, and if they don't fit other people's ideas, I will just evolve things until they work." This approach can only work fine in some well-supported mini domain applications, but in more complex application domains, it most often creates or neglects unsalvageable high-risk elements and leads the project down to disaster path.

♦ The risk-driven spiral model:

There are some restrictions on the applicability of the spiral model. In its present form, the model is intended exclusively for internal development of large-scale software [9]. Consider an internal project, that is, one where the developers and client are members of the same organization. If risk analysis leads to the conclusion that the project should be terminated, in-house software personnel can simply be reassigned to a different project. However, if a development organization and an external client have signed a contract, the termination of contract by either side will lead to a breach-of-contract lawsuit. Thus in the case of contract software, all risk analysis must be performed by both client and developers before the contract is signed, and not as in the spiral model.

A second restriction on the spiral model is related to the size of the project. It makes no sense to perform risk analysis if the cost of performing the risk analysis is comparable to the cost of the project as a whole, or if performing the risk analysis would significantly affect the overall profit potential.

A major strength of the spiral model is that it is risk-driven, but this is also a weakness. Unless the software organization developers are skilled enough at pinpointing the possible risks and analyzing them accurately, there is a danger that the development team may believe that all is well, but in fact, is headed for disaster.

As a whole, it can be seen that the causes of software problems occur throughout the software development life cycle. As such, systematic and proactive approaches must be taken to overcome these problems at each stage of the software development. While these problems cannot be eliminated totally, some of them can be considerably reduced or better controlled by applying systematic risk management procedures and techniques, in the early stages and during the software development process.

1.7 The Importance of Software Risk Management Tool

In view of the above-mentioned problems, it is strongly believed that a useful and cost-effective risk management tool will be able to solve most of the software development process problems. This tool will help software organizations to:

- Minimize disasters: Risks that are not identified, improperly assessed or badly dealt with will cause software disaster. The organization will sustain the failure of the project: cost of the undertaking project will be increased and possibly overrun, and the deadline of the project will not be met. With software risk management tool, the project manager will be able to identify and rectify risk items before they become a reality and threaten the project, and thus keep a project from the disaster file.

- Avoid rework: Rework of an erroneous software project consumes 40 – 50 percent of the total cost of a software development [9], [48]. Software risk management tool can help to diminish these rework costs by accenting project efforts on the early identification and elimination of high-risk items. Avoiding rework also brings to increase in overall software productivity.
- Produce high quality and more reliable software products: Many software project managers spend much of their effort and money than is necessary on redundant and low-leverage activities, while neglecting the high-risk items. Risk management tool focuses efforts on the critical areas and thus ensures that products produced are reliable and of high quality.
- Manage risk confidentially: With software risk management tool, organizations will be able to manage their own risks explicitly and confidentially, with no worry that their organization's data will be exposed to competitors.
- Secure company reputations and customers goodwill: A single missed delivery or delay in development will reduce a company's creditability. Risk management tool helps avoid this situation by effectively eliminating risk constituencies that can affect the success of a project.

1.8 Structure of the Thesis

This document contains six chapters and six appendixes. This chapter, Chapter One provides the introduction to this research work. It defines various software risk definitions and software risk management; provides some ideas on the benefits of performing risk management and why it is not performed; and discusses the various problems currently faced by software development organizations. The root causes of these problems are also analyzed and briefly presented. It is then followed by the explanation on why there is a need to have software risk management tool.

The research objectives and the methodology are covered in Chapter Two. Chapter Three, the Literature Reviews, elaborates on some of the effective risk management techniques and presents a comparative look at the various currently available automated risk management software tools.

Chapter Four constitutes the heart of this work. The employment of a systematic and statistical based risk management technique is considered to be the most effective way to reduce/eliminate potential software problems. This has led to the development of a software risk management tool, A Statistical Manager, which forms one of the objectives of this research study. This chapter also presents the development methodology, the objectives and requirements of the tool, the representation of the design and testing strategies as well as screen captures of the tool.

Chapter Five focuses on the evaluations of the tool. The feedback of evaluations, tool characteristics and shortcomings, and possible future enhancements to overcome these shortcomings are discussed. It is then followed by the comparisons of this tool with other similar tools. This chapter concludes with the achievements of Statistical Manager with respect to the specified requirements.

Chapter Six, the Conclusions and Future Work, summarizes the work done and a look ahead to possible future works on the tool.

Bibliography presents a collection of references on software risk management and other subjects relevant to this project.

Appendix A provides definitions of terms relevant to risk management and also fields used in the Statistical Manager.

Appendix B contains the detailed design of all the classes in the Statistical Manager.

Appendix C provides the tool's screen snapshots and the detailed explanations to each of these screens.

Appendix D is the evaluation questionnaires.

Appendix E acquaints the reader with a collection of software risk management principles, myths and FAQ.

Appendix F concludes this document with a tool installation guide.