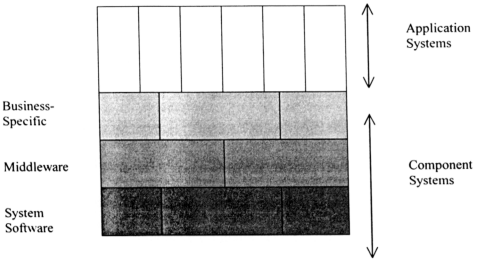
Chapter 3 System Architectural View and Analysis

3.0 Architectural View

The customisable production tool will be developed as a kind of layered system with the application systems on the top and the component system underneath as shown in Figure 3.1 [Jacobson, 1997]. A layered architecture is an architectural style that simplifies system evolution and enables substantial reuse of components [Jacobson et al., 1997]. The system development of this proposed customisable production planning tool is also based on the technique of software modelling developed by Ivar Jacobson [Jacobson et al., 1992]. The approach to this system will use Jacobson's use-case-driven Object-Oriented Software Engineering (OOSE) modelling notation and method [Jacobson et al., 1992]. The OOSE notation also conforms to the Unified Modeling Language (UML), [Booch et al.,1997] features. OOSE approach defines systems in terms of several object-oriented models which are the requirements model, the analysis model, the design model, the implementation model and the test model, which will be explained in the later section. This method is chosen as it produces significant levels of software reuse in the architecture and design of the software systems. Besides, this will also lead to a smoother implementation process and testing which ultimately produces a better and more maintainable system.

Figure 3.1 A Layered Architecture of Application and Component Systems



3.1 Object-Oriented (OO) Software Engineering Process

Object-orientation is an important approach to analysing problems, designing systems and building solutions. It is the process of systematically building systems using objects, types and classes. Through object-orientation, systems are designed in a more modular and extensible way enabling them to evolve more readily as user requirements changes. These software systems are composed of objects which have both data (state) and functional (behavioural) components. The ability to combine data structure and behaviour in a single entity is also known as encapsulation or information hiding, which can prevent a program from becoming so interdependent that a small change has massive ripple effects. These objects which can communicate with one another via messages, are categorised into classes with similar properties. Classes of these objects are themselves related in a hierarchy and therefore inherit general features from classes that are above them which are also known as super classes.

Through the use of objects, classes and message passing, object-orientation provides a more modular way of modelling problems. This in turn translates into clearer designs and ultimately leads to clearer implementations enabling better control of complexity with a significant reduction in maintenance costs.

Besides, inheritance provides a powerful means of reusing and creating specialisations as necessary. Specialisation refers to the fact that the subclasses refine or specialise the superclasses. By designing and building systems from existing reusable class hierarchies, the software engineering process can be reused from tried and tested classes perfected through use in previous developments. The benefits of OO engineering process which can also accommodate the dynamics of a changing environment, are very suitable for the development of a customisable OO tool for production planning because as requirements evolve over time especially so in the manufacturing industry, OO approach focuses first on identifying objects from the application domain, then fitting procedures around them, thus retaining the underlying framework of the application domain. The following sections will show how this OO approach is used to do the analysis and design of this customisable OO production planning tool.

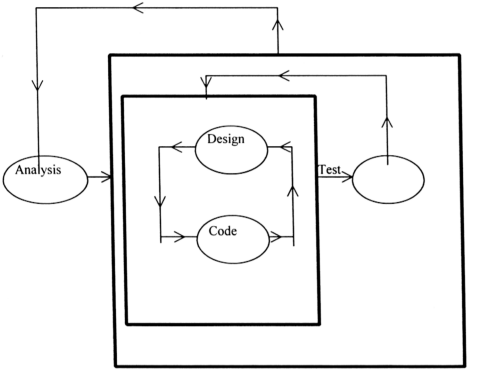
3.2 Object-Oriented Development Life Cycle

The development of this production planning tool uses an OO software development life cycle which is incremental and iterative in nature. Figure 3.2 [OMG,1992a] shows the evolutionary development life cycle which provides the best basis for OO analysis and design. Analysis is done in a top-down manner which results in a set of fairly high-level classes and the interactions between these classes. As the analysis

merges into design, lower-level class diagrams will be used to illustrate more internal details of the classes. The design cycle consists of a two-stage process that is the class design and application design. The class design involves developing functionality associated with specific classes and developing class hierarchies through abstraction of common data and function thus creating more generic classes. The class design involves both bottom-up and top-down procedures. The main purposes of this class design are to design for reuse and to design in a highly modular way. The process of identifying common behaviour or abstraction and specialisation creates better reuse possibilities. The application design is then achieved through bottom-up by connecting instances of the design classes so that they interact with each other resulting in the solutions to the stated requirements.

The main feature of this incremental model is that the design and code loop in the center provides a basis for fast innovative code creation. This produces quick versions of the production planning application which can be evaluated and fed back into cycle for further design which is the essence of the prototyping life cycle. This rapid prototyping is a highly iterative process which can be used to reduce risks in software development. Implementation of classes then takes places in parallel with much of the design work. Testing procedures then lead to results which must conform to the specified requirements otherwise they will be fed back for redesign and analysis. This process thus uses an incremental and iterative cycle which will be refined until all requirements are met.

Figure 3.2 Overview of the Production Planning Development Cycle [OMG. 1992a]



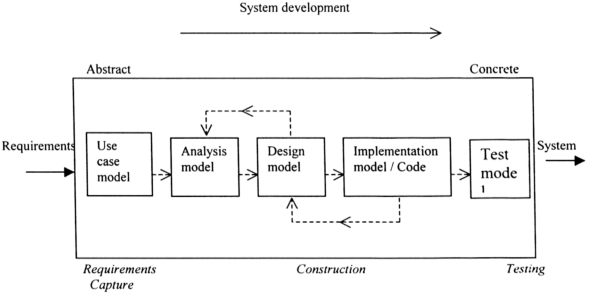
3.3 System Development Activities

The OO system development for this customisable production planning tool, as mentioned previously is based on the OO software engineering method modelled by Jacobson, 1992 [Jacobson et al., 1992]. This method encompasses analysis and design producing five different models from abstract to building a concrete system. Figure 3.3 [Jacobson, 1997] shows a summary of the development activities. The object models are :

- The requirements model – which captures the functional requirements through the use-case model.

- The analysis model – whose aim is to develop a sound, stable and extensible object structure.
- The design model – which refines the object structure to suit the current implementation environment.
- The implementation model – which is used in implementing the system.
- The test model – to verify the system.

Figure 3.3 Summary of the Development Activities



The different models and the elements defined in the respective models are connected to each other through traces shown in the dotted arrows in Figure 3.3. The figure also shows a summary of the activities in each of the phases of the development mentioned and their respective models of each phase in the whole development process. The links allows the developer to track any changes from requirements through to the different

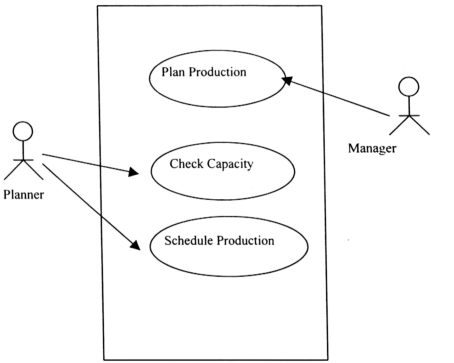
models all the way to coding. This will help in the analysis on the impact of any changes to be done. The system starts from the abstraction of the initial classes with the requirements defined in the analysis phase. During this phase, the essence of some domain within the real world is abstracted to provide a model from which the planning tool will be ultimately developed. In this case, a high-level use case model is used to capture the planning requirements for the customisable production planning tool. It identifies scenarios in which the planning processes are activated and controlled. This leads to the analysis model which then identifies the objects and classes based on the requirement specifications about what the system will do. The use cases are refined and made more detail and robust. The design model shows the interaction between the classes and subclasses which further refine the analysis model to the actual implementation environment. Detailed interaction diagrams are developed in the application design to show how the objects will carry out their responsibilities through the respective functions or methods to be implemented. Further stages involve developing the analysis, design and implementation models iteratively when more details are uncovered until the system development is complete with the testing model verified. The incremental and iterative analysis and design methods for the customisable production planning tool will be further explained in more detail in the next section.

3.4 Requirements Model Through Use Case Model

The development of the customisable production planning tool begins with a high-level use case model to capture the system's functional requirements. Figure 3.4 illustrates the use case model for the customisable production planning tool. The scenario shows two

actors, namely the manager and planner who are both directly involved in production planning.

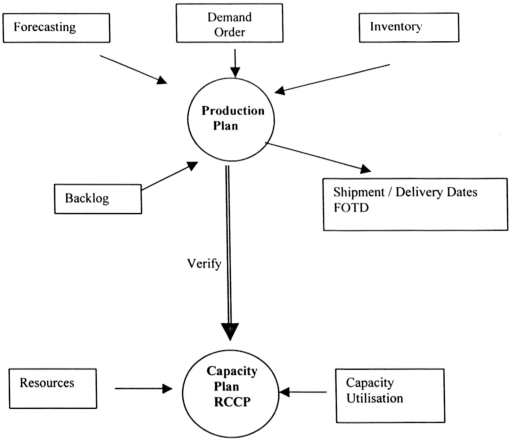
Figure 3.4 Use Case Diagram for Production Planning Tool



The use case model represents a high-level production plan that is controlled by the planning manager. For this plan to materialise, inputs to the plan must first be sorted out. These inputs are the components of the production plan. Figure 3.5 shows the components of the overall production plan. The components consist of inventory, demand order, forecasting, backlog and shipment. These inputs are all netted into the production plan together with the management policies before the plan is checked against sufficient rough cut capacity planning (RCCP) in order for production to proceed. The RCCP shows the resources such as machines, manpower and work hours that are available for the realisation

of this plan. This ultimately shows how much capacity is being utilised. The manager will have to consider all these components needed to plan for production to be executed by the planner. In the mean time, the planner's role is to check the capacity of the plant to ensure that there is sufficient capacity for the plan to proceed. This is shown in the planner's use case. If there is readily available capacity, the planner will then proceed with the production plan. He will then schedule the production quantity according to the plan. Otherwise, the planner will need to refer back to the planning manager about the insufficient capacity and the latter will need to make whatever adjustments to scale down the production plan.

Figure 3.5 Components of a Production Plan



In short, the use case model defines what the system should do for the users. The model specifies precisely how responsibilities are allocated to actors and their interactions with instances of the use cases. It models the aspect of the manufacturing domain that concerns the users and solves the production planning problem. The next section looks into the analysis model of this system followed by the design and the architecture of the production planning tool.

3.5 Analysis Model

The analysis model is a model of the system design at a high level, ignoring the specific low-level details of the target implementation environment. The analysis process evolves in a systematic model building that is based on the component systems of the planning tool. This analysis model is built upon a widely reusable structure that is robust when faced with the ever changing requirements within the manufacturing environment. A good application system with a flexible and scalable architecture for this production planning system is the aim of this object-oriented software engineering process. To achieve this, the analysis model considers how modifications will affect the system. In this stage, the use cases are refined and made more detailed. Each part of the system undertakes its responsibility according to the use case model represented by each component.

The model makes use of three types of objects to elaborate each use case and their respective interactions with the component systems. These analysis types represent an abstraction of classes in the system's implementation. Figure 3.6 shows three different kinds of analysis types (BCE) [Jacobson et al., 1997] which are :

i) Boundary (B) or Interface object which handles communication between the system and its environment.

ii) Control (C) object performs use case specific behaviour which controls or coordinate other objects.

iii) Entity (E) object which holds information about the system, often used to model business objects.

Figure 3.6 Three different kinds of analysis types (BCE)

Boundary/ Interface | Control | Entity

These three distinct types, BCE help in developing a more robust structure in this analysis model. In the development of this production planning tool, every component system is represented by its own respective use case model, which in turn has its own boundary, control and entity types to illustrate how objects perform the use case functions.

The analysis model gives shape to the production planning system's architecture. This model is made up of the analysis types and subsystems including their relationships through traces. A dynamic analysis model can lead to an ideal design and a smooth implementation environment. The problem domain when clearly stated in this analysis model will lead to a development structure that allows the system to evolve when requirements are added in the later part of the object-oriented software engineering cycle.

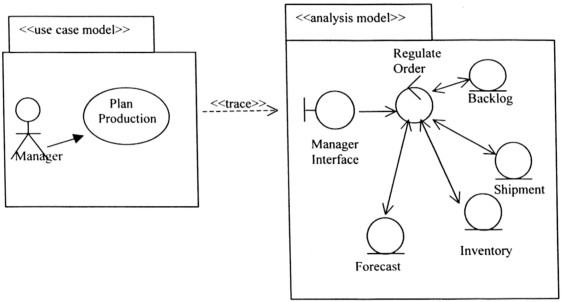
In this planning tool, for each use case, there is a collaboration diagram that illustrates how objects perform the use case. The use case model is traced by a dash line between the use case and the BCE types.

There are three specific use cases namely, plan production, check capacity and schedule production. The primary modelling activity during this analysis phase is to identify the various types and their relationships and to allocate responsibilities to these types. The responsibilities need to be allocated so that all the steps in instances of the use case are performed by interactions between the objects. A collaboration diagram is used to show these interactions in this analysis model.

3.5.1 Plan Production Use Case

The collaboration diagram to trace the Plan Production use case model is shown in Figure 3.7. The use case model is traced to the analysis model with a dashed line. The boundary object which is the *Manager Interface* interacts with the system to plan for production. The control object, *Regulate Order*, carries out the use case specific behaviour, which in this instance, is to confirm customers' orders and ensure that the production plan is sufficient to meet those orders. This control object interacts with the entity objects so that an accurate plan can be drawn up so as to fulfill all orders since the plan is actually controlled by the customers' demand order. The entity objects in this case are the four objects in the system which are *Backlog*, *Shipment*, *Inventory* and *Forecast*. *Backlog* takes into account those orders which have not been delivered. This could be due to delay in production or insufficient resources. *Shipment* includes when goods must be shipped after an order is taken from the customer. This delivery date is important so that they will be no delay to the order. Inventory is needed to buffer any sudden demand and production loss so that there will be no disruption to the production plan. *Forecast* will enable the system to predict the demand orders so that production goes according to predicted future demand orders. All the entity objects are important inputs to produce an exact production plan.

Figure 3.7 A collaboration diagram to trace the Plan Production Use Case Model

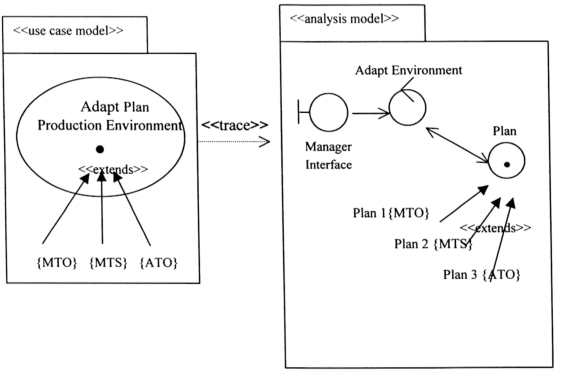


3.5.1 Customisation Through Variation Points

In planning for production, the system allows the manager to set a production environment of his choice. In this scenario, there are three production environments namely, Make-To-Order (MTO), Make-To-Stock (MTS), and Assemble-To-Order (ATO). MTO mode is chosen when the user plans for production according to customers' orders which are usually given in advance before production starts. Whereas MTS is chosen when a plant wants to shorten the delivery lead time by producing with increased inventory. Unlike ATO, is for the production of semi-finished products. The customisable planning tool allows the manager to choose from one of the three production environments in a certain production year which can be set at any date. The three modes of production are shown as three variant points in the collaboration diagram in Figure 3.8, indicating specialisation according to the three different environments. Specialisation technique is a

form of <<generalisation>> that refers to the variability mechanisms [Jacobson et al., 1992; Griss, 1995d]. In this instance, the variability mechanism use extension points to express three variants attached at a variation point in the **Adapt Plan Production Environment** use case and object components. An interface will be provided for the manager to select which environment he so chooses at that particular production period.

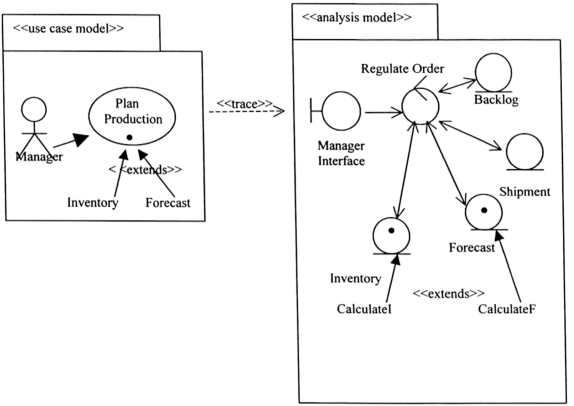
Figure 3.8 A collaboration diagram to trace the Plan Production Environment Use Case Model



Another customisable feature of the planning tool is the calculation of inventory and forecast which are entities of the production plan. The manager can calculate both the entities according to their own formulae through another interface where he can customise his own inventory and forecast. Another two variation points will be included into the plan

production use case as shown in Figure 3.9. The two variations for *Inventory* and *Forecast* are *CalculateI* and *CalculateF* respectively. The variability is again expressed as <<extensions>> that allows users to customise their own calculation of inventory and forecasting methods.

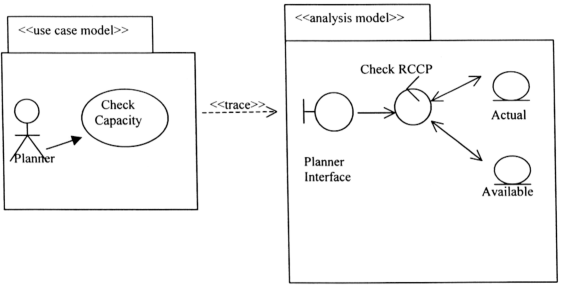
Figure 3.9 A collaboration diagram to trace the Plan Production Use Case Model with Variation Points



3.5.2 Check Capacity Use Case

A collaboration diagram is again used to trace the **Check Capacity** use case model. Figure 3.10 shows the analysis model to trace this particular use case. The use case instances interact through the BCE types in order to verify the rough cut capacity planning (RCCP). The control object, *Check RCCP* checks the actual capacity against the available capacity so as to determine whether there is sufficient capacity for production to proceed.

Figure 3.10 A collaboration diagram to trace the Check Capacity use case model



The boundary object which is the *Planner Interface*, handles communication between the system and its surrounding which is the planner who is given the access to interact with the capacity planning (RCCP). The control object, that is *Check RCCP* carries out the use case specific behaviour, which in this instance, is to check for capacity. The entity objects in this case are the two objects in the system, *Available* and *Actual* capacity. The *Available* entity defines the capacity that is available in terms of resources. Unlike the *Actual* entity,

which shows the actual capacity needed for this plan to be carried out. If the *Actual* capacity is below the *Available* capacity, then this will result in the plant operating under capacity, thus resources are under utilised. This is an indication for the plant to either cut down their resources or increase production accordingly. If there is readily available capacity, the planner will then proceed with the production plan. Otherwise, the adjustments must be made to scale down the capacity plan. The dashed line is a trace between the use case and the analysis types.

3.5.3 Schedule Production Use Case

In this use case, the planner again interfaces with the system after the RCCP has been verified against the plan and the plan now is ready to be executed. The control object, which is the *Plan Schedule* now determines the scheduling activities. The entity objects are represented by the schedule, type of product and the production week. The production schedule will run according to the plan. The production plan is done in weeks and in each production week, the schedule is again different for the different types of products to be produced. These products are in consistent to the customers' demand for the various types of products. The planner will schedule the production based on the control object which is the plan as shown in Figure 3.11.

Figure 3.11 A collaboration diagram to trace the Schedule Production use case model