Process Modelling to Support Software Development under the Capability Maturity Model

Kann-Jang Yang

Doctor of Philosophy
University of Edinburgh
1998

To My Father,
who died on January 5 1996.

Abstract

In 1913, the Ford Motor Company successfully introduced the assembly-line mass production of motor vehicles. The success of mass production came in the concept of interchangeable parts and manufacturing processes. Currently, after struggling with "software crisis" in the last decades, the software community is trying to imitate the concept of mass production.

Problems arise from the characteristics of software. Software is a logical rather than a physical system element. We develop software products but not manufacture them. However, researchers still believe in the practices used for assembling previously existing components into large software systems.

Before the technique of software components is mature, we believe that the software process is another essential topic for "manufacturing software products". The steps in the software process must be defined very precisely and carefully. Process-centred Software Engineering Environments (PSEEs) are viewed by many as a way to assist developers in the execution of their work. Research has produced a variety of PSEEs providing support for management and technical activities. However, it is hard to say which process is the most appropriate one.

The Capability Maturity Model for Software (CMM), developed by Software Engineering Institute in Carnegie Mellon University, provides software organisations with guidance on how to gain control of their processes for developing and maintaining software. For the last few years, some organisations have successfully improved their software process maturity by using the CMM.

This research builds a PSEE, called SPI (Software Process Improvement) PASTA , that models the CMM by using the process notation PASTA (Process and Artifact State Machine Transition Abstraction). There are two reasons for doing this research. Firstly, we believe that a PSEE must comply with a framework of continuous process improvement, such as the CMM, in order to improve project management in software organisations. Secondly, in any context in which the CMM is applied, a reasonable interpretation of the practices should be used. The CMM must be appropriately interpreted for different size projects and software organisations.

The SPI PASTA provides a framework for continuous improvement of the process. This framework complies with a supporting knowledge transfer and

implementation services architecture that makes it possible to achieve higher software process maturity. Therefore, the software organisation's productivity and quality can be improved over time through consistent gains in the discipline achieved by using the SPI PASTA. Furthermore, by means of a CMM-based appraisal method, the state of an organisation applying the SPI PASTA will be determined. As a result, the organisation's software process can be continuously improved.

Acknowledgements

After four years of PhD research, I can now add the final and most enjoyable part to my thesis; it is in these lines I can finally thank the many people that have enabled me to produce this work, by providing the fruitful environment that I have worked in for the last four years.

Firstly, I would like to use this thesis to memorialise my father who died on 5 January 1996. Without his support, I could not have finished my research career. Furthermore, I have to thank my mother, Hsin Yang-Huang, and my wife, Pei-Pei Tsai, whose patience, love and sacrifices made this dream come true. Next, special thanks go to Kirsti Withell and Paul Coe, who gave me such useful help in doing this research, and to Robert Lai, who gave me his continued help and encouragement as I was first learning the software process. I would also like to acknowledge the help and encouragement I have received from my friends, Marcio Fernandes, Ana Goldenburg, Nils Knafla, Rob Payne, Isabel Rojas-Mujica, Perdita Stevens, Thomas Zurek and so many more who deserve to be mentioned.

I am deeply indebted to my supervisor, Dr. Rob Pooley, who contributed so much to the numerous discussions that we had about my work. Supervising a foreign student cannot be an easy job; he must put in additional effort to improve my writing and listen carefully to my strange accent. Without his patience, I would not have completed this thesis, and for all this I am extremely grateful.

Thanks to all of you!

Table of Contents

Chapter 1

Introduction

On the 4th June 1996, the maiden flight of the Ariane 5 launcher ended in failure. Only 40 seconds after initiation of the flight sequence, at an altitude of about 3700 m, the launcher veered off its flight path, broke up and exploded. An independent Inquiry Board was immediately set up and finally submitted its report[LIO96]. The report indicated:

> *The failure of the Ariane 501 was caused by the complete loss of guidance and attitude information 37 seconds after start of the main engine ignition sequence (30 seconds after lift-off). This loss of information was due to specification and design errors in the software of the inertial reference system.*
>
> *The extensive reviews and tests carried out during the Ariane 5 Development Programme did not include adequate analysis and testing of the inertial reference system or of the complete flight control system, which could have detected the potential failure.*

On the 14th September 1997, Microsoft announced that it would delay until spring the release of Windows 98. The product, which features tight integration with the company's Internet Explorer software, is strategically important. Microsoft wants to make Windows 98 a key weapon in its battle with Netscape Communications to provide the main software package for using the Internet. The company will release Windows 98 in June 1998 rather than in March. Consequently, Wall Street traders pushed Microsoft's stock down 5 percent.

Software development has been a troublesome technology for a long time. A majority of large software projects tend to run late or out of control, or to fail to meet their target. The two cases above are good examples where a minor mistake could destroy the whole project or cost companies considerable sums. In this thesis, we try to help software organisations by building a software process framework, called SPI (Software Process Improvement) PASTA , that models the

CMM by using the process notation PASTA (Process and Artifact State Machine Transition Abstraction).

1.1 Is software development tricky?

In the first Industrial Revolution, the process of change from an agrarian, handicraft economy to one dominated by industry and machine manufacture had been well prepared for a long time. Towards the end of the nineteenth century, the second Industrial Revolution was linked with a sharp increase in scale of production and the size of companies. Today, the third Industrial Revolution is taking place. Electronic and computer-based technologies rapidly shift the locus of economic and industrial power.

Each Industrial Revolution brings new management theories to the new enterprise. For the first and second wave, people took a long time to adjust to the new idea. However, for the third wave, it seems that we have not enough time to sort out how to enter the digital era, especially in the software industry, a totally new industry for human beings.

Basically, software is a logical rather than a physical system element[Pre94]. This is the biggest difference between software and traditional mass production industries. In the early years, software programmers were viewed as craftsmen. They built software products for special customers. However, software systems are increasingly big and complex. Software development is not a craft any more. It is team work.

From his study, Jones[Jon96] described how a significant percentage of projects are cancelled before completion, fail to deliver expected features, run over budget and overshoot schedules. Table 1.1 shows the approximate frequency of various kinds of outcomes, based on the overall size of the project being attempted.

	Project Size Expressed in Function Points			
Project Outcome	< 100	100-1000	1000-5000	> 5000
Cancelled	3%	7%	13%	24%
Late by > 12 months	1%	10%	12%	18%
Late by > 6 months	9%	24%	35%	37%
Approximately on-time	72%	53%	37%	20%
Earlier than expected	15%	6%	3%	1%

Table 1.1: Software Project Outcome By Size of Project[Jon96]

Table 1.1 clearly shows that only one fifth of projects, which are larger than

5000 function points, are completed on time. This percentage is far less than for small projects. From this survey, conclude that we have to gain an understanding of the properties of software. What is the main reason for this "software crisis"?

Kraut and Streeter[KS95b] suggested that uncertainty is one of characteristics of software development. They listed the following:

- *Unlike much manufacturing, software development is a nonroutine activity.*

  The lifecycle of software development is not clear. The classic "waterfall" lifecycle model has been criticised for a long time since real projects rarely follow the sequential flow. Booch[Boo96] described real processes as both cyclic and opportunistic. This means that a well-managed iterative and incremental development life cycle might be a good paradigm for software development.

- *Uncertainty increases because specifications of the software's functionality change over time.*

  Bersoff[BHS80] said that no matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle. Change always happens when software is used by end users as this is the time when end users understand software's capabilities and limitations.

- *Software development is uncertain because specifications for it are invariably incomplete.*

  The software engineering textbook usually describes how software requirements, such as major functions, interfaces and information, must be fully understood before successive layers of detail are specified. However, the true story is that too few people working on a software project have sufficient knowledge about the domain in which they are working. Analysts with varying degrees of domain knowledge interview customers and users. The specifications seem to be inevitably incomplete. Even worse, some of the users' requirements will not be uncovered until the product is released. This is really a nightmare for software developers.

- *Software is uncertain because the different subgroups involved in its development often have different beliefs about what it should do and how it should do it.*

Professionals generally develop their products by using their own methods and techniques. They might do a good job; however, without disciplined frameworks, the project could become out of control.

Therefore, software development is very different from traditional industries. We can not simply adopt traditional management theories to fit the software industry. We have to discover a new method to enter this new digital era.

1.2 Motivation

Cusumano[Cus91] in his book described the concept of the software factory. With the increasingly popular Internet, the virtual corporation is becoming reality. This effort is building large international systems with multi-national participation. However, to create these system, it needs a common international framework for specifying the best of practices for software processes, activities, and tasks. Without this commitment, developers will struggle with communicating each other.

The effort of software process improvement has been verified to be a good solution to resolve the problems[PW96, HIW+95, DS97]. With this consensus, those multi-national participants may follow the rule to develop software projects. To run an international software project, Microsoft's secrets[CS95, CS97] can provide an effective method for those software organisations. This "synch-and-stabilise" approach complying with disciplined process will effectively and efficiently develop a big software project.

1.2.1 Software Process Improvement

In recent years, process modelling has become one of hottest topics in the issue of software engineering. Researchers focus on software processes which can effectively combine related staff, resources and activities. As mentioned in Section 1.1, the software community faces significant difficulties. Researchers have been trying to tackle these problems. Their efforts include object-oriented methodology[BJR97], Frameworks[FS97], Patterns[GHJV94] and so on. The goal of each effort is to provide an effective and efficient method to help the software community complete their mission. However, software projects are not only about technology but about management. Methodologies, tools and people all influence software development.

For the last decade, the software community has focused on the area of software process improvement, in particular for those who contract to governments.

An organisation such as NASA has to develop, maintain and manage complex flight systems. It is very important to develop a continual process improvement approach that allows NASA to fine tune its process for its particular domain. As a result, the Software Engineering Laboratory (SEL) was created in 1976 in NASA and concentrated on software process improvement for the purpose of understanding and improving the overall software process and products that were being created within the Flight Dynamics Division (FDD)[MPB94]. The SEL's recently completed 1996 organisational baseline shows across-the-board improvement in all measurement[PW96].

- Average mission costs decreased by 15% when compared with the 1993 baseline, totalling a 60% overall reduction in mission costs since 1985.
- The cost of developing a line of new code has decreased by nearly 35% since 1993.
- Ground system projects saw a modest 7% reduction in project cycle time, while simulators experienced a 20% reduction since 1993.
- Error rates continued to drop, with a 40% reduction in development error rates since 1993. This combines with earlier improvement to total an 85% drop in development error rates over the past 10 years.

The impacts of these process changes are evident in the resulting characteristics of FDD products. This results in a belief that software products can be improved by optimising the software engineering process used to develop them.

In 1987, the Software Engineering Institute (SEI) in Carnegie Mellon University released a software process maturity framework and maturity questionnaire to support organisations in improving their software process[PCCW93a, KCF+96, Pau95]. Four years later, the SEI released the Capability Maturity Model for Software (SW-CMM or CMM)[PCCW93b, PWG+93]. Since then, the CMM has become an important guide to help software organisations select process improvement strategies.

The CMM was originally developed to assist the U.S. Department of Defence (DoD) in software acquisition. However, the use of the CMM swiftly pervaded the wider software engineering community. Not only the DoD contracting community, but also commercial organisations adopted the CMM as a framework for their own internal improvement initiatives and gained significant benefit from the CMM.

Ratheon Electronic Systems (RES) began its software improvement activities in 1988, driven by compelling business reasons to improve the cost and schedule

SEL CMM Level	Number of Projects	Quality†	Cycle Time (X factor)	Productivity (Relative)
1	3	n/a	1.0	n/a
2	9	890	3.2	1.0
3	5	411	2.7	0.8
4	8	205	5.0	2.3
5	9	126	7.8	2.8
†In-Process Defects/Million assembly-equivalent lines of code				

Table 1.2: Motorola GED Project Performance by SEI CMM Level[DS97]

predictability of its major business areas' software components. These activities, guided by the CMM, include[Hal96]:

- to establish a strong and effective software process infrastructure for continuous improvement and to maintain the team's enthusiasm over time, and
- to measure and analyse process and project data to quantity the benefits of software process improvement.

In eight years, Raytheon has demonstrated significant improvements to its software engineering process. During the period, productivity of the development staff has increased by a factor of almost 2.8, and predictability of their development budget and schedule has been reduced to a range of +/- 3%[HIW+95].

Motorola[DS97] has long been a famous organisation which has adopted the CMM as a vehicle for software process improvement. In November 1995, the company's Government Electronics Division was independently assessed at SEI level 4. Table 1.2 summarises the Motorola GED improvement trends for quality, cycle time and productivity by SEI level. In typical projects on level 4, Motorola achieves a 5-fold reduction in product cycle time to accelerate the introduction of new products, and a 4-fold reduction in defects and 2.3 time productivity than projects on level 2.

This achievement brought Motorola a remarkable return on investment and implied that there is a good business case for those who follow the SEI software process improvement approach.

There are other samples from the survey conducted by the SEI[CLMZ96, CLM+97]. The result of the survey was a mixture, in particular for those organisations which appeared to be of low process maturity. Nevertheless, the benefits of adopting the CMM are significant.

However, some research has different opinions. Fayad[FC96] argued that adopting the CMM recommended practices is not especially easy and smaller or-

ganisations cannot afford the two- to three-year duration it normally takes to reach CMM level 3. In addition, the CMM is continuously being questioned by the software community. One of the common complaints concerning the CMM is the organisation of the information contained within the document. References to software process practice for a given CMM Key Process Area (KPA) are not located in the same section but are dispersed throughout the document. This makes the job of reviewing a defined process against the recommendations made by the CMM particularly difficult for an organisation trying to improve their CMM maturity rating or trying to define a CMM consistent process[AES95].

To resolve these arguments, the SEI Software Process Definition Project has developed the Software Process Framework (SPF) to support users access to the process maturity criteria, or key practices established in the CMM[ORO94]. The purposes of the SPF are[Gat97]:

- to present information recommended by the CMM in a format that is convenient for software process definition tasks,
- to identify the policies, standards, processes, procedures, training, and tools recommended by the CMM,
- to provide checklists for ensuring that process documents are consistent with the CMM.

The SPF comprises a set of templates derived from the CMM and maps all CMM KPA specific recommendations. The policies and standards checklists are used to verify that policies and standards are in place to guide the use of the process. Furthermore, the process checklists are used to review and analyse software process documents. However, for software development, the software organisations have to cover all the software processes which include[Pau97]:

- software technical processes
- software support processes
- software management processes
- organisational processes

These four processes, shown in Figure 1.1, provide services to and support the work required by one another. The main problem is that software organisations have to invest in considerable resources to fit the SEI's process improvement activities. This problem is not only for large organisations but also for small

companies which might be dispersed teams connected by the Internet and/or intranet.

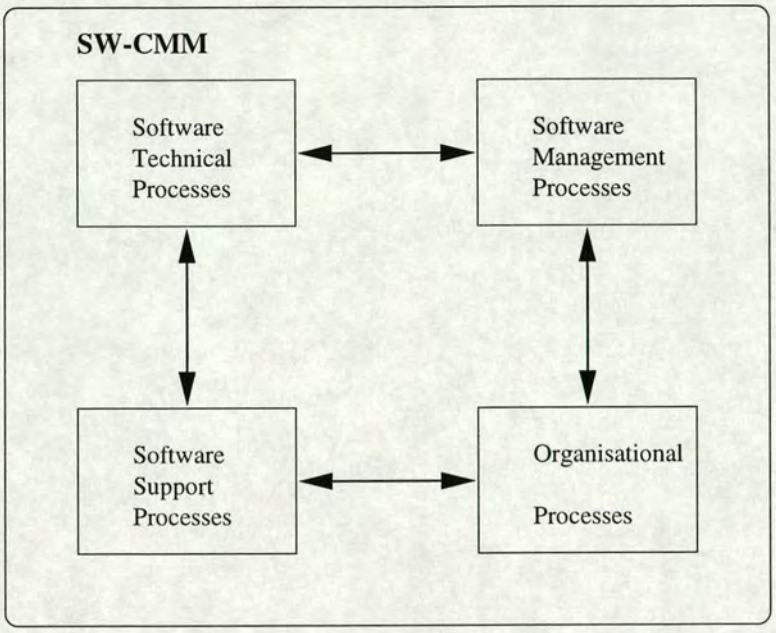

Figure 1.1: The Process Architecture of the CMM

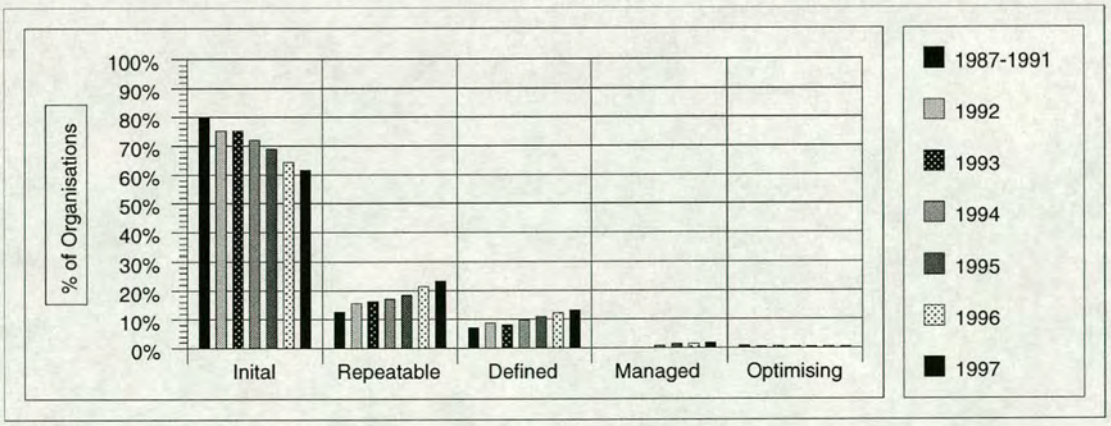

Figure 1.2: Trends in the Community Maturity Profile

However, as Figure 1.2[SEI97] shows, only 16 % of assessed organisations (13.9% on Level 3, 2.1% on Level 4 and 0.3% on Level 5) could reach higher than maturity Level 3 in 1997. This shows that adopting the CMM recommended practices is a big challenge for software organisations. The process maturity profile surveyed by the SEI[SEI97] showed that the time taken to move from maturity level 1 to 2 is 32 months for all organisations and 27 months for organisations that began their CMM-based SPI effort in 1992 or later. All groupings exhibit a

similar pattern for moving from maturity level 1 to 2 and level 2 to 3. This means that an organisation might take four or five years to achieve Level 3. CMM-based SPI is not a cheap nor a quick solution.

Diaz and Sligo concluded and found some reasons to explain why in lower maturity organisations it is much more difficult to implement software process improvements[DS97]:

- Keying process changes to metric analysis data is not addressed until CMM levels 4 and 5. Such data is critical to improving the effectiveness of SPI efforts.
- Lower maturity organisations focus on defining their core processes, not on improvement.
- Lower maturity organisations are just starting to improve their software processes. This requires significant effort, especially in the beginning.

We are faced with some basic problems in promoting CMM to software organisations. Although the SPF provides a framework to check the processes which are used in software organisations, it is still like a roadmap and cannot tell you how to design or how to analyse software process documents. There is still a gap between the CMM and the SPF. In this thesis, we try to use SPI PASTA to assist lower maturity organisations to implement software process improvement. By using SPI PASTA, those lower maturity organisations may easily apply key practices recommended by the CMM. This will significantly reduce their effort and earlier form a base to achieve higher maturity.

1.2.2 Japan's Software Factory

In 1913, the Ford Motor Company successfully introduced the assembly-line mass production of motor vehicles. The concept of mass production has dominated the automotive industry. Mass production methods are based on two general principles: the division and specialisation of labour and the use of tools, machinery and other equipment in the process combining precision, standardisation, interchangeability, synchronisation and continuity.

In the 1970's several Japanese firms, led by the Toyota Motor Corporation, developed radically different approaches to the management of inventories. By relying on careful scheduling and the coordination of supplies, just-in-time management ensured that parts and supplies were available in the right quantity, with proper quality, at the exact time they were needed in the manufacturing

or assembly process[Mon83]. Since then, the concepts of mass production and just-in-time management have influenced the software industry. The idea of the "software factory" has been widely discussed in the literature[Man84, TT84]. Cusumano[Cus91] studied some companies, such as IBM, System Development Corporation (SDC) and General Telephone and Electric (GTE) which pioneered variations of software factory approaches during the 1970's. The common activities of these companies were incorporated into a standardised set of engineering methods, controls and support tools. These standards specified a hierarchical architecture for all software systems, a formal engineering process based on a common life-cycle model, a list of required documents for each phase and a glossary of terms and symbols for developing programs. However, most of the companies abandoned their efforts after a couple of years of operation since information technology can not well support the concept of software factory.

Despite the unsuccessful implementations by American companies, we have to ask whether the concepts of "factory" provide solutions to problems in software production at all. In industries such as automobile manufacture, the de-skilling or routinsation of work, high levels of control over production tasks and work flows, division and specialisation of labour, interchangeable parts and automation have well been implemented. However, with respect to the software industry, the highly skilled programmers, wide variations in project contents and work flows, unclear requirements of the customer, ... were the major obstacles in implementation of factory concepts. In his book, Pressman[Pre94] suggested that software is developed or engineered rather than manufactured in the classical sense. This means that software projects cannot be managed as if they were manufacturing projects. Besides, in mass production, the use of interchangeable components is necessary and essential. Without standardisation of parts, mass production would be impossible to implement. However, some research has recently focused on this field:

- Object-Oriented Frameworks

  > *A framework is a reusable, "semi-complete" application that can be specialised to produce custom applications[FS97].*
  >
  > *Mattsson also defined an object-oriented framework as a (generative) architecture designed for maximum reuse, represented as a collective set of abstract and concrete classes; encapsulated potential behaviour for subclassed specialisations[Mat96].*

  The difference between an object-oriented framework and a class library is that an OO framework is targeted for particular business units and applic-

ation domains. A framework contains the basic application structure that application programmers previously had to develop on their own. By starting with a business framework, applications can be created more rapidly since developers have only to concentrate their development efforts on the unique differentiators they need for their customers.

Research into object-oriented frameworks has boomed since the introduction of object technology and a matured distributed environment. IBM's Commercial Shareable Frameworks initiative[Boh97], also called project San Francisco, is trying to build server-side core business process components that can be reused as a base for creating applications for specific industry domains. San Francisco will restructure the way applications can be built and sold by providing about 40% of a typical working application within the supported domains. ISVs (Independent Software Vendors) would develop the remaining 60% of the application business processes and services on top of San Francisco and bundle both the IBM and ISV code into a single solution which the ISV will then sell to customers.

Meanwhile, the Software Engineering Institute (SEI) in Carnegie Mellon University built the product line system[BC96, CFM+96, Wit96], a group of products sharing a common, managed set of features satisfying specific needs of a selected market or mission. All products in a product line share a common architecture[CN96] and control the variability inherent in a family of similar systems. The work of the product line system is focused in three areas: Domain Engineering, Software Architecture and Reengineering. Through careful management and engineering, a product line can be developed that exploits a common set of assets, ranging from reusable software components to work breakdown structures for individual projects.

Both systems share the same idea of developing a framework or an architecture with reusable components as a foundation for a specific application domain. By implementing a "standardised variety" approach, organisations can build a production system that can support the concurrent development of software for multiple projects. As in the concept of just-in-time management (a close coordination of information and plans with suppliers and vendors), project managers focus on domain engineering and receive components at the last minute.

- Distributed and Concurrent Development

  This is another type of just-in-time management. The concept of distrib-

uted and concurrent development arose because of the Internet. Virtual collaboration over the Internet is a new paradigm of software development. It enables multiple small teams, geographically distributed, to concurrently develop multiple functions for a family of large-scale software systems. Projects such as the Agile Software Process Model[Aoy93, Aoy90, Aoy97] and Fujitsu's Distributed and Concurrent Development Environment[NFK+97] paid attention to distributed development processes. To implement the concept, the Japanese developed a cyclic enaction model in which development of each enhancement has to be completed in a fixed time period, and iterated over multiple releases. The key point of the model is precisely controlling the process over multiple releases, since each process instance has to meet an exact development schedule. This is the key concept of just-in-time management.

To implement the concept of just-in-time management, software organisations can benefit from the use of object-oriented technology. Booch[Boo93] gave an object a definition:

> *An object has state, behaviour and identity; the structure and behaviour of similar objects are defined in their common class; the terms instance and object are interchangeable.*

Hence, the features of objects support the ability to create systems composed of independent parts and systems that can be extended without duplicating effort. However, if you are going to see the benefits of object-oriented technology, you have to create a work style that promotes identifying and working on well-defined and manageable system components. This is the crucial point which the Japanese software factory focuses on. By using components, software engineers can develop and release projects incrementally, create subteam structures that encourage parallel work and promote reuse opportunities.

Both approaches have to face crucial problems. Two of the toughest problems are *domain engineering* in frameworks and *partition* in distributed and concurrent development. In the framework approach, the first activity is to define the conceptual framework. This activity is based on a functional decomposition. In distributed and concurrent development, a good partition might closely coordinate the project manager (vendor) and developers (suppliers) and complete the products on time. Consequently, division of the software project into work components is a crucial task.

By surveying the Japanese software industry, Cusumano[Cus91] found that Japanese software companies attempted the *strategic management and integration* of activities required in software production, as well as *the achievement of planned economies of scope.* Cost reduction or productivity gains came from developing a series of products with one firm which is more efficient than building each product from scratch in a separate project, and planned scope economies required the deliberate sharing of resources across different projects. To manage their projects, Cusumano explained that Japanese software companies focused on several common elements[Cus91]:

- Commitment to process improvement
- Product-process focus and segmentation
- Tailored and centralised process R&D
- Skill standardisation and leverage
- Dynamic standardisation
- Systematic reusability
- Computer-aided tools and integration
- Incremental product/variety improvement

From this survey, we can determine some of the key points in the Japanese software industry. As in other industries, Japanese software producers first concentrated on process and quality control and then on process improvement. The managers who established software factories all believed they could improve software operations by using an institutionalised software process and quality control. In the meantime, the SEI's CMM might be viewed as the same issue[PCCW93b]. The CMM recommends institutionalising measures and procedures based on historical performance and statistical analysis as part of an organisational culture. Japanese software producers developed tailored processes for particular types of software products. The Japanese software companies established the organisation's set of standard software processes with centralised tools and methodology above the level of individual projects. The companies tried to establish baselines for software developers and product quality through a product focus and a standard process, as well as training in standard sets of tools, methods and management procedures. As a result, the variability of personnel skill does not change the progress of a project too much, since all activities are controlled by a baseline. This

concept was suited to the CMM which, in level 3, integrates the software engineering and management activities into a coherent, defined software process that is tailored from the organisation's set of standard software processes.

1.2.3 Microsoft Secrets

In their research, Cusumano and Selby [CS95, CS97] described how Microsoft uses the "synch-and-stabilise" approach to product development. First, Microsoft teams try to understand users' needs and structure those needs into individual features. They then assign priorities to these features and allocate them to subprojects that break up a development project into three or four milestone periods (builds). Microsoft managers also try to fix project resources - limiting developers and development time in any one project. The intended shipment date causes the whole development team to bound its creativity and effort.

Figure 1.3 shows that the life cycle contains three phases; planning, development and stabilisation. The planning phase takes three to twelve months, depending on the features of the project. The development phase takes six to twelve months and generally comprises three or four major milestone product releases. The stabilisation phase takes another three to eight months and comprises testing, buffer time and preparation for final release.

In the planning phase, Microsoft tries to use a high-level vision statement and outline specification to get projects going, rather than trying to write a complete specification at the outset. The program managers then write a functional specification, outlining the product features in sufficient depth to organise schedules and staffing allocations. However, the initial specification does not cover all the details as it is just a high-level vision statement. During the development phase, the program managers revise the functional specification when they learn more about what should be in the product. Of the total project time allocated for development and stabilisation, a project will generally spend about two-thirds of this time in the development phase and one-third in the stabilisation phase. The development phase consists of three or four "milestones" (builds). At these stages, program managers can revise their functional specification. In Microsoft, projects spend approximately two to four months developing each milestone release. Each release includes its own coding, testing, and debugging activities.

Microsoft also tried to fix shipment date to deliver products on time. Project managers schedule backwards from the shipment date and define the dates for the intermediate project milestones. Typical desktop applications, such as the next release of Office, Word, or Excel, are 12 to 24 months in duration. Microsoft is

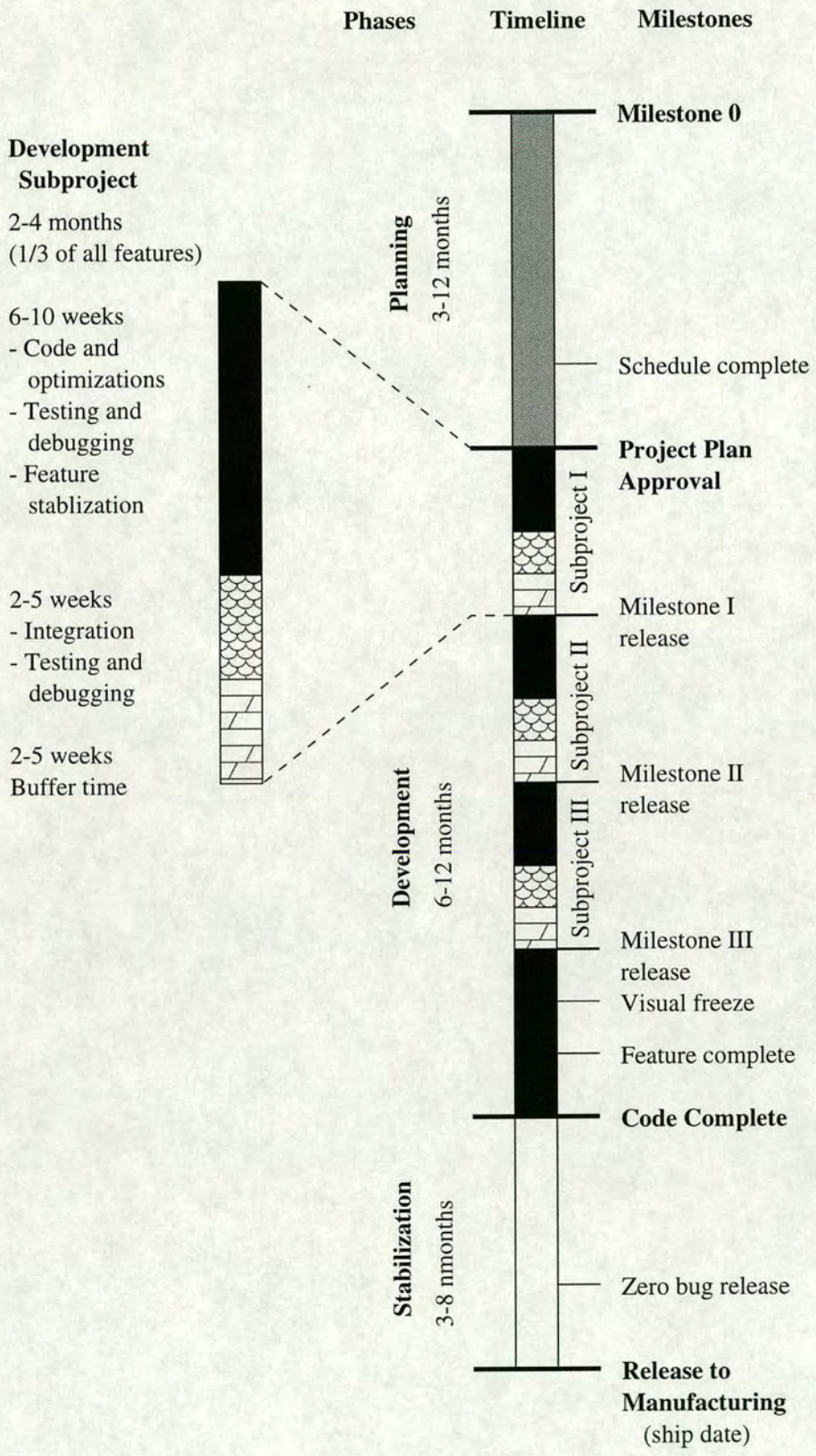

Figure 1.3: Microsoft's Synch-and-Stabilise Life Cycle[CS95]

moving toward alternating 12 and 24 month schedules for applications products, with the 12 month projects offering minor feature enhancements and 24 month projects offering major feature and architectural changes. As a result, Microsoft can ship a product release every 12 months[CS95]. The fixed shipment date keeps pressure on developers to cut down on features. However, rushing the shipment date generally leads to less time for testing and quality assurance activities at the end of the project.

In addition, the DSDM Consortium in UK suggested the Dynamic Systems Development Method (DSDM) as a framework of controls for the development of IT systems to tight timescales[DSD97]. The mechanism for handling flexibility of requirements in DSDM is the timebox. Each timebox is subdivided into three parts: investigation (a quick pass to see whether the team is taking the right direction), refinement (to build on the comments resulting from the review at the end of investigation) and finally consolidation to tie up any loose ends. The timebox must have an immovable end date and a prioritised set of requirements assigned to it.

Both cases show an essential concept that Yourdon[You96] argued the concept of "good enough" software. He suggested that functionality, quality and schedule are the three most important elements of "good enough" in most software today. These elements form a triangle and are interconnected. The balance between them shifts dynamically during a project. It has to be reevaluated by the customer and the project manager.

1.3 Research focus

To tackle the previously mentioned problems, my research aims to build a Process-centred Software Engineering Environment (PSEE) for software process improvement. According to ISO/IEC 15504, the software process is defined as following:

> *The process or set of processes used by an organisation or project to plan, manage, execute, monitor, control and improve its software related activities.[ISO96]*

This concept was originally from manufacturing physical products, in particular the automobile industry. From the beginning, software engineers tried to find solutions from industry. Although this is not an easy way to imitate industries' processes, at least, by following processes, the results can be planned and tracked. As a consequence, researchers tried to build an environment which

provides computer-based support and guidance for the enactment of software development processes. The implementation of PSEEs is increasingly popular in the software engineering community. Ben-Shaul and Kaiser[BSK95] define a PSEE as:

> *PSEEs are systems that support large scale software development by providing: (1) mechanisms and notations for explicitly modelling the process of development and maintenance of software, including task definitions, control integration such as global task ordering and local constraints on their activation, tool integration, data modelling and integration, and user modelling; and (2) mechanisms for enacting the modelled process by the PSEEs process-engine, where forms of enactment include process automation, consistency, monitoring, enforcement and guidance.*

Finkelstein, Kramer and Nuseibeh[FKN94] also suggest a PSEE as:

> *A PSEE is centred around an explicit process description, often called process model, that is defined using Process Modelling Languages (PMLs). These languages offer powerful capability to describe roles, manual and automated procedures, interaction among users, process artifacts, and constraints. The execution (enactment) of the process model within a PSEE provides support to process agents in the execution of their work, for example, by offering guidance to them or by automating some parts of the process.*

As a consequence, a PSEE might include:

- a process model
- a process definition language
- mechanisms or notations for process enactment
- tool integration mechanisms
- support for communications

Moreover, articles were focused on computer-supported cooperative work (CSCW) which combines a study of the organisational, psychological, and social aspects of people working together with the enabling technologies of groupware[TS96]. Bandinelli et al. argued that CSCW and PSEEs basically address the same issue, i.e., how to support cooperative activities in human-centred process[BNF96]. In this research, we do not restrict which environment belongs to CSCW or PSEEs. In principle, we are concentrating on supporting cooperative activities in software development.

Why do software organisations need PSEEs? Firstly, software systems have become increasingly more complex and larger in scale during the last decade. It results in crucial problems such as:

- Software organisations require more and more skilled people to develop and maintain their product.
- To manage these various experts, without a disciplined environment would be much more difficult.

As a result, researchers are looking for solutions to these problems. Ebert[Ebe97] in his article described that current software engineering practice is based on the uniqueness of projects. Knowing that something similar has been done before is considered to have no practical impact because some interfaces might differ and several environmental flavours could have changed. However, he suggested that software engineering should investigate how to copy what is good and use what already exists. This is why Gamma et al.[GHJV94] used design patterns to record experiences in designing object-oriented software. The goal of their study is simple. They would like to capture design experience in a form that people can use effectively. Design patterns make it easier to reuse successful designs. Expressing proven techniques as design patterns makes them more accessible to developers of new systems. If design patterns can help novices to learn by example to behave more like experts, why do we bother spending a significant amount of time hunting for any solution from scratch?

Design patterns have modified the ideal of using CASE tools. It is clear that simply using CASE tools which support specific activities is not sufficient. PSEEs which support software activities through the execution of the model of the software process will be a good solution for software development. Such a model describes the interaction between software developers and development environment, such as the role assigned to perform the activities, the anticipated work products needed by and produced by the activities of the process, as well as the tools, methods and control points related to the process. As far as developers are concerned, PSEEs provide them with an appropriate working context. This will result in an environment in which developers can inherit experience abstracted from experts. This can reduce the complexity of software development and the requirement for skilled people.

Secondly, the Internet has become one of the most popular innovations in the world. The Internet allows users access to selected information regardless of geographical distribution and heterogeneity within the physical computing en-

vironment. Since it is traditional that a software project might be composed of separate teams for requirement analysis, design, coding, testing and maintenance, the Internet could allow these teams developing software to be located around the world. As a result, the concept of "virtual corporation" is increasingly more essential for the software industry. A "virtual corporation" is a company that relies on outsourcing almost every aspect of a software project, from requirement analysis to maintenance. It is similar to setting up an assembly line around the world. We can imagine a scenario where teams in the United States and in India electronically collaborate to develop a software project. The States' team works on the project during the day and saves the work on a central computer. By nightfall, it is daytime in India, and India's team takes over, working with what the States' team has developed. The concept results in advantages such as:

- Cost saving: An Indian programmer is much cheaper than his/her American counterpart in Silicon Valley but is of similar quality. Since it is impractical to hire all skilled people in one geographical area and put them in the one physical building, the Internet enables companies to hire software engineers in different geographical area. This reduces the cost of hiring qualified people.

- Time-to-market saving: The project development is worked on in different time zone, shortening the time taking to develop a project.

Let us go back to Section 1.2. It is clear that the cases of Microsoft's development teams and the Japanese software factory have been adopted by these concepts. As a consequence, a disciplined environment with geographically dispersed teams, connected by the Internet and/or intranet, will be an essential feature of PSEEs.

This research will try to build a PSEE by using the CMM and there are two reasons for this. Firstly, we believe that a PSEE must comply with a framework of continuous process improvement, such as the CMM, in order to establish a disciplined environment. To date, we are not aware of any research building an environment for software process improvement. Secondly, in any context in which the CMM is applied, a reasonable interpretation of the practices should be used. The CMM must be appropriately interpreted for different size projects and software organisations. SPI PASTA adopts the concept of artifact-driven to build an environment for software development. In a development project, SPI PASTA provides guidance on what sort of product it is necessary to create.

In this research, we will concentrate on CMM levels 2 and 3 since the KPAs (Key Process Areas) on levels 2 and 3 include all basic processes for software development. This research will provide practical guidance on how to introduce the CMM into software organisations. We believe this PSEE will help lower maturity organisations to easily achieve higher levels of process maturity and effectively develop their software products.

1.4 Contributions

The main contribution of this research is that SPI PASTA (Software Process Improvement PASTA) builds a framework to provide software organisations with a defined process recommended by the CMM. This framework may help software organisations developing their own software development process to ensure that they are consistent with the CMM. Furthermore this framework may be a basis to develop large international systems with multi-national participation. With commitment to SPI PASTA, software engineers from different countries can follow the same rule to develop the software projects. Moreover, with SPI PASTA, software developers will better recognise what they have to do and accumulate the knowledge and experience to improve their software process. This is very helpful for new engineers who can use the SPI PASTA as a training resources in order to join the development team as soon as possible.

1.5 Thesis Organisation

This thesis describes how to define and model a software process by using a modelling notation under the software process improvement architecture. The organisation and content of the thesis is as following:

- Chapter 1, *Introduction*, shows the motivations why we are going to do this work, defines the research focus and presents what contributions have been done in the thesis.

- Chapter 2, *Background*, presents related work from several software organisations to set the context for the thesis and explores the standards and modelling method we used. These standards contain the CMM which we model, MIL-STD-498 providing uniform requirements for software development and documentation, which is used as a supplement for the CMM, and PASTA being a modelling notation, which we use to model the software process.

- Chapter 3, *Process Tailoring*, describes the KPAs for process tailoring. This will let the software organisation define the most appropriate software process for the organisation and the software project.

- Chapter 4, *The Processes in the CMM Level 2*, consists of the management process, focusing on the software project planning and explores software management activities before technical processes are implemented, and the support process, describing two essential KPAs, Software Quality Assurance and Configuration Management, which support implementation of management and technical processes.

- Chapter 5, *The Processes in the CMM Level 3*, consists of the technical process, describing software engineering activities which we use the UML to implement them, and the organisational process, describing four organisational topics which belong to the CMM level 3.

- Chapter 6, *Implementation and Assessment*, describes how to implement the SPI PASTA and how to assess the processes in order to improve organisation's software process.

- In Chapter 7, *Conclusion and future work*, finally, we make a conclusion for our work and suggest further work to be done in the future.

Chapter 2

Background

2.1 Related Work

In this section, we will summarise some related work on the software process improvement field. We firstly discuss the infrastructure that guides organisations in planning and implementing an effective software process improvement program. The Process Improvement Strategy from SEL (Software Engineering Laboratory) in NASA and the SEI's IDEAL (Initiating, Diagnosing, Establishing, Acting and Learning) model provide a concept of the life cycle for software process improvement. To implement these strategies, secondly, we focus on process-centred software development environments. Four projects, the Software Technology for the Adaptable, Reliable Systems (STARS) program from DoD in the US, EPOS (Expert System for Program and ("og") System Development), SPADE and Oz, define the process modelling language to model software processes and build a software engineering environment to support an organisation's development practices. Finally, we will introduce a commercial software development environment, Objectory, which defines a process to control software development during the software life cycle.

2.1.1 The Software Process Improvement Programs

2.1.1.1 SEL Process Improvement Strategy

The SEL has long been a pioneer for software process improvement because of the characteristics of its products. The SEL defined a standard paradigm to illustrate its concept of software process improvement. This paradigm is a three-phase model which includes the following steps[MPB94]:

1. **Understanding:** Improve insight into the software process and its products by characterising the production environment, including types of software

developed, problems defined, process characteristics and product characteristics.

2. **Assessing:** Measure the impact of available technologies and process change on the products generated. Determine which technologies are beneficial and appropriate to the particular environment and, more importantly, how the technologies (or processes) must be refined to best match the process with the environment.

3. **Packaging:** After identifying process improvements, package the technology for application in the production organisation. This includes the development and enhancement of standards, training and development policies.

In the SEL process improvement paradigm, these steps are addressed iteratively, and form a base for the software process community.

2.1.1.2 The IDEAL Model

The IDEAL model[McF96, GM97], developed by the SEI in Carnegie Mellon University, is an organisational improvement model that serves as a roadmap for initiating, planning and implementing software process improvement actions. As in the CMM for software, IDEAL provides an approach to continuous improvement by outlining the steps necessary to establish a successful improvement program. The model provides a disciplined engineering approach for improvement, focuses on managing the improvement program and establishes the foundation for a long-term improvement strategy. The IDEAL model is composed of five phases:

- **Initiating:** During the initiating phase, the business reasons for undertaking the effort are clearly articulated. The effort's contributions to business goals and objectives are identified, as are its relationships with the organisation's other work. The support of critical managers is secured, and resources are allocated on an order-of-magnitude basis. Finally, an infrastructure for managing implementation details is put in place.

- **Diagnosing:** The diagnosing phase builds upon the initiating phase to develop a more complete understanding of the improvement work. During the diagnosing phase, two characterisations of the organisation are developed, the current state of the organisation and the desired future state. These organisational states are used to develop an approach for improving business practice.

- **Establishing:** The purpose of the establishing phase is to develop a detailed work plan. Priorities are set that reflect the recommendations made during the diagnosing phase as well as the organisation's broader operations and the constraints of its operating environment. An approach is then developed that honours and factors in the priorities. Finally, specific actions, milestones, deliverables and responsibilities are incorporated into an action plan.

- **Acting:** The activities of the acting phase help an organisation implement the work that has been conceptualised and planned in the previous three phases. These activities will typically consume more calendar time and more resources than all of the other phases combined.

- **Learning:** The learning phase completes the improvement cycle. One of the goals of the IDEAL Model is to continuously improve the ability to implement change. In the learning phase, the entire IDEAL experience is reviewed to determine what was accomplished, whether the effort accomplished the intended goals and how the organisation can implement change more effectively and/or efficiently in the future. Records must be kept throughout the IDEAL cycle with this phase in mind.

2.1.2 Process-centred Software Development Environments

2.1.2.1 STARS

The STARS program[RE95, KS95a, Uzz96] is sponsored by the Defence Advanced Research Projects Agency (DARPA) in the United States. The goal of the STARS project is to increase software productivity, reliability and quality by integrating support for modern software development processes and reuse concepts within software engineering environment technology.

The STARS program uses a megaprogramming (or product-line) concept[BBB95] for software development and life-cycle support characterised by an architecture-based approach to software engineering with application domains. Basically, the STARS program builds a process-centred software engineering environment (PSEE) which includes the definition and enactment of disciplined processes for the development of applications and the evolution of the product-line as a whole. In addition to the PSEE, the crucial feature in the STARS program is domain-specific reuse which addresses the systematic creation of domain models and domain-specific architecture and their use in building applications[BC96, CFM+96].

To date, there are three STARS Demonstration Projects, one with each of the three services (Army, Navy and Air Force), which are currently engaged in applying the principles of megaprogramming to real systems. The major objectives for each of the projects are:

- Apply megaprogramming principles to the development of software for an actual DoD application, to establish the credibility of the approach.
- Collect and document experience about the benefits and costs of megaprogramming as well as the effectiveness of the specific tools and techniques used on the project, to help other organisations plan for and implement similar approaches.
- Transition to the Demonstration Project's parent organisation, to establish the capability to apply megaprogramming to other applications in their product-line.

2.1.2.2 EPOS

EPOS[NaC96, Con95, CLM+95] is a Software Engineering Environment with emphasis on process modeling, software configuration management, and support for cooperative work. The rationale for the scientific initiative is to improve software quality through better process support for the software production process.

EPOS defined a reflexive, object-oriented software process modelling language called SPELL. By using SPELL, the Planner can execute the task network. In addition, EPOS also creates the following meta-process tools to support software projects:

- **Schema Manager** is responsible for textually/graphically browsing, editing, defining, analysing, translating and evolving the Process Schema and can be used on all the process models.
- **Task Network Editor** makes it possible to directly manipulate the task network before and during execution, and supports features such as add/remove/mov Tasks and Products.
- **Planner** is incrementally invoked by the Process Engine to decompose high-level tasks into a task network.
- **Project Manager** is used to start and stop a project and to retrieve useful project metrics from the EPOS-database.

To perform a successful improvement program, EPOS defined a meta-process which consists of four steps:

1. Planning and Instantiation: A new project is initiated with a specific project context. The Planner will automatically build a task network based on the generic process model. The Project Manager will then use the project context to retrieve a set of previous similar projects from the Experience database.

2. Execution and Tracking: The Process Engine interprets the task network and enacts the task with sufficient and available resources.

3. Packing and Assessment: The current performance progress is compared and assessed against previous models, historical profile and experiences of the baseline project by the Project Manager.

4. Evolving and Learning: Improvement achievements from a completed project are generalised, formalised and stored for future use.

2.1.2.3 SPADE

The goal of the SPADE project[CNFG96, BNF96, NF95] is to provide a software engineering environment to support Software Process Analysis, Design and Enactment. The environment is based on a process modeling language, called SLANG (SPADE Language), which is a high-level Petri net based formalism. SLANG offers features for process modeling, enaction and evolution. In addition, it describes interaction with external tools and humans in a uniform style.

The architecture of SPADE is based on three separate layers:

- Process Enactment Environment (PEE): The main component of the PEE is the Process Engine which executes a SLANG process model.

- User Interaction Environment (UIE): The goal of the UIE is to manage the interaction between SPADE and its users. Users coordination and interaction is achieved through tools that are integrated into SPADE.

- The SPADE Communication Interface (SCI): The SCI is a filter which allows communication between the PEE and the UIE.

SPADE-1 is an implementation of the SPADE environment. It supports the enaction of SLANG process models. SPADE-1 includes a process interpreter which is able to enact process models written in SLANG, a SLANG editor which

creates and modifies SLANG process models, a monitor which controls the enaction state of the process, and an agenda which interacts with process agents. SPADE-1 supports tool integration at different granularity levels. In particular, it is possible to integrate stand-alone tools.

2.1.2.4 Oz Project

Oz[BSK96, BSK95, KDJY97] was developed by the Programming Systems Laboratory in Columbia University. It is a multi-site collaborative workflow management system (WFMS) that supports interoperability among heterogeneous and autonomous processes. The basic idea of the Oz project is that a large software project may be decomposed into teams that are each responsible for full development of a distinct component of the system, exhibiting intra-group heterogeneity. In a multi-team development, it may be desirable to allow teams to use their own set of software tools and hardware, their own private files or databases and their own development policies and process. Consequently, the development teams need to collaborate in order to develop the product.

The internal architecture of Oz consists of three main runtime computational entities: the Environment Server, the Connection Server and the Client. The Environment Server is composed of three components: process, transaction and data managers. The process manager loads the process model, the transaction manager is parameterised by lock tables and concurrency control policies and the data manager loads the schema for the product data and process state. The client is composed of four major subcomponents: (1) access to information about rules and built-in commands, (2) objectbase representation, (3) activity execution, and (4) an ad hoc query interface. The Connection Server's main responsibility is to establish connections to a local server from local clients, remote clients and remote servers.

A local process in Oz is defined using a rule-based language. Each activity is enclosed in a rule with formal typed parameters, and optional condition and effects that serve two purposes: to enforce and assert conditions that pertain to the activity itself; and to connect to other related activities and specify automation and/or atomicity requirements across activities. Related activities can be invoked automatically as part of either backward chaining to satisfy the predicates in a rule's condition, or forward chaining as a result of the assertions in a rule's selected effect. A rule thus defines a process step, and the set of all chains emanating from that rule define a task.

2.1.2.5 Trillium

The Trillium Model[Tri94] was developed by Bell Canada to assess the product development and support capability of prospective and existing suppliers of telecommunications or information technology-based products. In principle, the Trillium Model provides key industry practices which can be used to improve an existing process or life-cycle. The Trillium Model is mainly based on the CMM version 1.1 and incorporates international standards, such as ISO 9001, IEEE Software Engineering Standards Collection, and so on. However, the big difference is that its architecture is based on roadmaps, rather than key process areas. Basically, the Trillium Model consists of Capability Areas, Roadmaps and Practices. In the top level, eight Capability Areas are defined in the Trillium Model:

- Organisational Process Quality
- Human Resource Development and Management
- Process
- Management
- Quality
- System Development Practices
- Development Environment
- Customer Support

Each Capability Area incorporates one or more roadmaps. A roadmap is a set of related practices that focus on an organisational area or need, or a specific element within the product development process. Within a given roadmap, the level of the practices is based on their respective degree of maturity. Since these practices are taken from standards, meeting the requirements of a Trillium practice means meeting the requirements of the corresponding referenced standards.

2.1.2.6 Objectory

The Objectory Process[Obj97] is a Software Engineering Process which is originally defined by Jacobson[Jac87]. It provides a disciplined approach to assigning tasks and responsibilities within a development organisation. Basically, the process description integrates the method description into a framework, stating how the work should be carried out as interacting processes within each phase of the development. In general, Objectory can be described in two dimensions:

- Along time, the life cycle aspects of the process as it will unroll itself.
- Along process components, which groups activities logically by nature.

The first dimension represents the dynamic aspect of the process, as it is enacted, and is expressed in terms of cycles, phases, iterations, and milestones.

The software life cycle is broken into cycles, each cycle working on a new generation of the product. The Objectory process divides one development cycle into four consecutive phases:

- Inception
- Elaboration
- Construction
- Transition

Each phase is concluded with a well-defined milestone – a point in time at which certain critical decisions must be made, and therefore key goals must have been achieved.

The second dimension represents the static aspect of the process: how it is described in terms of process components, activities, workflows, and so on.

The Rational Objectory Process is composed of seven process components, four engineering process components:

- Requirement capture
- Analysis & Design
- Implementation
- Test

and three supporting components:

- Management
- Deployment
- Environment

Each process component comprises a set of correlated activities. An activity describes the tasks done by workers to create or modify artifacts, together with the techniques and guidelines to perform these tasks, and possibly including the use of tools to automate some of these tasks.

2.2 Standards and Modelling Method

The SEI's Capability Maturity Model for Software has popularised the notion of measuring the software process maturity of organisations. In addition, some efforts have been done in this field. In UK the Central Computer and Telecommunications Agency (CCTA) developed PRINCE (Projects in Controlled Environments) which is a project management method covering the organisation, management and control of projects[PRI97]. Furthermore, ISO 15504 project is currently creating a set of international standards for software process management that attempts to harmonise existing approaches. One of the ISO 15504 objectives is to create a way of measuring process capability, while not using a specific approach such as the SEI's maturity levels. The approach selected is to measure the implementation and institutionalisation of specific processes; a process measure rather than an organisation measure. Maturity levels can be viewed as sets of process profiles using this approach. During the development of version 2 of the CMM, one of the technical issues to be decided is whether to re-architect the CMM by layering organisational maturity on top of the ISO 15504 process capability framework.

2.2.1 The Capability Maturity Model

In 1987, the SEI released a brief description of the process maturity framework and a maturity questionnaire. Originally, the CMM was developed to assist the U.S. DoD in software acquisition. After four years, the SEI evolved the software process maturity framework into the Capability Maturity Model for Software (CMM). Then, the SEI released Version 1.1 of the CMM for Software in 1993. Since then, the CMM has been widely used by the software engineering community for appraising software processes and guiding software process improvement. Currently, the SEI recognises that the CMM should continue to evolve because continuous improvement applies to the CMM, just as it does to the software process[PGC96]. The new version would be released consistent with the CMM Integration Framework.

The two documents that provided the foundation for Version 1.1 of the CMM are:

- Capability Maturity Model for Software, Version 1.1[PCCW93b], and
- Key Practices of the Capability Maturity Model, Version 1.1[PWG+93].

The first one contains an introduction to the model, descriptions of the five

maturity levels, an operational definition of the CMM and its structure, a discussion of how organisations can use the maturity model, and some remarks on the future directions of the CMM. The second one contains the key practices that correspond to the key process areas at each maturity level of the CMM and information to help interpret the key practices.

The CMM is a descriptive model in the sense that it describes essential (or key) attributes that would be expected to characterise an organisation at a particular maturity level. It is a normative model in the sense that the detailed practices characterise the normal types of behaviour that would be expected in an organisation doing large-scale projects in a government contracting context. The intent is that the CMM is at a sufficient level of abstraction that it does not unduly constrain how the software process is implemented by an organisation; it simply describes what the essential attributes of a software process would normally be expected to be.

In any context in which the CMM is applied, a reasonable interpretation of the practices should be used. The CMM must be appropriately interpreted, using informed professional judgement, when the business environment of the organisation differs significantly from that of a large contracting organisation.

The CMM is not prescriptive; it does not tell an organisation how to improve. The CMM describes an organisation at each maturity level without prescribing the specific means for getting there.

The CMM is composed of five maturity levels. With the exception of Level 1, each maturity level comprises of several key process areas. In Version 1.1, there are 18 key process areas. In Version 2, the SEI restructures the CMM in particular on Levels 4 and 5. There will be 19 key process areas in new version. Each key process area is organised by a set of goals and the key practices that accomplish the goals of the key process area. The key practices are belonged to common features which contain five sections. The structure of the CMM is illustrated in Figure 2.1.

2.2.1.1 Maturity Levels

A maturity level is a well-defined evolutionary plateau toward achieving a mature software process. Each maturity level provides a layer in the foundation for continuous process improvement. Maturity levels are a staged architecture. As organisations establish and improve the software processes by which they develop and maintain their software work products, they progress through levels of maturity. Achieving each level of the maturity model institutionalises a different

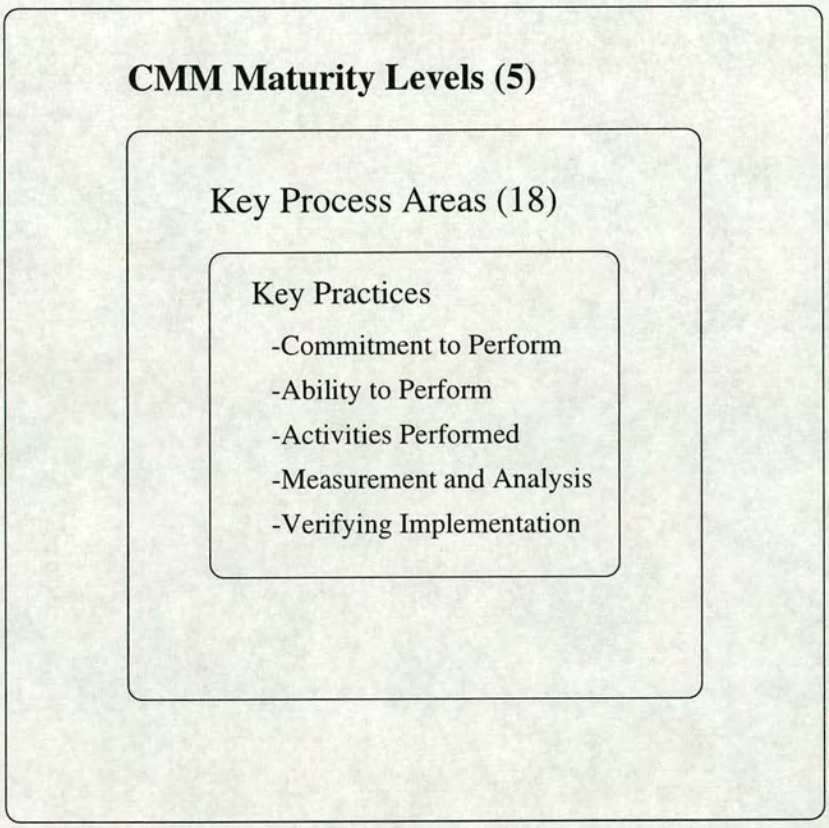

Figure 2.1: The CMM Structure

component in the software process, resulting in an overall increase in the process capability of the organisation.

The CMM is structured into five maturity levels[PCCW93b], as Figure 2.2:

1. **Initial:** The software process is characterised as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.

2. **Repeatable:** Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

3. **Defined:** The software process for both management and engineering activities is documented, standardised, and integrated into a standard software process for the organisation. All projects use an approved, tailored version of the organisation's standard software process for developing and maintaining software.

4. **Managed:** Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.

5. **Optimising:** Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

2.2.1.2 Key Process Areas

As illustrated in Figure 2.2, with the exception of Level 1, each maturity level comprises of several key process areas which indicate the areas an organisation should focus on to improve its software process. Key process areas are described in terms of a set of goals and the key practices. The goals summarise the key practices of a key process area and can be used to determine whether an organisation or project has effectively implemented the key process area. All the goals of a key process area must be achieved for the organisation to satisfy that key process area. When the goals of a key process area are accomplished on a continuing basis across projects, the organisation can be said to have institutionalised the process capability characterised by the key process area.

The key practices describe the infrastructure and activities that contribute most to the effective implementation and institutionalisation of the key process

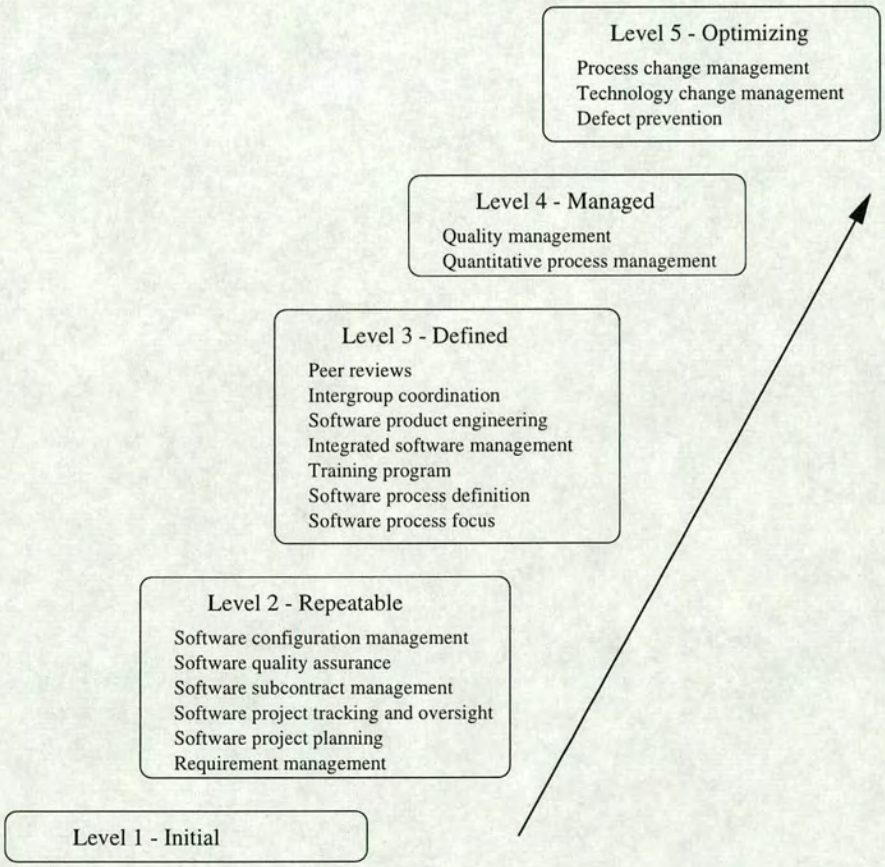

Figure 2.2: SEI Capability Maturity Model

area. To ensure consistent accomplishment of the goals of a key process area, the organisation can establish a documented procedure extracted from key practices of the key process area. However, the key practices describe "what" is to be done, but they should not be interpreted as mandating "how" the goals should be achieved. The key practices should be interpreted rationally to judge whether the goals of the key process area are effectively achieved.

For convenience, each of the key process areas is organised by common features. The common features are attributes that indicate whether the implementation and institutionalisation of a key process area is effective, repeatable, and lasting. The five common features, followed by their two-letter abbreviations, are listed below[PCCW93b]:

- Commitment to Perform (CO): Describes the actions the organisation must take to ensure that the process is established and will endure. Includes practices on policy and leadership.
- Ability to Perform (AB): Describes the preconditions that must exist in the project or organisation to implement the software process competently. Includes practices on resources, organisational structure, training, and tools.
- Activities Performed (AC): Describes the roles and procedures necessary to implement a key process area. Includes practices on plans, procedures, work performed, tracking, and corrective action.
- Measurement and Analysis (ME): Describes the need to measure the process and analyse the measurements. Includes examples of measurements.
- Verifying Implementation (VE): Describes the steps to ensure that the activities are performed in compliance with the process that has been established. Includes practices on management reviews and audits.

However, Bach[Bac94] argued that the SEI process maturity model has a number of limitations and weaknesses and that it may actually be dangerous in some circumstances. Bollinger and McGowan[BM91] also argued that unexpected problems pop up in the detailed implications of levels 4 and 5. The process-instrumentation approach causes software processes to fossilise into inflexible configuration. Furthermore, in level 5, the traceback methods seem in the case of software to be belated and poorly focused. Humphrey and Curtis[HC91] insisted that a defined engineering process cannot overcome the instability created by the absence of sound management practices. SPI PASTA are developed to lay the foundation on which effective practices for the higher level are built.

2.2.2 Software Life Cycle Processes

Currently, two standards, MIL-STD-498 and ISO 12207, are widely accepted by the software community for software life cycle processes. Both standards offer a framework for software life cycle processes from concept through retirement. They provide a structure of processes using mutually accepted terminology, rather than dictating a particular life cycle model or software development method. In spite of the similarity, both standards are very different in their scope.

2.2.2.1 ISO 12207

ISO 12207, published in August 1995, was created to establish a common international framework to acquire, supply, develop, operate and maintain software. It is especially suitable for acquisitions because it recognises the distinct roles of acquirer and supplier. The standard is intended for two-party use where an agreement or contract defines the development, maintenance or operation of a software system.

ISO 12207 consists of three main processes, primary life cycle processes, supporting life cycle processes and organisation life cycle processes. The contents of these processes as following:

Primary Life-Cycle Processes

- Acquisition Process
- Supply Process
- Development Process
- Operation Process
- Maintenance Process

Supporting Life-Cycle Processes

- Documentation Process
- Configuration Management Process
- Quality Assurance Process
- Verification Process
- Joint Review Process
- Audit Process

- Problem Resolution Process

Organisation Life-Cycle Processes

- Management Process
- Infrastructure Process
- Improvement Process
- Training Process

These processes would not be fit for any organisation. Consequently, ISO/IEC 12207 describes how to tailor the standard for an organisation or project.

2.2.2.2 MIL-STD-498

MIL-STD-498[DOD94], developed by U.S. DoD, identifies a set of software development activities and defines the software products to be generated by those activities. This standard is written in terms of Computer Software Configuration Items (CSCIs) which is an aggregation of software that satisfies an end use function and is designated for separate configuration management by the acquirer.

The MIL-STD-498 package consists of the standard and 22 Data Item Descriptions (DIDs). Basically, Nineteen major software development activities compose of the detailed requirements of MIL-STD-498, as following:

1. Project planning and oversight
2. Establishing a software development environment
3. System requirements analysis
4. System design
5. Software requirements analysis
6. Software design
7. Software implementation and unit testing
8. Unit integration and testing
9. CSCI qualification testing
10. CSCI/HWCI integration and testing

11. System qualification testing
12. Preparing for software use
13. Preparing for software transition
14. Software configuration management
15. Software product evaluation
16. Software quality assurance
17. Corrective action
18. Joint technical and management reviews
19. Other activities

In comparison with ISO 12207, there are no the acquisition process, the supply process, the operation process, the maintenance process or the training process in MIL-STD-498. However, ten activities, from software requirement analysis to preparing for software transition, in MIL-STD-498 correspond to only a single process in ISO 12207, the development process. MIL-STD-498 provides more detailed requirements for the software development process.

In addition to the activities and DIDs, MIL-STD-498 also suggests the program strategies for the system. These strategies are:

Grand design: The "grand design" strategy is essentially a "once-through, do-each-step-once" strategy.

Incremental: The "Incremental" strategy determines user needs and defines the system requirements, then performs the rest of the development in a sequence of builds.

Evolutionary: The "Evolutionary" strategy also develops a system in builds, but differs from the Incremental strategy in acknowledging that all requirements cannot be defined up front.

The grand design is similar to the "waterfall" approach with projects seeking to "freeze" the specification at the beginning, then developing design, coding, and testing. This approach has gradually lost favour, especially in the object-oriented field.

In Incremental and Evolutionary strategies, MIL-STD-498 suggests the concept of a build (see Figure 2.3). It means:

1. A version of software that meets a specified subset of the requirements that the completed software will meet.

2. The period of time during which such a version is developed.

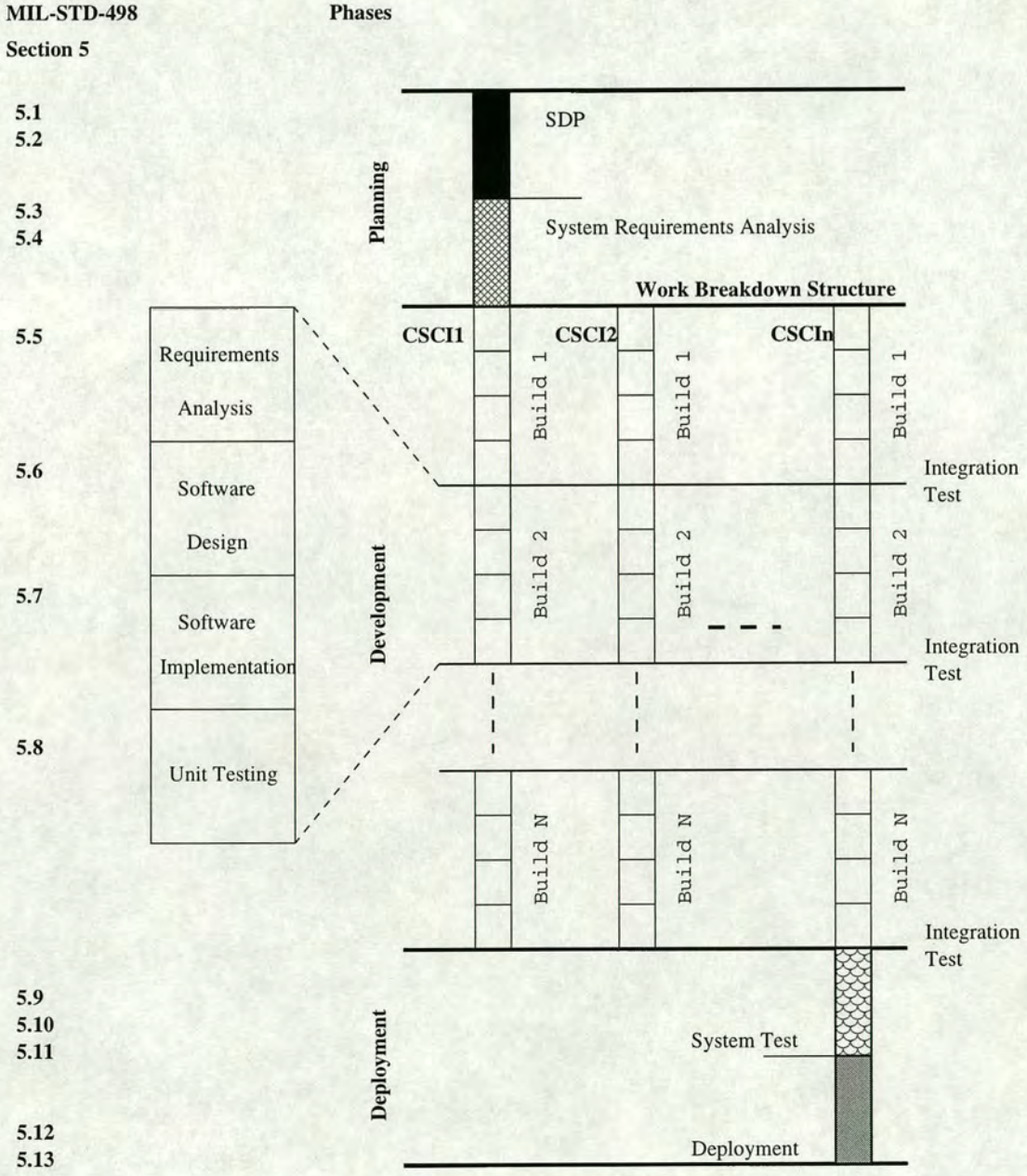

Figure 2.3: The Software Engineering Process in MIL-STD-498

Basically, both Incremental and Evolutionary strategies are based on the spiral model[Boe88]. The model defines four major activities:

1. Planning: determination of objectives, alternatives and constraints,

2. Risk analysis: analysis of alternatives and identification/resolution of risks,

3. Engineering: development of the "next-level" product, and

4. Customer evaluation: assessment of the results of engineering.

The spiral model is much more flexible and realistic than the classic "waterfall" life cycle. Both Incremental and Evolutionary strategies adopt these four activities to develop a project. However, there is a difference between Incremental and Evolutionary strategies.

The Incremental strategy is suitable for the concept of the software factory. When using the incremental strategy, the system requirements must be available and clear, such as bidding for government contracts. Project managers make reasonable estimates of schedules and arrange the appropriate resources. Under the schedule, the managers can establish three or four builds for the project to complete the system.

The Evolutionary strategy is suitable for highly competitive products like word processors and spreadsheets in mass markets. Such highly competitive products have the same characteristics. They all have great pressure on time-to-market. If you miss your shipment date, you lose the market. Furthermore, at the beginning, the managers cannot properly predict the specification of the project. As soon as project managers have any kind of specification, they develop the system as quickly as possible.

2.2.3 Process and Artifact State Machine Transition Abstraction (PASTA)

A process is a sequence of decision making activities. Software development is a process of making decisions about what programs should implement the software requirements and what the required properties of those programs are, including properties such as their structure and interfaces. A particular software process may be defined as a sequence of decisions made in different states. The process modeller may model a methodology by a set of predefined states, i.e., it is a prescription for the artifacts to be used, the activities to be performed and their sequencing, and the roles that people play. Process modellers provide software developers with guidance on what to do next based on the state of the development. A software developer using the methodology proceeds by following the activities prescribed by the states to produce the prescribed artifacts in the prescribed order.

In this thesis, we use the PASTA model[Lai91] to define a process for software development. The reason we chose the PASTA model is because the CMM model

provides a roadmap for software organisations to develop their software projects. As a result, there are many difficulties for software organisations using the CMM since the CMM contains so many key practices for developing and maintaining software. The software development teams are easily confused in this "roadmap". We try to use an artifact-driven approach to tackle this problem. Basically, the PASTA model uses artifacts, process states and roles to describe the software development process. In the PASTA model, artifacts capture the decisions made during the software development process. To characterise the state of a software development process, the software developer must characterise the state of the artifacts produced during the software process. However, merely characterising the state of the artifacts is insufficient to describe a complete software process. The process modeller must also describe the activities that may be performed on artifacts, the conditions under which those activities are performed, and the roles of the people who may perform them. As a consequence, it is possible to know exactly what the state of completeness of all artifacts is. By modelling the CMM, software organisations will clearly get through the maze to the right destination.

In Figure 2.4, the state model shows two levels. The lower level is based on the states of the artifacts produced during the software process; such states are called artifact states (A-states). Because A-states alone are insufficient to describe the software process completely, descriptions of activities, operations on artifacts, analyses that the software developer can perform on artifacts within the state, and the roles of the people involved augment them. The augmented states in the upper level state model are called process states (P-states).

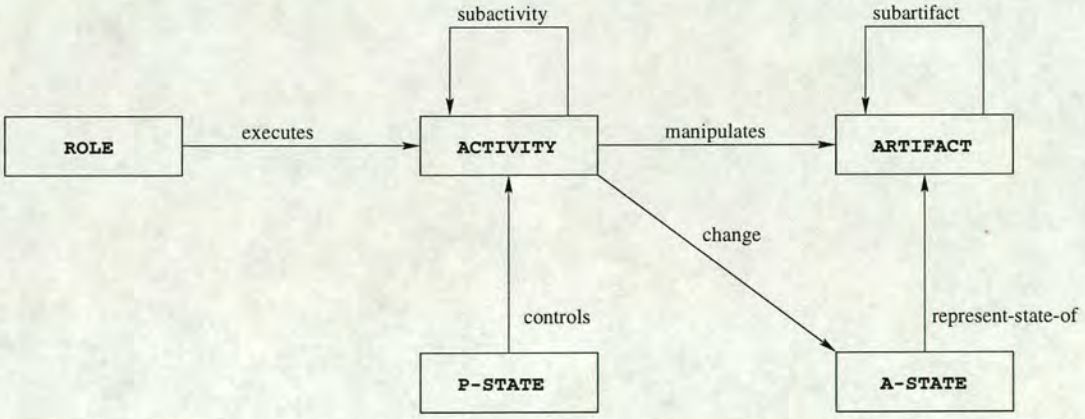

Figure 2.4: Relationships Defining the Design Model

PASTA uses tree-diagrams, forms, and state transition diagrams to describe process elements. The hierarchy of process states, the hierarchy of artifacts,

and the hierarchy of roles are represented using trees. Forms define process, artifacts and their states. Moreover, transition diagrams describe the process state machines and the artifact state machines. Figure 2.5 shows the different types of diagrams that PASTA uses for different process elements.

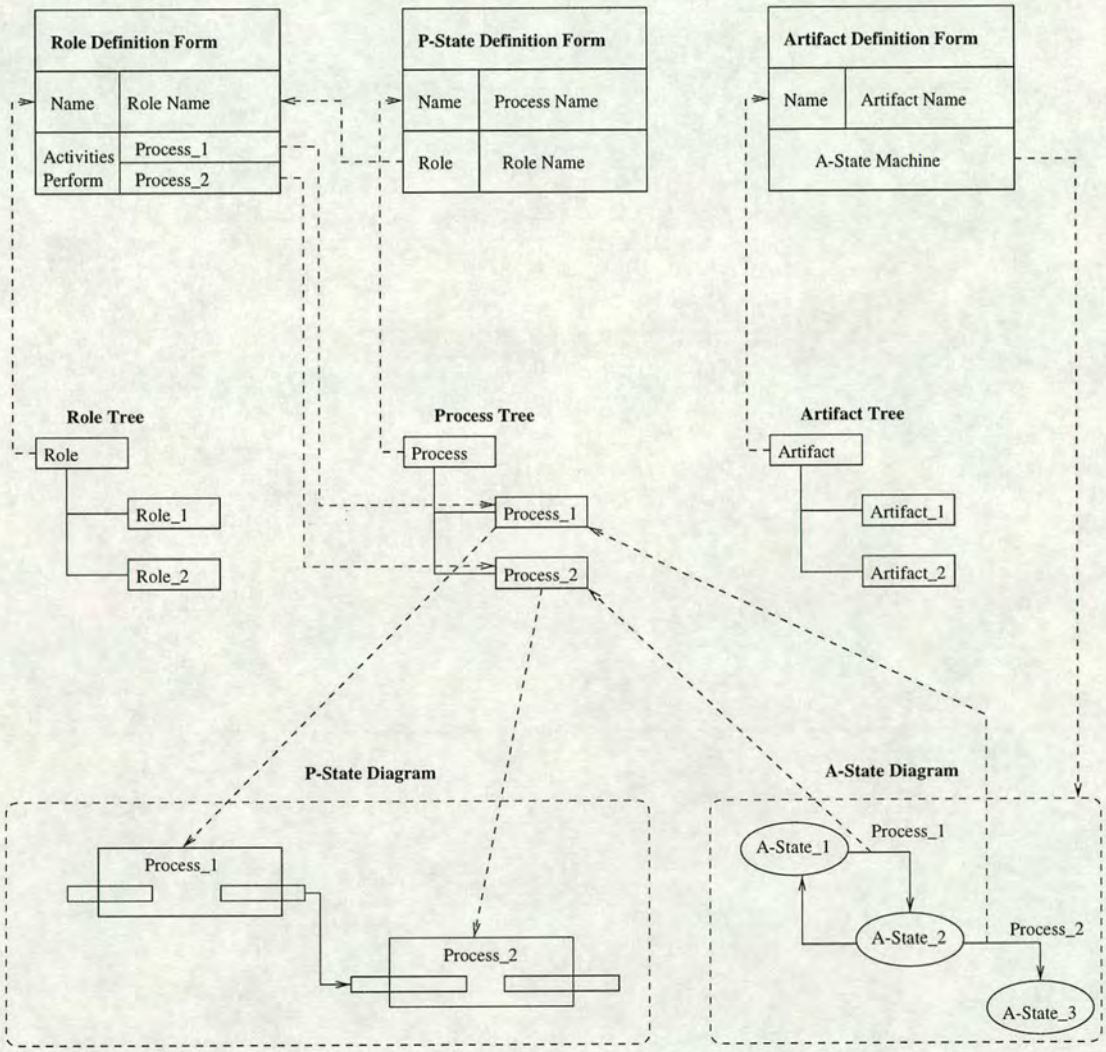

Figure 2.5: Representation of PASTA elements

Among these process elements, definition forms act as a key role in process modelling. They provide all required information for the process model. PASTA adopts a variety of forms to define many of the elements of a process model. These forms are described as following:

- Artifact Definition Form: Defines the artifacts used by a process.
- P-State Definition Form: Defines a process state.

Name	Artifact name
Synopsis	Prose description of artifact
Complexity	Either ELEMENTARY or COMPOSITE
Data Type	Data type used to store information about the artifact, predefined or user-defined
Artifact State Machine	A list of ordered pairs of states that defines the possible state transitions.
A-State Name	For artifacts that are ELEMENTARY, i.e., not decomposed into subartifacts, this cell is blank. For composite artifacts, a logical function of the states of other artifacts.
Subartifacts	
Sub-artifact Name	Synopsis of the sub-artifact.
Relations	
Relation Name	Names of related artifacts.

Table 2.1: The Artifact Definition Form Template

- Operation Definition Form: Defines an operation that may be performed in a P-state.
- Analysis Definition Form: Defines an analysis that may be performed in a P-state.
- Relation Definition Form: Defines the relationships between pairs of artifacts. Mostly used for situations where the state of one artifact may depend on the state of another artifact.
- Role Definition Form: Defines a role used in a process.

Here, we focus on the artifact and process state definition forms and state diagrams because they are central to PASTA models. Furthermore, we also present the operation definition form since it describes activities for the process model.

The Artifact Definition Form

Table 2.1 shows the Artifact Definition Form used to record information about artifacts. An artifact has states, may have subartifacts, and may be related to other artifacts.

The A-State Transition Diagram

In addition to the tabular description of the A-state machine for each artifact, PASTA also shows the machine as a state transition diagram. Figure 2.6 shows

the A-state transition diagram. An A-state transition diagram is similar to a conventional state transition diagram, where each arc represents the operation that causes the A-state change, and the node represents the A-state.

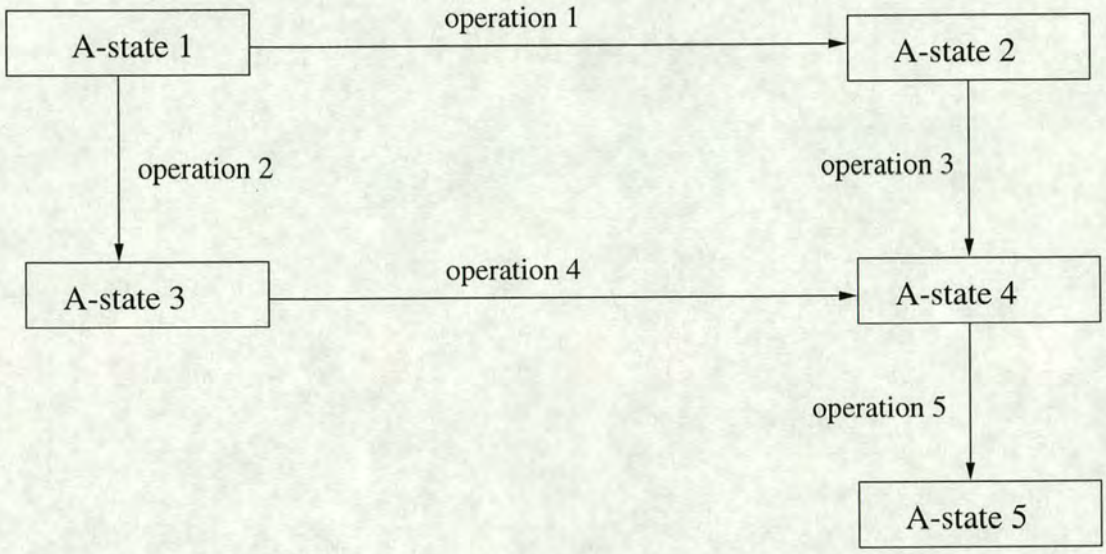

Figure 2.6: The A-State Transition Diagram

The Process State Definition Form

Table 2.2 shows the template for the Process State Definition Form used to record information about process states. A P-state may have substates, operations that may change artifacts, analyses that provide information about artifacts but do not change them, and roles that may perform the operations and analyses.

The P-State Transition Diagram

Figure 2.7 shows the P-state transition diagram. The larger rectangles with bold names are the sub-states of the P-state. The smaller rectangles at the entrance and exit to each substate represent the entry and exit conditions for the substate and are called condition boxes. A set of cells divides the condition box. Each cell represents one artifact in one A-state. The intersection lines between the condition boxes and the P-state box divide the cells into two parts. The parts outside of the P-state contain the name of the artifact, and the parts within the P-state box represent the A-state of the artifact. Each artifact cell may have more than one line connected to it. This means that operations in several different P-states may change the state of an artifact, thereby affecting the P-state for which

Name	The name of the state.
Main Role	Names of the roles which are principally concerned with activities in this state.
Synopsis	Prose description of process state.
Entrance Condition	Condition required for entry of an activity. The developer uses this to determine when a process can enter a particular P-state.
Artifact List	A list of artifacts upon which work may proceed in the P-state.
Information Artifacts	The artifact which holds information required to support operations and analyses in this state.
Activities	
Operations	Operation name and description.
Analyses	Analysis name and description.
Exit Condition	Predicated on A-states that determine when the process exits the P-state.

Table 2.2: The Process State Definition Form Template

the artifact state is a precondition. Also, a single A-state change may result in several P-state transitions.

The Operation Definition Form

Table 2.3 shows the operation definition form. Operations may change artifacts, often resulting in an A-state transition, or may just read them to obtain information needed to perform the operation. Each operation has an entry condition and an exit condition. The specification of an operation describes how to perform the operation and is given in both informal and formal terms.

In general the definition forms are intended to define the complete semantics of the model. The diagrams summarise the forms and act as a visual complement to them. Those who prefer visual representations may look at the diagrams first and the forms second; those who prefer textual representations may do the reverse.

Conclusion

In this section, we described those standards and methods that are used in this thesis. An artifact-driven approach is adopted by using PASTA to model the CMM. The CMM is a descriptive model in the sense that it describes essential attributes that would be expected to characterise an organisation at a particular

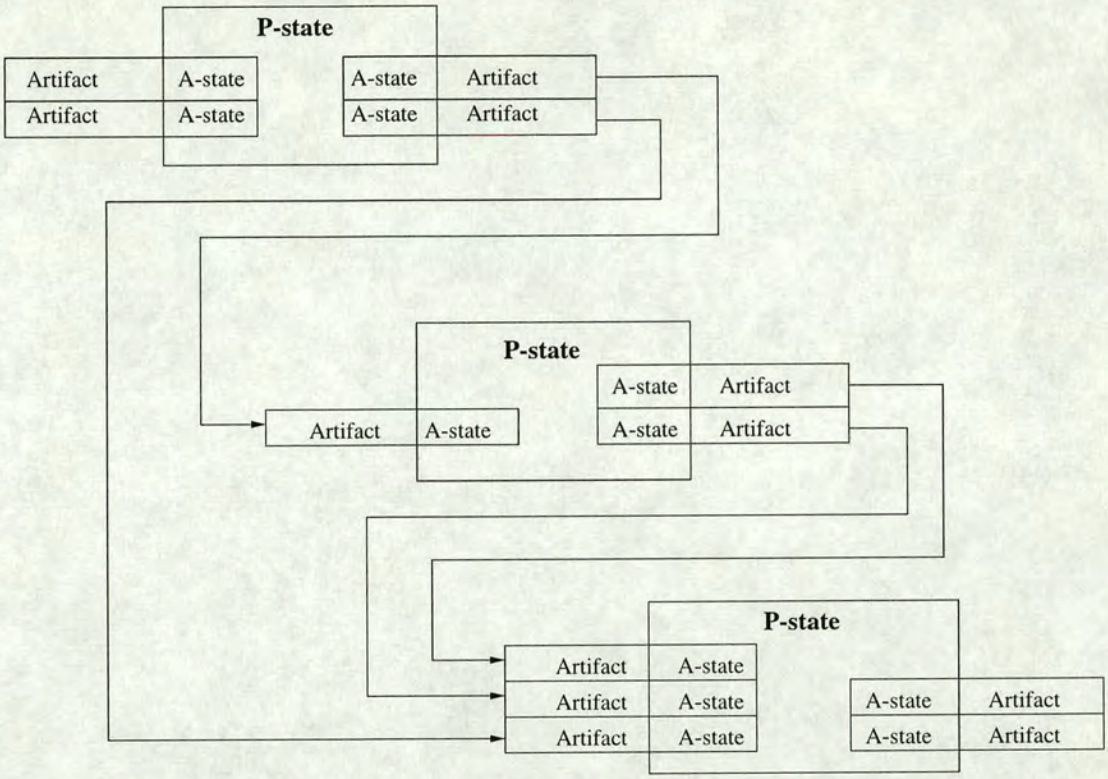

Figure 2.7: The P-State Transition Diagram

Name	Operation Name
Synopsis	Prose description of operation
Role List	List of roles that may perform the operation
Operation Type	Either Manual or Automated
Entrance Condition	Pre-condition for performing the operation
Artifact List	Artifacts that may be modified by the operation
Information Artifacts	Artifacts that may be read but not changed by the operation
Exit Condition	Post-condition for the operation
Informal Specification	Prose description of the procedure used to perform the operation
Formal Specification	Formal description of the procedure used to perform the operation

Table 2.3: The Operation Definition Form Template

maturity level. It does not prescribe any standard or methodology to develop software products. As a result, we adopted MIL-STD-498 to define a set of software development activities. These efforts will establish a framework to assist software organisations to develop their software projects.

Chapter 3

Process Tailoring

The CMM comprising a set of key practices suitable for use by all potential organisations and projects would be too general to be easily applied to any one organisation. It has long been a criticism that the CMM is only appropriate for large defence or avionics environments, and more difficult to apply to small organisations. This is because the CMM covers so many materials during developing a software project. As a consequence, by tailoring those key practices presented in the CMM, small software organisation would be easier to apply the CMM.

Paulk[PWG+93] suggested that a fundamental concept that supports the approach taken by the SEI in its process definition work is that processes can be developed and maintained in a way similar to the way products are developed and maintained. They include:

- requirements that define what process is to be described,
- an architecture and design that provide information on how the process will be defined,
- implementation of the process design in a project or organisational situation,
- validation of the process description via measurement, and
- deployment of the process into widespread operation within the organisation or project for which the process is intended.

As Figure 3.1 shows, using the analogy of product development, a framework for software process development and maintenance has evolved that translates these concepts into ones which are more specific to the process development discipline.

SPI PASTA (Software Process Improvement PASTA) provides a process-centred software engineering environment mainly based on the CMM Version

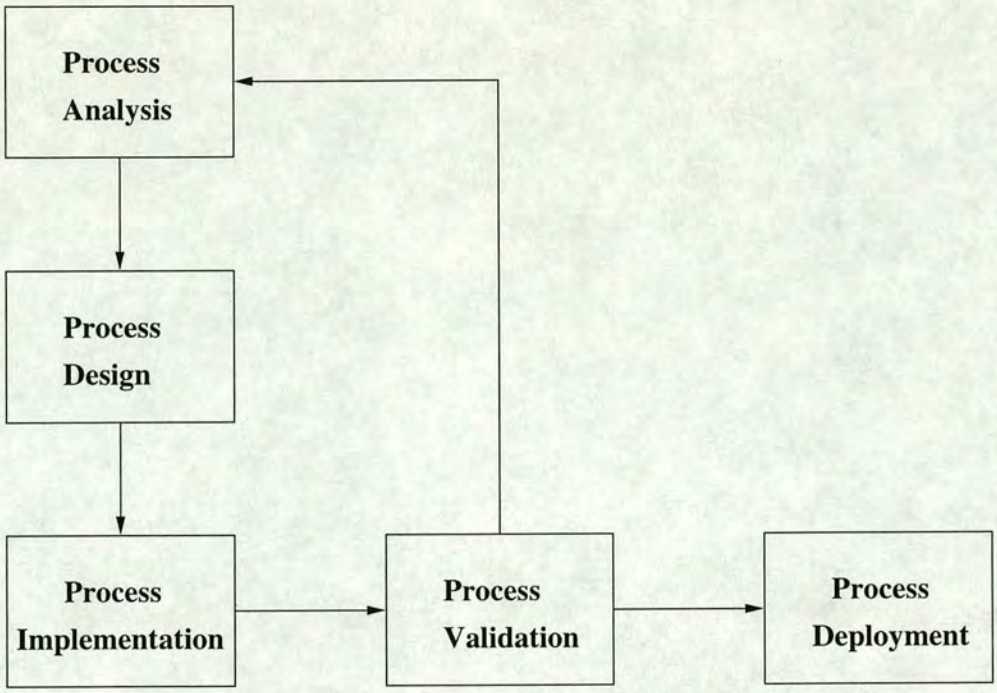

Figure 3.1: Process Definition Life Cycle

2.0 Draft C[Pau97]. The fundamental task to implement SPI PASTA is tailoring software processes to fit the software projects. Figure 3.2 shows the tailoring framework of SPI PASTA. However, the CMM is not only the process requirement at the starting point. We have to use other software processes and product standards to comply with the CMM. Since the CMM doesn't indicate whether the software organisation has the right process, these standards, such as ISO 12207, MIL-STD-498, ISO 9001 and so on, will give the organisation a good guide to complete its work.

The software organisations may view SPI PASTA as their organisation's standard software process which establishes a consistent way of performing the software activities across the organisation.

3.1 The Relevant Activities and Products in the CMM Levels 2 and 3

The SPI PASTA focuses on the CMM Levels 2 and 3. Before we go through the SPI PASTA, It is better to recognise the relevant activities and products in the CMM Levels 2 and 3. Figure 3.3 shows a framework of activities and products in the CMM Levels 2 and 3.

The SEI defined the CMM, which is a framework describing the key elements

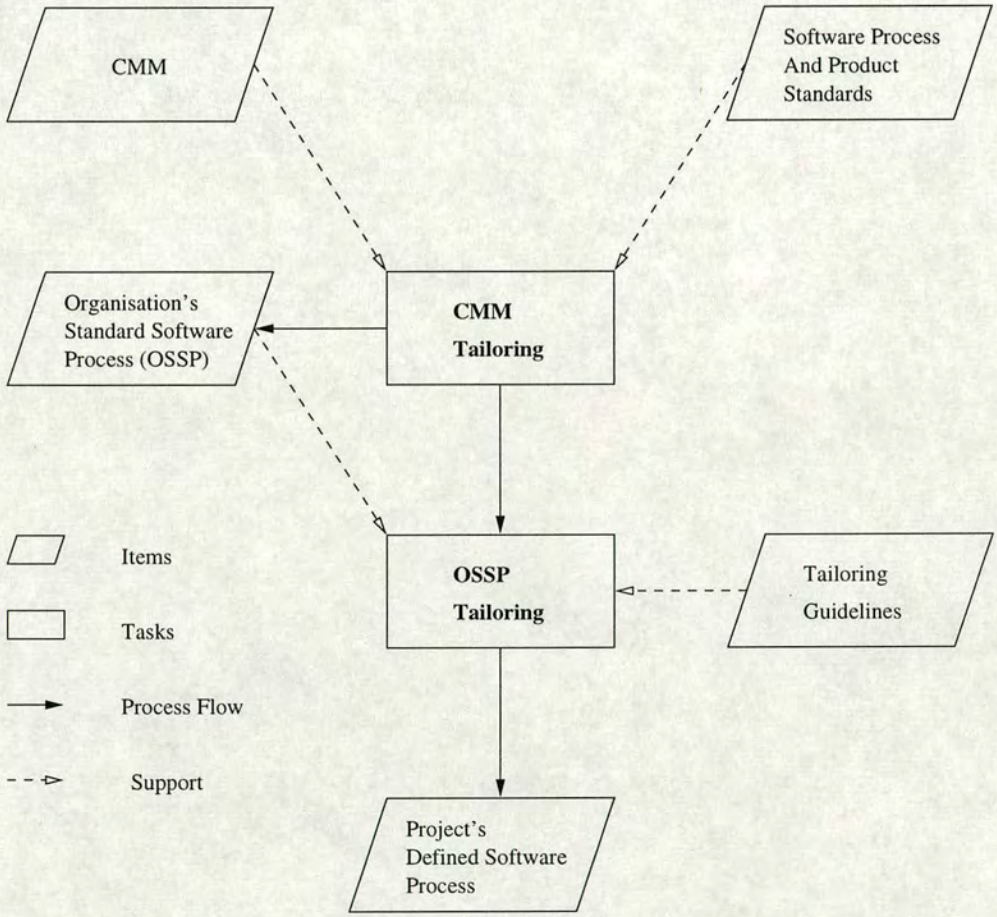

Figure 3.2: A Tailoring Framework

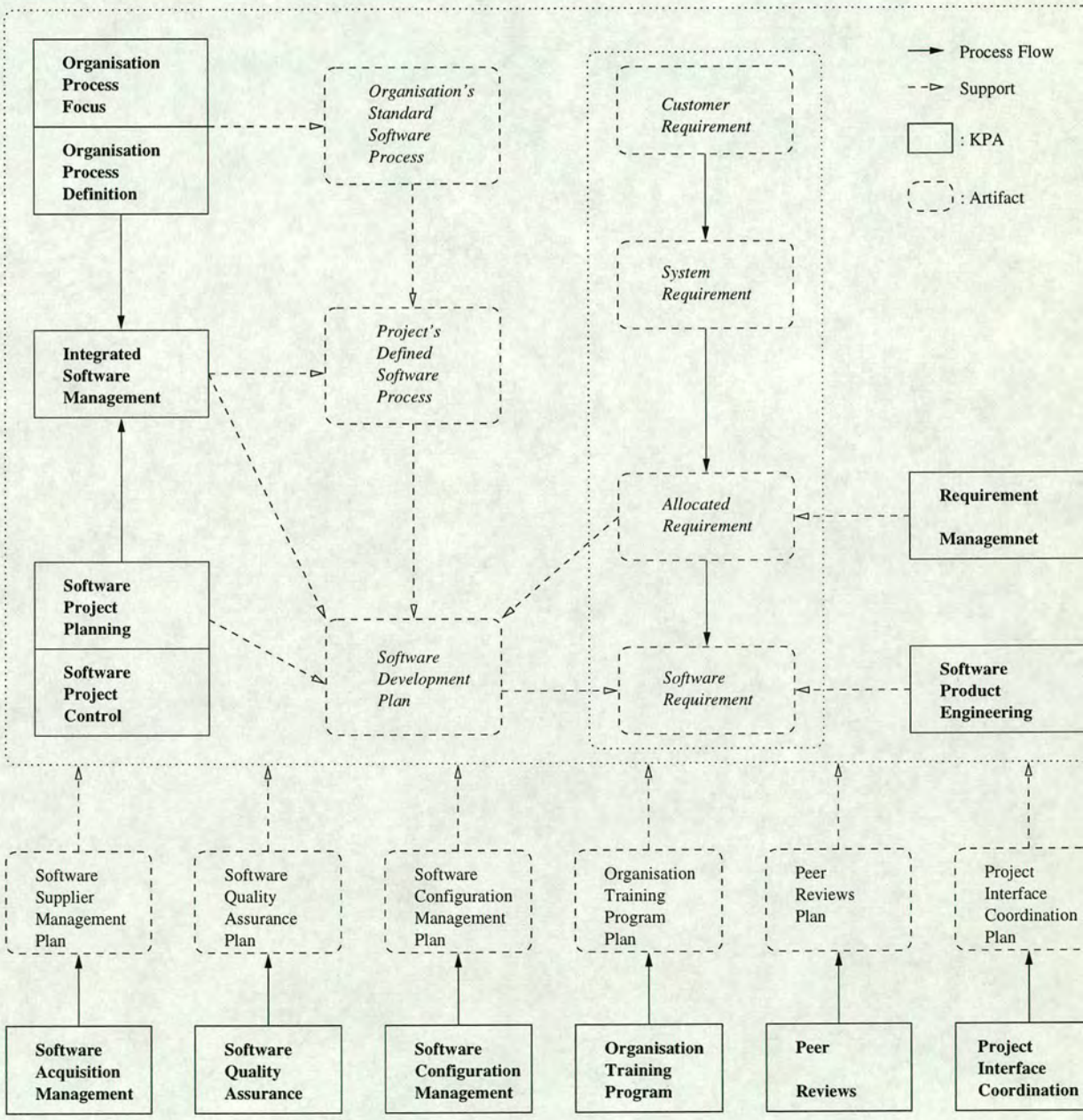

Figure 3.3: The Relevant Activities and Products in the CMM Levels 2 and 3

of an effective software process. The SEI suggested that a software process can be defined as a set of activities, methods, practices and transformations that people use to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases and user manuals). As a consequence, the CMM provides software organisations with guidance on how to gain control of their processes for developing and maintaining software and how to evolve toward a culture of software engineering and management excellence.

A fundamental concept of process definition in the CMM is the organisation's standard software process. An organisation's standard software process is the operational definition of the basic process that guides the establishment of a common software process across the software projects in the organisation. It describes the fundamental software process elements that each software project is expected to incorporate into its defined software process.

The primary purpose of the standard software process is to support the sharing of software process assets and experiences across the organisation. The organisation's standard software process covers the entire software process, including:

- software technical processes,
- software management processes,
- software support processes, and
- organisational processes.

The organisation's standard software process given in SPI PASTA provides a general but not exhaustive definition which can be modified by an organisation. The project's defined software process is developed by tailoring the organisation's standard software process to fit the specific characteristics of the project. It is a well-characterised and understood software process, described in terms of software standards, procedures, tools and methods.

However, the description of the project's defined software process will usually not be specific enough to be performed directly. It does not specify the individual who will assume the roles, the specific software work products that will be created, nor the schedule for performing the tasks and activities. The project's software development plan provides the bridge between the project's defined software process (what will be done and how it will be done) and the specifics of how the project will be performed (e.g., which individuals will produce which software work products according to what schedule). The combination of the project's

defined software process and its software development plan makes it possible to actually perform the process.

For completely establishing the project's software development plan, the system requirements allocated to software, simply as the "allocated requirements" in the CMM, must be a primary input to the software development plan. The allocated requirements are a subset of the system requirements which are an elaboration of the customer requirements to a level of detail needed to plan the project's activities and work products.

Once the software development plan has been established, the software engineering group can start analysing and elaborating the allocated requirements to complete the software requirements. The software requirements are the technical requirements for the software project and cover the software functions and performance and the interfaces to hardware, other software components and other system components. The software requirements form the basis for the software design, coding, testing, documentation, delivery, support and maintenance.

In addition to the artifacts and activities described above, some support artifacts have to be established. The software supplier management plan identifies acquisition needs for the software project. Software to be acquired falls into two main categories. Firstly, some components of the project's software products may be acquired externally rather than being developed by the software project. Secondly, the software tools in the software engineering environment must be acquired.

The purpose of software quality assurance is to ensure that software project's activities and work products comply with the applicable requirement, process descriptions, standards and procedures. The purpose of software configuration management is to establish and maintain the integrity of the products of the software project throughout the project's software life cycle.

The purpose of an organisation training program is to develop the skills and knowledge of individuals so they can perform their roles effectively and efficiently. The purpose of peer reviews is to remove defects from the software work products early and efficiently. The purpose of software risk management is to identify and mitigate software risks throughout the life cycle of a software product.

3.2 Organisation Process Focus

> *The purpose of Organisation Process Focus is to establish and maintain an understanding of the organisation's software processes and coordinate the organisation's software process improvement activities[Pau97].*

3.2.1 The Software Process Improvement Plan

Software process improvement is a non-stop business. SEL[MPB94] reported some attributes of the development organisation were an increasingly significant driver for the overall definition of process change. These attributes include the types of software being developed, goals of the organisation, development constraints, environment characteristics, and organisational structure. This means software organisations have to establish a baseline understanding of the software process, products, and goals. This is the purpose of Organisation Process Focus.

Currently, two famous examples for software process improvement are NASA's SEL process improvement paradigm and the SEI's IDEAL model. Figure 3.4 shows the SEL process improvement paradigm that includes three phases, Understanding, Assessing and Packaging. This paradigm starts with improving insight into the software process and its products, then measuring the impact of available technologies and process change on the products generated, and finally implementing the technology for application. In the SEL process improvement paradigm, these steps are addressed sequentially, and iteratively, for as long as process and product improvement remains a goal within the organisation.

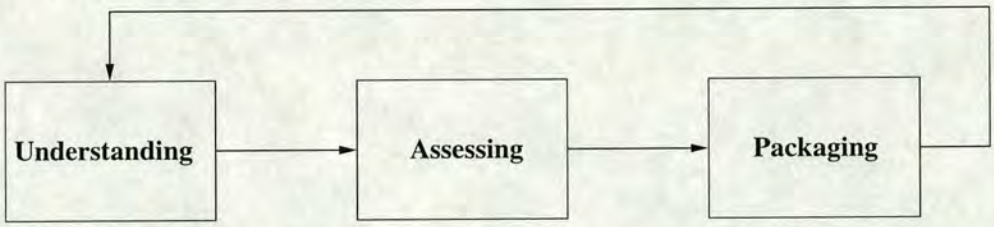

Figure 3.4: The SEL Process Improvement paradigm

Figure 3.5 shows the phases of the SEI's IDEAL model. Each phase includes activities and resources needed for a successful process improvement effort. Unlike the SEL, the IDEAL model starts with building an infrastructure for managing implementation details, and securing the support and resources from the top level of the organisation. After the Initiating phase, software organisations have to develop a more complete understanding of the improvement work. Then, a detailed plan for doing the work is developed and implemented. Finally, the entire IDEAL experience has to be reviewed to determine whether the effort accomplished the intended goals, and how the organisation can implement change more effectively and efficiently in the future.

Both strategies give us an idea of Total Quality Management on software process improvement. The idea is not fixed by any standard and must be encompassed by the whole organisation. One of the most important ingredients

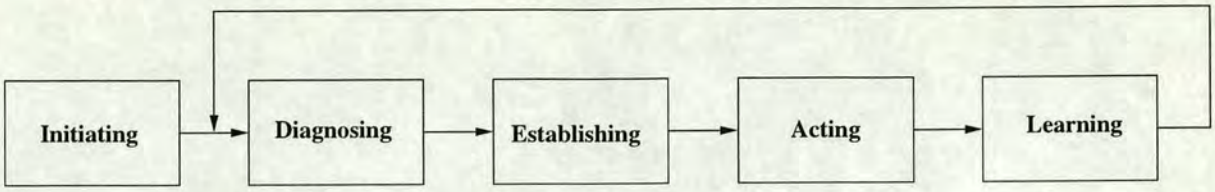

Figure 3.5: The IDEAL Model for Software Process Improvement

in Organisation Process Focus is to check necessary competencies and commitments. It must ensure that ability to perform is put aside to make sure that implementation works and that people are committed to the solution.

3.2.2 Processes in Software Process Improvement

The tree-diagrams present the hierarchy of artifacts, roles and process states. By expanding the P-state tree diagram, users can see the relationship between P-state and easily recognise all relevant operations. Figure 3.6 shows the P-state tree of Software Process Improvement which describes the relations of Sub-P-State.This is the highest level in the model. From the beginning, an understanding of organisation's software processes must be established. The organisation's software process assets are then built by a software engineering process group (SEPG). Finally, the set of standard software processes is tailored by the software projects and support groups to create their defined software processes. Figure 3.7 shows the processes of developing software process improvement.

The processes of developing software process improvement start at Organisation Process Focus described in the CMM Level 3. The SEI emphasised that a repeatable process (Level 2) must be finished before implementing a defined process (Level 3). However, we believe that the organisation's standard software process should be established as soon as software process improvement is developing. Each project needs its defined software process to guide all activities. Without a defined software process, the software development plan is hardly complete and management processes could be in jeopardy.

In the rest of the thesis, we use the following template to describe each KPA:

- **Main Roles:** A list of roles of people who are allowed to perform this operation.
- **Entrance Conditions:** Conditions required for entry of an activity.
- **Artifact List:** A list of artifacts upon which work may proceed in the P-state.

- **Information Artifacts:** The artifacts which hold information required to support operations.
- **Activities:** A list of operations that developers may perform.
- **Exit Condition:** Conditions required for exit of an activity.

This is a P-State style template and corresponds to the P-State diagram to show the relevant information for the KPAs.

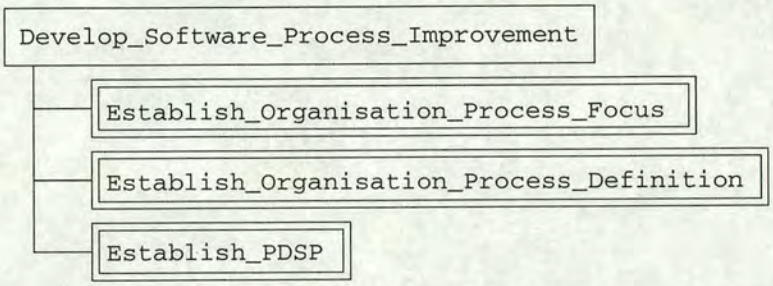

Figure 3.6: The P-State Tree of Software Process Improvement

3.2.3 Processes in Organisation Process Focus

The first step in developing software process improvement is to establish and maintain an understanding of the organisation's software process. Figure 3.8 shows the P-state tree of Organisation Process Focus. It describes the relationship between P-states and operations in this KPA.

Main Roles: The main roles participating in the operations are senior managers and SEPG who establish the policy for the organisations.

Artifact List: The artifact in this KPA is the Software Process Focus which includes three sub-artifacts, Process Improvement Plan, Action Plan and Software Process Definition as shown in Figure 3.9.

Entrance Condition: state-of(Organisation_Process_Focus) = Referenced

Once the need to establish and maintain an understanding of the organisation's software process is referenced, the processes of Organisation Process Focus are started. Firstly, developers must check the artifact list to survey the relationship between operations and artifacts. To do so, developers can use the artifact tree and link to artifacts forms which list the details of the artifact (See Appendix).

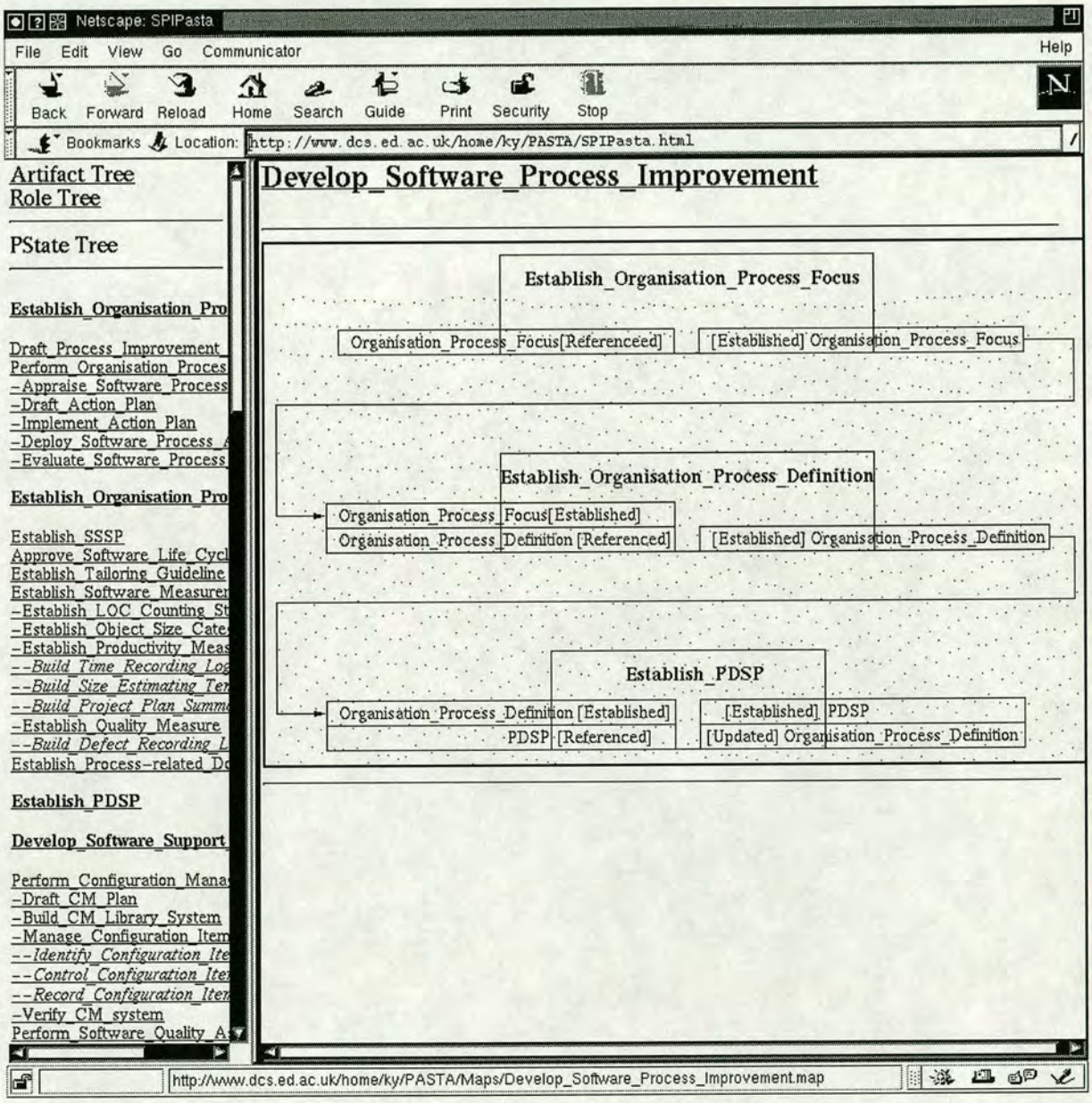

Figure 3.7: The P-State Diagram of Software Process Improvement

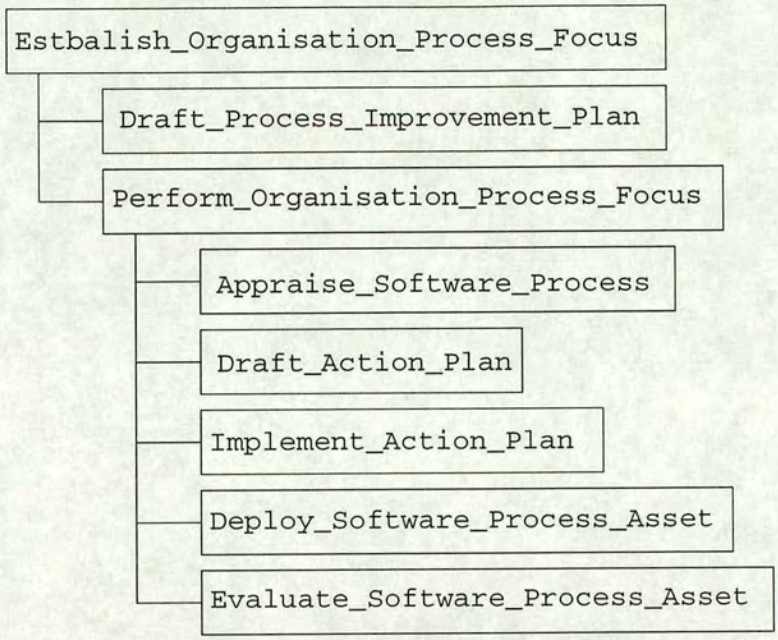

Figure 3.8: The P-State Tree of Organisation Process Focus

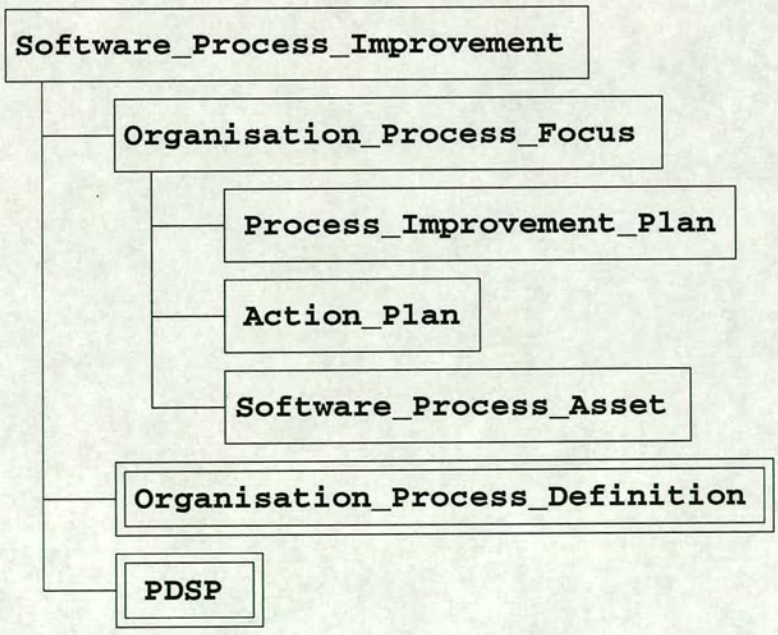

Figure 3.9: The Artifact Tree of Organisation Process Focus

Activities: Two operations are presented in the P-state diagram of Organisation Process Focus as shown in Figure 3.10. Before performing Organisation Software Focus, the software process improvement plan must be established. The SPI PASTA model might be a software process improvement plan, since it documents the process for software process improvement. The SPI PASTA can be tailored in order to align with the organisation's strategic business objectives.

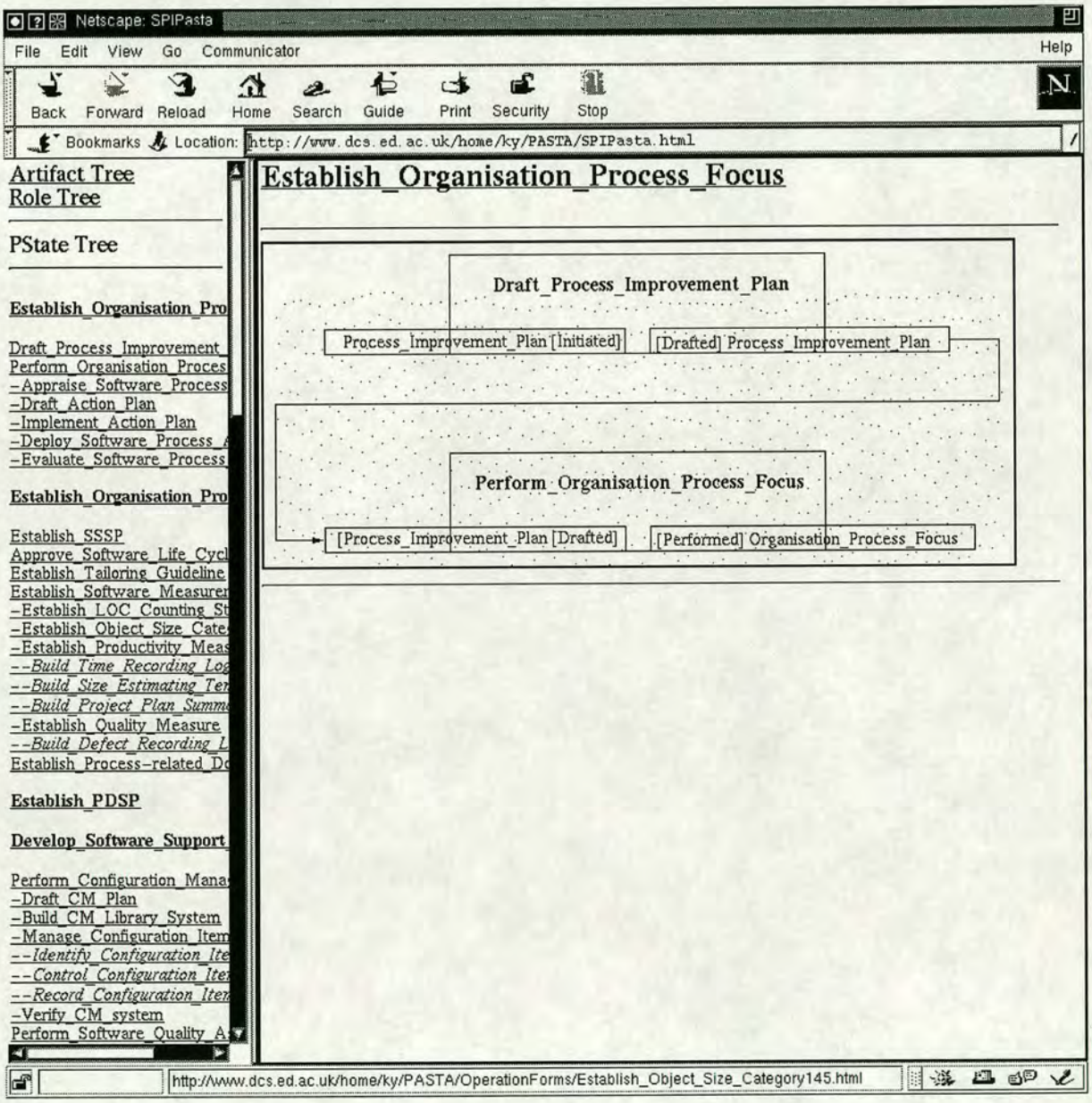

Figure 3.10: The P-State Diagram of Organisation Process Focus

In the next step, the activities performed in Software Organisation Focus are included in the Sub-P-State, Perform_Organisation_Process_Focus. When the

process improvement plan is drafted, developers can go down to the next layer to perform Organisation Process Focus. This P-state consists of five operations: Appraise Organisation Process, Draft Action Plan, Implement Action Plan, Deploy Software Process Asset and Evaluate Software Process Asset. After appraising the organisation's software process to identify strengths and weaknesses, SEPG and senior managers should establish action plans to address the findings of the software process appraisals. Then, in accordance with the priority of the recommendations from the software process appraisal, SEPG implements the software process action plan across the organisation and deploys the organisation's software process assets. Finally, SEPG should conduct the evaluation of the action plan and derive lessons learned from these operations. In SPI PASTA, all P-states and operations have their own P-state and operations forms (See Appendix) describing the relevant activities and conditions. Developers can easily refer to these forms to perform those activities.

Exit Condition: state-of(Organisation_Process_Focus) = Established

After completing the P-states, SEPG must check whether the exit condition has been reached.

3.3 Organisation Process Definition

> *The purpose of Organisation Process Definition is to establish and maintain a usable set of software process assets that help to ensure consistent process performance across the organisation and that provide a basis for cumulative, long-term benefits to the organisation[Pau97].*

Organisation Process Definition is the action of the Organisation Process Focus key process area. It involves establishing and maintaining the organisation's software process assets, including

- the organisation's set of standard software processes,
- descriptions of software life cycle models approved for use by the projects,
- guidelines for tailoring the organisation's set of standard software processes,
- the organisation's software measurement database, and
- the organisation's library of software process-related documentation

However, the SEI does not prescribe what kind of models or processes may be organisation's software process assets. Those assets are chosen by project managers or senior managers in an organisation.

As Figure 3.11 shows, the Organisation Process Definition key process area provides a base to establish process commonality across the organisation's projects. The activities managing the organisation's software process assets are in the Organisation Process Focus key process area. Once the software process assets are established, the project managers can use them in developing, maintaining and implementing their project's defined software process.

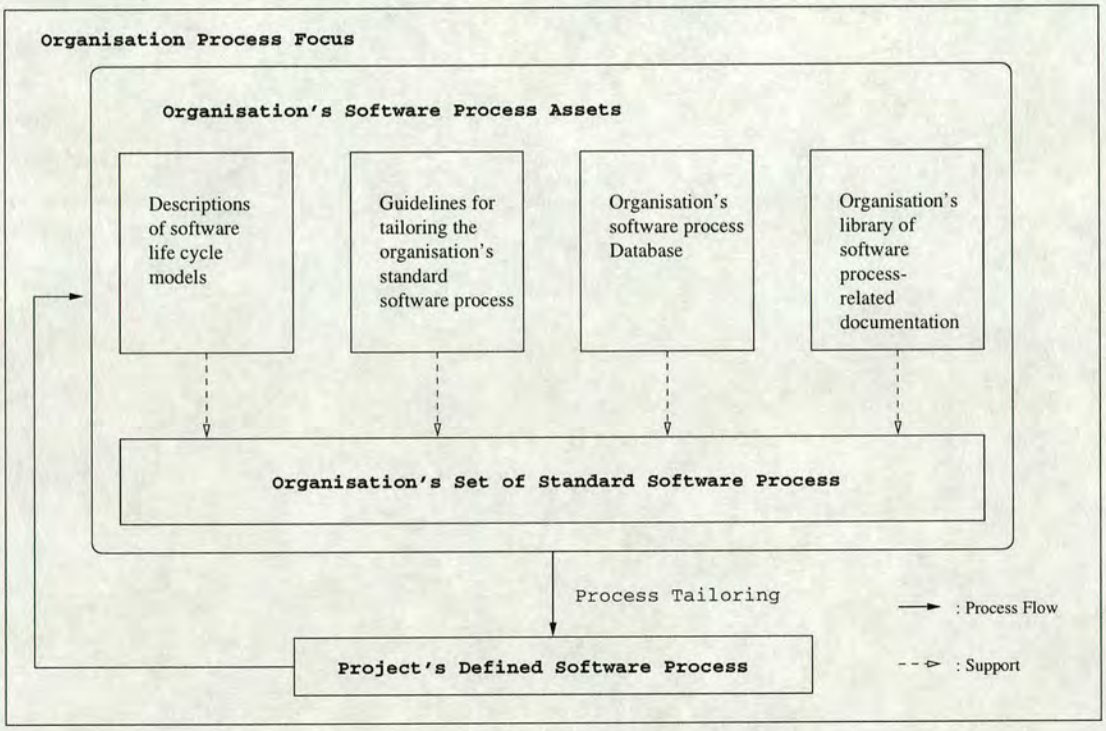

Figure 3.11: The Framework of the Organisation's Standard Software Assets

3.3.1 Organisation's Software Process Assets

- Organisation's Standard Software Process

  An organisation's standard software process is the operational definition of the basic process that guides the establishment of a common software process across the software projects in the organisation[PWG+93]. It contains process elements that may be interconnected according to one or more software process architectures that describes the relationships among these process elements. SPI PASTA provides this software process architecture

that describes the ordering, interfaces, interdependencies and other relationships between the software process elements of the organisation's standard software process. Each artifact is a software process element which is a constituent element of a software process description. These artifacts are incorporated into the relevant P-State performed by appropriate roles.

- Software Life Cycle Model

  Software life cycle models partition the life of the software product into phases that guide the project through the major steps of identifying customer needs, developing, testing, installing, operating and retiring the software product[Pau97]. Software organisations have their software life cycle model, such as a waterfall model, a spiral model and so on. Furthermore, the standards have been published to support software development, such as ISO/IEC 12207, MIL-STD-498, IEEE-STD-1074, etc. These models or standards suggest activities, tasks and products for developing software projects. The CMM does not prescribe software organisations to use any standard or model. This task must be done in the Organisation Process Definition key process area. SPI PASTA uses MIL-STD-498 as its software life cycle model. MIL-STD-498 was developed by the US DoD and well incorporated into the CMM. It has been modelled as SPI PASTA in management processes and technical processes. Each Data Item Description (DID) has been deployed at the appropriate key practice. However, this standard has to be tailored again to fit specific software projects. The project manager must decide which DID is really appropriate for the project.

- Tailoring Guidelines

  Tailoring guidelines are used by software projects and support groups to tailor the organisation's set of standard software processes to fit their specific needs[Pau97]. Since the organisation's standard software process defines common processes for all projects, a tailoring guideline must be built in order to tailor SPI PASTA to fit the projects. Ginsberg[GQ95] suggested that some areas, such as the organisational structure, customers relationships and requirements, business goals and so on, are essential for tailoring the organisation's standard software process. With regard to DIDs, it is not necessary that every project needs all DIDs defined in MIL-STD-498. Tailoring guidelines have to give project managers a trade-off to select the appropriate process items for their projects.

- Organisation's Software Measurement Database

The organisation's software measurement database is used to collect and make available measures and data on the software processes used in the organisation and on the resulting work products, especially the measures relating to the organisation's set of standard software processes[Pau97]. In SPI PASTA, we adopt Humphrey's Proxy-based Estimating (PROBE)[Hum95] to assist project managers with establishing a plan for building the software product. This work may generate useful data for software organisations. SPI PASTA does not prescribe a typical database for storing this data. Software organisations may use any commercial database to define their database scheme to store all information from software process improvement. This database will be viewed as an organisation's software measurement database.

- Process-related Documentation

  The organisation's library of software process-related documentation is used to collect, store and make available process documentation that is potentially useful to current and future projects[Pau97]. The process-related documentation, such as software development plans, software test plans, training records and so on, might be a pilot process item for software organisations. In particular, it helps future project managers to manage their projects. Without an appropriate library of software process-related documentation, the development team may take much time to figure out the relevant documents.

3.3.2 Processes in Organisation Process Definition

Figure 3.12 shows the P-state tree of Organisation Process Definition. This P-state consists of five operations: Establish SSSP (Set of Standard Software Processes), Approve Software life cycle, Establish Tailoring Guideline, Establish Software Measurement Database and Establish Process-related Documentation.

Main Roles: The main roles participating in the operations are senior managers, SEPG, project managers and software product managers who will establish the software process assets for the organisation.

Artifact List: The artifact list in the KPA is Organisation Process Definition which consists of five sub-artifacts: SSSP, Software Life Cycle, Tailoring Guideline, Software Measurement Database and Process-related Documentation. The artifacts in this KPA are shown in Figure 3.13.

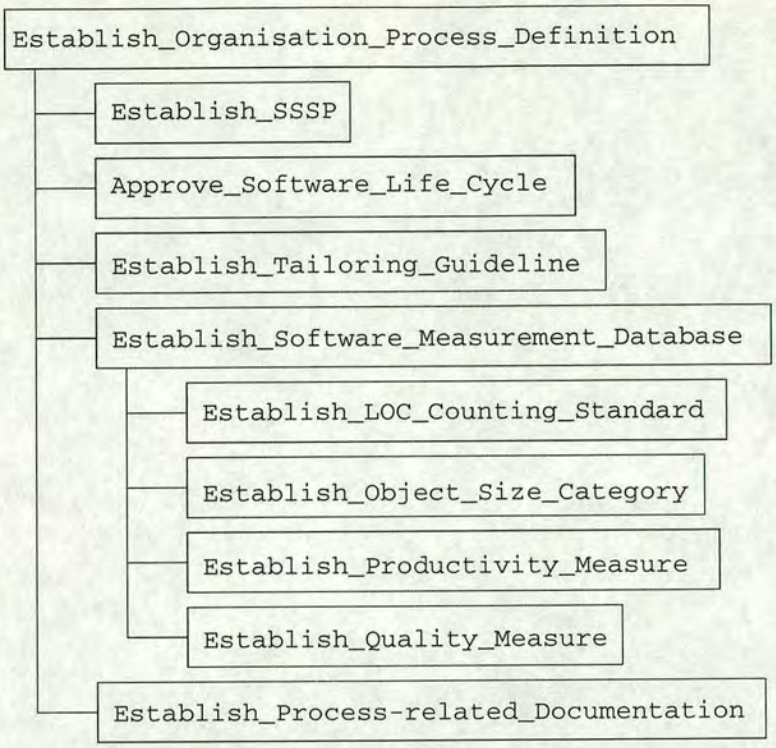

Figure 3.12: The P-State Tree of Organisation Process Definition

Information Artifacts: The artifact, Organisation Process Focus, as an information artifact give the information to manage those artifacts.

Entrance Condition: state-of(Organisation_Process_Focus) = Established and state-of(Organisation_Process_Definition) = Referenced

Firstly, the need to perform Organisation Process Definition must be identified. Then, the activities of Organisation Process Focus should be completed. Once the two conditions are satisfied, developers may enter the Organisation Process Definition process.

Activities: Five operations are presented in the P-state diagram of Organisation Process Definition as shown in Figure 3.14. These operations do not have relationship between them. This means each operation can be independently performed by developers.

Firstly, SPI PASTA can be viewed as the organisation's set of standard software processes for software organisations, since it has tailored the software processes and decomposed each standard software process into constituent process elements (artifacts). The SEPG and managers can use SPI PASTA to specify

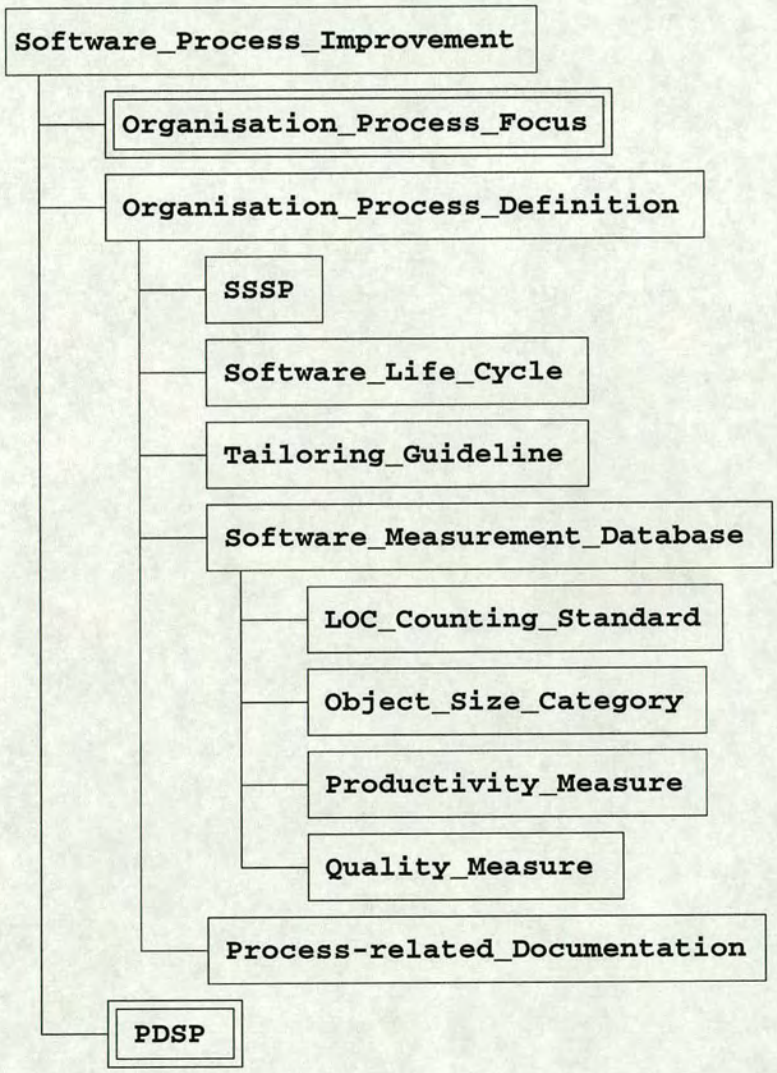

Figure 3.13: The Artifact Tree of Organisation Process Definition

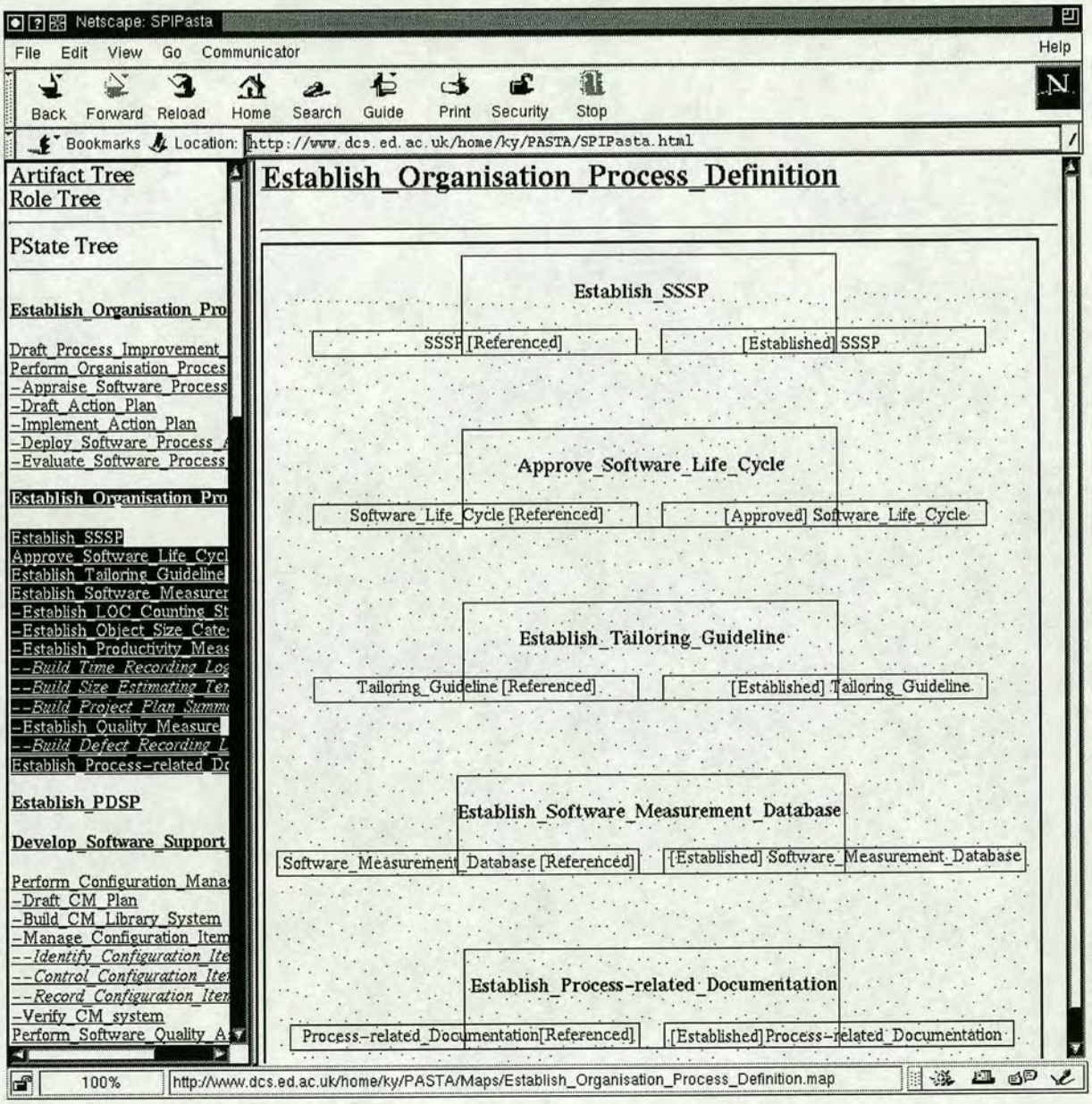

Figure 3.14: The P-State Diagram of Organisation Process Definition

the critical attributes and relationships from artifacts and P-states for the organisation. Moreover, this operation must confirm that all process elements and processes adhere to the organisation's policies and strategies. This set of standard software processes will be tailored by the project managers to fit the software projects' need in the Integrated Software Management key process area. Furthermore, with Organisation Process Focus, the organisation's set of standard software processes should be checked periodically and receive feedback from the active project managers.

Secondly, since the CMM does not prescribe a specific life cycle model, the SEPG and managers must select the most appropriate life cycle model for the organisation. MIL-STD-498 adopts three program strategies, grand design, incremental and evolutionary for developing software. In addition, iterative and spiral models are also very popular life cycle models.

Thirdly, SPI PASTA is described at a general level that might not be directly usable by a project. In this operation, the SEPG and managers have to build a tailoring guideline providing the criteria and procedures for selecting artifacts and P-states. The project managers can adhere to the guideline tailoring SPI PASTA to accommodate the project's characteristic and needs.

Fourthly, historical data is the most essential information for estimating software size, effort and cost. Humphrey in his book[Hum95] designed templates to collect software measures. These measures have to be stored in the software measurement database as the historical data for next estimation. In addition to the time and defect record, SEPG and managers must build the LOC counting standard and object size category in order to estimate the productivity.

Finally, SEPG and managers must design a library to collect process-related documentation. These documents can be reused in future projects

Exit Condition: state-of(Organisation_Process_Definition) = Established

After completing the P-states, SEPG and managers must check whether the exit condition has been reached.

3.4 Integrated Software Management

The purpose of Integrated Software Management is to proactively manage the software project according to an integrated, coherent, and defined software process that is tailored from the organisation's set of standard software processes[Pau97].

As Figure 3.18 shows, the main activity in the Integrated Software Management key process area is tailoring the project's defined software process from the organisation's standard software process. Moreover, the implementation and management of the activities of the project's defined software process is primarily described in the software development plan.

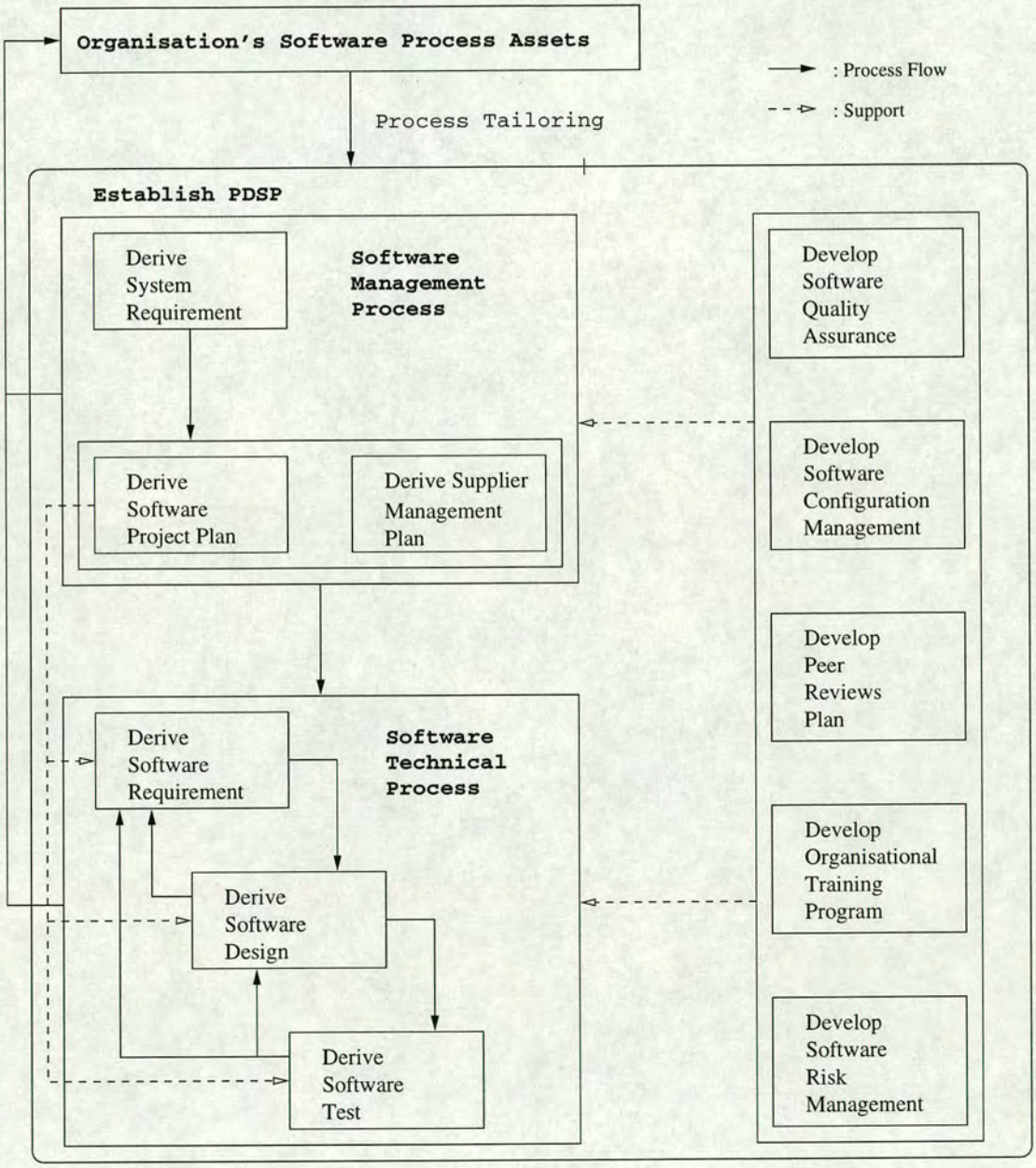

Figure 3.15: The Framework of the Project's Defined Software Process

3.4.1 The Project's Defined Software Process

The project's defined software process (PDSP) is a well-characterised and understood software process, described in terms of software standards, procedures, tools and methods. The PDSP provides the basis for planning, performing and improving the activities of the managers and technical staff performing the project's tasks and activities.

In the CMM[PWG+93], the work to be performed is broken down into tasks. Within the context of process definition, a task is a well-defined component of a defined process. In SPI PASTA, tasks are described as the P-state of a process which is the completeness of the process. Tasks can be divided into activities which are steps taken or functions performed toward achieving some objective. Operation forms provide a complete description for these activities. The results of P-states and operations primarily consist of artifacts, called software work products in the CMM. Artifacts can be anything created in the software process, which include process descriptions, plans, procedures, computer programs and associated documentation.

3.4.2 Process Tailoring

Since the PDSP is tailored to fit the specific characteristics of the project, the project manager must carefully decide what artifacts are essential for the project. A set of tailoring guidelines must be established in the Organisation Process Definition key process area. Ginsberg[GQ95] suggested that the organisation's tailoring guidelines must be developed and applied in a manner that will preserve the benefits of having common practices based on the organisation's standard software process. Moreover, the guidelines must grant projects the flexibility to operate efficiently, while also preserving the maximum amount of commonality possible.

SPI PASTA is built assuming a full development life cycle. We do not suppose all organisations can fit this model. For example, it is not necessary to adopt the Software Acquisition Management key process area for all projects. Furthermore, the artifacts from MIL-STD-498's DIDs, such as System/Subsystem Design Description (SSDD), Software Design Description (SDD), Interface Design Description (IDD), may be tailored to one description. The SEPG and managers have to depend on features of the project and the culture of the organisation to tailor these artifacts.

3.4.3 The Software Development Plan

MIL-STD-498[DOD94] defines the Software Development Plan (SDP) as a developer's plans for conducting a software development effort. Moreover it provides the acquirer insight into, and a tool for monitoring, the processes to be followed for software development, the methods to be used, the approach to be followed for each activity, and project schedules, organisation and resources. NASA's SEL[NAS90] suggests that the SDP provides a disciplined approach to organising and managing the software project. A successful plan serves as

- a structured checklist of important questions,
- consistent documentation for project organisation,
- a baseline reference with which to compare actual project performance and experiences, and
- a detailed clarification of the management approach to be used.

The description of the PDSP will usually not be specific enough to be performed directly. It describes "what" the project will be performed rather than "how" the project will be performed. It does not specify the individual who will assume the roles, the specific artifacts that will be created, nor the schedule for performing the tasks and activities. The SDP should be living documents that represent how the project work will be performed.

The SDP might be either a single document or a collection of plans collectively referred to as a software development plan. The typical SDP is composed of a project mission plan, an organisation and responsibility plan, a software engineering activity plan, a schedule and resource plan, a configuration management plan and so on. The combination of the PDSP and its software development plan makes it possible to actually perform the process.

3.4.4 Processes in the PDSP

We use the title "Processes in the PDSP" instead of "Processes in Integrated Software Management", because the main activity in Integrated Software Management is to establish the PDSP. There are other activities in the Integrated Software Management key process area such as risk management. We would like to focus on establishing the PDSP and split risk management as an operation in the organisational processes. Therefore, in this P-state, the entire software processes must be tailored to fit the project.

Figure 3.16 shows the P-state tree of PDSP. This P-state consists of four operations: Develop Software Support Process, Develop Software Management Process, Develop Software Technical Process and Develop Organisational Process.

Main Roles: The main roles participating in the operations are senior managers, SEPG, project managers and software product managers who will develop defined software processes for the projects.

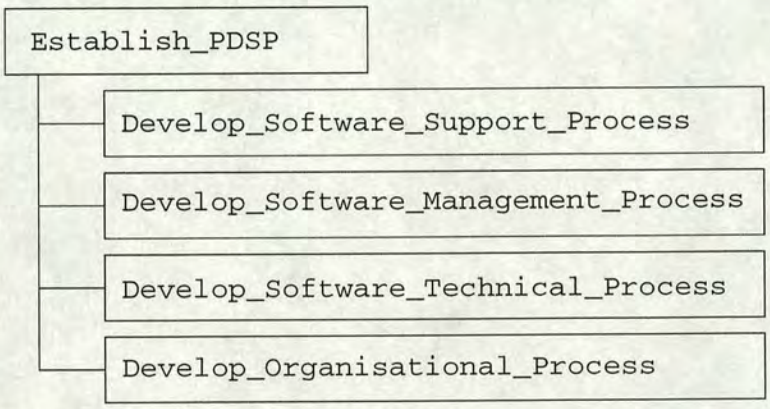

Figure 3.16: The P-State Tree of the PDSP

Artifact List: The artifact list in the KPA is the PDSP which consists of four sub-artifacts: Software Support Process, Software Management Process, Software Technical Process and Organisational Process. The artifacts in this KPA are shown in Figure 3.17.

Information Artifacts: The artifact, Organisation Process definition, provides organisation's standard software processes, selected software life cycle models, tailoring guideline, the software measurement and process-related documentation to perform the project's defined software process.

Entrance Condition: state-of(Organisation_Process_Definition) = Established
and state-of(PDSP) = Referenced

In addition, the need to perform the PDSP must be identified, the activities of Organisation Process Definition should first be completed. Once two conditions are satisfied, developers may enter the Project's Defined Software Process.

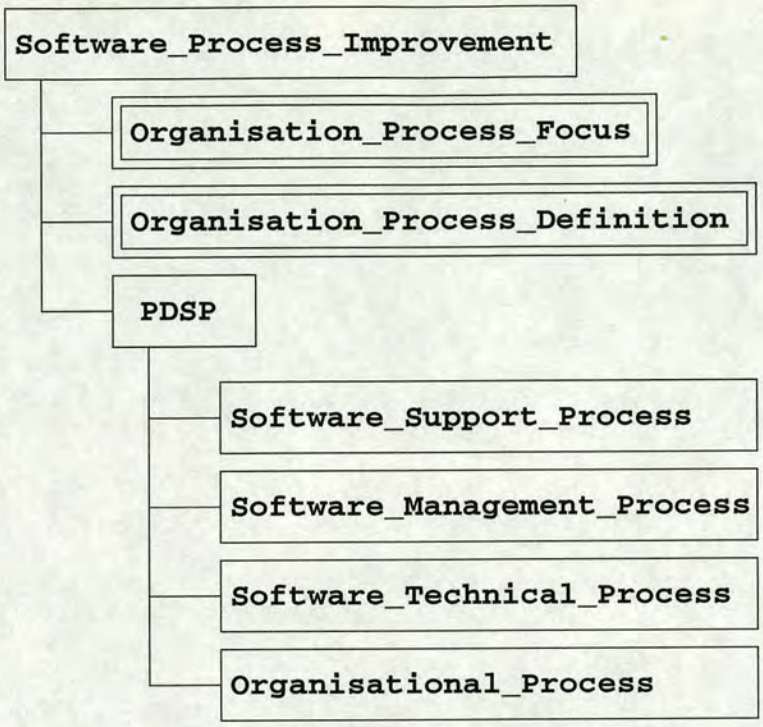

Figure 3.17: The Artifact Tree of the PDSP

Activities: Four operations are presented in the P-state diagram of the PDSP as shown in Figure 3.18. These operations cover the entire software processes. However, as Humphrey[Hum88] mentioned, it is better to focus on management processes before engineering processes.

To tailor the PDSP for the entire software process, SEPG and managers should first select a software life cycle model appropriate to the scope, magnitude and complexity of the project from those available from the organisation. Then, SEPG and managers tailor SPI PASTA according to the tailoring guideline. Unfit artifacts and P-states may be disabled and all activities in the P-state should be carefully checked. In the meantime, it is better to concurrently build the software development plan and ensure that SPI PASTA is appropriately reflected in the software development plan. Other activities can be started at this KPA, or linked to relevant P-state such as Organisation Training Program.

Exit Condition: state-of(PDSP) = Established and
state-of(Organisation_Process_Definition) = Updated

After completing the P-states, SEPG and managers must check whether the exit condition has been reached. In addition the PDSP must be completely established, and all information performed in the PDSP, such as product measures,

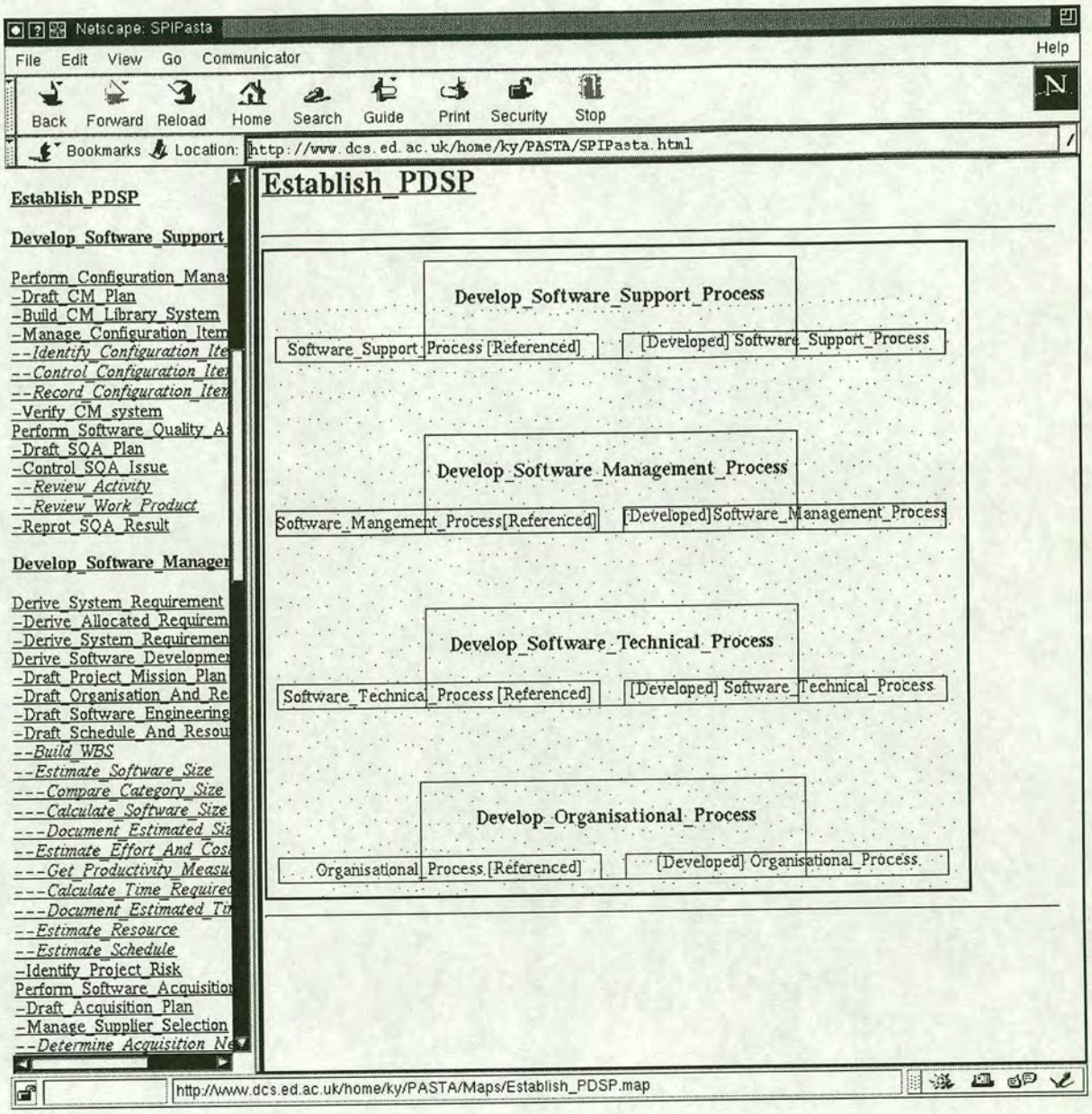

Figure 3.18: The P-State Diagram of the PDSP

should be contributed to the Organisation Process Definition key process area as the historical data for the future projects.

Chapter 4

The Processes in the CMM Level 2

PRINCE process model[PRI97] describes how a project is divided into manageable stages enabling efficient control of resources and regular progress monitoring throughout the project. The various roles and responsibilities for managing a project are fully described and are adaptable to suit the size and complexity of the project, and the skills of the organisation. Meanwhile, the purpose of the ISO 15504 project[Gar98, Kit97] is to provide software organisations pursuing multiple improvement approaches. This project tries to establish a framework for understanding the state of the organisation's processes for process improvement. In the same way as the ISO 15504 project and PRINCE process model, the key process areas at Level 2 also focus on the software project's concerns related to establish basic project management controls[Pau97]. To complete the activities of Level 2, software costs, schedules, and functionality should be well controlled, software requirements and the work products developed to satisfy them should be clearly defined, and the relationship with contractors should be established. This chapter contains two parts, software management process and software support processes. These processes will help software organisations become disciplined development teams and effectively control software development.

4.1 The Software Management Process

Software management processes are described on the CMM Level 2 and have become a basis of software development. Humphrey[Hum88] emphasised that software management processes must be done before engineering processes, since without management discipline, the engineering process is sacrificed to schedule and cost pressures. He did not agree that a Level 1 organisation tries to implement

a defined process (Level 3) before it has established a repeatable process (Level 2).

Figure 4.1 shows the framework of software management processes. The Software Project Planning key process area concentrating on planning and scheduling activities is a main part of software management processes. Firstly, the software project is divided into the work breakdown structure which is based on the allocated requirements from the Requirement Management key process area. In the meantime, the activities of tracking the software project are concurrently performed. In addition, software production can also be done by other organisations working under contract with the project.

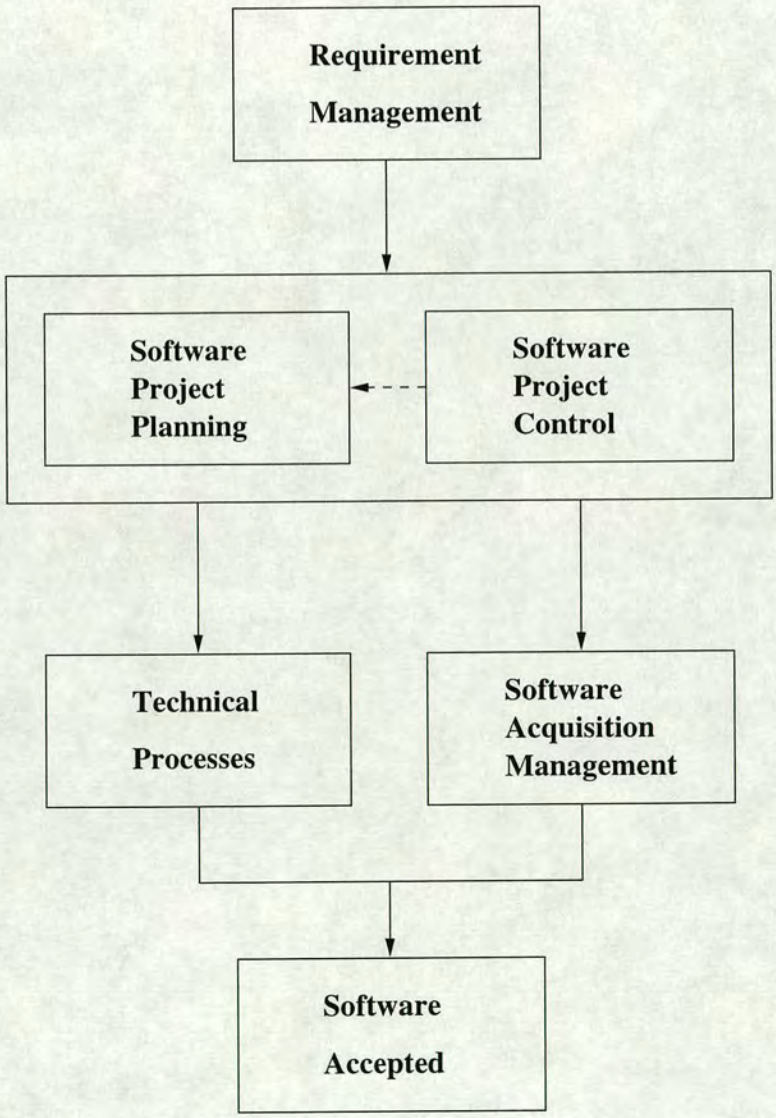

Figure 4.1: The Framework of the Software Management Processes

Processes in the Software Management Process

Figure 4.2 shows the P-state tree of the Software Management Processes. This P-state consists of five operations: Derive System Requirement, Derive Software Development Plan, Perform Software Acquisition Plan, Perform Software Project Control and Establish Commitment. The first four operations belong to the KPAs of the CMM Level 2. Moreover, these activities should establish the software project's internal and external commitments.

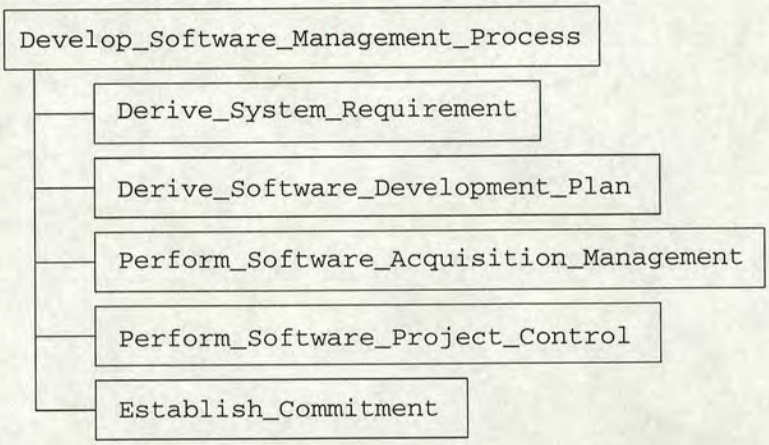

Figure 4.2: The P-State Tree of the Software Management Process

Main Roles: The whole Management Group should be involved in developing software management processes. Furthermore, the software product manager conducting the technical processes and customers are also involved in performing activities in the Software Management Process.

Artifact List: The artifact list in this part is the Software Management Process which consists of five sub-artifacts: System Requirement, Software Development Plan, Software Acquisition Plan, Software Project Control and Commitment. The artifacts in the software management process are shown in Figure 4.3.

Information Artifacts: The artifact, PDSP, provides the tailored software management processes for the management group.

Entrance Condition: state-of(Software_Management_Process) = Referenced

Once the need to develop a Software Management Process has been identified, the process roles start to perform all activities in the management processes.

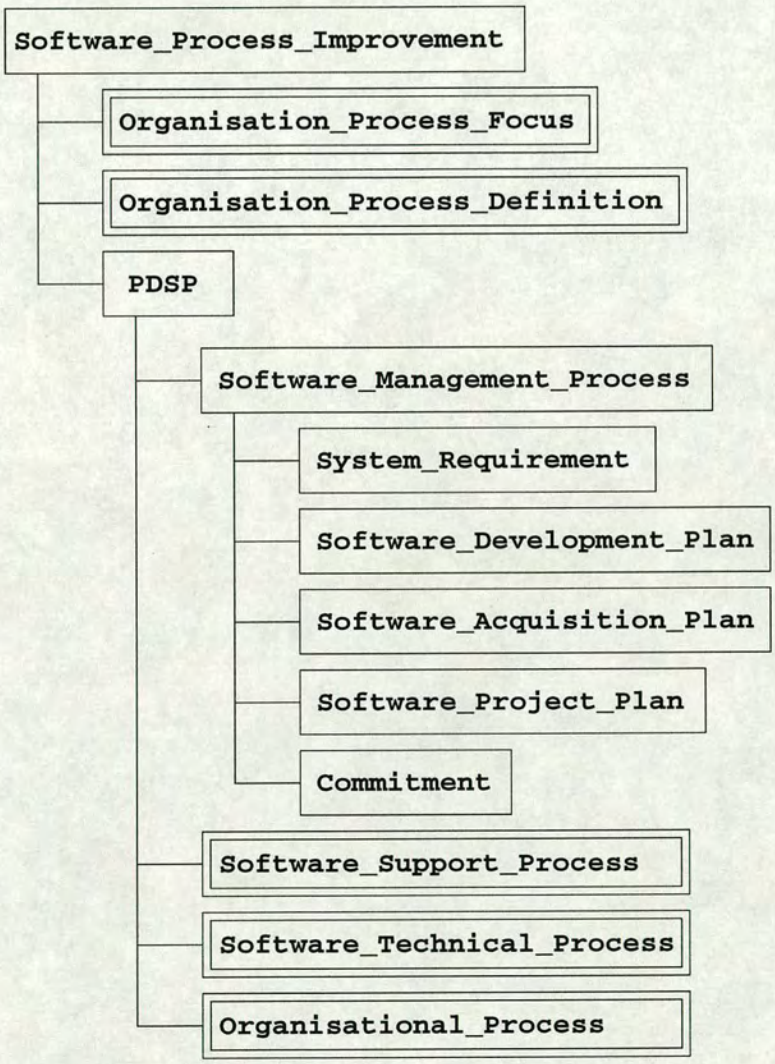

Figure 4.3: The Artifact Tree of the Software Management Process

Activities: Five operations are presented in the P-state diagram of the Software Management Process as shown in Figure 4.4. System requirements should firstly be derived; however, it is not necessary to complete system requirements prior to the software development plan. Software project control must rely on the software development plan to track software project performance and take corrective actions. When the needed acquisition of software and associated work products is identified, the software acquisition management shall be performed; otherwise this operation can be omitted. In the former operations, process roles should establish internal and external commitments to ensure all activities are reliable.

Exit Condition: state-of(Software_Management_Process)=Developed

After completing the P-states, managers must check whether the exit condition has been reached and the final commitments have been reached.

4.1.1 Requirements Management

The purpose of Requirements Management is to establish and maintain a common agreement between the customer and the software project regarding the customer's requirements that will be addressed by the software project[Pau97].

4.1.1.1 System Requirements Allocated to Software

Figure 4.5 presents the architecture of requirements. Originally, a statement of the customer's needs and expectations for the project is gradually formed and abstracted. The customer requirements are stated from the customer and end user perspectives, and are intended to achieve a shared understanding between the customer and the project, and provide the criteria to determine whether the products satisfy the customer's needs and expectations. However, if the customer requirements are not suitable for the project development, it should be elaborated to a level of detail needed to plan the project's activities and work products and should be objective and verifiable. The system requirements are abstracted from the customer requirements and can be divided into two parts, the system requirements allocated to software and the system requirements allocated to hardware. Since the CMM focuses on the software process, discussion of customer requirements centres on those customer requirements to be implemented in software. The system requirements allocated to software, usually referred to as the "allocated requirements" in the CMM, are a primary input to the software development

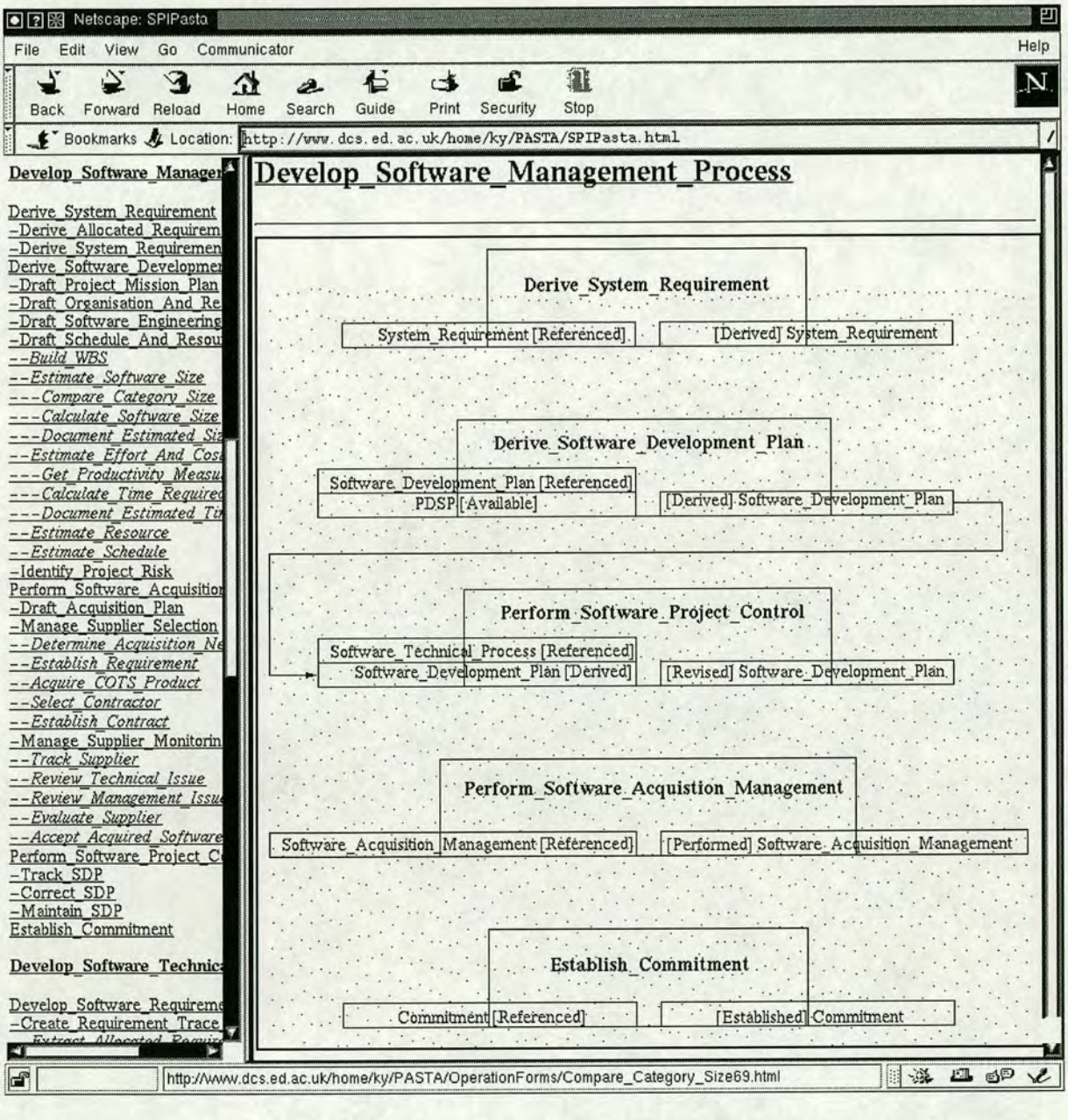

Figure 4.4: The P-State Diagram of the Software Management Process

plan. Software requirements analysis refines the allocated requirements and results in software requirements which are documented. The system requirements allocated to hardware are typically done by a system engineering group as part of the overall system design. SPI PASTA does not focus on this topic for real-time and control systems. Those who are interested in the system engineering can refer to the System Engineering CMM (SE-CMM) from the SEI[BKW95].

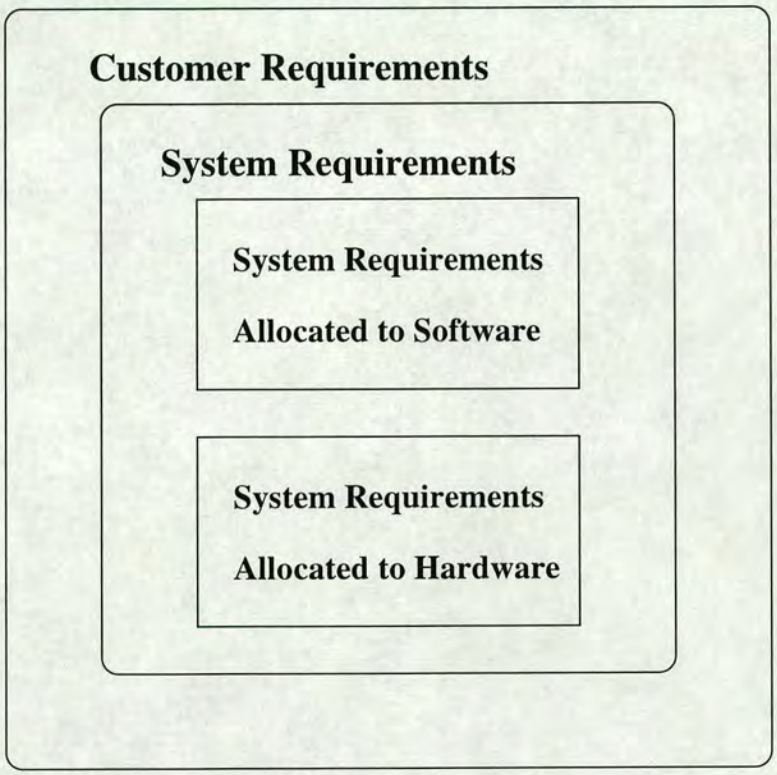

Figure 4.5: The Architecture of Requirements

Requirements Management is the first KPA in the CMM, since the allocated requirements form the basis for planning, performing and tracking the software project's plans and activities throughout the software life cycle. All plans and software work products in the CMM activities must be consistent with the requirements. Researchers[DL95, KS95b] believe that the major causes of software projects' failure are poor requirements and change management. Therefore, the CMM focuses on requirements management at the beginning of software process improvement.

4.1.1.2 Processes in Requirements Management

Figure 4.6 shows the P-state tree of Requirements Management. This P-state consists of two operations: Derive Allocated Requirement and Derive System

Requirement To Hardware.

Main Roles: The main roles participating in the operations are project managers, software product managers, system engineers and customers who will derive system requirements allocated to software and hardware.

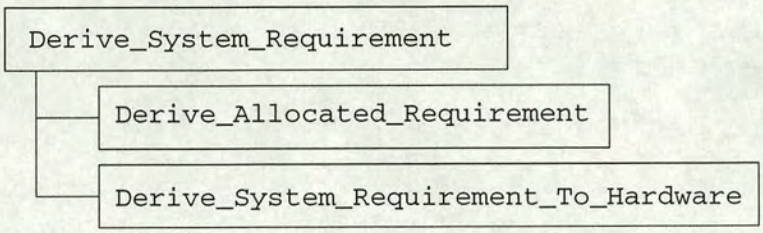

Figure 4.6: The P-State Tree of Requirements Management

Artifact List: The artifact list in the KPA is System Requirement which consists of two sub-artifacts: Allocated Requirements and System Requirements Allocated to Hardware. The artifacts in this KPA are shown in Figure 4.7.

Information Artifacts: No information artifacts in Requirements Management.

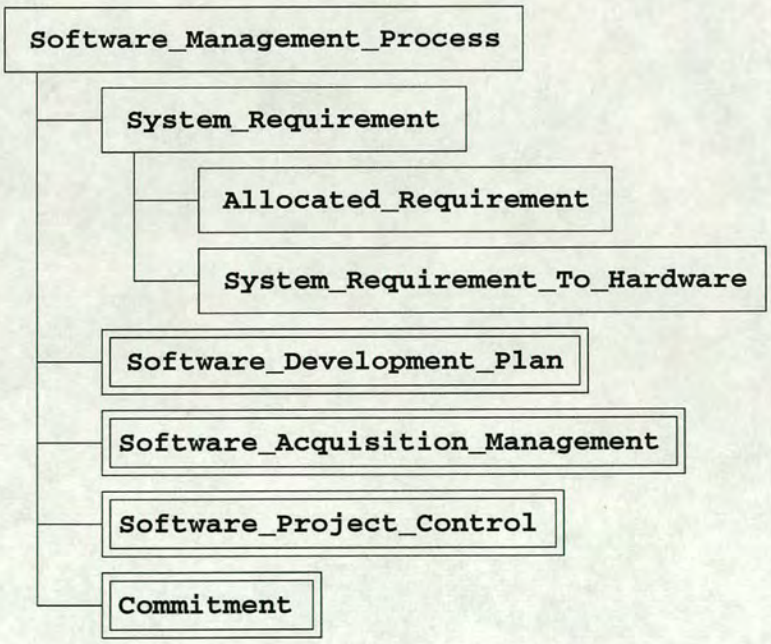

Figure 4.7: The Artifact Tree of Requirements Management

Entrance Condition: state-of(System_Requirement) = Referenced

Deriving system requirements is at beginning of the software project. Software organisations should have their own methodology to derive customer requirements. Once the customer's statements are collected, the system requirements should be derived and reviewed.

Activities: Two operations are presented in the P-state diagram of Requirement Management as shown in Figure 4.8.

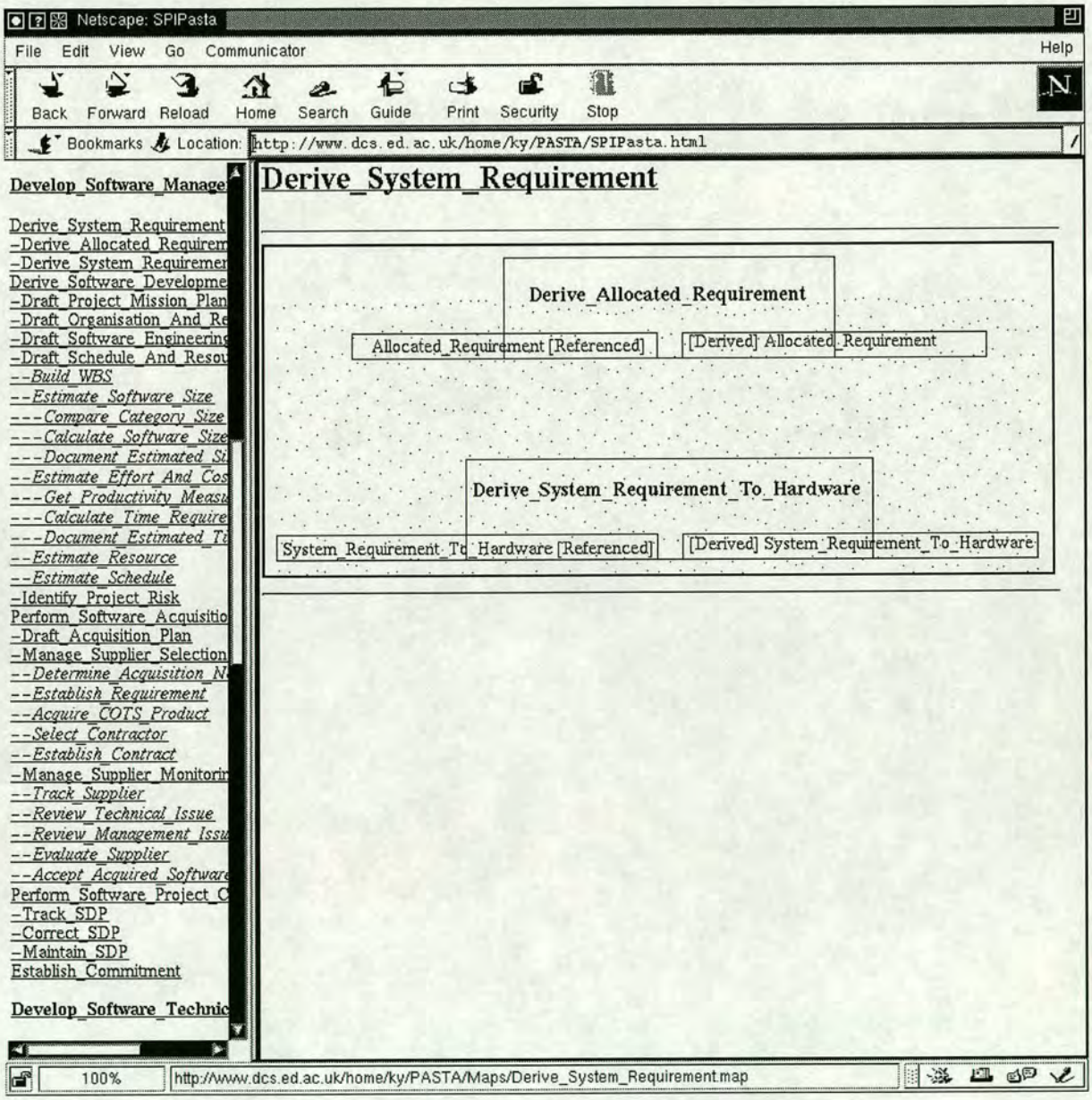

Figure 4.8: The P-State Diagram of Requirement Management

The main activities performed in the Requirements Management key process

area are to document the allocated requirements for the software project and to document changes to the allocated requirements throughout the life cycle. The CMM recognises that change is an integral part of software activity, which software organisations usually ignore it. The concept of freezing the specifications is almost impossible to apply to the software development. Software organisations must carefully assess the impact of change and make a good decision for the software development.

Currently, there are some commercial products to manage requirements. In addition to storing all requirements, these tools also provide a powerful ability to track changes during the development life cycle. The organisation can choose which tools they need to derive system requirements, but they will need to link it into SPI PASTA.

Exit Condition: state-of(System_Requirement) = Derived

After completing the P-states, managers and customers must check whether the exit condition has been reached. This means that the system requirements should be documented and controlled before they are incorporated into the software project.

4.1.2 Software Project Planning

> *The purpose of Software Project Planning is to establish reasonable plans for building the software product and for managing the software project[Pau97].*

After making an agreement with the customer on the requirements for the software project, the project's software development plan should be built and software risks should be analysed. This includes steps to estimate the size of the software work products and the sources needed, negotiate commitments, produce a schedule and identify and assess software risks. A good software development plan is crucial for the success of a software project. However, the quality of a software development plan generally depends on the quality of the size estimate. How to appropriately estimate the size of the software work products is becoming the most essential task in the Software Project Planning key process area.

To build the software development plan, two of the most critical resources are development staff and time[NAS90]. The project manager is concerned with how much time will be required to complete the project and what staffing level will be necessary over the development cycle. However, the degree to which you can accurately and precisely plan a job depends on what you know about

it. At the earliest, or preproposal stage, you have only a general idea of the product requirements. To make an accurate estimate, you must start with a design specification. You then examine and estimate each part of the job. This estimate requires separate estimates for each software component, each major document, the test cases, installation planning, file conversion and user training.

Nevertheless, in his book, Pressman[Pre94] described:

> *Software cost and effort estimation will never be an exact science. Too many variable - human, technical, environmental, political - can affect the ultimate cost of software and effort applied to develop it. However, software project estimation can be transformed from a black art to a series of systematic steps that provide estimates with acceptable risk.*

For years, researchers tried to find an effective method to tackle the estimation process problem. To date, the estimation process can be classified as either model based or analogy model. Model based estimation usually uses statistical analysis to build a model for estimation. Boehm[Boe81] introduced a hierarchy of software estimation models, called COCOMO. Albrecht's function point[AG83] identified five basic functions that occur frequently in commercial software development. Putnam[PM92] used the Rayleigh curve to derive his software equation. However, Vigder and Kark[VK94] found that informal analogy was the most commonly used estimating method for software organisations. There were two major reasons given for organisations not using formal models. Firstly, there was a lack of confidence in the ability of a model to outperform an expert. Secondly, the historical data is not available. Software managers would like to use "rules-of-thumb" to estimate their new projects. Nonetheless, a majority of large software projects tend to run late or overrun their budget, or even to be cancelled.

Most models of software estimation view the estimation process as being a function computed from a set of cost drivers and in most of the advocated software estimation techniques, the primary cost driver is assumed to be the software requirements(as Figure 4.9).

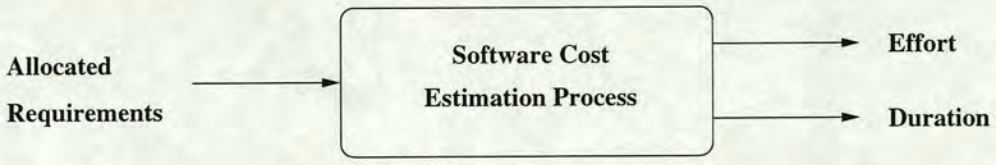

Figure 4.9: Classical view of software estimation process

Consequently, allocated requirements are viewed as constraints which must be satisfied. The initial requirements analysis, however, inevitably produces a

requirements definition that is incomplete and ambiguous and will be changed significantly throughout development. Furthermore, another problem is that it is hard to compare one program with another. As programs are increasing their size and complexity, they are almost impossible to compare in any orderly or consistent way. One way to address this comparison issue would be to break the new product into pieces. From this concept, the object-oriented methodology seems to be a good choice.

4.1.2.1 Size and Effort Measurement

For making a good estimation, partitioning is a crucial point. For making a good partition, allocated requirements are essential and necessary. In the CMM, a work breakdown structure (WBS) for the software project will be established at the beginning of the process. A partition is a set of capabilities that identifies the boundaries of a subproblem. Project managers coordinate the development of a set of partitions, planning resources for each individual partition, setting milestone dates, facilitating re-partitioning efforts and assuring the quality of each partition. Goldberg and Rubin[GR95] in their book suggested Breadth-First and Depth-First approaches. A depth-first approach partitions the allocated requirements and takes each partition all the way from analysis to implementation and testing. This approach is suitable for a project whose allocated requirements are very clear and where each subproject is nearly independent. Since this approach partitions at the beginning, each partition is totally independent. It might be difficult to be integrated, since project managers have to partition under an unclear condition. Moreover, the requirements seem to be frequently changed during the development period. A breadth-first approach starts with a high-level analysis. After analysis, the project manger creates partitions for subteams and the partitions are fully developed in parallel. A variant of the breadth-first approach (Figure 4.10) lets developers partition the problem, and do an analysis of each partition in parallel. When all analyses are completed, they are combined into a single set of analysis artifacts. Then the problem is repartitioned for development based on this unified analysis.

This approach is suitable for using object technology and use case driven development. At the outset of a project, the project manager does preliminary partitioning from allocated requirements. The first partitioning will be an approximation since the boundaries might need to be reassessed. Requirement analysts can do analysis in parallel by using the use case model. When all analyses are completed, the project manager collects all use case models and repartitions them

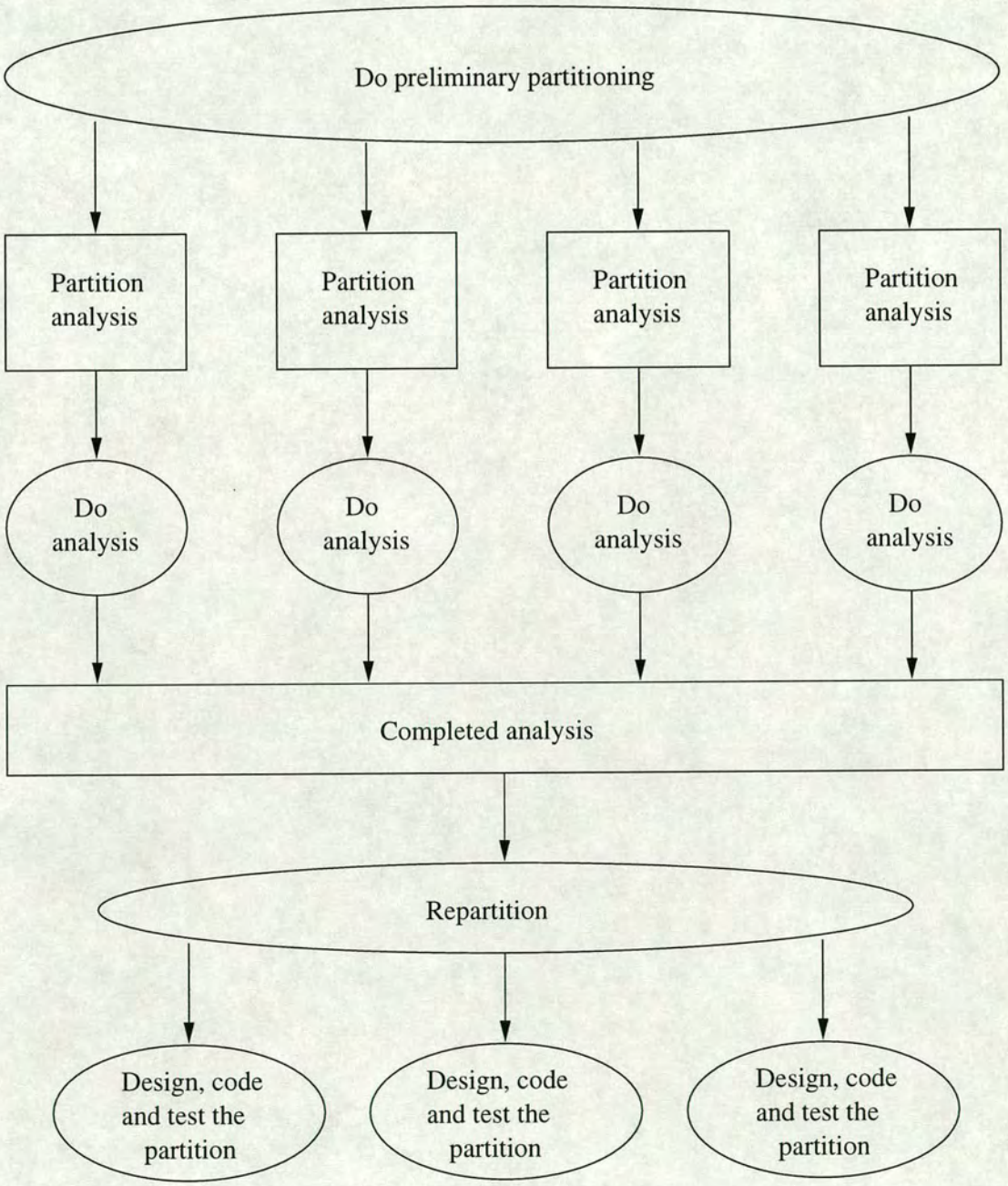

Figure 4.10: Alternative View of Breadth-First Partitioning [GR95]

Phase	Percent of Time Schedule	Percent of Effort
Requirement Analysis	12	6
Preliminary Design	8	8
Detailed Design	15	16
Implementation	30	40
System Testing	20	20
Acceptance Testing	15	10

Table 4.1: Distribution of Time Schedule and Effort Over Phase in NASA[NAS90]

to subteams.

In addition to partitioning, estimation of schedule for software product development and delivery is also a very challenging task. There are many factors that affect the schedule, and development progress is difficult to measure.

However, for software projects contracted with a government, clients and contractors decide the delivery date during negotiations. To avoid the project running late the initial estimate is a crucial step, nevertheless, it is the most uncertain since the allocated requirements are still unclear. As mentioned in Section 1.2.3, Microsoft roughly divides the life cycle into three phases. It is usual to take one fourth to one third of a project's schedule for planning. Before the allocated requirements are clearly partitioned, this could be an effective method for schedule estimation. By monitoring their developed software projects, the Software Engineering Laboratory in NASA[NAS90] collected the information of the expected schedule consumption and effort expenditure in each phase of the life cycle. These efforts are very helpful for estimating the contract software. In Table 4.1, the requirement analysis and preliminary design needs 20 percent of the time schedule. This means that the development team can fix 20 percent of the time schedule to complete its use case models. For those software organisations contracted with a government, it is relative easier to collect their historical data to establish their own distribution of time schedule.

After preliminary partition and estimation, making a software development plan starts with estimating the size of partitioned parts. By estimating the size of the product you plan to build, you are better able to judge the amount of work required to build it. Figure 4.11 shows the steps for making a schedule estimation.

For a schedule estimation, first and foremost, a framework for size measurement must be established. Line of code counts can easily be misinterpreted and misused if the organisation hasn't got a common policy. However, to help organisations obtain clear and consistent reports of software size, the Software Pro-

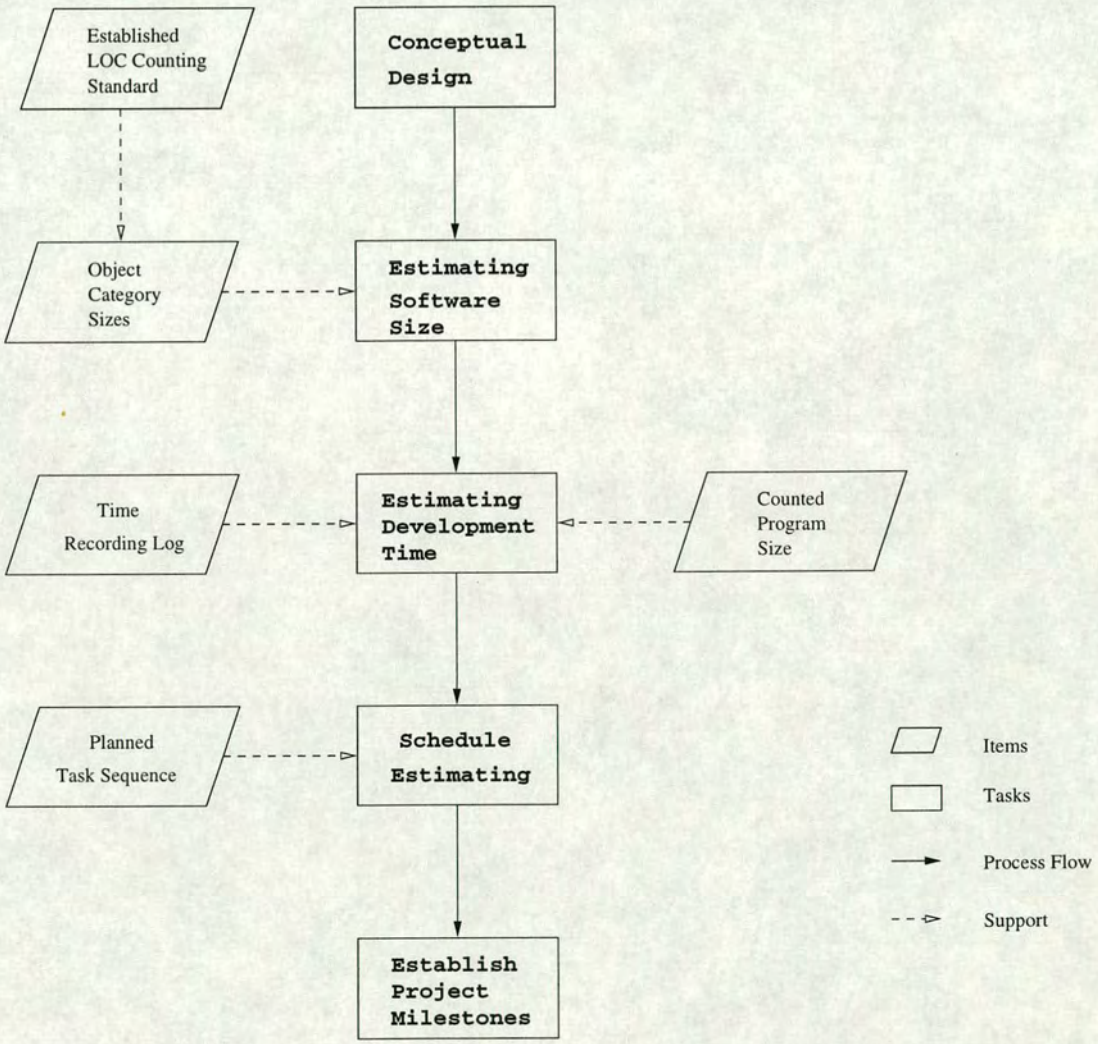

Figure 4.11: The Steps for Making a Schedule Estimate

cess Measurement Project at the SEI has developed a framework for describing software size measurements[Par92]. Once you have established the framework for counting lines of code, you can start estimating the software size. Researchers[Jon91, PM92, Hum95] suggest that using size estimating methods can help get a better quality size estimate.

The Personal Software Process (PSP), developed by Watts Humphrey[Hum95], is a defined and measured software process designed to be used by an individual software engineer or small developing team. PSP makes engineers aware of the processes they use to do their work and the performance of those processes. They learn to set personal goals for improvement, measure and analyse their work, and adjust their process to meet their goals. To date, data from the surveys provides convincing evidence of the benefits of the PSP. In their study, Hayes and Over[HO97] found the following result from using the PSP:

- Effort estimates improved by a factor of 1.75 (median improvement).
- Size estimates improved by a factor of 2.5 (median improvement).
- The tendency to underestimate size and effort was reduced. The number of overestimates and underestimates were more evenly balanced.
- Product quality, defects found in the product at unit test, improved 2.5 times (median improvement).
- Process quality, the percentage of defects found before compile, increased by 50% (median improvement).

In addition, the survey conducted by Ferguson *et al.*[FHK+97] also supported this conclusion.

A critical factor in the PSP is size and effort estimating. The PSP uses the Proxy-Based Estimating (PROBE) method for size and effort estimating. Instead of directly using LOC as the size measure, Humphrey uses a proxy to judge product size. The properties of the object-oriented methodology are good examples for a proxy. To use objects as proxies, firstly, organise your historical object data into categories and size ranges. After grouping these objects into functional categories, you can make estimates by deciding which functional category of object you are considering, then judge how many methods it is likely to contain, and finally determine where it falls into the size range.

Once the object category has been established, as in Figure 4.12, the next step will be building a work breakdown structure for the software project. The

work breakdown structure (WBS) divides the overall software project into work packages that represent singular work units that are assignable and for which accountability can be expected.

After you have subdivided the product into parts, you check to see if you have historical data on them. If a part does not resemble any element in the category, you have to reexamine it to see if you have refined it to the proper level. If an object is at the right level and does not belong to any of the existing categories, then you estimate its size as the first of a new category. This is an iterative process and you must finish it completely.

You now have the conceptual design, having named each object and having determined its category. Next you need to determine new object type and size. For each new object, you judge how its size compares with those in the database in its category. On the basis of this judgement, you estimate roughly what the new object's size will be.

After estimating a software size, you next have to estimate the time the work will take, judge the accuracy of this estimate and generate a schedule. You do this by relating the time you spent on prior projects to the estimated sizes of the programs you produced.

From the PSP's description(Figure 4.13), there are three choices for estimating development time. Firstly, if you do not have at least three historical data points, you have to calculate historical productivity in LOC per hour. This means that you divide the total LOC by the total hours to get your average productivity for the new project. Then, with Choice C, you estimate the time for the new program by dividing your estimated new program size by your productivity rate to get your new estimated time.

Secondly, if you just have data on actual development hours and object LOC for at least three projects and that the actual object LOC and actual development hours correlate with an $r^2 \geq 0.5$. Then, you do the regression calculation for total actual LOC and actual hours. With Choice B, you will use the regression method to calculate the estimated development time for the new program.

Thirdly, you have got at least three projects where the object LOC and actual development time correlate with an $r^2 \geq 0.5$. Then, with Choice A, you will use the regression parameters to calculate the estimated development time for the new program.

Finally, you have to make a good schedule plan.

A good size and effort estimation depends on good historical data collected from prior projects. Before you collect this useful data, you could make a good

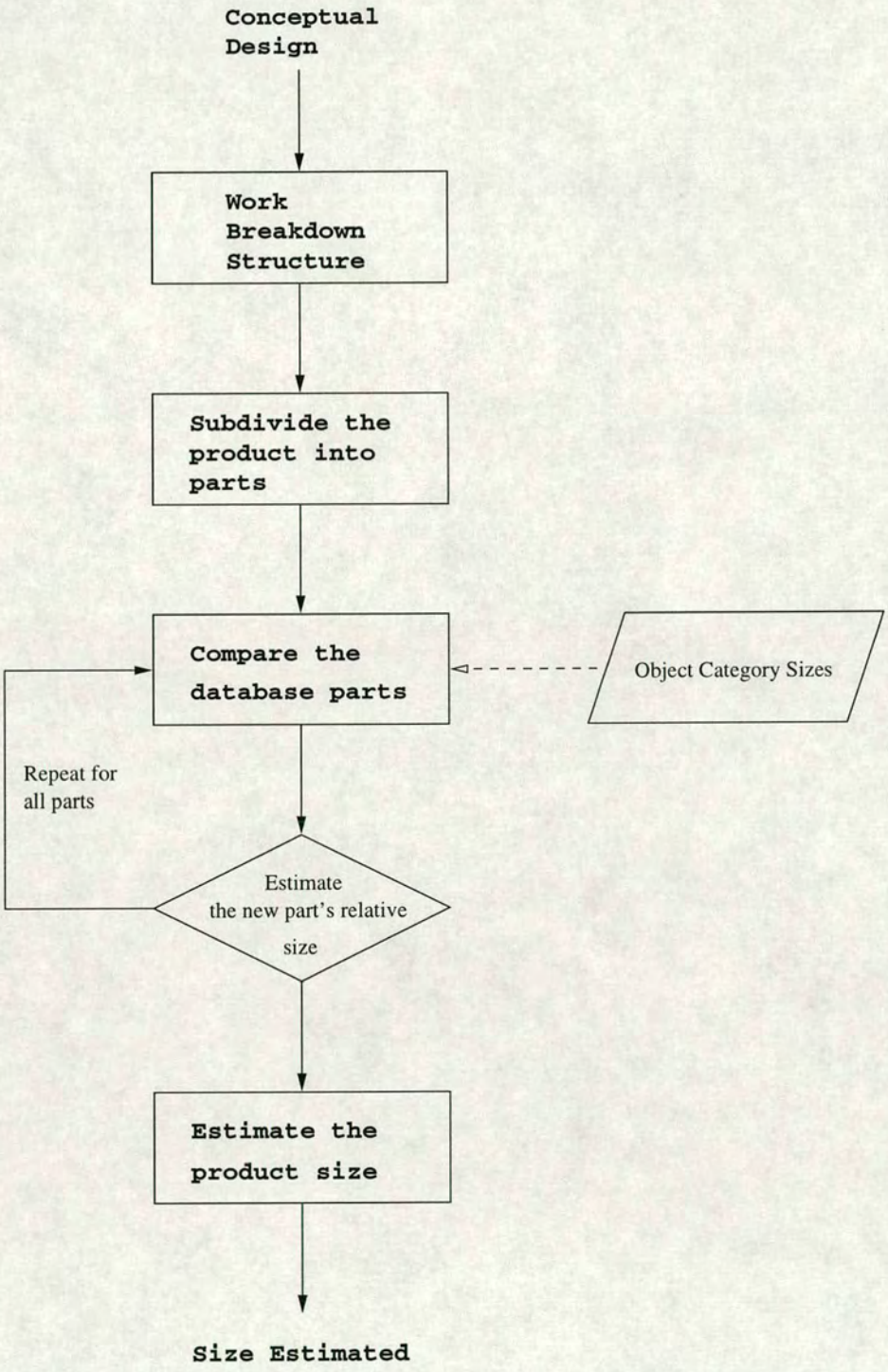

Figure 4.12: The flowchart for estimating software size

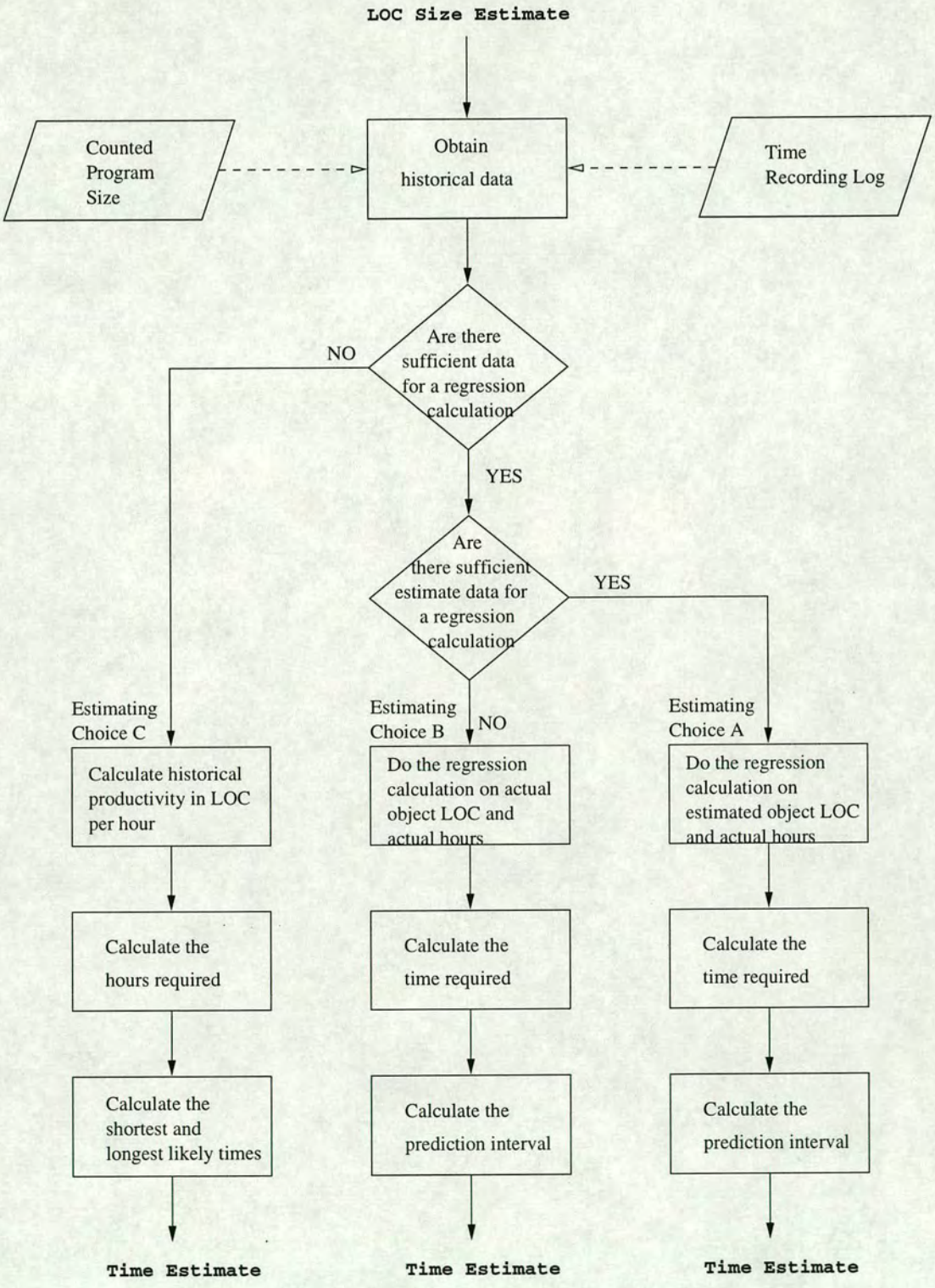

Figure 4.13: The flowchart for estimating development time

Project Type	Environment Type	Effort Multiplier
Old	Old	1.0
Old	New	1.4
New	Old	1.4
New	New	2.3

Table 4.2: Complexity Guideline[NAS90]

Teams Years of Application Experience	Effort Multiplier
10	0.5
8	0.6
6	0.8
4	1.0
2	1.4
1	2.6

Table 4.3: Development Team Experience Guideline[NAS90]

guess. NASA's SEL suggests that the estimates should be adjusted before the uncertainty proportion is applied. Some factors, such as project type, development environment and development team experience, can affect the estimates. Table 4.2 presents the recommended percentage adjustment to the effort estimate due to the complexity of the problem.[1] Table 4.3 presents an adjustment to the effort estimate for the effect of different team experience levels.[2]

4.1.2.2 Processes in Software Process Planning

Figure 4.14 shows the P-state tree for Software Process Planning. This P-state consists of five operations: Draft Project Mission Plan, Draft Organisation And Responsibility Plan, Draft Software Engineering Activity Plan, Draft Schedule And Resource Plan and Identify Project Risk.

Main Roles: The main roles participating in the operations are project managers, software product managers, senior managers, SEPG, system engineers, testing staff and quality assurance staff who will derive the software development

[1] The project type and environment type is old when the organisation has more than 2 years experience with it.

[2] Average of team member's years of application experience weighted by member's participation on the team. Application experience is defined as prior work on similar applications and member's participation is defined as time spent working on the project as a proportion of total project effort.

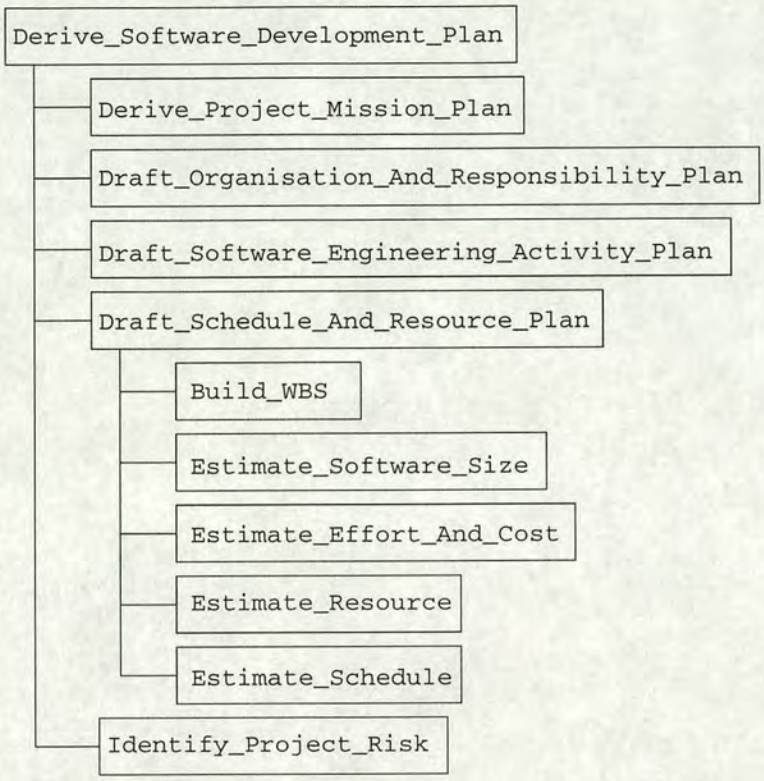

Figure 4.14: The P-State Tree for Software Process Planning

plan for the software project.

Artifact List: The artifact list in the KPA is the Software Development Plan which consists of four sub-artifacts: Project Mission Plan, Organisation And Responsibility Plan, Software Engineering Activity Plan and Schedule And Resource Plan. MIL-STD-498 provides a Software Development Plan description for both the software acquirer and supplier. Developers can adopt it as a framework complying with the P-state to complete the project's software development plan. The artifacts in this KPA are shown in Figure 4.15.

Information Artifacts: Two artifacts, Allocated Requirement and PDSP, provide the information to derive the software development plan. The allocated requirement is a primary input to the software development plan. In the meantime, deriving the software development plan should rely on the project's defined software process, if the organisation has established its own project's defined software process by following SPI PASTA.

Entrance Condition: state-of(Software_Development_Plan) = Referenced and state-of(PDSP) = Established

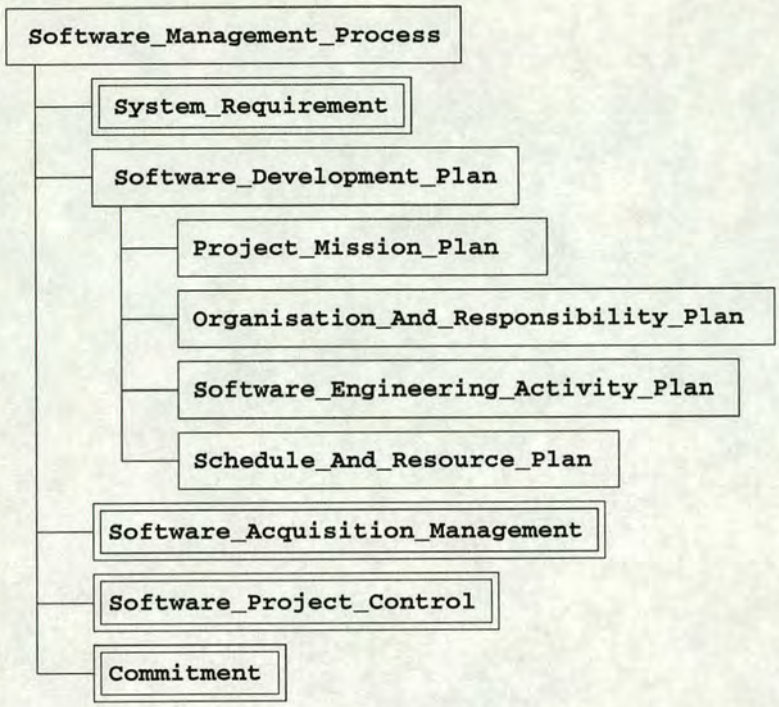

Figure 4.15: The Artifact Tree for Software Process Planning

Developers should ensure that the project's defined software process is appropriately reflected in the software development plan. Once the need to derive the software development plan has been identified, process roles may perform the operations.

Activities: Five operations are presented in the P-state diagram for Software Process Planning as shown in Figure 4.16.

Process roles firstly draft the project mission plan and the organisation and responsibility plan based on the allocated requirements. Moreover, the software life cycle must be selected from the project's defined software process. Process roles build the plan addressing sequencing and interdependencies of software engineering activities. Then process roles perform the activities according to Section 4.1.2.1, establishing the work breakdown structure to divide the overall software project into work units, estimating the size of each work unit, estimating the effort and cost for the software project, estimating the project's resources and establishing the project's schedule. The schedule and resource plan will provide a basis for the software project. Finally, risks associated with the software project should be identified. This operation could be linked to Risk Management which is described it on the Organisational Process, since it belongs to the Integrated

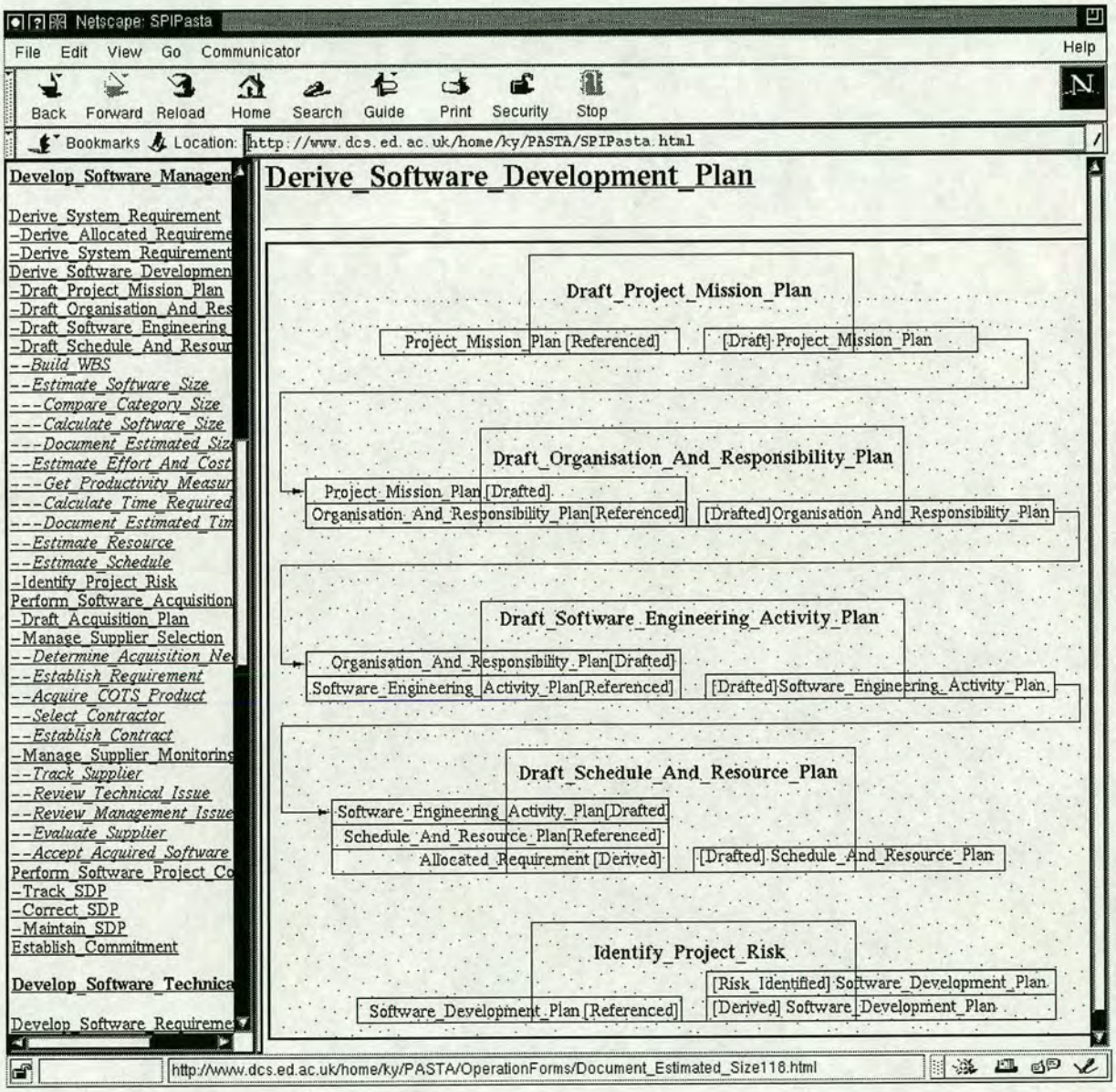

Figure 4.16: The P-State Diagram for Software Process Planning

Software Management key process area in the CMM Level 3. If Risk Management is not available or if the organisation is still on Level 2, process roles must perform this operation without risk mitigation strategies.

Exit Condition: state-of(Software_Development_Plan) = Derived

After completing the P-states, process roles must check whether the exit condition has been reached.

4.1.3 Software Project Control

> *The purpose of Software Project Control is to provide adequate visibility into progress of the software project so that appropriate corrective actions can be taken when the software project's performance deviates significantly from the plan[Pau97].*

4.1.3.1 Management of the Software Project

Without tracking, the software project might become out of control. The activities of Software Project Control involve tracking and reviewing the software performance and results against the plan and taking corrective action as necessary based on actual performance and results. Management of the software project should be based on the software development plan.

Why does the software project have to be tracked? The main reason is software estimation. As mentioned in Section 4.1.2, the development team use the allocated requirements to estimate the software size in order to build the software development plan. In the very beginning, the development team have only the allocated requirements that are elaborated from the customer requirements. This rough requirement is not good for estimating software size. When the development team starts to elaborate the software requirements, creating the use case diagrams and class diagrams, the software effort and cost estimation should be modified from time to time. Furthermore, any problem happening in the software development must be detected as soon as possible, otherwise the project cost will increase.

Currently, project management tools can conveniently support software project control. However, Steve McConnell[McC97] suggested creating a project intranet home page with links to general project information. With regard to project tracking, the home page might include the following:

- Percentage of schedule used (actual)
- Percentage of resources used (actual and planned)

- Percentage of defects found (actual and Planned)
- Graphs of actual vs. planned resources and defects
- Current task list
- Current defect list
- Top 10 risk list
- Anonymous feedback bulletin board

This information can be integrated as a part of PSEEs and will provide appropriate visibility into actual progress of the software project.

4.1.3.2 Processes in Software Project Control

Figure 4.17 shows the P-state tree for Software Project Control. This P-state consists of three operations: Track SDP, Correct SDP and Maintain SDP.

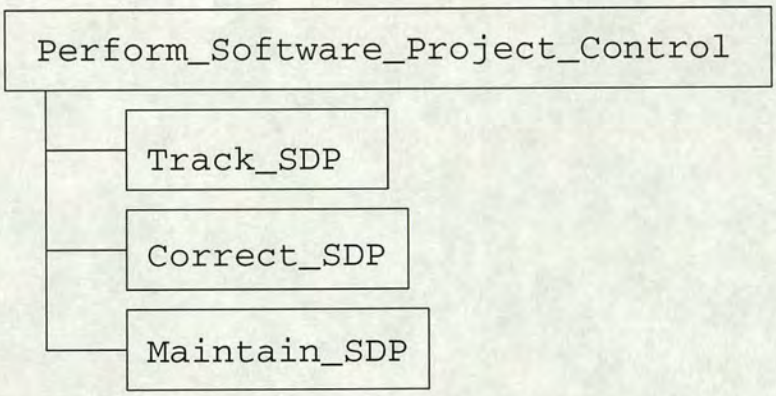

Figure 4.17: The P-State Tree for Software Project Control

Main Roles: The main roles participating in the operations are project managers, customers and the development group who will develop the software project. In principle, project managers should take charge of managing the software project.

Artifact List: The artifact list in the KPA is the Software Development Plan, since the plan is the basis for tracking software activities and taking corrective action.

Information Artifacts: The project manager tracks software project performance against the software development plan. All activities in management and technical processes will be tracked and the project manager will take corrective actions from time to time.

Entrance Condition: state-of(Software_Development_Plan) = Derived and state-of(Software_Technical_Process) = Referenced

When the development group starts to develop the software project according to the software development plan, software project performance and risks should be carefully tracked.

Activities: Three operations are presented in the P-state diagram for Software Project Control as shown in Figure 4.18.

The main activities performed in the Software Project Control key process area are to track software project performance and results in accordance with the software development plan. When the software requirements are elaborated from allocated requirements, the use case diagrams and class diagrams are created by software analysts. The content and functions of classes are increasingly clear. Therefore, it is better to re-estimate the software size, effort and costs. As a result, the computer resources, software engineering facilities and the project's schedule must make a correction to fit the actual software project development. These activities should be performed from time to time in order to make a best estimation for software project. Furthermore, the software development plan may also be revised to reflect accomplishments, progress, changes and corrective actions as appropriate.

Exit Condition: state-of(Software_Technical_Process) = Developed

The activities of the Software Project Control key process area must be performed until the software project is completed. This means the software project should be well controlled during the whole software life cycle.

4.1.4 Software Acquisition Management

> *The purpose of Software Acquisition Management is to effectively manage the acquisition of software from sources external to the software project[Pau97].*

This key process area applies to acquisition of software work products for which there exists a formal agreement between the supplier and the software

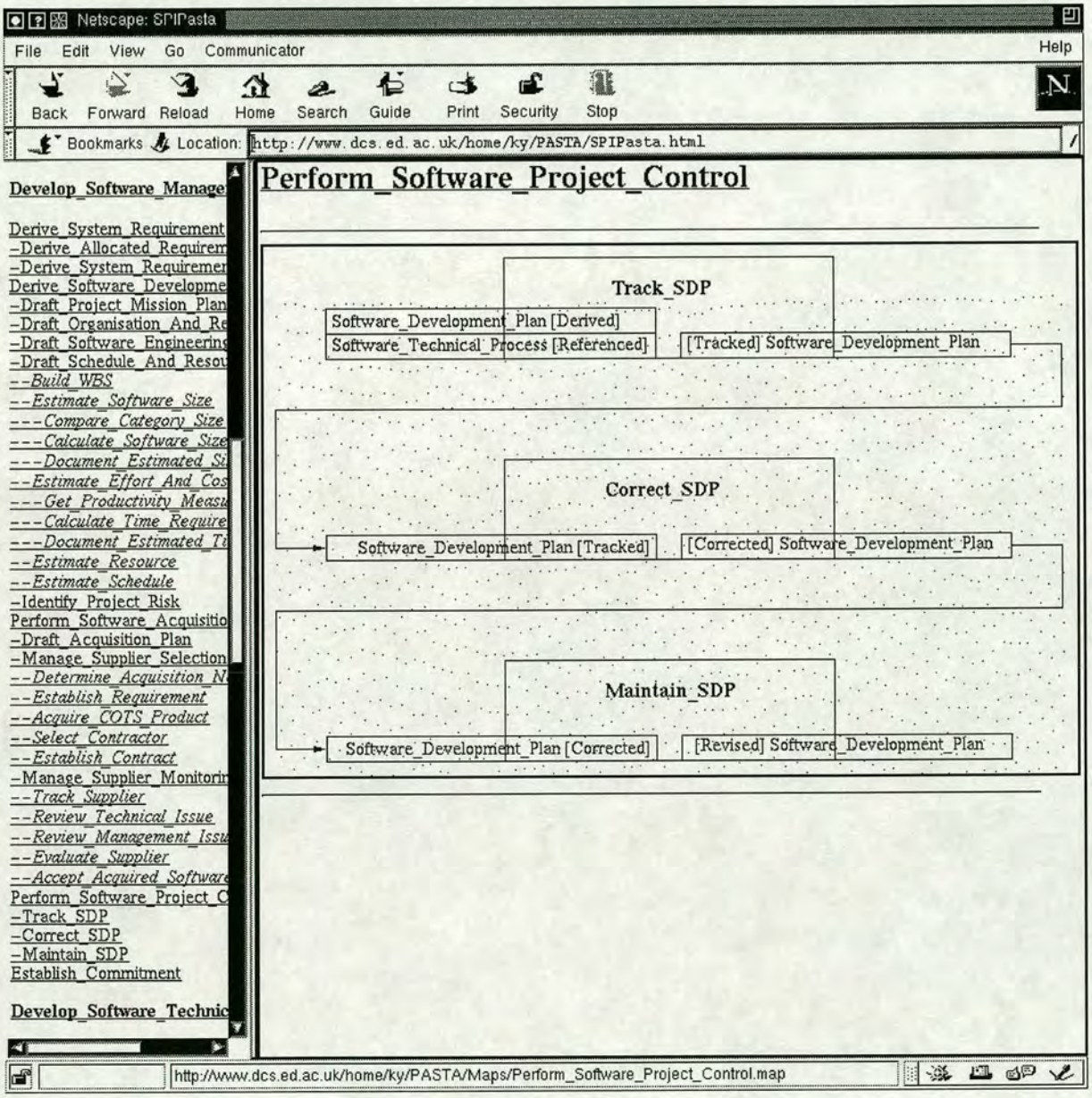

Figure 4.18: The P-State Diagram for Software Project Control

project. The acquired software is delivered to the software project from the supplier and becomes part of the software products delivered to the project's customer.

This key process area is not necessary for those organisations which are developing the software work products by themselves. However, software projects are increasingly sophisticated and software development costs are becoming a major trade-off for project managers. As a result, the software community has been motivated to find more effective and efficient ways to develop software in the last few years. The use of commercial off-the-shelf (COTS) products as elements of larger systems is becoming a consensus of reducing development costs. Therefore, this key process area can be applied not only for outsourcing software projects but also for acquiring COTS products.

4.1.4.1 Component-Based Software Development

The software community has tried to imitate the concept of mass production for a long time. Currently, the concept of interchangeable parts, similar to Integrated Circuit in the microelectronics industry, is emerging in the software industry.

In the last few years, the research of component-based software development has become one of the most important topics in the area of software engineering[Ber97]. This research focuses on building large software systems by integrating previously existing software components. Developing component-based systems is becoming feasible due to the following:

- The rapid evolution of the Internet/Intranet is making distributed projects increasingly viable.
- The middleware technology which manages communication and data exchange between objects, such as Common Object Request Broker Architecture (CORBA) from the Object Management Group[OMG97] and Microsoft's Component Object Model (COM), is emerging and increasingly mature.

As a result, the software development might move to a large-scale manufacturing and engineering process. Software developers may assemble software products from purchased fine-grained software components. The COTS-based systems (CBS)[Car97], conducted by the SEI, are focused on improving the technologies and practices used for assembling previously existing components into large software systems. The CBS Initiative is developing component-based systems practices that effectively qualify and integrate COTS components into critical

systems within business constraints. The CBS approach relies on the existence of an inventory of existing software components, the emergence of component integration technologies such as CORBA and COM, and the development of organisational capabilities for CBS trade-off analysis and design. This initiative will result in an innovative software development approach. The roles will be changed from being a developer and producer of systems to being a consumer and integrator instead. Figure 4.19 shows a new life cycle for the CBS.

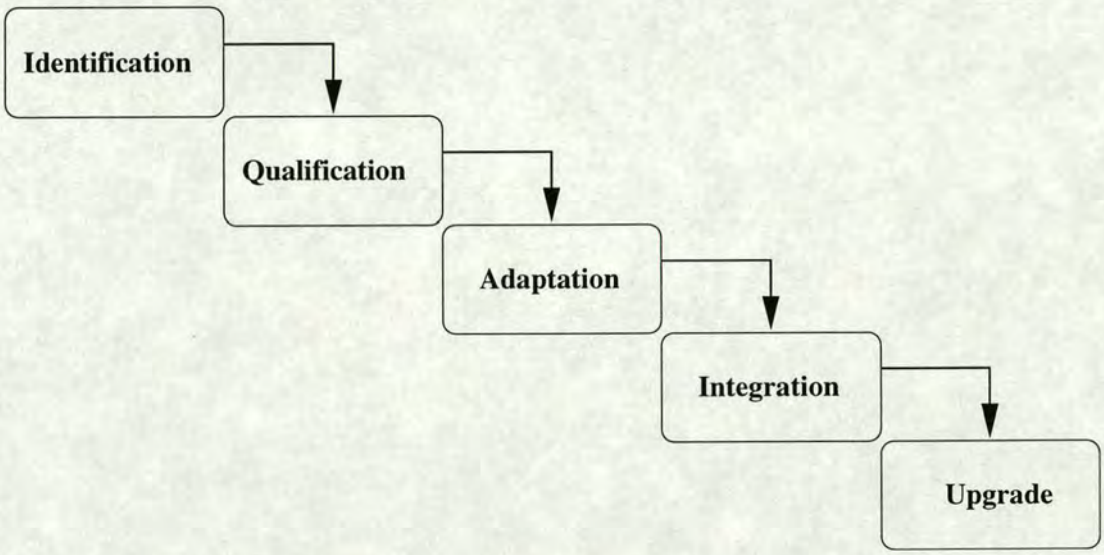

Figure 4.19: The Life Cycle of COTS-Based Systems

The challenge of a CBS consumer is how to build sufficient flexibility into procurement and contract documents to allow a variety of creative solutions while at the same time constraining bidders to selecting appropriate components and strategies. Software Acquisition Management will be a key role for the success of the CBS.

4.1.4.2 Software Acquisition Management

As Figure 4.19 shows, the activities of the software acquisition management contain five steps. Firstly, software acquisition management plans which encompass the total software acquisition effort must be established. This planning should identify the process for software acquisition management, which involves such items as early budgetary action, schedule determination, acquisition strategy, risk identification and software requirement definition. With the acquisition plans, software managers have to make a crucial decision – make or buy. This decision should come from analysing the allocated requirements to identify software that will be acquired. Once the requirements for the acquired software are established,

software managers must enter the second step, qualifying the contractors. With selecting, contracting, tracking and reviewing contractors, project managers have to ensure the software work products appropriately adhere to the requirements. Then, project managers conduct acceptance reviews and test and integrate all software components to make a complete software system. Finally, if necessary, they make an evolution plan.

However, as outsourcing, especially international outsourcing, becomes more common, software acquisition management is increasingly complicated. In order to minimise or eliminate the risk that software contracts will end up in dispute or in court, Jones[Jon96] gave the software community some recommendations as follows:

1. The sizes of software contract deliverables must be determined during negotiations.
2. Cost and schedule estimation must be formal and complete.
3. Creeping user requirements must be dealt with in the contract in a way that is satisfactory to both parties.
4. Some form of independent assessment should be included.
5. Anticipated quality levels should be included in the contract.
6. Effective software quality control steps must be utilised by the vendor.

4.1.4.3 Processes in Software Acquisition Management

Figure 4.20 shows the P-state tree for Software Acquisition Management. This P-state consists of three operations: Draft Acquisition Plan, Manage Supplier Selection and Manage Supplier Monitoring.

Main Roles: The main roles participating in the operations are project managers, software product managers, contract management staff, requirement analysts and software suppliers who will make the "make-or-buy" decision.

Artifact List: The artifact list in the KPA is Software Acquisition Management which consists of three sub-artifacts: the Acquisition plan, Supplier Selection and Supplier Monitoring. The artifacts in this KPA are shown in Figure 4.7.

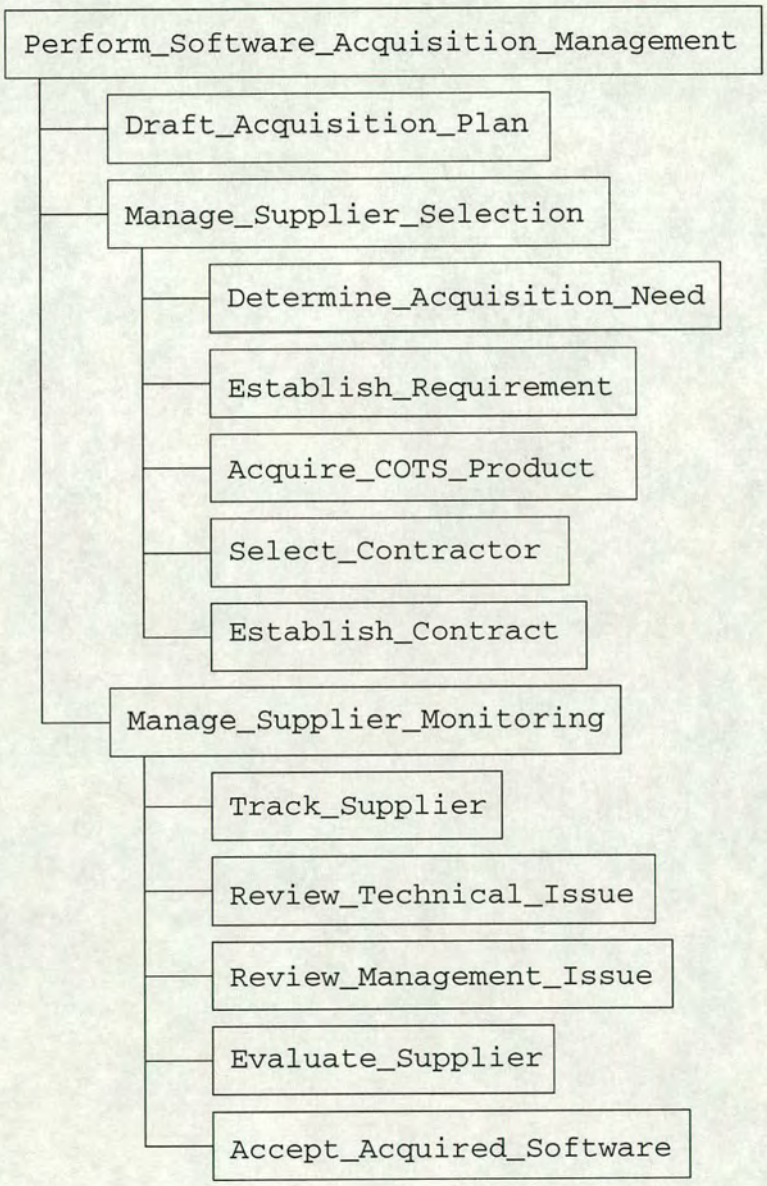

Figure 4.20: The P-State Tree for Software Acquisition Management

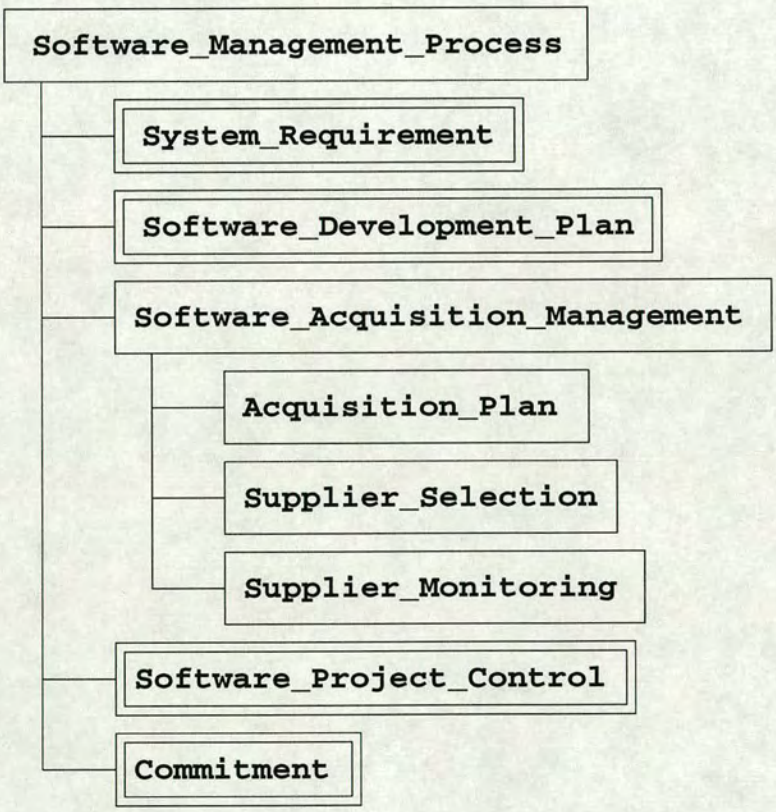

Figure 4.21: The Artifact Tree of Software Acquisition Management

Information Artifacts: The artifact, PDSP, as information artifacts provides a guideline to perform the software acquisition management.

Entrance Condition: state-of(Software_Acquisition_Management) = Referenced

The software analysts and managers according to the allocated requirements make a "make-or-buy" decision. The decision includes what kind of software work products will be acquired, if the project manager decides to buy software work products from external suppliers.

Activities: Three operations are presented in the P-state diagram for Software Acquisition Management as shown in Figure 4.22.

Firstly, managers should draft the software acquisition plan according to SPI PASTA. The plan with SPI PASTA may be a basis for performing the software acquisition.

Then the activities will be split into two parts, selecting the supplier and monitoring the supplier. In the supplier selection, when the acquisition decision has been made, the managers have to select the acquisition option. In accordance with the characteristics of the software project, the managers should decide to purchase COTS software products, obtain software from a contractor or other options. Then the organisation should establish the requirements for the acquired software. These requirements must comply with the allocated requirements referenced to the Requirements Management key process area. If the managers decide to purchase COTS software products, they need to perform the "Acquire_COTS_Product" operation and select the COTS products to satisfy the software project's needs. Otherwise, the managers should evaluate and select software contractors. Finally, the agreement must be established with the software contractors.

In the next step, the software supplier's activities should be monitored. The managers must review technical and management issues from time to time and periodically evaluate the performance of the software supplier in order to control the quality and progress of software products. Finally the organisation conducts acceptance reviews and tests to verify that the software products match the requirements.

Exit Condition: state-of(Software_Acquisition_Management) = Derived

After completing the P-states, managers and customers must check whether the exit condition has been reached. This means that the acquired software products are completely satisfied.

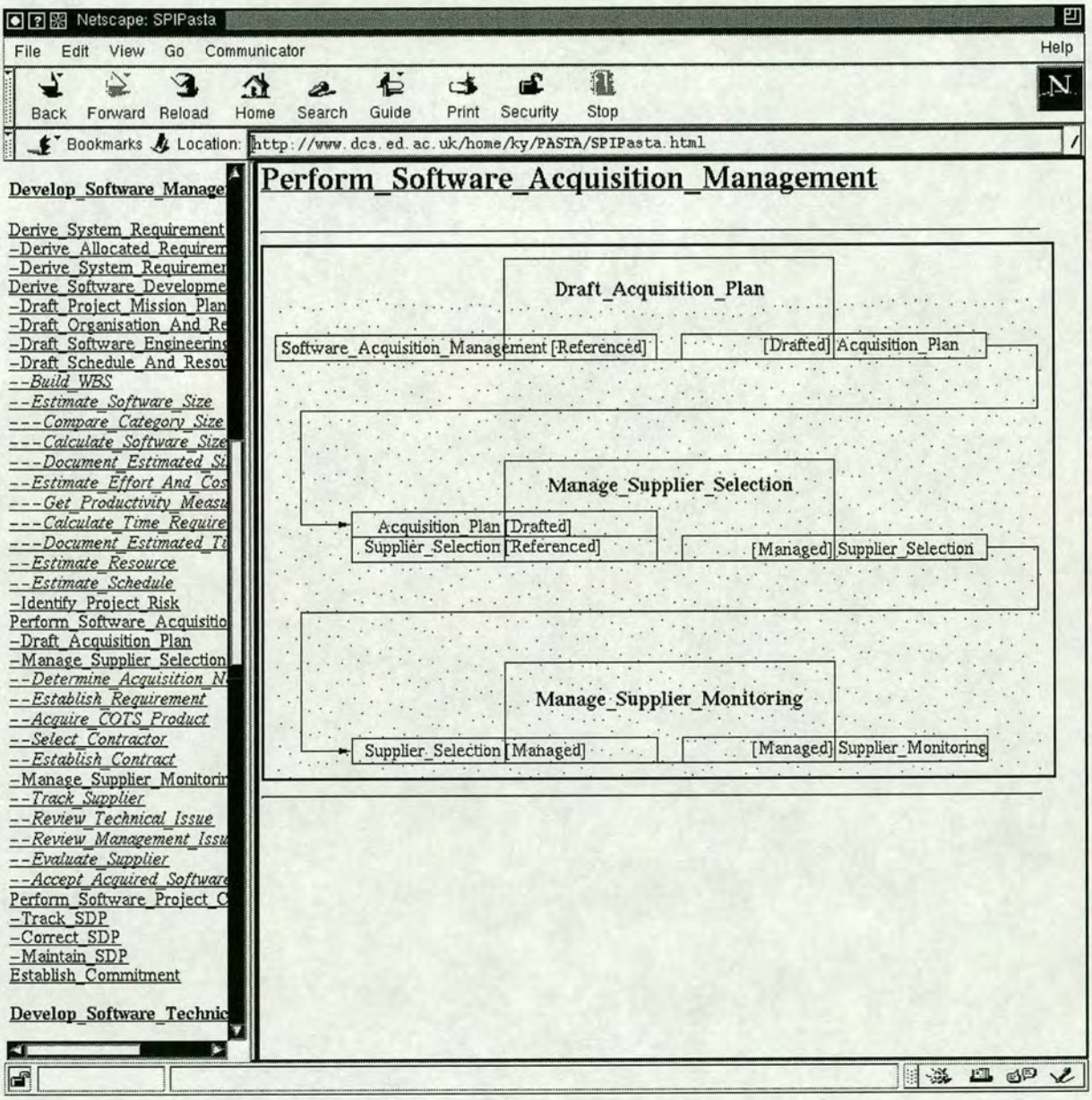

Figure 4.22: The P-State Diagram for Software Acquisition Management

4.2 The Software Support Process

After developing the software management process, policies for managing a software project and procedures to implement those policies should be established. In the meantime, the software support process should be developed in order to support the software management process. Software quality assurance will review the software project's activities and work products in order to provide managers with appropriate visibility to manage the software project throughout the software development life cycle. Configuration management will record and report the status of project configuration items which are produced from activities of the software management process and later on the operations throughout the software development life cycle.

Processes in the Software Support Process

Figure 4.23 shows the P-state tree of the Software Support Process. This P-state consists of two operations: Perform Configuration Management and Perform Software Quality Assurance.

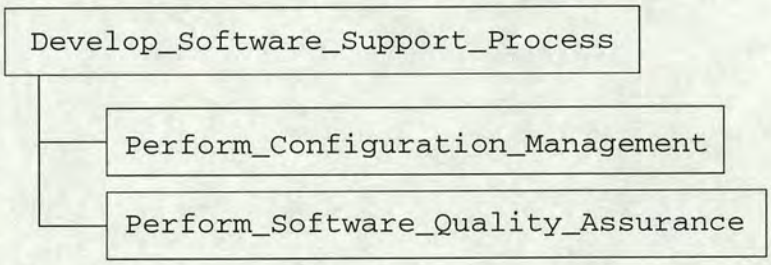

Figure 4.23: The P-State Tree of the Software Support Process

Main Roles: The main roles participating in the operations are project managers, software product managers, configuration management staff and quality assurance staff who will develop the software support process.

Artifact List: The artifact list in this part is Software Support Process which consists of two sub-artifacts: Configuration Management and Software Quality Assurance. The artifacts in the Software Support Process process are shown in Figure 4.24.

Information Artifacts: Users developing the software support process should rely on the project's defined software process.

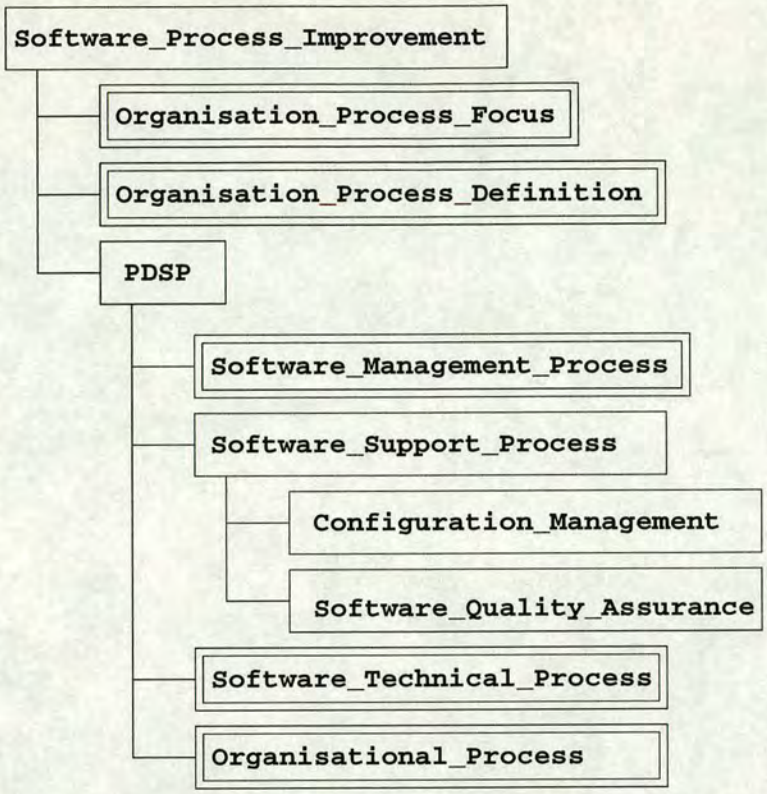

Figure 4.24: The Artifact Tree of the Software Support Process

Entrance Condition: state-of(Software_Support_Process) = Referenced

Developing the software support process should be at the beginning of the software project. As soon as the project starts, the need of developing the software support process should also be identified.

Activities: Two operations are presented in the P-state diagram of the Software Support Process as shown in Figure 4.25. There are no relationships between two operations. Both operations can be independently performed and should support the software management process to build a foundation for the software project. The activities can be performed under the project's defined software process. However, those organisations which have not reached level 3 or the project's defined software process is not available should directly perform the activities to support the software project.

Exit Condition: state-of(Software_Support_Process) = Developed

After completing the P-states, managers must check whether the exit condition has been reached. This means that the artifacts have been well performed and recorded.

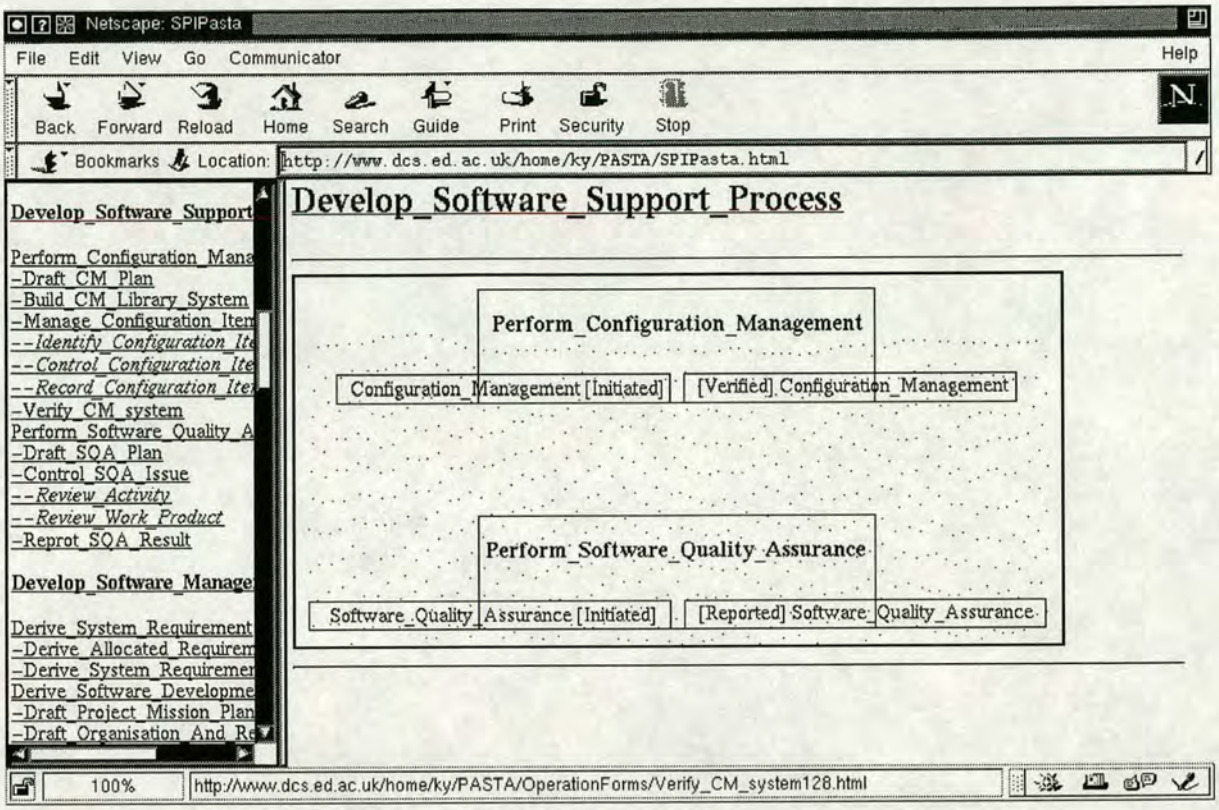

Figure 4.25: The P-State Diagram of the Software Support Process

4.2.1 Software Quality Assurance

The purpose of Software Quality Assurance (SQA) is to objectively review the software project's activities and work products for adherence to the applicable requirements, process descriptions, standards, and procedures[Pau97].

4.2.1.1 ISO 9000

ISO 9000 is a series of standards, published by the International Organisation for Standardisation (ISO), which define a framework of minimum requirements for the implementation of quality systems.

There are three different ISO 9000 certifications: ISO 9001, 9002 and 9003. For the software community, the most comprehensive of the standards is ISO 9001, Quality system - model for quality assurance in design, development, production, installation and servicing. It applies to industries involved in the design and development, manufacturing, installation and servicing of products or services. The core of ISO 9001 lies in Chapter 4, which consists of 20 quality elements. Since ISO 9001 is a high level standard and created to be used by all kinds of industries, there is significant room for interpretation in using ISO 9001 in the

software industry. Therefore, ISO 9000-3, Guidelines for the application of ISO 9001 to the development, supply and maintenance of software, is created in order to interpret ISO 9001.

To comply with ISO 9001, first and foremost, the organisation must have a comprehensive quality policy. ISO 9001 requires a real quality policy that identifies specific goals and methods. The quality policy has to be clearly described throughout the organisation. Once this organisation-wide policy is in place, a well-planned and managed quality system must be defined and documented. ISO 9000-3 characterises this quality system as an integrated process during the entire life cycle. This quality system based on ISO 9001 and 9000-3 provides an accurate description of the organisation and advice on the best practice adopted in order to consistently satisfy customer expectations. Figure 4.26 shows the architecture of the quality system.

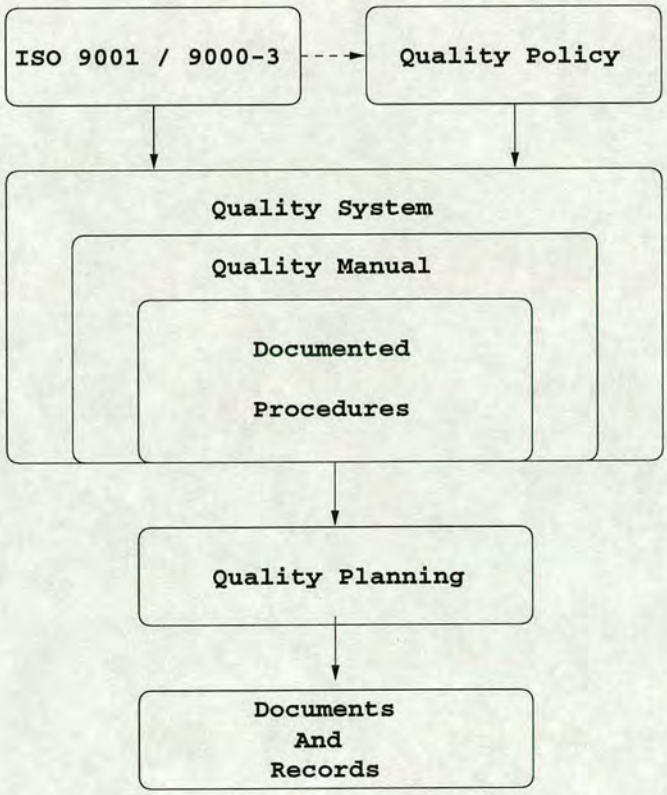

Figure 4.26: The Architecture of Software Quality Assurance

The concrete details of a quality system will be contained in a quality manual. The quality manual is an essential part of the quality system. Such a manual will require documented procedures. A procedure is a detailed step-by-step set of instructions describing how a particular quality assurance activity is to be carried

out. The procedures must be consistent with the organisation's quality policy and well documented in the quality manual.

Then, in accordance with the quality system, the quality plan must be established by the SQA staff. Each project should have its own quality plan which addresses appropriate procedures and the specific quality factors that are important to the customer and the developer. These include scheduling activities, assigning equipment and resources to the process of performing these activities, and providing the appropriate training to SQA staff. Once a quality plan is in place, it acts as a formal contract between the customer and the developer and as an informal commitment between the development team and SQA staff.

4.2.1.2 ISO 9001 and the CMM

Paulk[Pau94] compared ISO 9001 and the CMM, and pointed out " The biggest difference between these two documents is the emphasis of the CMM on continuous process improvement. ISO 9001 addresses the minimum criteria for an acceptable quality system. It should also be noted that the CMM focuses strictly on software, while ISO 9001 has a much broader scope: hardware, software, processed materials, and services." In spite of the difference, both documents are driven by similar concerns with quality and process management. There is still a strong correlation between the two documents.

People will argue whether a level 2 or 3 organisation is considered compliant with 9001 or which level an ISO 9001-compliant organisation should be at. Paulk suggested that, given a reasonable implementation of the software process, an organisation that obtains and retains ISO 9001 certification should be close to level 2. However, he also described how even a level 3 organisation would need to ensure that the delivery and installation process described in clause 4.15 of ISO 9001 is adequately addressed and should consider the use of an included software product, as described in clause 6.8 of ISO 9000-3. This would be comparatively trivial for a level 3 organisation[Pau94].

However, we believe that the Software Quality Assurance key process area provides a good opportunity to comply with ISO 9001. To deal with this key process area, the software organisation should use ISO 9001 as a standard to guide all activities performed in the key process area. As a result, when the organisation reaches level 2 of the CMM, it should be benefited to prepare for an ISO 9001 audit.

4.2.1.3 Processes in Software Quality Assurance

Figure 4.27 shows the P-state tree of Software Quality Assurance. This P-state consists of three operations: Draft SQA Plan, Control SQA Issue and Report SQA Result.

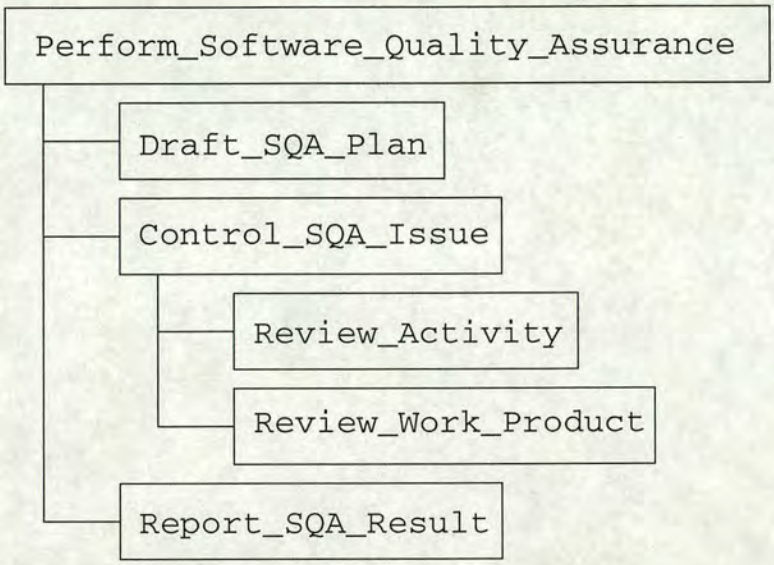

Figure 4.27: The P-State Tree of Software Quality Assurance

Main Roles: The main roles participating in the operations are senior managers who contribute their experiences of SQA, and software quality assurance staff who will perform the activities of SQA. Although it is not necessary to assign a different group to perform the SQA function, an independent SQA group is usually needed to ensure objectivity in the SQA reviews.

Artifact List: The artifact list in the KPA is Software Quality Assurance which consists of three sub-artifacts: SQA Plan, SQA Issue and SQA Result. The artifacts in this KPA are shown in Figure 4.28.

Information Artifacts: There are no information artifacts in Software Quality Assurance.

Entrance Condition: state-of(Software_Quality_Assurance) = Initiated

The activities of Software Quality Assurance should be initiated in the beginning of the software project. The SQA staff, as consultants, must participate in developing the project's defined software process and deriving the software development plan, and provide the consultation relying on the standards.

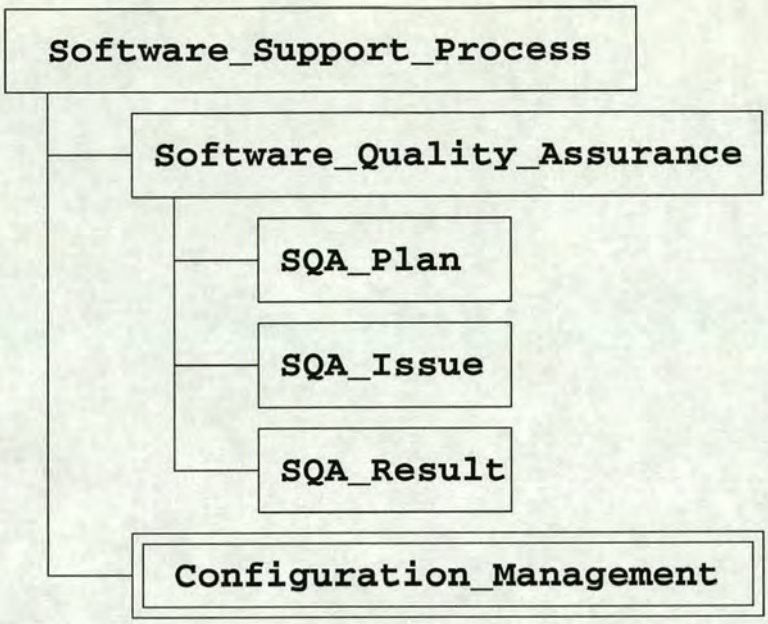

Figure 4.28: The Artifact Tree of Software Quality Assurance

Activities: Three operations are presented in the P-state diagram of Software Quality Assurance as shown in Figure 4.29.

The preliminary activity performed in the Software Quality Assurance key process area is drafting the SQA plan. The plan should comply with the organisation's quality policy and quality manual, and document the process for SQA including procedures performed by the SQA staff, and resources and responsibility assigned.

In the next step, the SQA staff may review designated software activities and software work products against the applicable requirements, process descriptions, standards, and procedures. The software activities depend on the project's defined software process and the software development plan. The software work products can be reviewed at selected milestones to check whether the customer requirements are satisfied.

The SQA staff then reports those deviations identified from reviewing software activities and software work products to the project manager and related software staff. The project manager and software staff must find a solution to resolve these deviations. Furthermore, the SQA staff should periodically review these identified deviations in order to validate them to comply with the standards.

Exit Condition: state-of(Software_Quality_Assurance) = Reported

After completing the P-states, managers and the SQA staff must check whether the exit condition has been reached. This means that the identified deviations

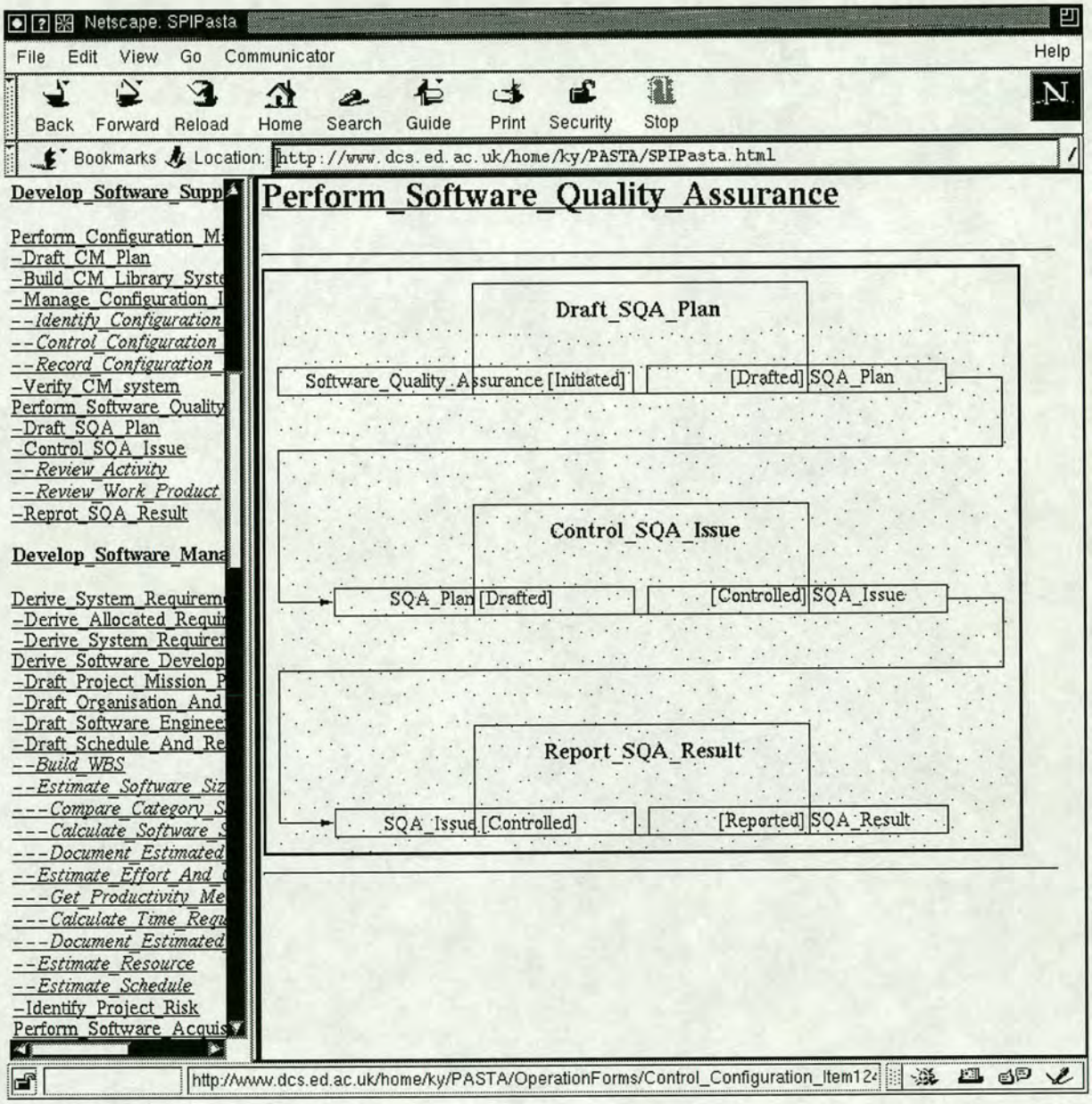

Figure 4.29: The P-State Diagram of Software Quality Assurance

have been reported to related software staff who will take the steps to resolve these deviations.

4.2.2 Software Configuration Management

> *The purpose of Software Configuration Management (SCM) is to establish and maintain the integrity of the work products of the software project throughout the software life cycle[Pau97].*

4.2.2.1 The Activities of Software Configuration Management

Software configuration management is a set of activities that have been developed to manage change throughout the software life cycle. In accordance with IEEE Standard 828-1990, Software Configuration Management Plans, the activities of software configuration management consist of four functions[IEE90]:

- **Identification:** identify, name, and describe the documented physical and functional characteristics of the code, specifications, design, and data elements to be controlled for the project.
- **Control:** request, evaluate, approve or disapprove, and implement changes.
- **Status accounting:** record and report the status of project configuration items (initial approved version, status of requested changes, implementation status of approved changes).
- **Audits and reviews:** determine to what extent the actual configuration item reflects the required physical and functional characteristics.

SCM should be performed by using a "configuration item". A configuration item is an entity designated for configuration management, which may consist of multiple related work products[Pau97]. At the beginning of software development, the developers should identify configuration items. Since SCM covers the entire software life cycle, configuration items could be any sort of work products. Pressman in his book suggested a list of configuration items as follows[Pre94]:

1. System Specification
2. Software Project Plan
3. - Software Requirements Specification
   - Executable or "paper" prototype
4. Preliminary User Manual

5. Design Specification
    - Data design description
    - Architecture design description
    - Module design descriptions
    - Interface design descriptions
    - Object descriptions
6. Source code listing
7.
    - Test Plan and Procedure
    - Test cases and recorded results
8. Operation and Installation Manuals
9. Executable program
    - Modules - executable code
    - Linked modules
10. Database description
    - Schema and file structure
    - Initial content
11. As-Built User Manual
12. Maintenance documents
    - Software problem reports
    - Maintenance requests
    - Engineering change orders
13. Standards and procedures for software engineering

Another essential function is to manage change, including version control and change control.

Version Control Version control manages different versions of configuration items created during the software life cycle by using automated tools. These tools can provide the facilities to reconstruct specific file states, objects and the entire application.

Change Control SCM adopts a baseline as a milestone for software development. A baseline represents the assignment of an identifier to a configuration item and its associated entities. A baseline that is delivered to the customer is called a "release"[Pau97]. Before a configuration item becomes a baseline, change may be made informally. However, once a baseline is built, change control will be necessary and compulsory. Change control is a set of procedures for the control of change. Any change of configuration items must follow the procedures in order to unify the version of configuration items.

4.2.2.2 Processes in Software Configuration Management

Figure 4.30 shows the P-state tree of Software Configuration Management. This P-state consists of four operations: Draft CM Plan, Build CM Library System, Manage Configuration Item and Verify CM System.

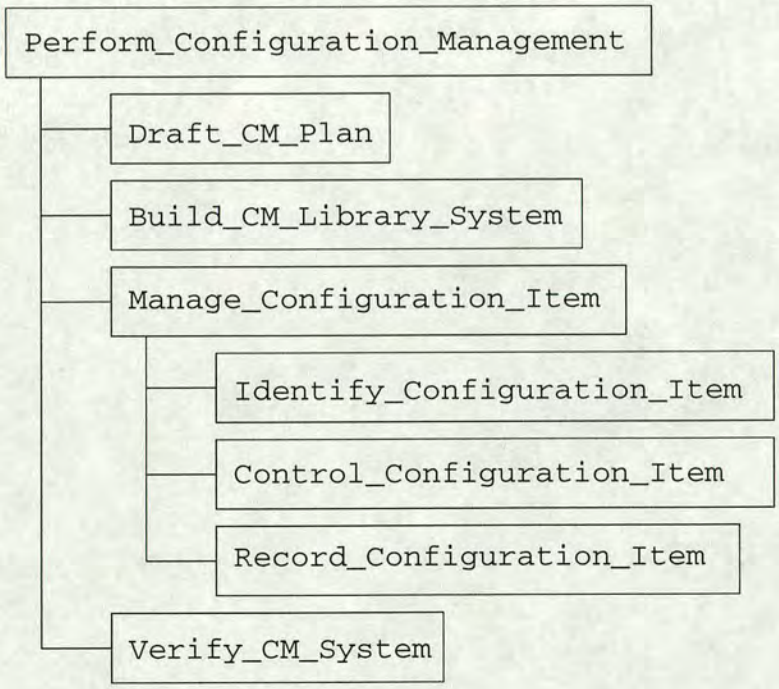

Figure 4.30: The P-State Tree of Software Configuration Management

Main Roles: The main roles participating in the operations are Configuration Management Staff who will perform activities of configuration management.

Artifact List: The artifact list in the KPA is Configuration Management which consists of three sub-artifacts: CM Plan, CM Library System and Configuration Item. The artifacts in this KPA are shown in Figure 4.31.

Information Artifacts: There are no information artifacts in Software Configuration Management.

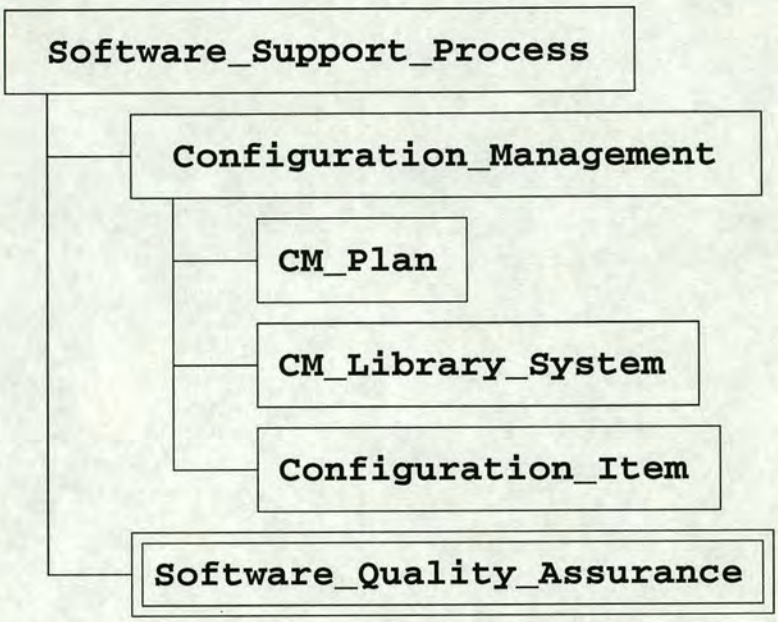

Figure 4.31: The Artifact Tree of Software Configuration Management

Entrance Condition: state-of(Configuration_Management) = Initiated

Since the activities of Software Configuration Management are covered throughout the software life cycle, the activities of Software Configuration Management should be initiated at the beginning of the software project.

Activities: Four operations are presented in the P-state diagram of Software Configuration Management as shown in Figure 4.32.

The preliminary activity performed in the Software Configuration Management key process area is drafting the SCM plan. The plan should document the process for SCM including activities performed by the SCM staff, resources and responsibility assigned and SCM-related activities performed by other staff in the organisation. In the next step, a software configuration library system must be established. This library system provides for the storage as well as the recording of changes of configuration items. Currently, commercial SCM tools provide facilities not only for the storage of configuration items but also for tracing relationships between versioned configuration items. The SCM staff can easily link those tools to the P-state of SCM to perform the activities.

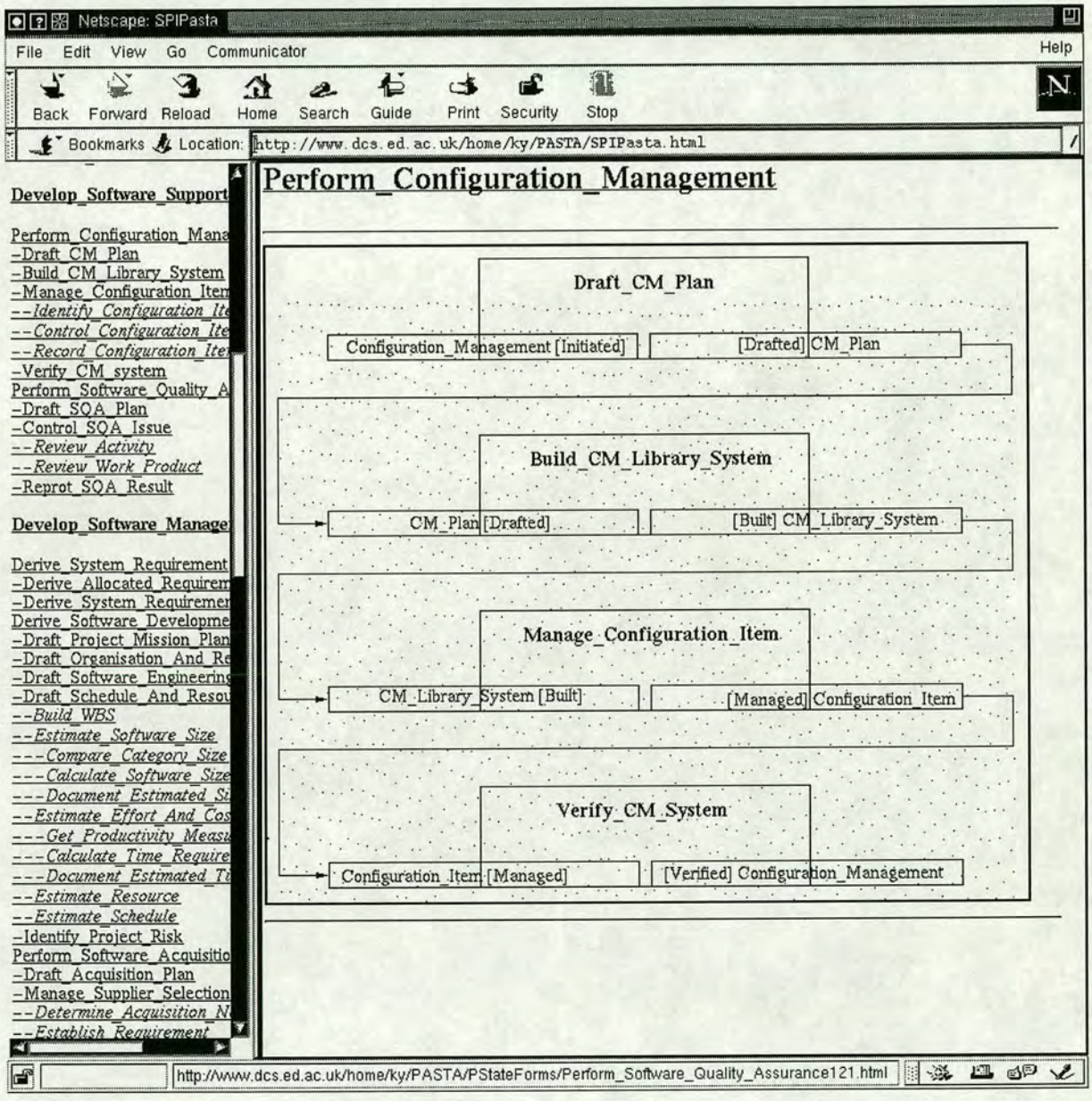

Figure 4.32: The P-State Diagram of Software Configuration Management

The SCM staff then perform the activities prescribed in the SCM plan, identifying configuration items, following the change control process to control changes of configuration items, and recording detailed information about configuration items. Finally, configuration audits should be appropriately performed.

Exit Condition: state-of(Configuration_Management) = Verified

After verifying completeness and correctness of the configuration items, the exit condition has been reached. However, the activities should be performed until the software project is completely developed. All configuration items should be effectively managed throughout the software life cycle.

4.3 Summary

This chapter discussed how to plan and how to control a software project based on key process areas of the CMM Level 2. We use the incremental and iterative strategies for software development and suggest Humphrey's Personal Software Process to plan the software project. The purpose of planning a software project is to organise work effort, to communicate how work will proceed toward meeting projects goals, and to form a basis to track project's progress. In addition, the relationship with contractors, if any, is emphasised on software acquisition management. Two support processes, software quality assurance and software configuration management, are also described to control the project's quality and to treat version and change control for the software project. Once these activities have been completed, the processes for software projects should be institutionalised. The experiences of software development can be adopted and repeated. An effective process will gradually become a disciplined paradigm to improve software development for software organisations.

Chapter 5

The Processes in the CMM Level 3

The key process areas at Level 3 address both project and organisational issues, as the organisation establishes an infrastructure that institutionalises effective software engineering and management processes across all projects[Pau97]. Processes established at Level 3 are used to help the software managers and technical staff perform more effectively. The organisation exploits effective software engineering practices when standardising its software processes. Once the activities at Level 3 have been completed, an organisation-wide training program should ensure that the staff and managers have the knowledge and skills required to fulfil their assigned roles, the software engineering staff should effectively coordinate and collaborate with other development teams, defects of software work products should be removed early and efficiently, and project's risk should be under control. This chapter consists of two parts, software technical processes and organisational processes. The former focus on the life-cycle of software development, the latter will support development teams to effectively and efficiently develop software work products.

5.1 The Software Technical Processes

> *The purpose of Software Product Engineering is to consistently perform a well-defined engineering process that integrates all the technical activities of the software project to produce correct, consistent software work products effectively and efficiently[Pau97].*

Software engineering includes both management and technical activities. The SEI separates them into Level 2 and Level 3 of the CMM and focuses on management processes before engineering processes. The fundamental reason is that in

the absence of management discipline, engineering process is sacrificed to schedule and cost pressures.

There are some software life cycle models in place, such as the Waterfall Model, the Spiral Model and so on. It is better that the organisations select a software life cycle model as part of the organisation's standard software process. In his book[Boo96], Booch suggested an iterative and incremental process for the software development life cycle. An iterative process is one that involves the successive refinement of a system's architecture, from which we apply the experience and results of each major release to the next iteration of analysis and design. The process is incremental in the sense that each pass through an analysis/design/implementation cycle leads us to gradually refine our strategic and tactical decisions, extend our scope from an initially skeletal architecture, and ultimately leads to the final, deliverable software product. In the meantime, DSDM[DSD97] also adopts iterative and incremental process as its life cycle model.

Yet what is the destination for software development? Every iteration and increment means that the product is refined again and again. Yourdon[You96] recommended that functionality, quality and schedule are the three most important elements of software development. He promoted a "good enough software" idea, meaning that an iterative and incremental process is the trade-off between functionality, defects and speed of delivery. These is no doubt that some software systems, such as nuclear reactor systems, will focus on zero-defect software. However, for most of the shrink-wrap software, time-to-market would be an essential point. In the corporate MIS application software industry, the schedule is also a pressure for the project manager. Booch's macro development process answers this problem. In the object-oriented process, we start with what we know, devise a skeletal object-oriented architecture, study the problem some more, improve upon our architecture, and so on, until we expand to a solution that satisfies our project's essential minimal characteristics. So, a deadline-based iterative and incremental process would be a practical one for software development. Booch's macro process is explicitly iterative and incremental and very close to the spiral model. The project manager improves the software product by the deadline and decides the final iteration and increment for releasing the products on time.

As Figure 5.1 shows, Booch's macro process is from conceptualisation to maintenance. This is quite similar to the traditional waterfall approach; however Booch insisted that the macro process is explicitly iterative and incremental, and it is closer to Boehm's spiral model.

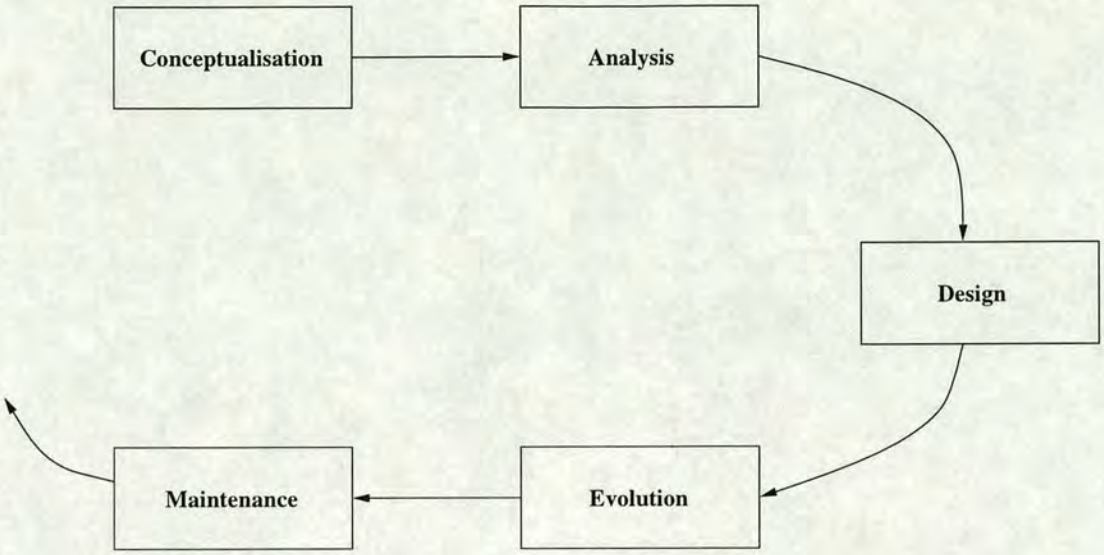

Figure 5.1: The Macro Development Process

In this thesis, we use MIL-STD-498 to comply with the CMM as the software engineering processes. MIL-STD-498 adopts the concept of "build" which is a version of software that meets a specified subset of the requirements that the completed software will meet, and the period of time during which such a version is developed. This concept is similar to Booch's macro process. Each build incorporates a specified subset of the planned capabilities of the software. The builds might be prototypes, versions offering partial functionality, or other partial or complete versions of the software[DOD94]. In addition to "build", MIL-STD-498 also suggests three basic program strategies: Grand Design, Incremental and Evolutionary strategies. The grand design strategy is similar to the waterfall life cycle, but will not be discussed in this thesis. The primary difference between incremental and evolutionary strategies is whether all requirements are defined first. The incremental strategy first defines the system requirements, then performs the rest of the development in a sequence of builds, whereas the evolutionary strategy does not define all requirements first, and the system requirements are partially defined, then refined in each succeeding build. Both strategies have their own advantages. As mentioned in Section 1.2.3, because of the "time-to-market" pressure for Microsoft's products, Microsoft adopts a "synch-and-stabilise" approach which is similar to the evolutionary strategy. Furthermore, if the software project must define its requirements for the acquirer before the project is started, the incremental strategy may be a good choice.

Figure 5.2 shows the software engineering process in MIL-STD-498. This is split into three phases: Planning, Development and Deployment. Those activities

correspond to Section 5 Detailed Requirements in MIL-STD-498.

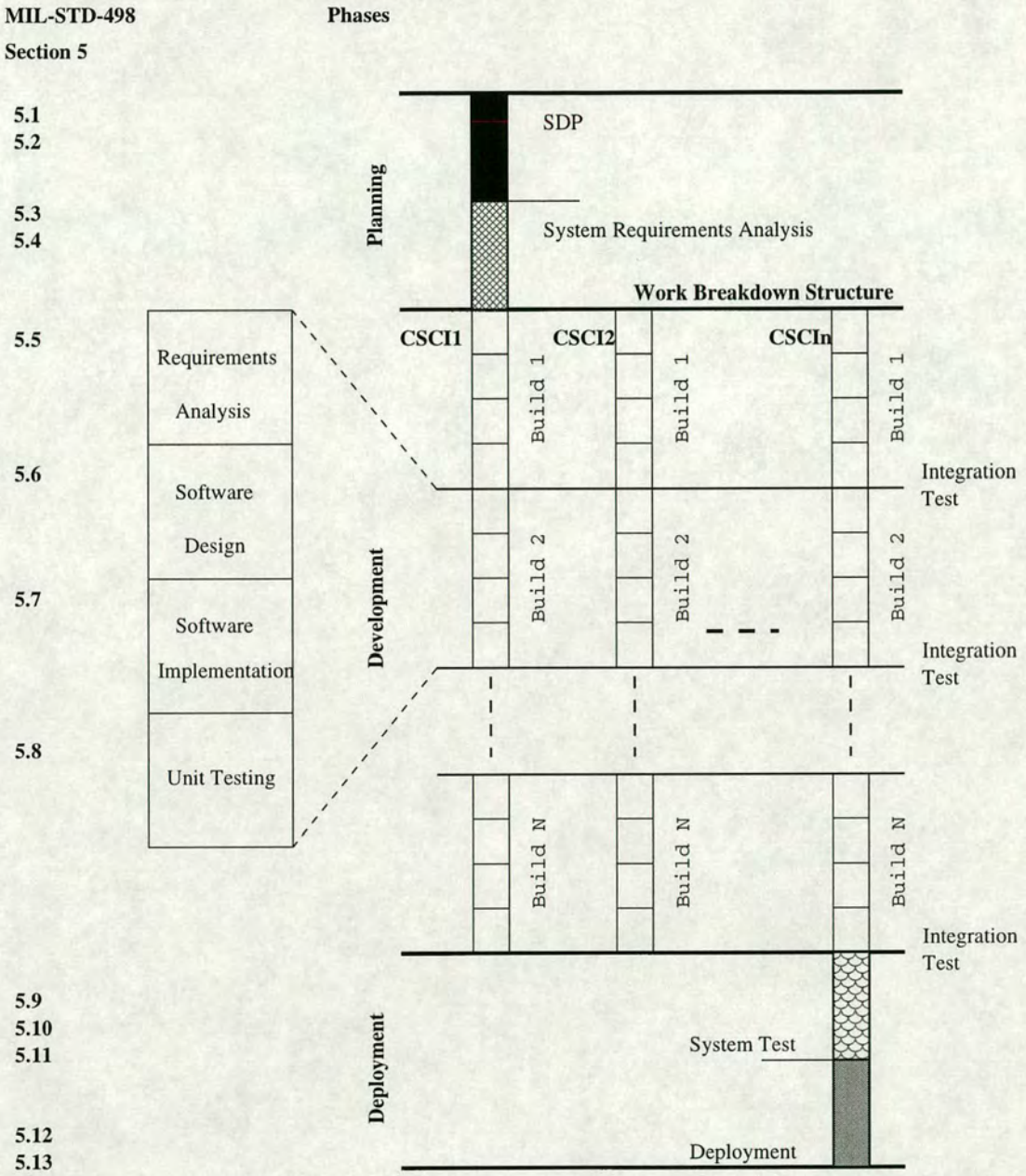

Figure 5.2: The Software Engineering Process in MIL-STD-498

Planning Phase

The first step of the planning phase is building the software development plan which we have mentioned in the management processes, and establishing a software development environment.

The next step is to define and record the requirements for the system. The system requirements consist of the allocated requirements and the system re-

quirements allocated to hardware. Those activities, mentioned in Section 4.1.1 Requirements Management, will comply with the SDP as a foundation of performing the technical processes.

Development Phase

The development phase is the core of the software life cycle and is described as the technical processes in the CMM. Before developers perform the technical processes, the CSCIs must be defined in order to parallel development. MIL-STD-498 also adopts the "build" concept which splits the development phase into several builds. Each build consists of four steps, from requirement analysis to unit testing. At the end of unit testing, all CSCIs should do integration testing. The results produced during each build may be given to the customers or the development team for evaluation. This is the time to make decisions to either go forward or end the development.

Deployment Phase

The first step of the deployment phase is a system qualification test which is performed to demonstrate to the acquirer that the system requirements have been met. The next step is to prepare for software use, which includes writing the software manuals and preparing for software transition.

Processes in the Software Technical Process

Figure 5.3 shows the P-state tree of the Software Technical Processes. This P-state consists of six operations: Develop Software Requirement, Develop Software Design, Develop Software Code, Develop Software Test, Develop Operation Documentation and Perform Software Enhancement. These operations compose of a complete software life cycle and guide to develop software products.

Main Roles: The whole Development Group should be involved to develop software technical processes. Furthermore, the configuration management staff and quality assurance staff should take responsibility for supporting development of the software technical processes.

Artifact List: The artifact list in the KPA is Software Technical Process which consists of six sub-artifacts: Software Requirement, Software Design, Software Code, Software Test, Operation Documentation and Software Enhancement. The artifacts in this KPA are shown in Figure 5.4.

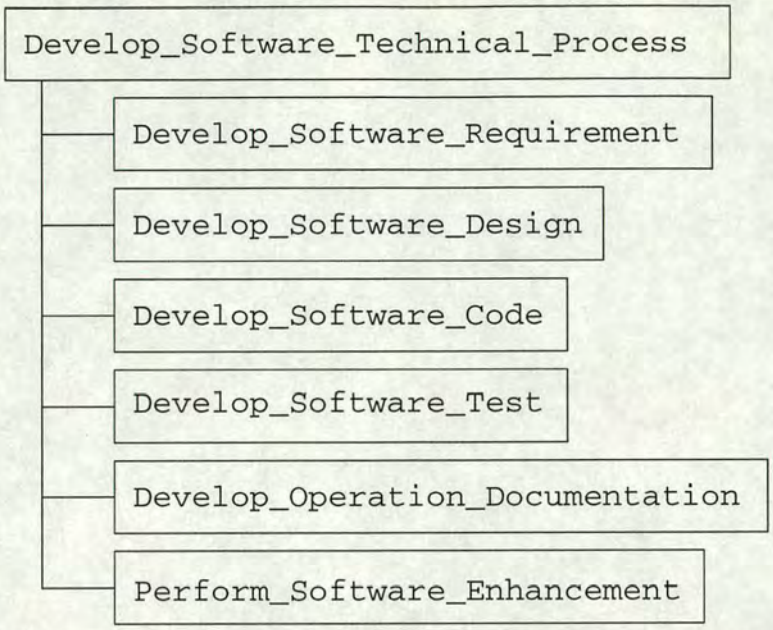

Figure 5.3: The P-State Tree of the Software Technical Process

Information Artifacts: The artifact, Allocated Requirement, as an information artifact provides information for developing the software technical process.

Entrance Condition: state-of(Software_Technical_Process) = Referenced

Once the need to develop a Software Technical Process has been identified, the process roles start to perform all activities in the technical processes.

Activities: Six operations are presented in the P-state diagram of Software Technical Process as shown in Figure 5.5. The project manager conducts activities of software engineering by following the program strategy. Moreover, the development group performs the software technical process according to the software development plan. In the meantime, the operation documents should also be developed. Finally, for both incremental and evolutionary strategies, the software products may need to be updated or refined. This decision will be made by the project manager.

Exit Condition: state-of(Software_Technical_Process)=Developed

After completing the P-states, managers must check whether the exit condition has been reached. This means that the software work products are completely developed and are satisfactory to customers.

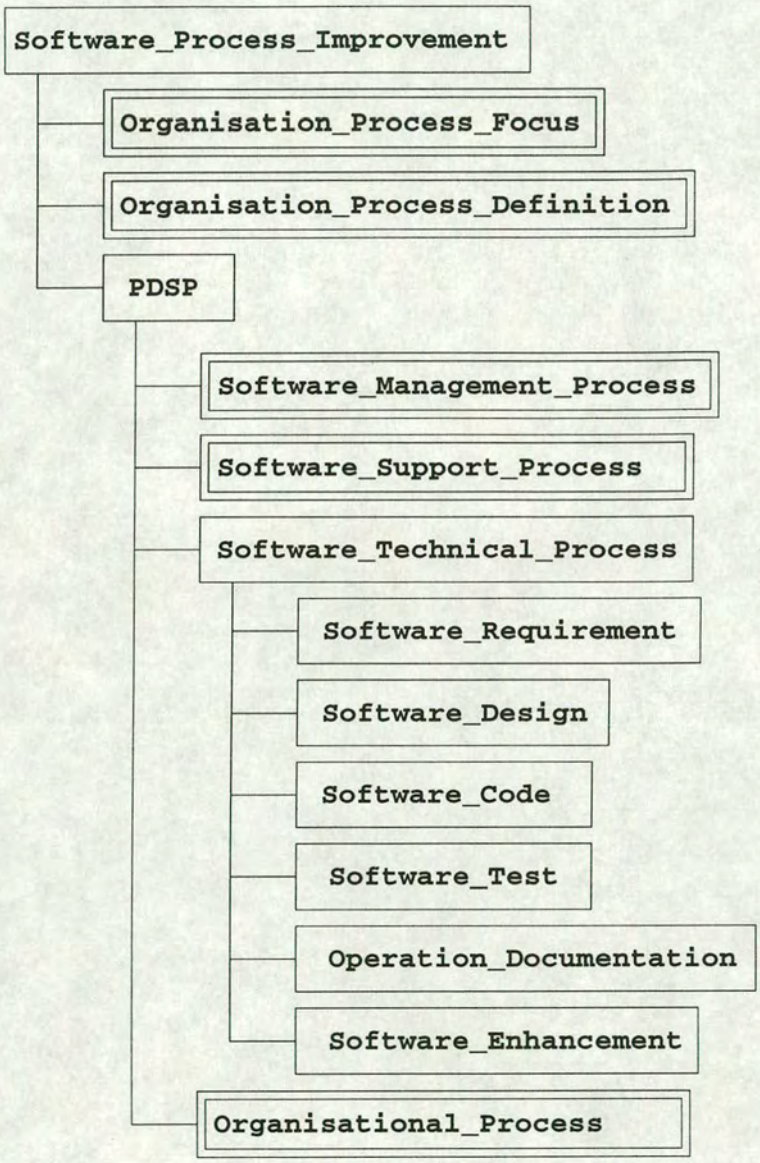

Figure 5.4: The Artifact Tree of the Software Technical Process

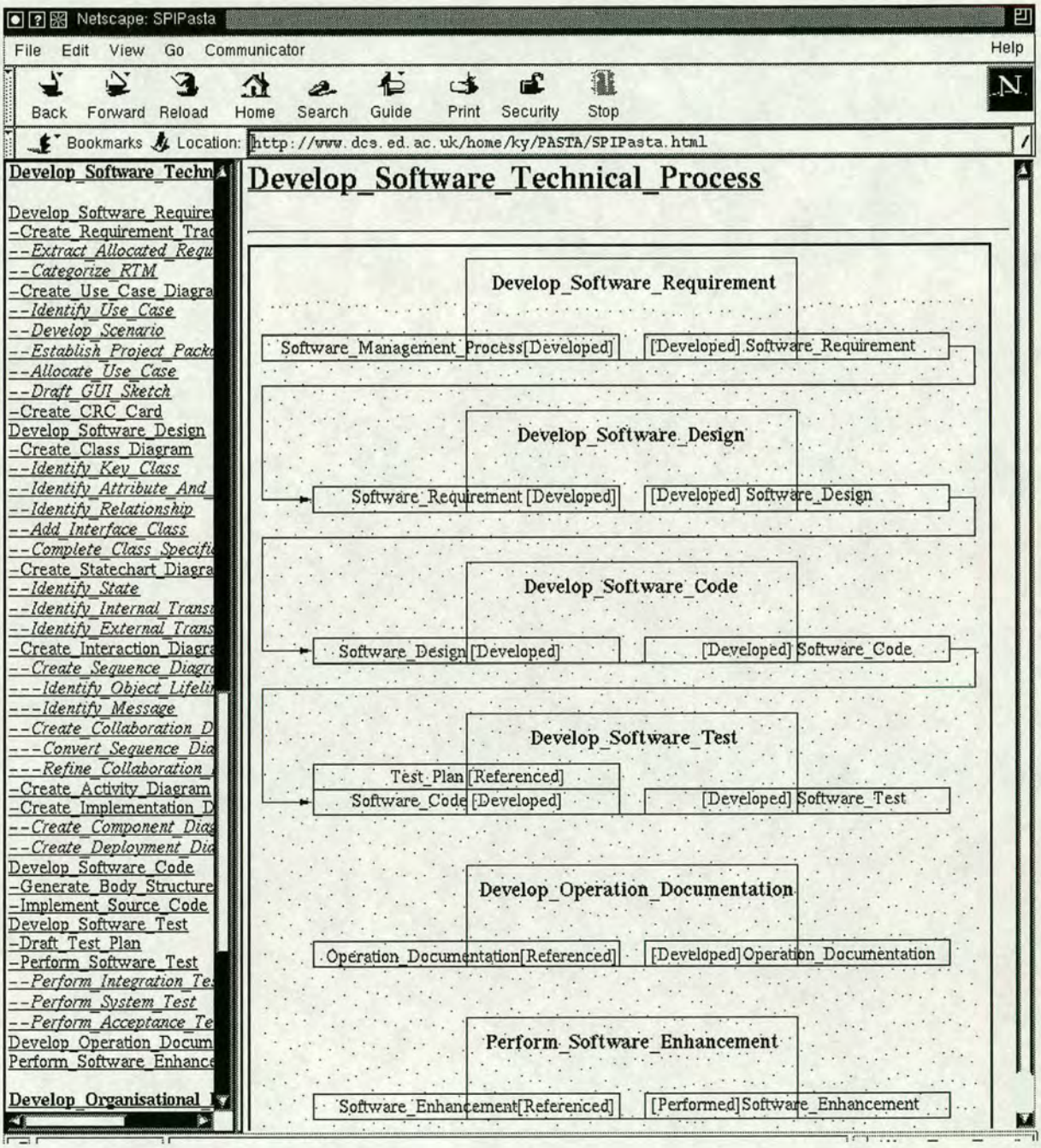

Figure 5.5: The P-State Diagram of the Software Technical Process

5.1.1 The Unified Modelling Language

In the Software Product Engineering key process area, the activities performed describe that should "use effective methods to design the software". However, the problem is what are the effective methods? The software community has debated about the methodology for a long time. We usually have so many "effective" methods to develop our software products. Especially as object orientation has become the mainstream of software development, the OO methods overlap the OO concept, but are not identical. This leads to a big problem - communication. It is difficult to communicate with people when they use different OO methods.

Now, the debate seems to have ended. The Unified Modelling Language (UML), which unifies the methods of Grady Booch, James Rumbaugh and Ivar Jacobson, provides a set of notations for software analysis and design and has been endorsed by the Object Management Group. We might predict the other methods will finally be eliminated in the next few years. However, is it really good for the software community? Basically, most of us will agree so. In this thesis, SPI PASTA would be built by using OO methodology. Especially we adopt the UML as the notation for software analysis and design.

A fundamental change in the UML is that it is a standard modelling language rather than a method. The developers of the UML focus on a common metamodel and a common notation, not on the development process. The primary idea behind the UML is to provide sufficient semantics and notation to address a wide variety of contemporary modelling issues in a direct and economical fashion[BJR97]. With the standard diagrams, a foundation for communication between individuals is built. It results in better interoperability between tools, more available developers who are skilled in using that notation, and lower overall training costs.

The UML basically defines a notation and its semantics. The notation is the syntax of the UML. It is a representation of a user-level model. Users use the notation to present all artifacts of a software system and communicate with other users. The UML also defines a metamodel to provide a single, common and definitive statement of the syntax and semantics of the elements of the UML.

In terms of the views of a model, the UML defines the following graphical diagrams:

- use case diagram
- class diagram
- behaviour diagrams:

- statechart diagram
- activity diagram
- sequence diagram
- collaboration diagram

- implementation diagram
  - component diagram
  - deployment diagram

In the next two sections, we will briefly introduce these diagrams which are viewed as a basis in the technical process.

5.1.2 Requirements Analysis

> *The software requirements cover the software functionality and performance requirements and the interfaces to hardware, other software components, and other system components[Pau97].*

5.1.2.1 The Use Case Model and Use Case Diagram

In software management processes, the allocated requirements are elaborated from customer requirements to plan the project's activities and work products. In software technical processes, the software requirements are derived by analysing the allocated requirements. For a long time, particularly in object oriented methods, use cases have been used to help developers understand requirements. Jacobson[JGJ97] in his book pointed out that the main purposes of the use case model are to define "what" the system should do, and to allow the software engineers and the customer to agree on this. A use case is used to define specific requirements for the behaviour of the system and to drive the rest of the development work where the object modelling activities are performed with the use case model as a starting point.

Jacobson[JEJ95] in his book defined use cases as follows:

> *A use case is a sequence of transactions in a system whose task is to yield a result of measurable value to an individual actor of the system.*

The use case model is composed of actors and use cases. In the UML, the use case model can be presented in the use case diagram. As shown in Figure 5.6, a use case diagram is a graph of actors, a set of use cases, and communication associations between the actors and the use cases.

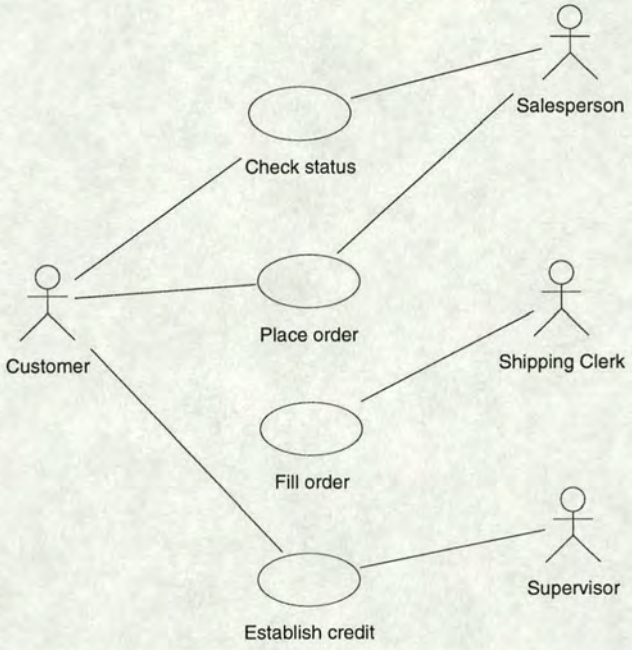

Figure 5.6: The Use Case Diagram[BJR97]

Actors, shown as stick figures in the use case diagram, represent everything that needs to exchange information with the system[JCJO92]. Alternatively, in his later book, Jacobson[JEJ95] described that an actor represents a role that someone or something in the environment can play in relation to the business. The actor can have various roles with regard to a use case; it might be a person, an organisation, or even an external system.

A use case, shown as a named oval in the use case diagram, is a coherent unit of functionality provided by a system or class as manifested by sequences of messages exchanged among the system and one or more outside interactors (called *actors*) together with actions performed by the system[BJR97]. Basically, the "sequence of messages" are described by using plain or structured English and viewed as a "scenario" to implement the application.

5.1.2.2 Use Cases for Requirements Capture

The use case model is good at capturing the requirements for a software project. Developers usually capture a use case by talking to typical users and discussing the various things they might want to do with the system . However, faced with a big system, it can often be difficult to come up with a list of use cases. Software analysts start with the allocated requirements after completing manage-

Entry #	System Specification Text	Type	Build	Use Case Name
1	System Specification Text	SW	B1	Usecase1
2	System Specification Text	HW	B2	Usecase2
...	.....	...	...	.....
SW: Software, HW: Hardware				

Table 5.1: The Requirements Trace Matrix

ment processes. Unfortunately, allocated requirements usually contain a variety of information that is not suitable for describing the use cases.

Texel and Williams[TW97] use a Requirements Trace Matrix (RTM) to extract the allocated requirements. An RTM, shown in Table 5.1, is a matrix that initially contains the set of requirements for a system. Firstly, Texel and Williams suggest extracting the "shall" sentences from the allocated requirements and developing an initial RTM. However, this is not the only rule for developing the RTM. Requirements can be discovered through discussions with domain experts. When an initial RTM has been developed, software analysts should categorise each entry in the RTM according its type. Then, the requirements are prioritised in order to allocate the use case for the "build". Finally, the RTM is reformatted into use case diagrams and continually maintained throughout the life cycle of a project.

The RTM provides a clear vision for developing use cases. The software analysts may perform software design by using use cases. However, analysts need more experience to find classes from use cases. For most software engineers, CRC cards may be a good solution to develop software design.

CRC (Class, Responsibility and Collaborators) cards were invented by Ward Cunningham and Kent Beck[CB89] as a way to help a group of people agree on objects that represent the problem. A class represents a collection of similar objects. A responsibility describes what the object does in the system. Sometimes a class will not have enough information to complete its responsibility, therefore a collaboration is needed to comply with other classes that are involved in carrying out the responsibilities.

For a long time, software analysts have been struggling to find appropriate classes for an application. CRC cards provide a simple but effective technique for extracting the allocated requirements in order to find classes. As Figure 5.7 shows, Ambler[Amb95] suggested that the definition of use cases and prototypes come before a CRC model, which in turn comes before a class diagram. In Ambler's process, the CRC cards are viewed as a bridge between use cases and

class diagrams. In addition to finding classes, the CRC cards are often used to validate the information gathered by use cases and refine these use cases.

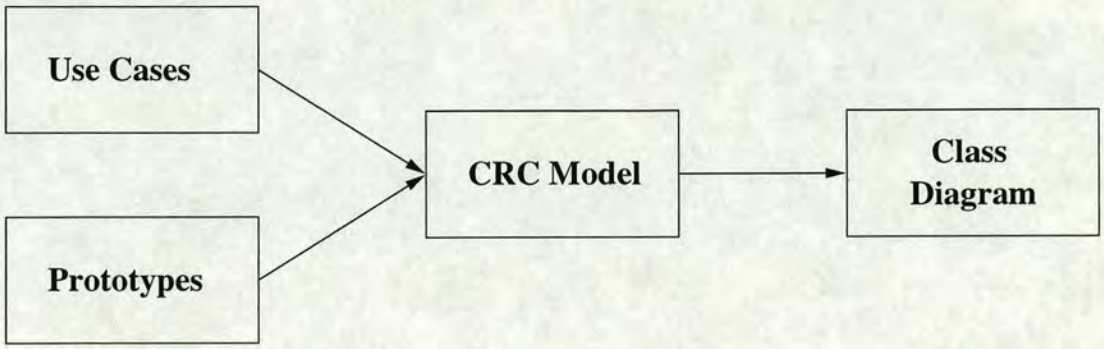

Figure 5.7: How CRC Modelling Fits In[Amb95]

To comply with the Software Product Engineering key process area, a complete process of requirements engineering should be similar to those in Figure 5.8. Before using CRC cards, we suggest that allocated requirements may be extracted and grouped into use cases by using the RTM. Then, as in Ambler's process, the CRC card team uses CRC cards to find classes. However, this is not the "waterfall process" in the RTM, use cases and CRC cards. CRC cards might refine use cases which in turn might redevelop the RTM or even the allocated requirements.

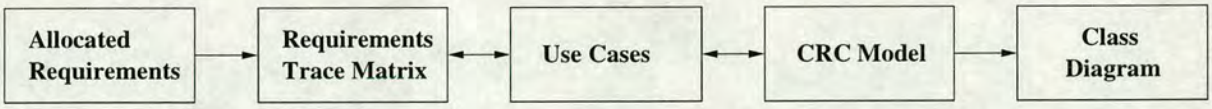

Figure 5.8: The Process of Requirements Engineering

5.1.2.3 Processes in Requirements Analysis

Figure 5.9 shows the P-state tree of Requirements Analysis. This P-state consists of three operations: Create Requirement Trace Matrix, Create Use Case Diagram and Create CRC Card.

Main Roles: The main roles participating in the operations are the project manager and software product managers who conduct the activities, requirement analysts who perform the activities and the configuration management staff who manage the configuration items.

Artifact List: The artifact list in this part is Software Requirement which consists of three sub-artifacts: Requirement Trace Matrix, Use Case Diagram and CRC Card. The artifacts in the software requirement are shown in Figure 5.10.

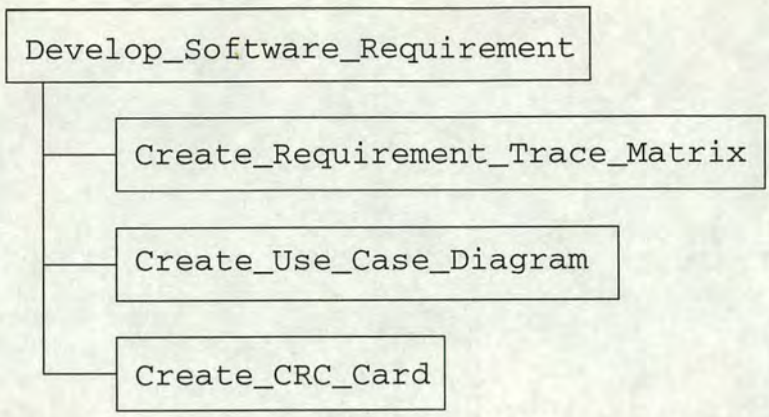

Figure 5.9: The P-State Tree of Requirements Analysis

Information Artifacts: The artifact, Allocated Requirement, as an information artifact provides the requirements for software analysis. Software analysts will analyse these requirements into software requirements.

Entrance Condition: state-of(Software_Management_Process) = Developed

After completing software management processes, the managers and staff can develop software technical processes. The development group perform the activities according to the software development plan which is developed in software management processes.

Activities: Three operations are presented in the P-state diagram of Software Requirement as shown in Figure 5.11. Software analysts can firstly analyse the allocated requirements by using the requirement trace matrix. Analysts list the appropriate sentences and categorise them as a foundation for the use cases. When the requirement trace matrices are created, analysts can build the use case diagrams by using CASE tools. Before developing software design, the managers, analysts and designers may use the CRC cards to discover classes. CRC cards are the bridge between requirement analysis and software design. Analysts and designers might take the round-trip between use case diagrams and CRC cards until use cases are appropriately refined and classes are completely discovered.

Exit Condition: state-of(Software_Requirement) = Developed

After completing the P-states, managers must check whether the exit condition has been reached. This means that the software requirements are appropriately analysed.

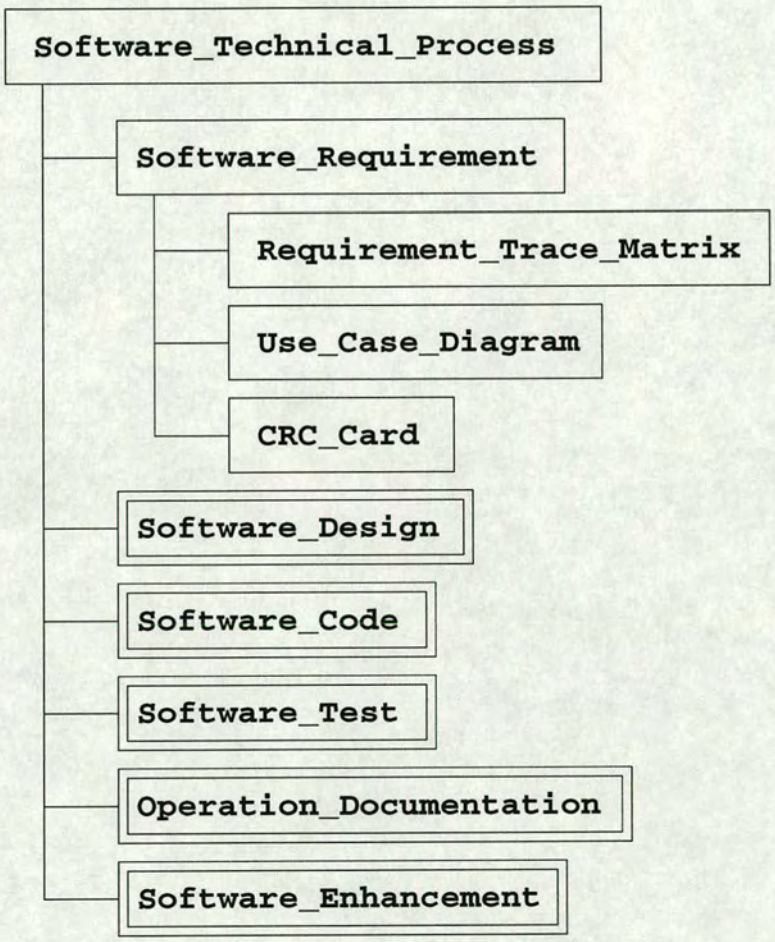

Figure 5.10: The Artifact Tree of Requirements Analysis

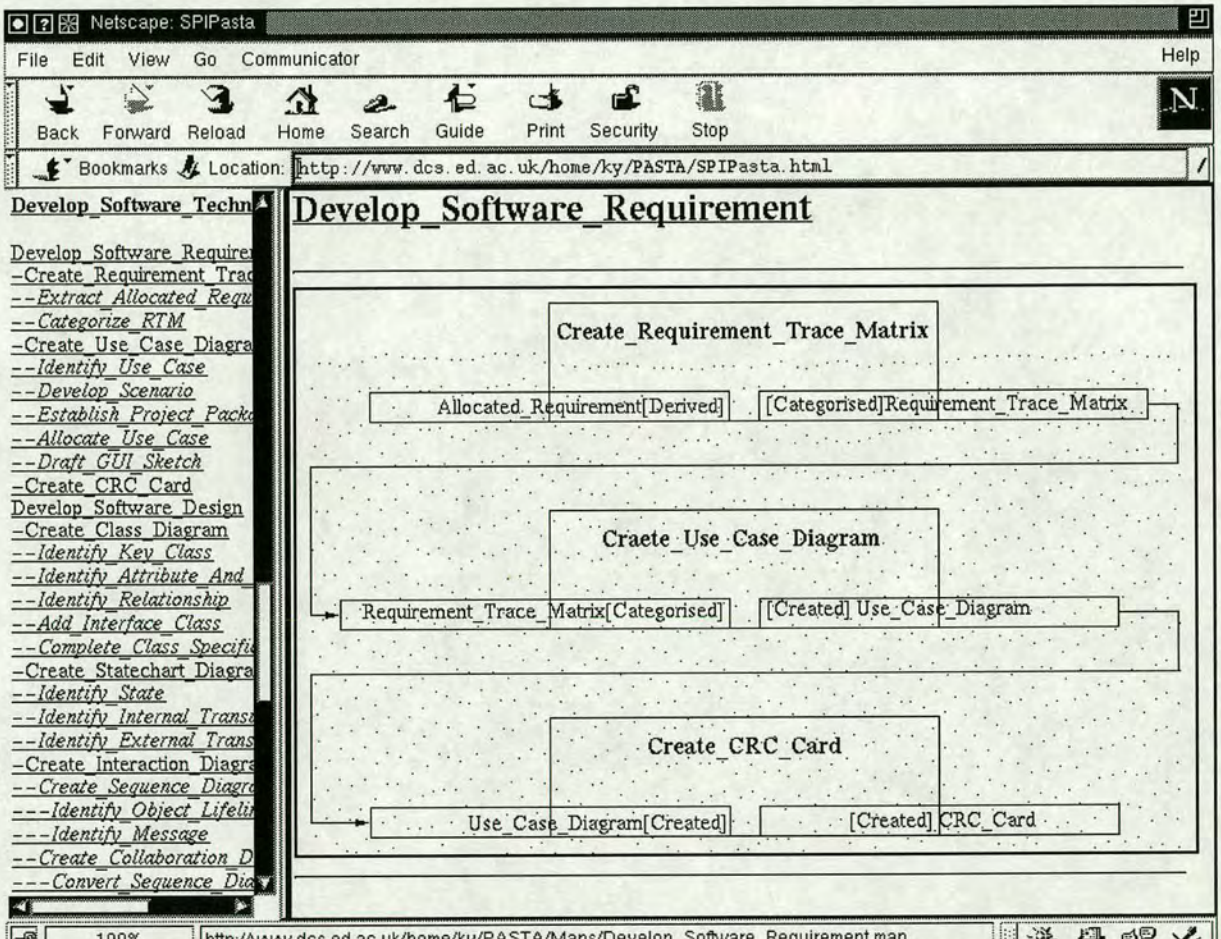

Figure 5.11: The P-State Diagram of Software Requirement

5.1.3 The Software Design

> *The software design consists of the software architecture and the detailed software design (and there may be multiple levels of detailed design). It covers the software components; the internal interfaces between software components; and the software interfaces to other software systems, to hardware, and to other system components (for example, people). The software design is derived from the software requirements and forms the framework for coding.*[Pau97]

5.1.3.1 The Visual Modelling Language

The UML is a language for visualising, specifying, constructing and documenting the artifacts of a software system. It provides multiple perspectives of the software system under analysis and development. By using its diagrams, a software system can be completely and consistently presented.

Class Diagrams

Class diagrams, as shown in Figure 5.12, play a very important role in the UML. They are the bridge between software requirements and software implementation. The UML defines a very rich set of class diagram features, most of which are intended to support analysis, design and implementation

Software analysts use CRC cards to find classes, their responsibilities and collaborations from use cases. This information, collected in the class diagrams, is the basis for software coding to implement the application. In the UML, a class diagram is shown as a rectangle including a class name which is documented with its properties, attributes which are documented by a description of what they contain, and operations which are services that an instance of the class may be requested to perform. Additionally, some relationships between classes are defined in order to capture the coupling in software design. The main one is an association which is drawn as a solid path connecting two classes. Composition shows ownership between two classes, which is drawn as a solid filled diamond. Generalisation is the taxonomic relationship between a more general element and a more specific element, which is drawn as a solid-line path from the more specific element to the more general element with a hollow triangle at the end of the path.

Interaction Diagrams

In the UML, class diagrams show the static structure of the system, moreover, interaction diagrams capture the dynamic behaviour of the system. An interaction

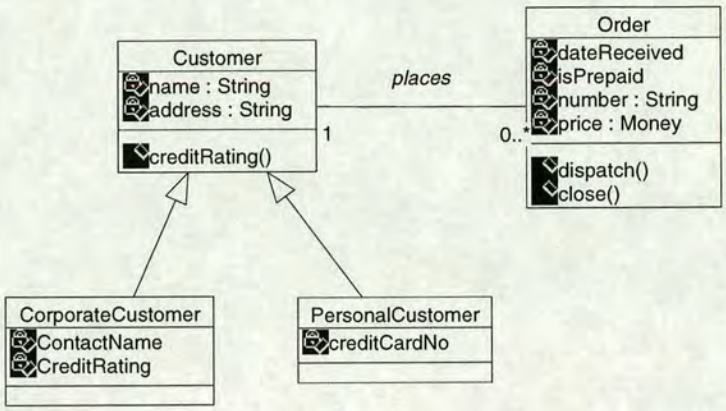

Figure 5.12: The Class Diagram

diagram presents the interactions between objects. With regard to a complex system, interaction diagrams provide a logic view for the system. This results in an integrated solution. However, interaction diagrams are less important than class diagram. In a simple system it might not be necessary to create these diagrams.

The UML defines two kinds of interaction diagrams: sequence diagrams and collaboration diagrams. Developers do not need to create both diagrams in the same system. Sequence diagrams show the explicit sequence of messages and are better for real-time specifications and for complex scenarios. Collaboration diagrams show the relations among objects and are better for understanding all of the effects on a given object and for procedural design[BJR97].

Sequence Diagrams Sequence diagrams, as shown in Figure 5.13, show the details of the interactions between objects. Originally, Jacobson defined the sequence diagram for showing how the participating objects realize the use case through their interaction[JCJO92]. Sequence diagrams provide a complete logic presentation for the use case scenario. The developers can get a picture of how sequence progresses over the objects participating in the system.

Within a sequence diagram, objects are represented on the horizontal dimension with a vertical line which is called the object's "lifeline". The object can be created or destroyed during the period of time shown on the diagram. The lifeline represents the existence of the object at this period. The behaviour which the objects will perform is described on the leftside of the diagram. Furthermore, a message, being a communication between objects, is shown as an arrow between

the lifelines of two objects. The order of these messages is normally shown top to bottom with sequence numbers on the diagram. Each message also has its own name.

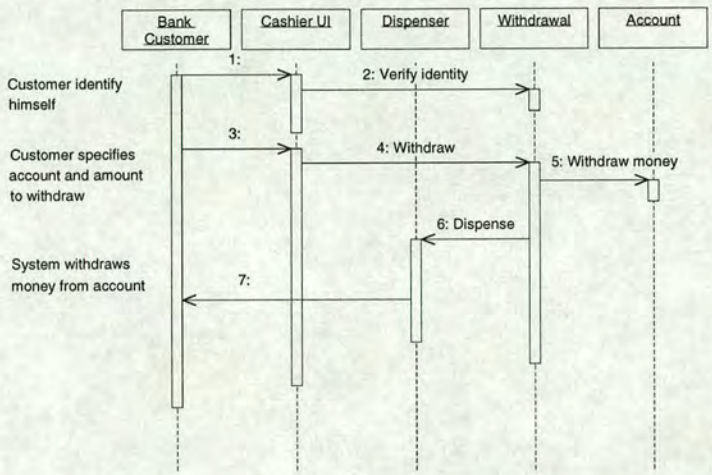

Figure 5.13: The Sequence Diagram[JGJ97]

Collaboration Diagrams Collaboration diagrams, as shown in Figure 5.14, show the relationships among the objects rather than the time frame within the use case. This is why collaboration diagrams are not suitable for real-time or concurrent systems. However, the difference between collaboration diagrams and sequence diagrams might be only the layout of the objects and messages. Although Jacobson originally did not define the collaboration diagram for the use case, collaboration diagrams provide a supplement for the class diagrams. Developers can use collaboration diagrams to get the big picture of the system, incorporating the message flow of use case scenarios.

A collaboration diagram is a graph of references to objects and links with message flows attached to its links. The rectangles represent the various objects within the system and the line linking objects represents the relationship between them. This layout makes it more difficult to see the sequence between objects, however, it shows how the objects are linked together and provides a clear picture to show the relationships among the objects.

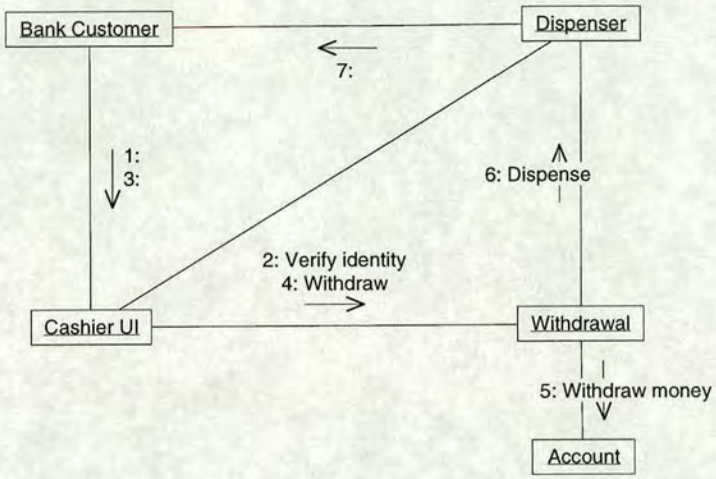

Figure 5.14: The Collaboration Diagram

Statechart Diagrams

Interaction diagrams show the interactions among the objects; however, the internal behaviour of an object needs another diagram to complement this. A statechart diagram, as shown in Figure 5.15, is the graph of states and transitions that describes the response of an object of a given class to the receipt of outside stimuli.

A state, shown as a rectangle with rounded corners, is a condition during the life of an object. A transition, shown as a solid arrow from one state to another state, is a relationship between two states. The relationship represents the action which should be done before entering the second state. Transactions might have multi target states especially in concurrent systems. However, it is not necessary to have a statechart diagram for each class diagram. Sometimes a statechart diagram might be useless; it depends on the complexity of the class.

Activity Diagrams

Since interaction diagrams present the behaviour of several objects within a single use case, developers cannot recognise a precise definition of the objects' behaviour. Fowler[Fow97] in his book suggested that if developers want to look at the behaviour of a single object across many use cases, they should use a state diagram, and if developers want to look at behaviour across many use cases or

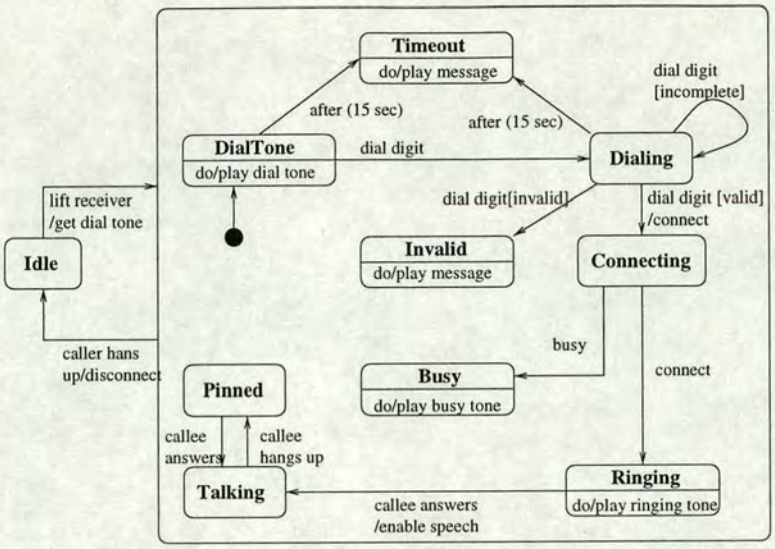

Figure 5.15: The Statechart Diagram[BJR97]

many threads, they should consider an activity diagram.

An activity diagram, as shown in Figure 5.16, is a special case of a state diagram. The purpose of this diagram is to focus on flows driven by internal processing. Basically, it consists of action states and transitions triggered by completion of the actions in the source states. Action states do not have internal transitions or outgoing transitions based on explicit events, since statechart diagrams present these transitions.

Implementation Diagrams

Implementation diagrams present how the system is implemented. The UML defines two forms for implementation diagrams. The component diagram shows the structure of the code itself and the deployment diagram shows the structure of the run-time system.

Component Diagrams A component diagram shows the dependencies among software components. Currently, component-based software development, in which software items can be assembled as hardware, is increasingly presented in the software community. With the standards (such as CORBA) published and matured, the component-based software development will be one of the most important research fields in software engineering. Component diagrams are just used to show the software components.

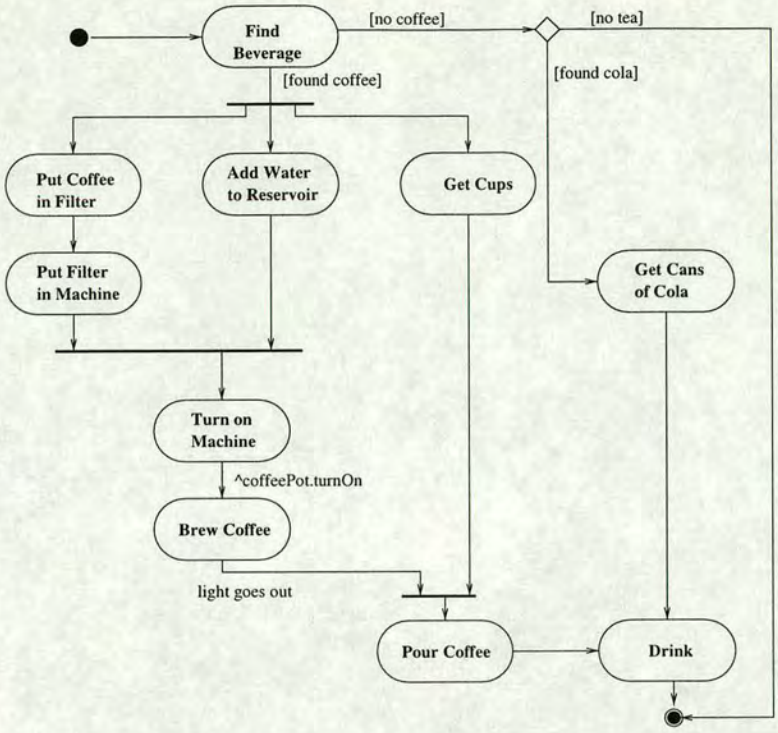

Figure 5.16: The Activity Diagram[BJR97]

Deployment Diagrams A deployment diagram is a graph of nodes connected by communication associations. The node is a run-time physical object which represents a processing resource. The node may contain component instances which live or run on the node.

5.1.3.2 Processes in the Software Design

Figure 5.17 shows the P-state tree of Software Design. This P-state consists of five operations: Create Class Diagram, Create Statechart Diagram, Create Interaction Diagram, Create Activity Diagram and Create Implementation Diagram.

Main Roles: The main roles participating in the operations are software product managers who conduct the activities, software designers who perform the activities, requirement analysts who provide the requirement details to software designers and the configuration management staff who manages the configuration items.

Artifact List: The artifact list in this part is Software Design consists of five sub-artifacts: Class Diagram, Statechart Diagram, Interaction Diagram, Activity

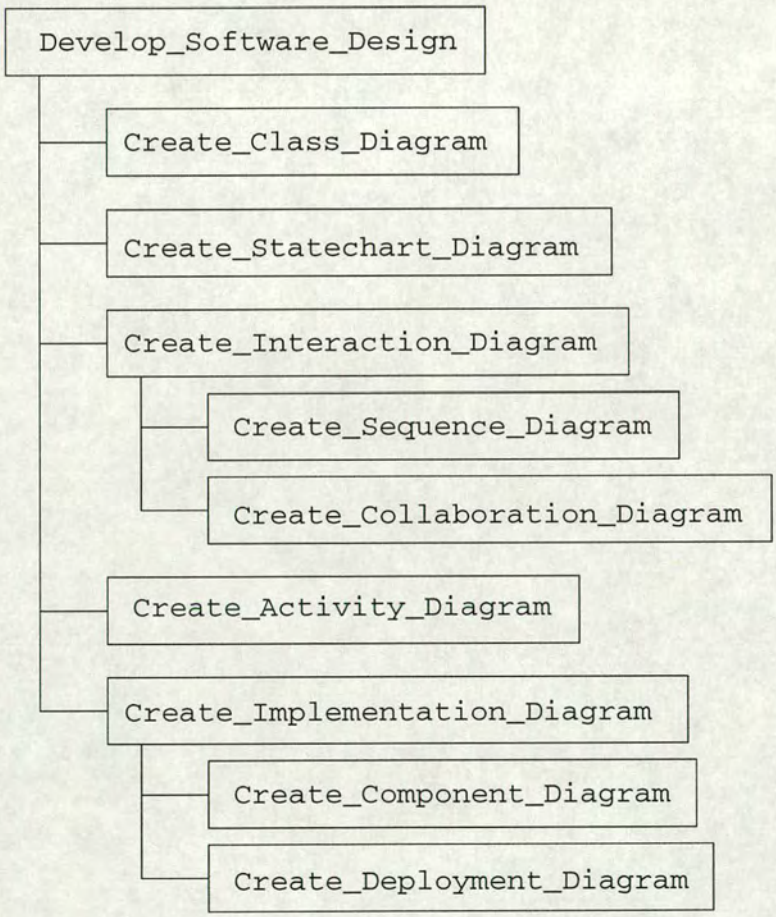

Figure 5.17: The P-State Tree of the Software Design

Diagram and Implementation Diagram. The artifacts in the software requirement are shown in Figure 5.18.

Information Artifacts: The artifact, Software Requirement, as an information artifact provides the details for software design. This could be use case diagrams or CRC cards. Software designers create the diagrams according to use case diagram and CRC cards.

Entrance Condition: state-of(Software_Requirement) = Developed

After creating use case diagrams and CRC cards, the software designers can develop the software design.

Activities: Five operations are presented in the P-state diagram of Software Design as shown in Figure 5.19. Use case diagrams and CRC cards, created in software requirement analysis, provide the details of classes for software designers. Therefore, class diagrams should be created first. Furthermore, statechart diagrams present the behaviour of an object of a given class. Normally, a statechart diagram is attached to a class.

The other diagrams can be independently created. For example, software designers may decide what kind of interaction diagrams will be used, sequence diagrams or collaboration diagrams. This decision relies on the characteristic of the software project. If designers want to look at behaviour across many use cases, they may create activity diagrams. If the project is split into components, designers may need component diagrams. Furthermore, if the project will be run in different types of machines, deployment diagrams should be created during the software design.

Exit Condition: state-of(Software_Design) = Developed

After completing the P-states, product managers must check whether the exit condition has been reached. This means that the relevant diagrams are appropriately created.

5.1.4 Software Implementation

When the software design is completed, a detailed design representation of software should be translated into a programming language realisation. Since software implementation begins after the software design has been defined, the source code should be directly generated from the software design.

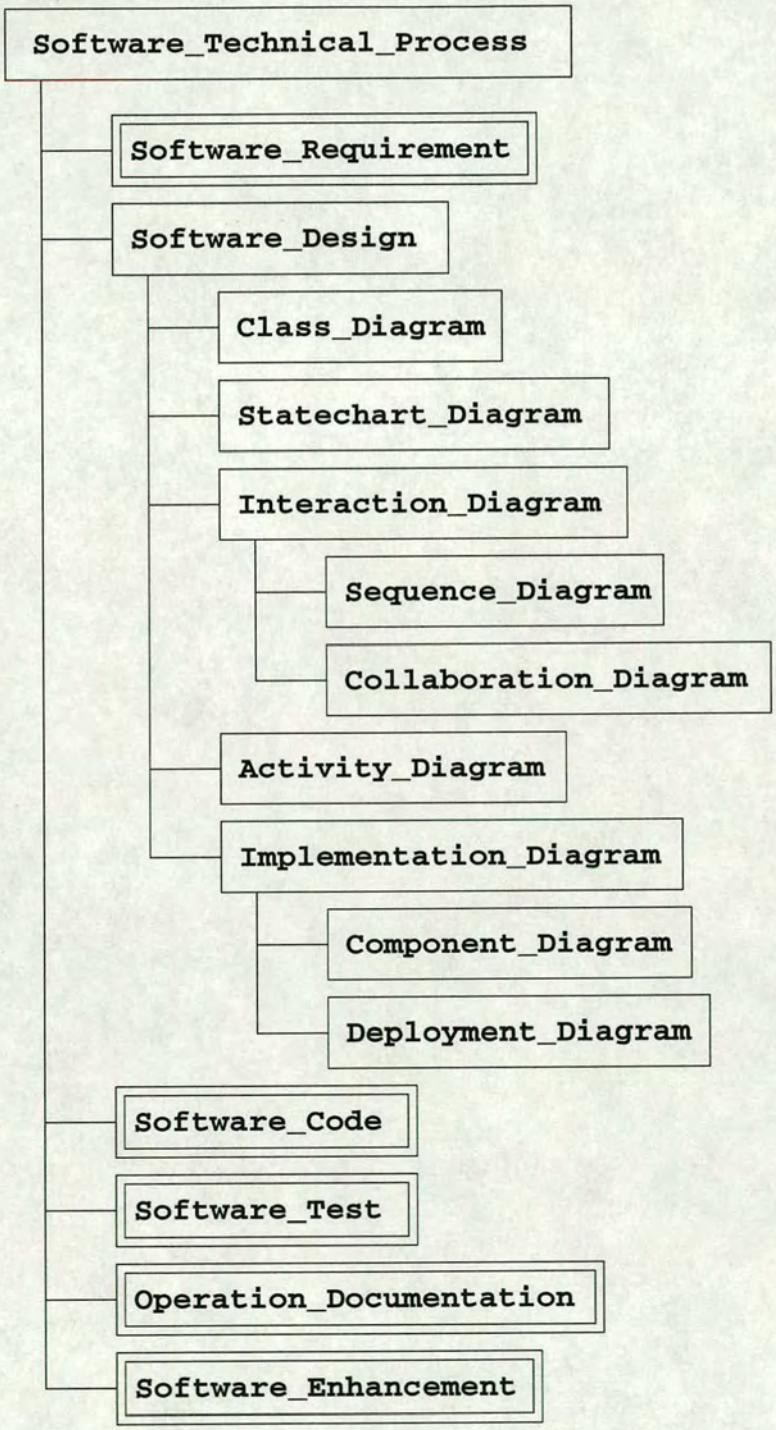

Figure 5.18: The Artifact Tree of the Software Design

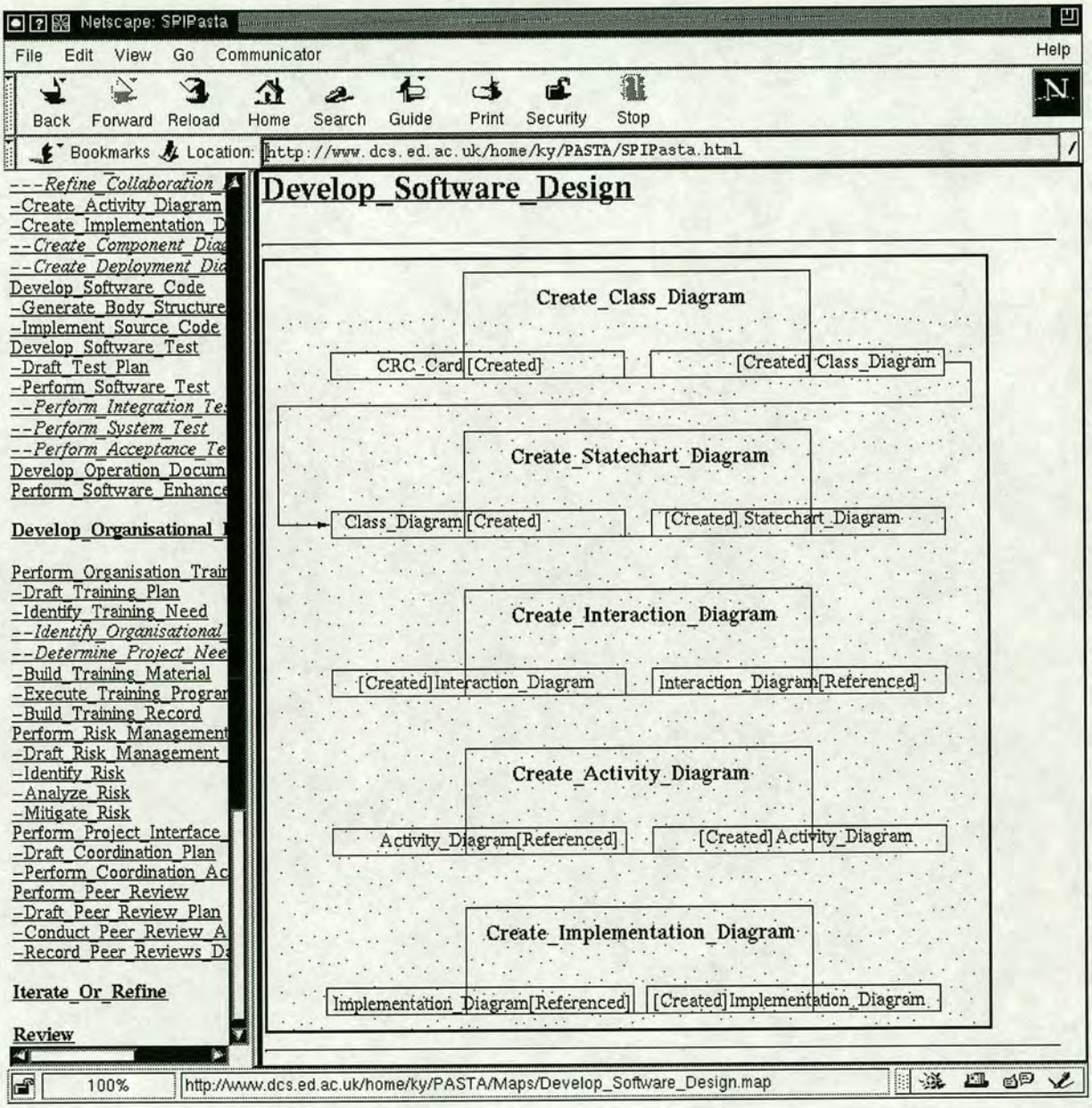

Figure 5.19: The P-State Diagram of the Software Design

For the last two decades, the software community has been faced with maintaining aging software systems that are constructed to run on a variety of hardware types and programmed in obsolete languages. Particularly, those systems were developed with poor design, unstructured programming methods, poor code quality and poor documentation. As a result, the task of software maintenance becomes more complex and more expensive. Furthermore, every aspect of computing changes rapidly so that software maintenance is increasingly essential for the software organisations. Therefore, for the sake of maintenance, software implementation is not only coding. The research of reverse engineering provides a foundation for software implementation. If it is necessary for identifying software artifacts, discovering their relationships and generating abstractions in the future, why software organisations do not firstly focus on these topics?

Currently, some software organisation define their own coding standards to facilitate the maintenance, portability, and reuse of programming language. These standards are based on proven software engineering principles that lead to code that is easy to understand, maintain and enhance. Furthermore, following a common set of coding standards results in greater consistency, making the development team significantly more productive.

In addition, CASE tools are concentrating on code generation. Case tools usually generate source code from a software design model. They produce a code architecture by using properties such as the class model's attributes, operations and so on. As a result, the code architecture complying with the organisation's coding standard can lead to software implementation which is much more controlled and productive. Moreover, as project parameters change or new requirements are added, reverse engineering tools can extract data and design information from an existing program. This can reduce a significant maintenance effort.

5.1.4.1 Processes in Software Implementation

Figure 5.20 shows the P-state tree of Software Implementation. This P-state consists of two operations: Generate Body Structure and Implement Source Code.

Main Roles: The main roles participating in the operations are software programmers who perform the activities and configuration management staff who manage the configuration items.

Artifact List: The artifact list in this part is Software Code which consists of two sub-artifacts: Body Structure Code and Source Code. The artifacts in the

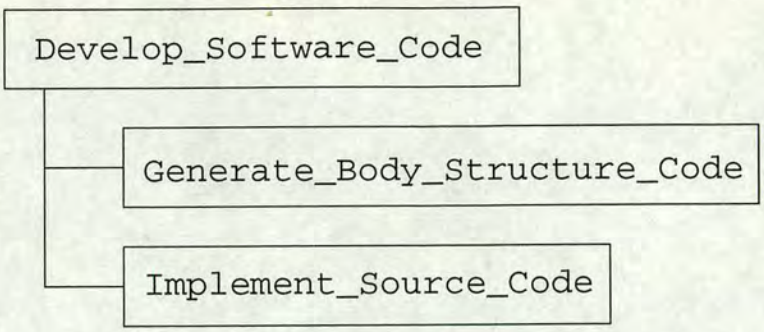

Figure 5.20: The P-State Tree of Software Implementation

software implementation are shown in Figure 5.21.

Information Artifacts: The artifact, Software Design, as an information artifact provides the details for programming source codes. Software programmers will rely on them to perform the activities.

Entrance Condition: state-of(Software_Design) = Developed

After completing software design, the programmers can develop source codes. In this thesis, programmers will use class diagrams and related UML diagrams as a foundation to develop source codes.

Activities: Two operations are presented in the P-state diagram of Software Implementation as shown in Figure 5.22. Software programmers can firstly generate the body structure code of class diagrams by using CASE tools or depending on the coding standards. As the basic structures are built, programmers may develop source codes for each class according to statechart diagrams, behaviour diagrams and implementation diagrams. This step will not complete until all source codes are developed.

Exit Condition: state-of(Software_Code) = Developed

After completing the P-states, programmers must check whether the exit condition has been reached. This means that the software codes are completely developed.

5.1.5 Software Testing

MIL-STD-498 separates software testing into three phases: unit testing, integration testing and system testing.

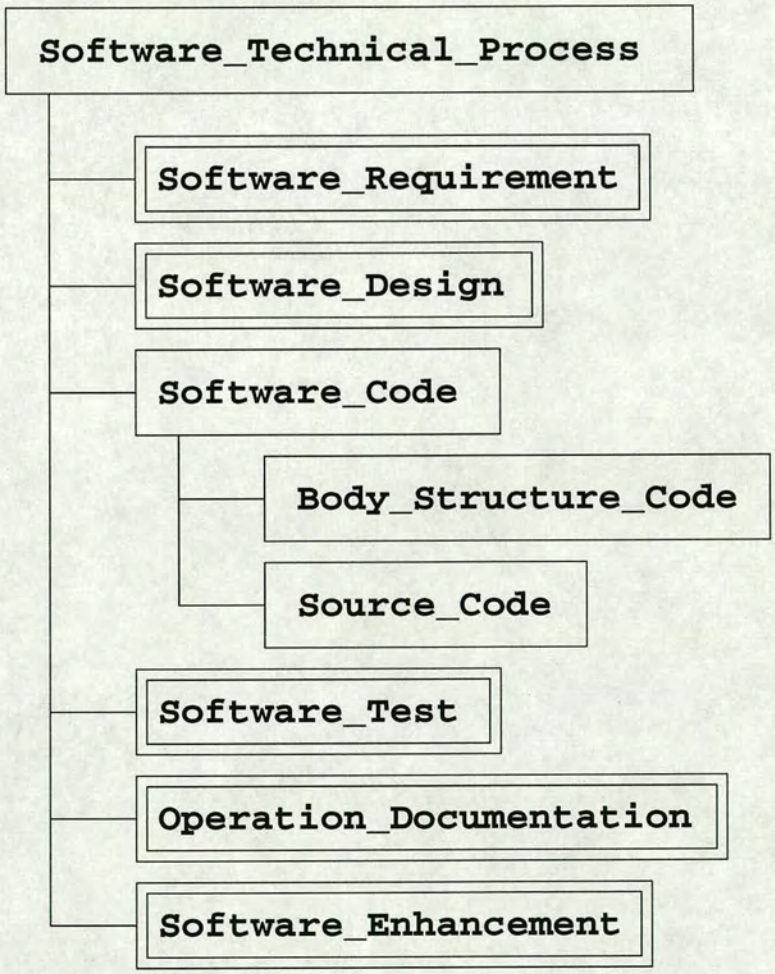

Figure 5.21: The Artifact Tree of Software Implementation

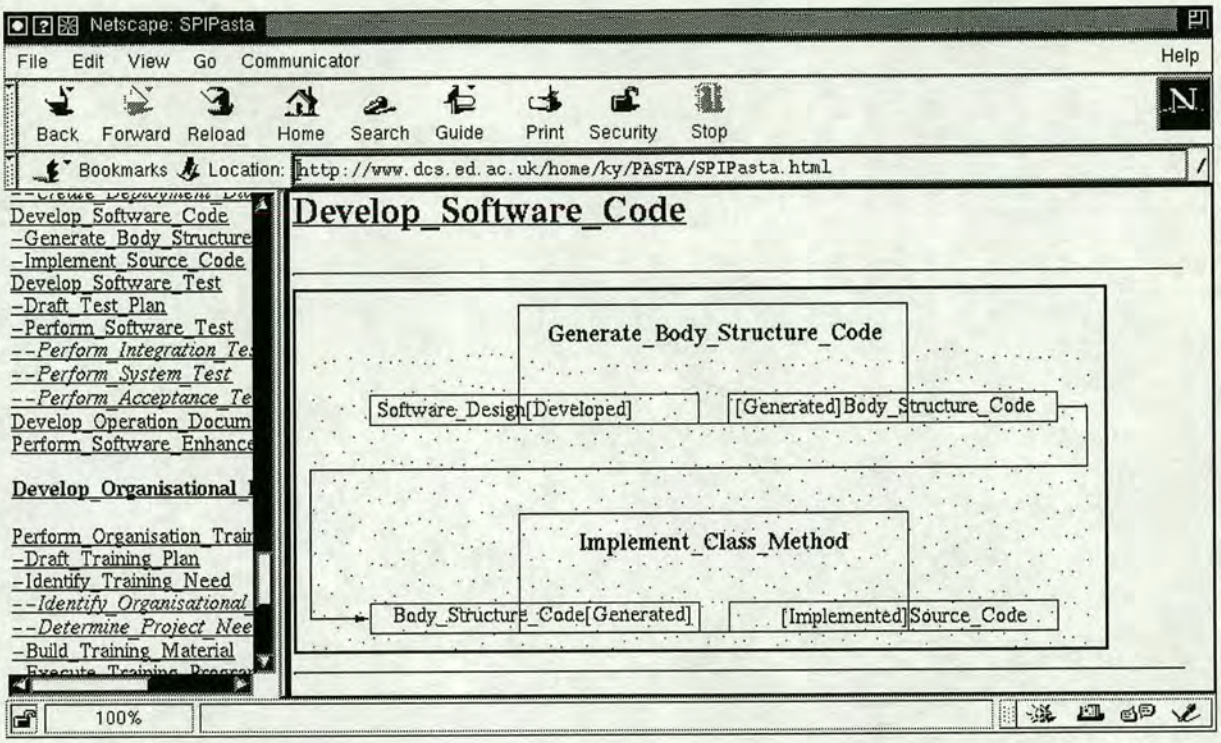

Figure 5.22: The P-State Diagram of Software Implementation

Unit testing is normally considered an adjunct to the software implementation. In OO methods, unit testing concerns classes, which implies that unit testing in OO systems must be carried out at a higher level. Before performing unit testing, the developers should establish test cases, test procedures and test data for testing the software corresponding to each software unit. When software implementation is completed in each build, developers begin to perform unit testing in accordance with the unit test case and procedure.

The purpose of integration testing is to test whether different units that have been developed are working together properly. As we mentioned in Section 4.1.2, the software project is split into several partitions in order to be concurrently developed. Integration testing is normally performed at the end of software implementation. It may include the testing of use cases, subsystems and the entire system. Once all partitions of the software project have been completed and unit testing for each partition has also been performed, developers should take integration test procedures. Moreover, an organisation could use "incremental" or "evolutionary" development strategy. In incremental strategy, developers perform the software project in a sequence of builds. Therefore, integration testing will not be complete until the final build. However, in evolutionary strategy, the customer requirements are partially defined up front, then are refined in each

succeeding build. Thus, integration testing could be performed in each build, since the project could be completed in every build. Furthermore, both software components developed internal to the software project and software components obtained externally to the software project must be appropriately integrated.

System testing is a series of different tests whose primary purpose is to fully exercise the computer-based system. Pressman[Pre94] in his book suggested the types of system tests as follows:

Recovery Testing Recovery testing is a system test that forces the software to fail in a verity of ways and verifies that recovery is properly performed.

Security Testing Security testing attempts to verify that protection mechanisms built into system will protect it from improper penetration.

Stress Testing Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency, or volume.

Performance Testing Performance testing is designed to test the run-time performance of software within the context of an integrated system.

Furthermore, MIL-STD-498 defines three step for system testing: CSCI qualification testing, CSCI/HWCI integration and testing, and system qualification testing. Since we are focusing on software development, CSCI/HWCI integration and testing is beyond the topic of this thesis. Therefore, CSCI qualification testing and system qualification testing could be combined together to demonstrate to the customer that customers' requirements have been met. The system testing in the CMM concentrates on validating the software satisfies the allocated requirements.

Finally, acceptance testing is performed to demonstrate to the customer that the software system satisfies the customer requirements for the software project.

5.1.5.1 Processes in Software Testing

Figure 5.23 shows the P-state tree of Software Testing. This P-state consists of two operations: Draft Test Plan and Perform Software Test which contains three activities: Perform Integration Test, Perform System Test and Perform Acceptance Test.

Main Roles: The main roles participating in the operations are the project manager and software product managers who conduct the activities, the testing

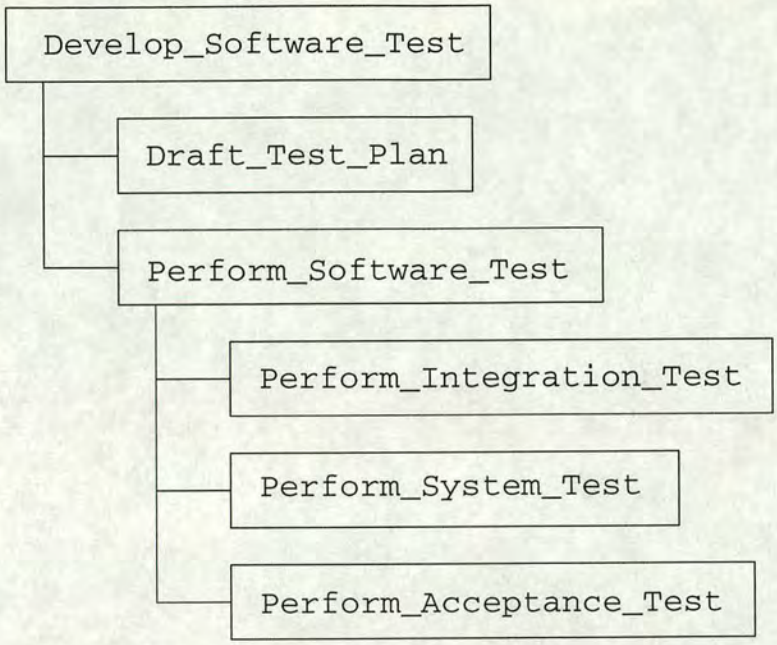

Figure 5.23: The P-State Tree of Software Testing

staff who perform the activities, the configuration management staff who manage the configuration items, and customers who are involved in testing activities to provide their opinions.

Artifact List: The artifact list in this part is Software Test which consists of four sub-artifacts: Test Plan, Integration Test, System Test and Acceptance Test. The artifacts in the software Testing are shown in Figure 5.24.

Information Artifacts: The artifact, System Requirement, as an information artifact that provides the details for software testing. This is because system requirements are abstracted from customer requirements, and software work products must be tested to satisfy the customer's need.

Entrance Condition: state-of(Test_Plan) = Referenced or state-of(Software_Code) = Developed

When the need to draft the software test plan has been identified, the testing staff should take the responsibility to develop the test plan. Furthermore, after developing software codes, the testing staff will perform software tests according to the test plan.

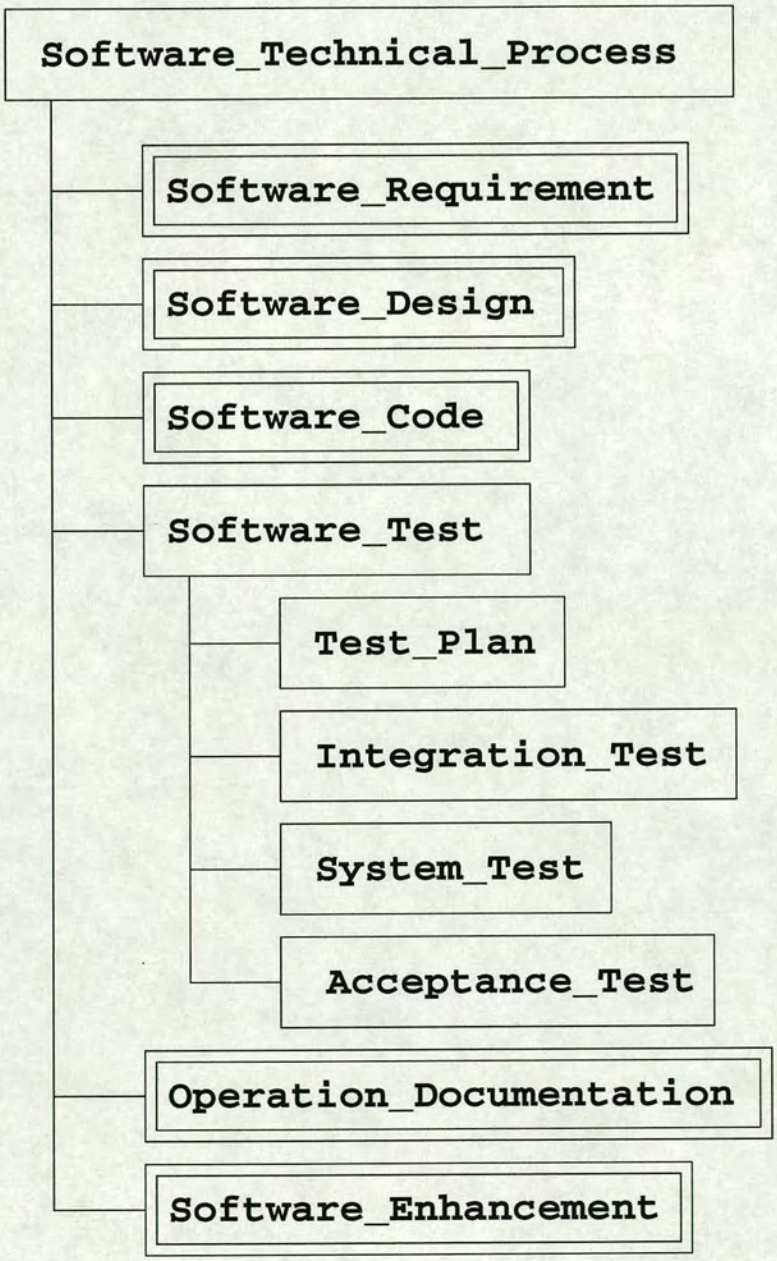

Figure 5.24: The Artifact Tree of Software Testing

Activities: Two operations are presented in the P-state diagram of Software Testing as shown in Figure 5.25. Firstly the testing staff should draft the software test plan. This plan might be drafted as part of the software development plan. In the meantime, the software codes should be developed when the testing staff perform the activities of software test. The testing activities contain integration test, system test and acceptance test. The testing staff must test the software work products step by step. Finally the software system will be demonstrated to the customers to ensure that the system satisfies the customer requirements.

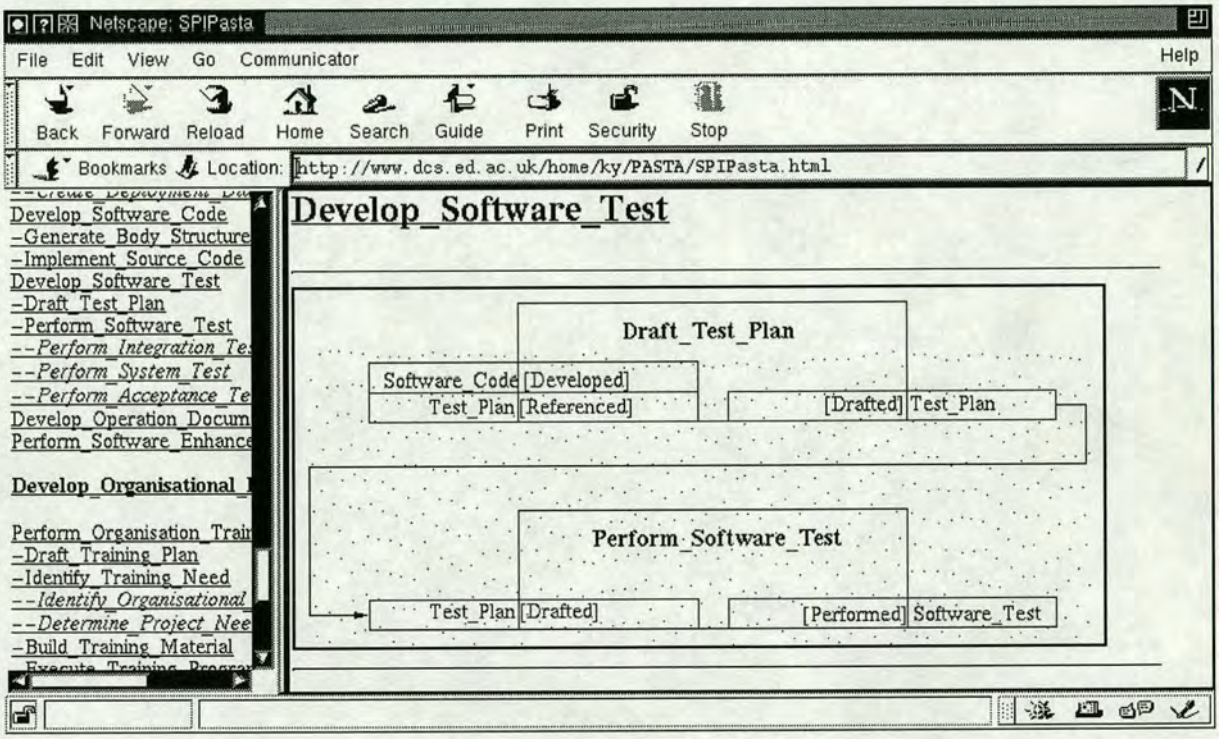

Figure 5.25: The P-State Diagram of Software Testing

Exit Condition: state-of(Software_Test) = Developed

After completing the P-states, process roles must check whether the exit condition has been reached. This means that the software work products are completely tested and satisfy the customer's need.

5.2 The Organisational Process

The Organisational Process consists of four artifacts: Organisation Training Program, Risk Management, Project Interface Coordination and Peer Reviews. All artifacts except risk management are the KPAs of the CMM level 3. Risk management is described in Activities 6 and 7 of the Integrated Software Management

key process area. The reason we put risk management in the organisational process is that organisations need special efforts to manage software project risks. Moreover, the primary purpose of the Integrated Software Management key process area is to tailor the project's defined software process. It is better to make a clear vision for developers.

At level 3, improvement efforts are coordinated and focused at the organisational level. These artifacts will provide the necessary activities to direct the software technical process.

Processes in the Organisational Process

Figure 5.26 shows the P-state tree of the Organisational Process. This P-state consists of four operations: Perform Organisation Training Program, Perform Risk Management, Perform Project Interface Coordination and Perform Peer Reviews.

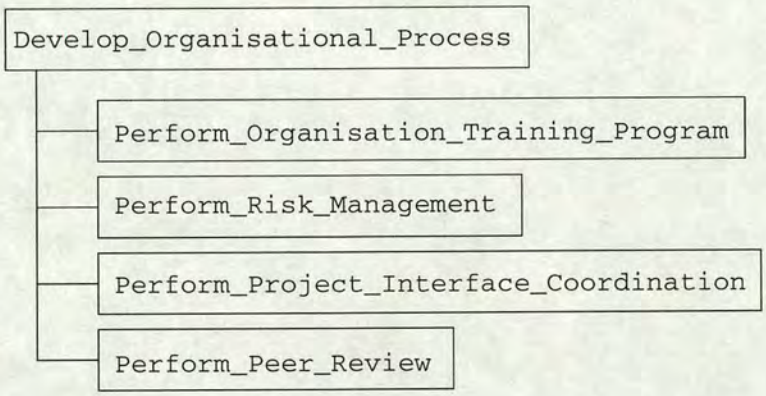

Figure 5.26: The P-State Tree of the Organisational Process

Main Roles: The main roles participating in the operations are project managers, software product managers, training staff and reviewers who will derive the organisational process.

Artifact List: The artifact list in this part is Organisational Process which consists of four sub-artifacts: Organisation Training Program, Risk Management, Project Interface Coordination and Peer Reviews. The artifacts in the organisational process are shown in Figure 5.27.

Information Artifacts: Users developing the organisational process should rely on the project's defined software process.

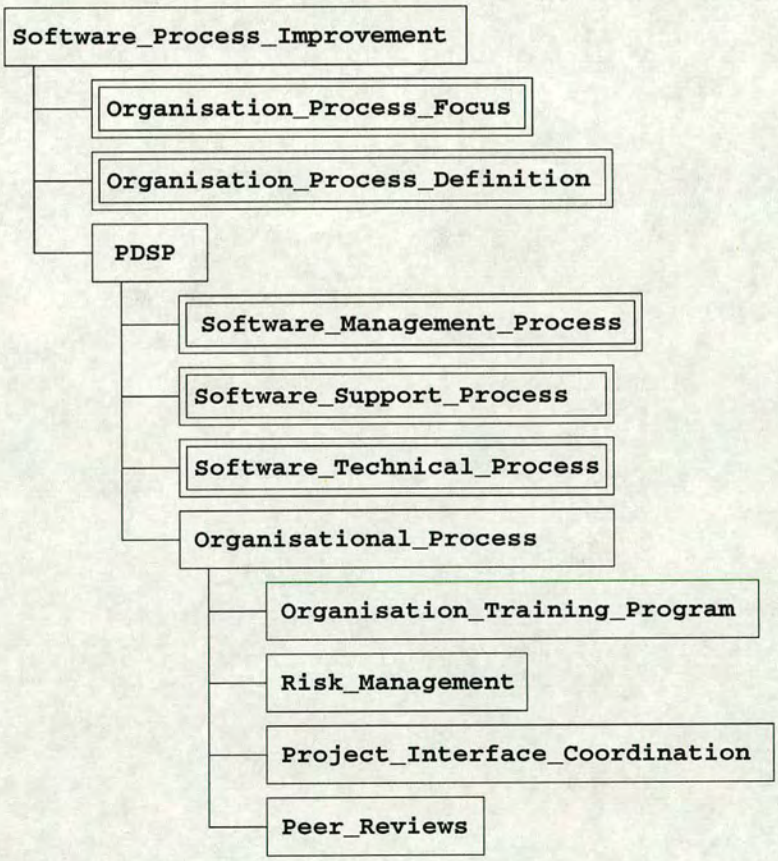

Figure 5.27: The Artifact Tree of the Organisational Process

Entrance Condition: state-of(Organisational_Process) = Referenced

After developing the software management process, the basic project management practices have been established. Complying with the technical process, the need to develop the organisational process should be identified.

Activities: Four operations are presented in the P-state diagram of the Organisational Process as shown in Figure 5.28. There are no relationships between these operations, therefore these operations can be independently performed. The project's defined software process defines the processes for performing these operations. All activities should be performed when the project's defined software process has been developed and throughout the whole development life cycle.

Exit Condition: state-of(Organisational_Process) = Developed

After completing the P-states, managers must check whether the exit condition has been reached. This means that the artifacts have been well performed and recorded.

5.2.1 Organisation Training Program

> *The purpose of the Organisation Training Program key process area is to develop the skills and knowledge of people so they can perform their software roles effectively and efficiently[Pau97].*

5.2.1.1 Organisation Training Program in the CMM

Training is one of the most important aspects of improving a software organisation. The quality of the software engineering workforce is a direct function of the quality of software engineering training. Consequently, the SEI developed the KPA in the CMM. Furthermore, in order to continuously develop the human assets of a software organisation, the SEI also developed People CMM (P-CMM)[CHM95]. The P-CMM provides guidance on how to develop an organisation whose practices continuously improve the capability of its workforce. This effort primarily focuses on the training program.

However, both models' training efforts become focused upon the entire organisation at maturity level 3. It is curious that the topic of a training program doesn't show up until level 3. Carpenter and Hallman[CH95] pointed out that at level 3 improvement efforts are coordinated and focused at the organisational level, and are no longer a loose collection of bottom-up improvement efforts.

In order to perform the training program, Mead *et al*[MTC96] suggested that organisations should have some key practices. These practices include:

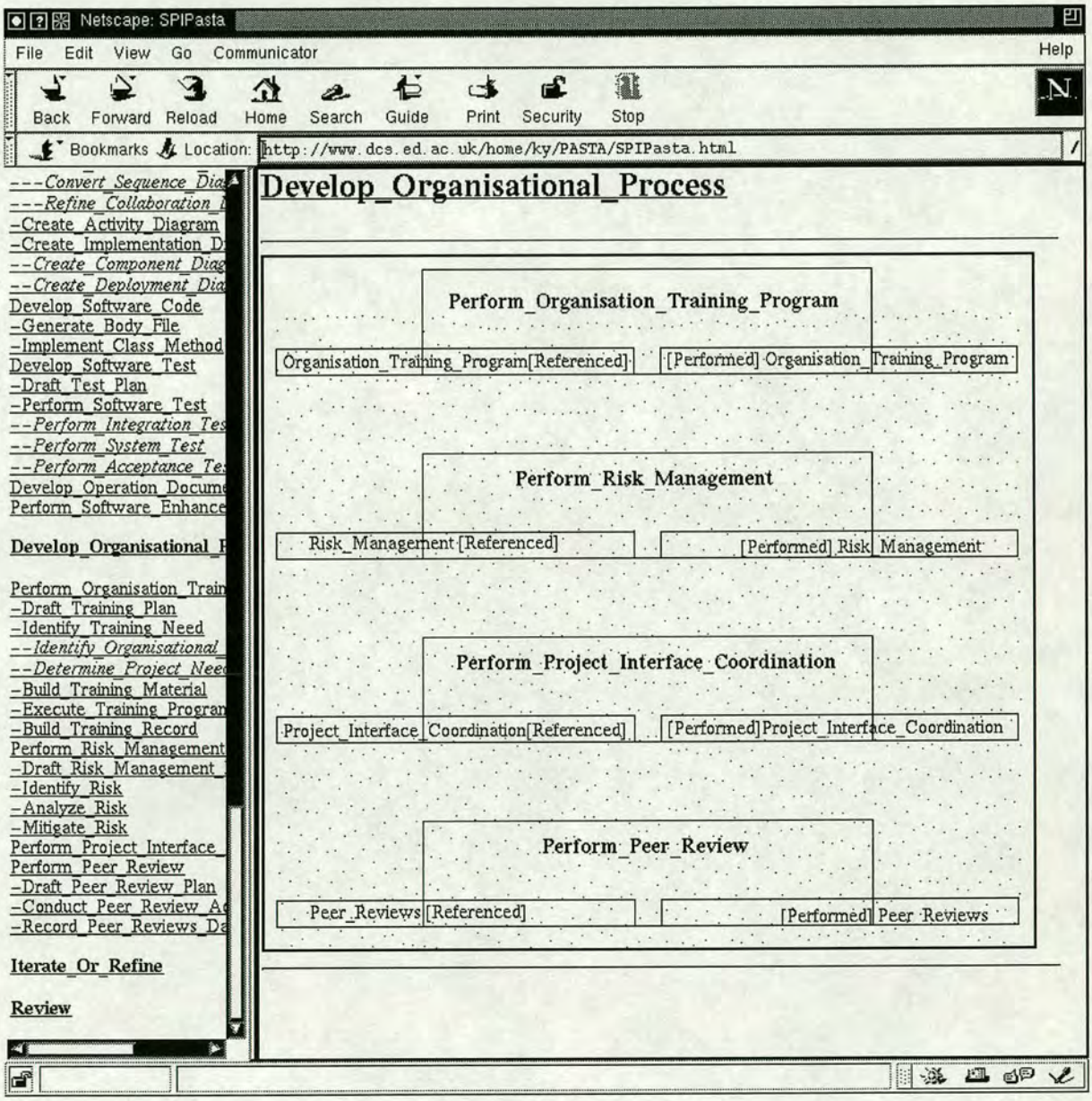

Figure 5.28: The P-State Diagram of the Organisational Process

- a defined process for software engineering education,
- a formal needs analysis activity,
- availability of a wide variety of courses from different sources, and
- training by a local, respected organisation.

Among these practices, the CMM and the P-CMM provide a guideline to develop a training process for software organisations. Moreover, the identification of training needs is primarily based on the skills needed for the organisation's set of standard software processes, as described in the Organisation Process Definition key process area. The specific training needs are identified by software projects, as described in the Integrated Software Management key process area. Furthermore, the organisations should have courses available from a wide variety of sources, such as in-house instructors, training vendors and universities. Although some experts[BCKM97, PDHT97] did not satisfy with the quality of academic software engineering education, training by a respected organisation, such as a university, is an effective practice for software organisations.

In addition to the above practices, the creation of a training plan is also an essential element within the training process of a software organisation. Without a good training plan, the training program would not be effectively performed. Carpenter and Hallman suggested the following information should be included in the training plan[CH95]:

- Scope of the Training Plan
- Responsibility for the Plan
- Training Objectives
- Technical Strengths and Weaknesses of the Software Organisation
- Software Engineering Curriculum
- Course Development and Acquisition Process
- Estimated Training Costs
- Student Selection and Enrolment Procedures
- Course Delivery Standards
- Training Evaluation and Tracking Procedures

These items cover the necessary information for the organisational training program and provide a foundation to complete the training process.

5.2.1.2 Processes in the Organisation Training Program

Figure 5.29 shows the P-state tree of the Organisation Training Program. This P-state consists of five operations: Draft Training Plan, Identify Training Need, Build Training Material, Execute Training Program and Build Training Record.

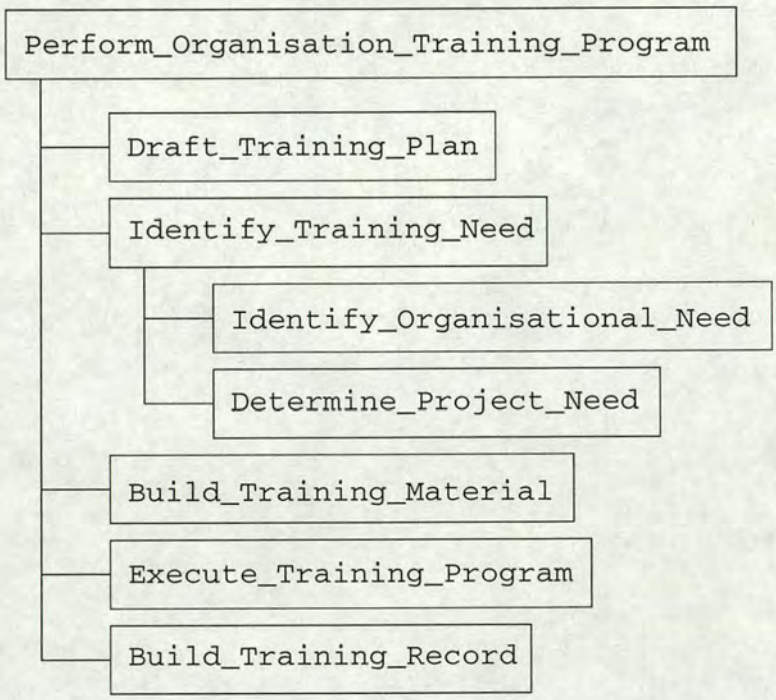

Figure 5.29: The P-State Tree of the Organisation Training Program

Main Roles: The main roles participating in the operations are project managers and the training staff who will derive the organisation training program.

Artifact List: The artifact list in this KPA is the Organisation Training Program which consists of five sub-artifacts: Training Plan, Training Need, Training Material, Training Program and Training Record. The artifacts in the Organisation Training Program are shown in Figure 5.30.

Information Artifacts: The project's defined software processes provide a guideline for performing the activities of the organisation training program.

Entrance Condition: state-of(Organisation_Training_Program) = Referenced

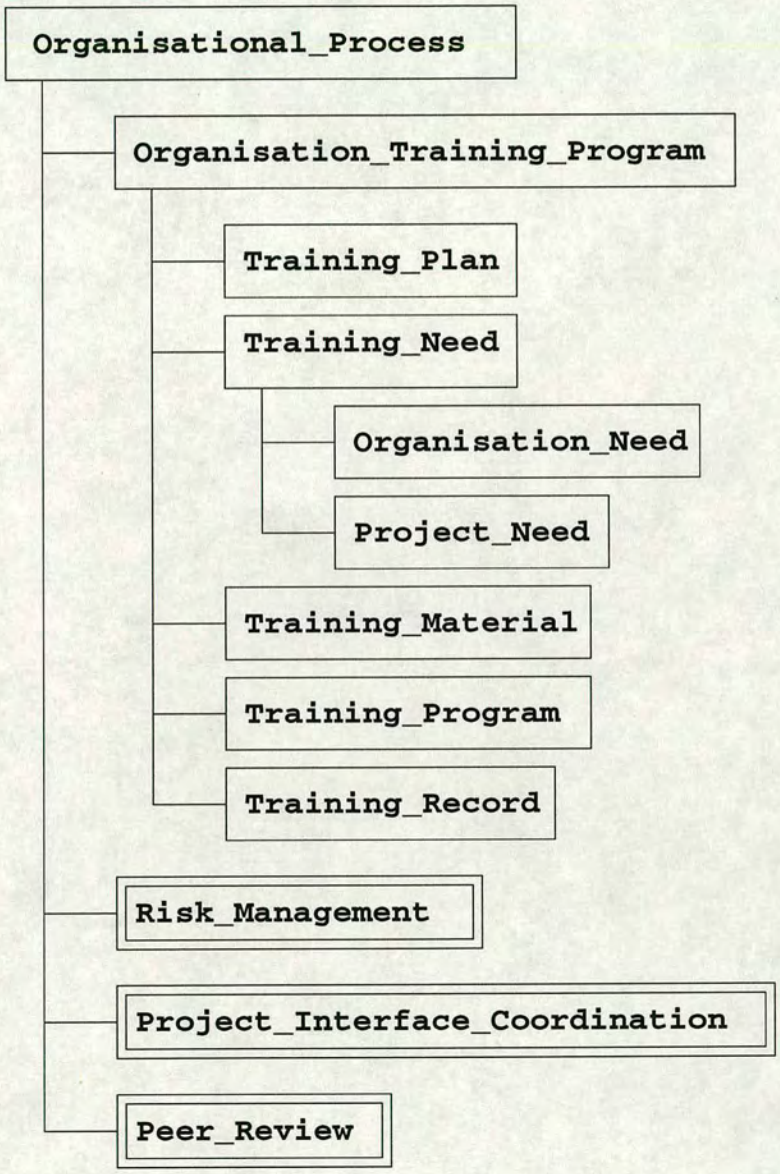

Figure 5.30: The Artifact Tree of the Organisation Training Program

Performing the organisation training program should start at early stage of the software project. From tailoring the project's defined software process to performing the software technical process, all activities need the trained people to perform them.

Activities: Five operations are presented in the P-state diagram of the Organisation Training Program as shown in Figure 5.31. First and foremost the software organisations must create the organisation training plan. An organisation training plan documents the objectives of the training program, the training need of the organisation and procedures for carrying out training activities. The training needs should then be analysed. This is the most critical part of the training activities. The training needs consist of two parts, organisational needs and project needs. The organisational training needs may contain process tailoring, software management, software engineering, and so on. Moreover, different projects might need some special training needs. The managers and training staff should identify these training needs for each software project.

After identifying training needs, the training staff may establish training materials that address the needs of the organisation. The typical training material is training courses. The training courses may have different types, formal or informal, external or internal; it depends on the organisation condition. In the next step, the managers and training staff should select the people who will receive the training, and conduct the training. Finally, the training records must be kept as a reference to assign people an appropriate job.

Exit Condition: state-of(Organisation_Training_Program) = Performed

After completing the P-states, the managers and training staff must check whether the exit condition has been reached. This means that the training program has appropriately been performed and all training records have also been kept.

5.2.2 Risk Management

5.2.2.1 Risk Management in the CMM

The SEI did not define Risk Management as a KPA in the CMM but it is currently a KPA in the Systems Engineering CMM, and the Software Acquisition CMM. The topic of risk management is primarily defined in the Integrated Software Management key process area of the CMM. However, a disciplined and systematic method of managing software development risk is necessary and feasible to control

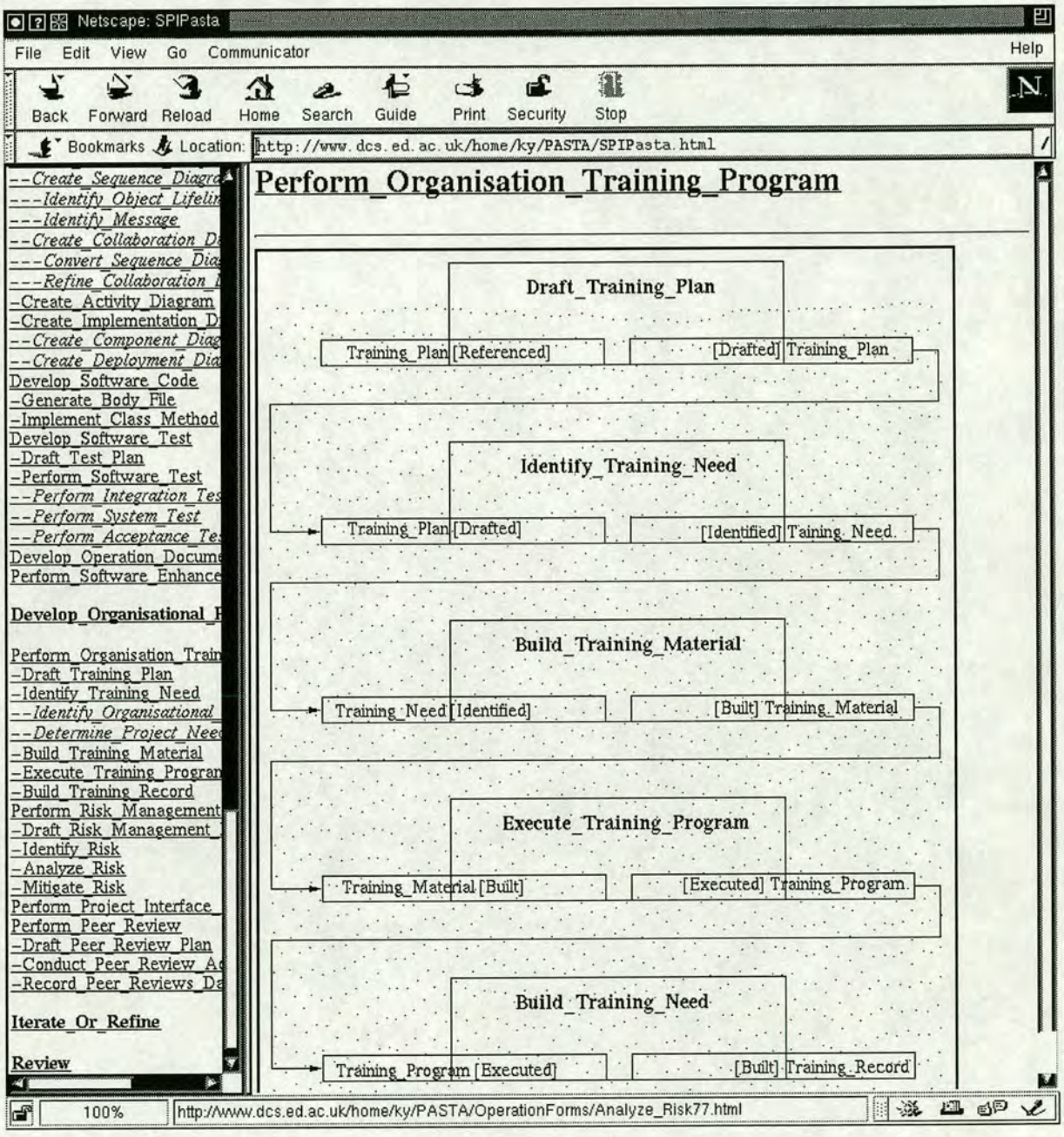

Figure 5.31: The P-State Diagram of the Organisation Training Program

the quality, cost and schedule of software products. Therefore, we separate this KPA into two parts: one is in Section 3.4 describing the project's defined software process and another is risk management.

In the last few years, the SEI has made a big effort to study risk management both of software development and software acquisition. To date, the SEI defined three groups of practices to support software risk management[HH96]:

- Software Risk Evaluation (SRE): The SRE practice is a formal method for identifying, analysing, communicating, and mitigating software technical risk. It is used by decision makers for evaluating the technical risks associated with a software-intensive program or project. The SRE has to be conducted at major milestones early and periodically in the development or acquisition life cycle. This practice consists of primary and support functions. Primary functions are Detection, Specification, Assessment, and Consolidation. Support functions are Planning and Coordination, Verification, and Training and Communication[SJ94].

- Continuous Risk Management (CRM): The CRM practice is a principle-based practice for managing project risks and opportunities throughout the lifetime of the project. These principles are composed of three groups: core, sustaining and defining. The core principle focuses on creating an open communication environment in the organisation. The sustaining principles focus on how project risk management is conducted on a daily basis. The defining principles focus on how project staff members identify risks, and the extent to which staff and management are ready to address uncertainty[HH96].

- Team Risk Management (TRM): The TRM practice defines the organisational structure and operational activities for collectively managing risks throughout all phases of the life cycle of a software-dependent development program such that all individuals within the organisations, groups, departments, and agencies directly involved in the program are participating team members. Team risk management practices bring together individuals within and between organisations to form working teams[HGD+94].

Basically, these practices are based on the risk management paradigm, which depicts the different activities involved in the management of risk associated with software development. The paradigm, being a circular form with communication at the centre, is a model of how the different elements of software risk management interact and a framework for describing how software risk management can be

implemented. These elements include identification, analysis, planning, tracking, control and communication as follows[Sco92]:

- Identification: Risk identification is the first element in the risk management paradigm. Identification highlights risks before they become problems and adversely affect a project. Without identification, risk management cannot be effectively performed. Consequently, the SEI developed a method, the Risk Taxonomy, to identify risks[CKM+93]. The taxonomy is organised into three major classes: Product Engineering, Development Environment and Program Constraints. With the taxonomy-based questionnaire, experts may follow the life cycle of software development and elicit risks potentially affecting the software product.

- Analysis: Risk analysis is the conversion of risk data into risk management information. Sometimes this step can be combined with identification. Risk analysis sifts the known risks, and places the information to allow a manager to make decisions. Therefore, providing a quantitative analysis of risks might be a good solution to analyse risks for managers.

- Planning: Risk planning develops actions to address individual risks, prioritising risk actions, and orchestrating the total risk management plan.

- Tracking: Tracking consists of monitoring the status of risks and the actions taken to improve them. Appropriate risk metrics are identified and monitored to enable the evaluation of the status of risks themselves as well as of risk mitigation plans.

- Control: Risk management should meld into program management and relies on program management processes to control the risk action plans, correct for variations from the plans, respond to triggering events, and improve the risk management process.

- Communication: Risk communication lies at the centre of the model to emphasise both its pervasiveness and its criticality. Without effective communication, no risk management approach can be viable. In order to be analysed and managed correctly, risks must be communicated to and between the appropriate organisational levels. This includes levels within the development project and organisation, within the customer organisation, and across that threshold between the developer and the customer.

5.2.2.2 Processes in Risk Management

Figure 5.32 shows the P-state tree of Risk Management. This P-state consists of four operations: Draft Risk Management Plan, Identify risk, Analyse Risk and Mitigate Risk.

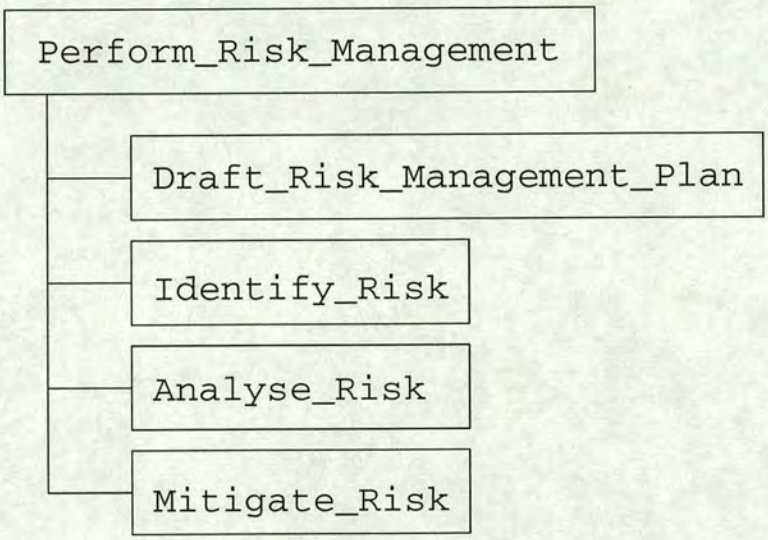

Figure 5.32: The P-State Tree of Risk Management

Main Roles: The main roles participating in the operations are project managers, software product managers and system engineers who will manage project risk.

Artifact List: The artifact list in this KPA is Risk Management which consists of two sub-artifacts: Risk Management Plan and Project Risk. The artifacts in risk management are shown in Figure 5.33.

Information Artifacts: All system development efforts have inherent risks. Managers should identify risks from the software management process and software technical process.

Entrance Condition: state-of(Risk_Management) = Referenced

When the need to manage project risks is identified, managers perform the activities of risk management. This need may be identified by the project's defined software process.

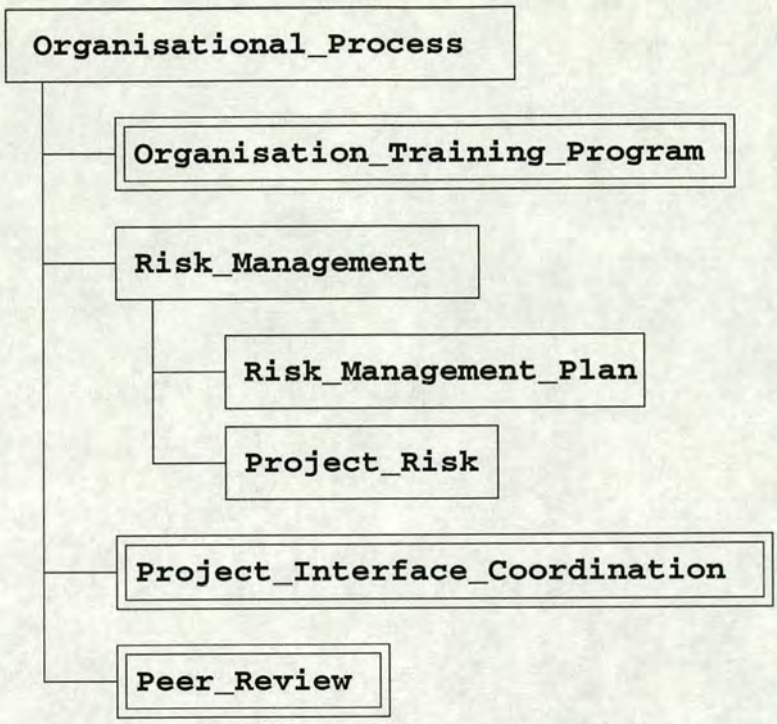

Figure 5.33: The Artifact Tree of Risk Management

Activities: Four operations are presented in the P-state diagram of Risk Management as shown in Figure 5.34. First, managers should develop a plan for activities of risk management. The plan is a basis to guide the activities managing project risks. The risk management plan can be part of the software development plan or a project risk management plan. Project risks should then be effectively identified. Since risk management covers throughout the software development life cycle, it is better to use a risk identification method. SEI's Risk Taxonomy may be a good method to be used to help identify possible problems. However, the software organisation can also define its own risk identification method. After identifying project risks, the identified risks should be documented and listed. From the list of risks, managers analyse risks and determine their priority. In accordance with the risk priority, managers should mitigate the project risks by using the documented risk mitigation strategies in the risk management plan. The activities of risk management will be performed iteratively until the project is completed.

Exit Condition: state-of(Risk_Management) = Performed

After completing the P-states, the managers must check whether the exit condition has been reached. This means that the project risks have been mitigated.

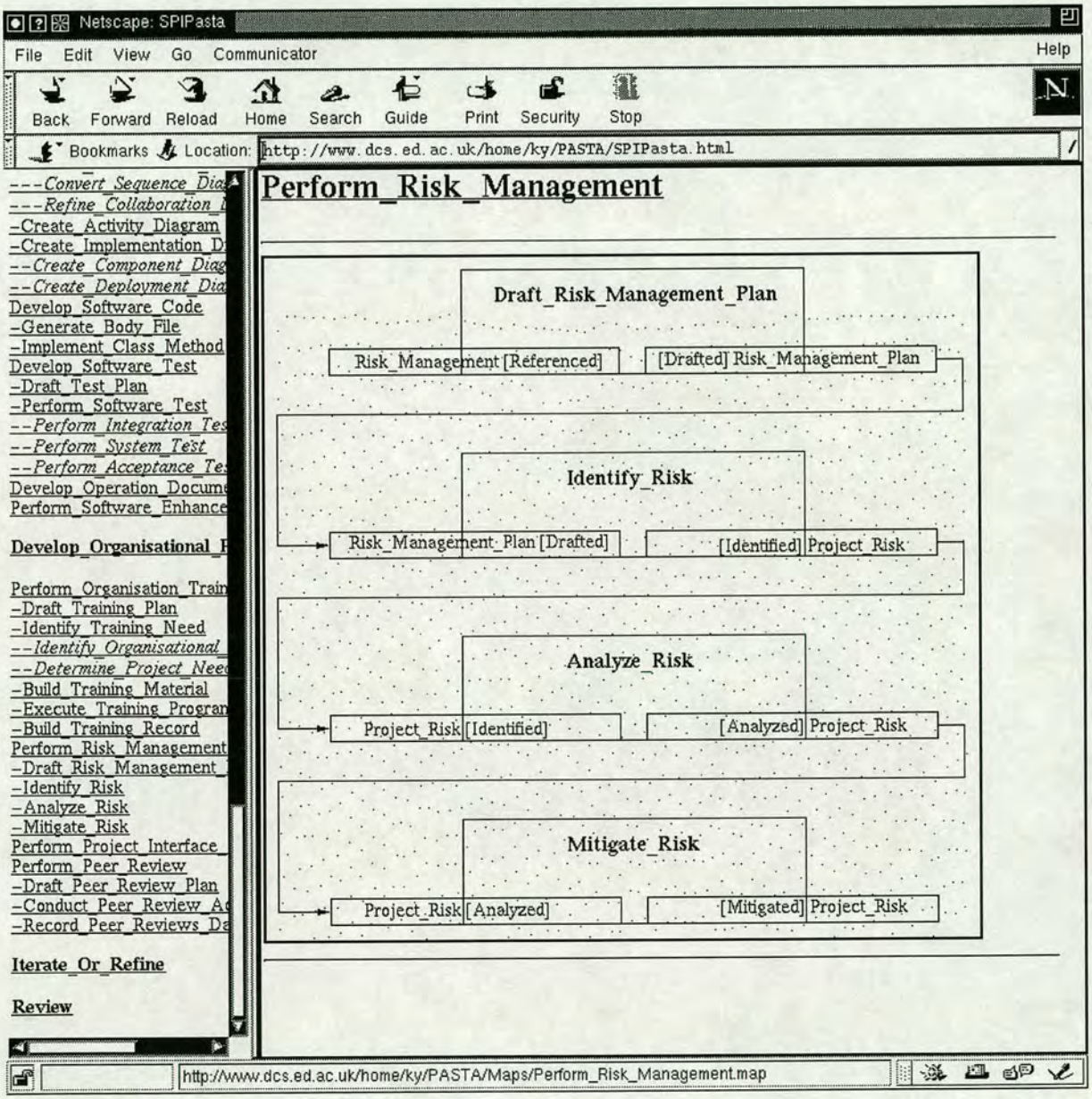

Figure 5.34: The P-State Diagram of Risk Management

5.2.3 Project Interface Coordination

The purpose of Project Interface Coordination is to ensure that software managers and staff effectively communicate, coordinate, and collaborate with other functions in the organisation to satisfy the customer's needs[Pau97].

5.2.3.1 Project Interface Coordination in the CMM

Developing a large complex software project must involve the efforts of many engineers with expertise from different areas. In order to structurally tackle the problem, as mentioned in Section 4.1.2, the software project is divided into many partitions. Different teams work concurrently on these partitions, which later on merge together to build the complete system. These teams could belong to line organisations, matrix organisations, integrated product teams, etc, since the type of organisational structure is not limited by the CMM. However, a wave of virtual enterprises is emerging. Engineers will work at different locations all over the world and complete the project together.

One of the biggest hurdles in developing a large software project is coordination of the activities of different teams. This is especially true of concurrent development. Maurer suggested that a project coordination support system should consist of four components[Mau96]:

- A project repository stores all information on the project.
- A project planning component allows users to plan and schedule activities, determines dependencies between information items, and supports resource allocation.
- A project execution component handles the worklists of the users, supports task execution, and is responsible for constraint and change management.
- A project control component supports the monitoring of the project.

Moreover, a common interface, Web browsers, should be used to integrate all components.

Clearly, the CMM covers these components and creates this KPA to coordinate the project's teams. Furthermore, the SEI is now developing an integrated product development (IPD) framework which is a systematic approach to product development that achieves a timely collaboration of necessary disciplines throughout the product life cycle to better satisfy customer needs. It

typically involves a teaming of the functional disciplines to integrate and concurrently apply all necessary processes to produce an effective and efficient product that satisfies the customer's needs.

5.2.3.2 Processes in Project Interface Coordination

Figure 5.35 shows the P-state tree of Project Interface Coordination. This P-state consists of two operations: Draft Coordination Plan and Perform Coordination Activity.

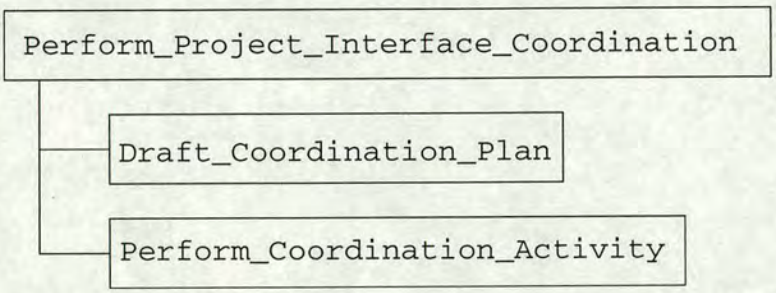

Figure 5.35: The P-State Tree of Project Interface Coordination

Main Roles: The main roles participating in the operations are the management group and development group since the activities of project interface coordination involve all of the project's staff.

Artifact List: The artifact list in this KPA is Project Interface Coordination which consists of two sub-artifacts: Coordination Plan and Coordination Activity. The artifacts in Project Interface Coordination are shown in Figure 5.36.

Information Artifacts: The activities of software development should be coordinated between the software engineering groups. The software management process and the software technical process provide necessary information to support coordination.

Entrance Condition: state-of(Project_Interface_Coordination) = Referenced

When the need to perform project interface coordination is identified, managers and staff in different groups should actively coordinate with each other. This need may be identified by the project's defined software process.

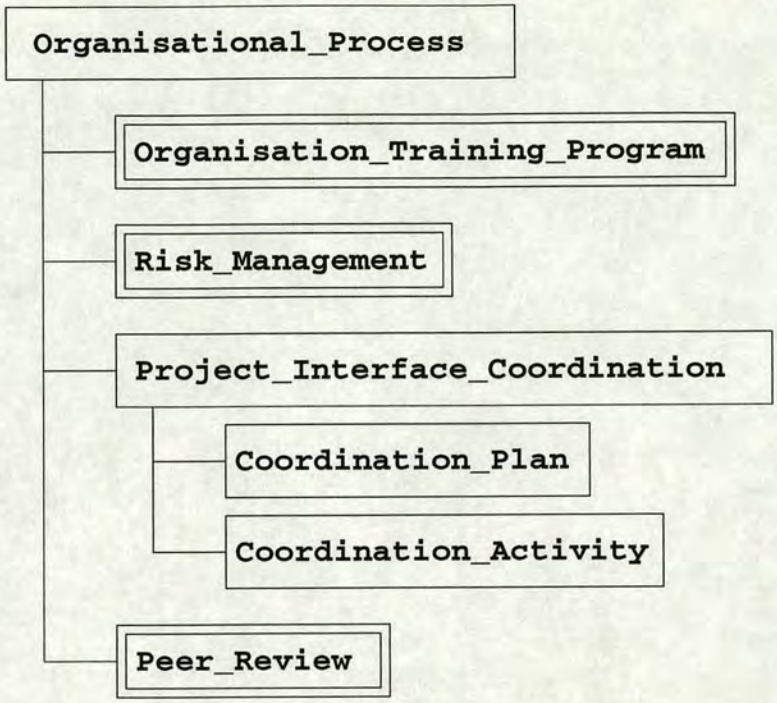

Figure 5.36: The Artifact Tree of Project Interface Coordination

Activities: Two operations are presented in the P-state diagram of Project Interface Coordination as shown in Figure 5.37. First, managers should develop a plan of activities for project interface coordination. The plan is a basis to guide the activities managing project interface coordination. In the next step, managers and software staff evolve an understanding of the customer requirements. The allocated requirements are then appropriately partitioned. All development groups should be carefully coordinated during the software management process and technical process. Each group must ensure that work products meet the needs of the receiving group and all system problems are effectively resolved.

Exit Condition: state-of(Project_Interface_Coordination) = Performed

After completing the P-states, the managers and staff must check whether the exit condition has been reached. This means that the work products are properly delivered between development groups.

5.2.4 Peer Reviews

> *The purpose of Peer Reviews is to remove defects from the software work products early and efficiently. An important corollary is to develop a better understanding of the software work products and the process that produced them so that defects can be prevented[Pau97].*

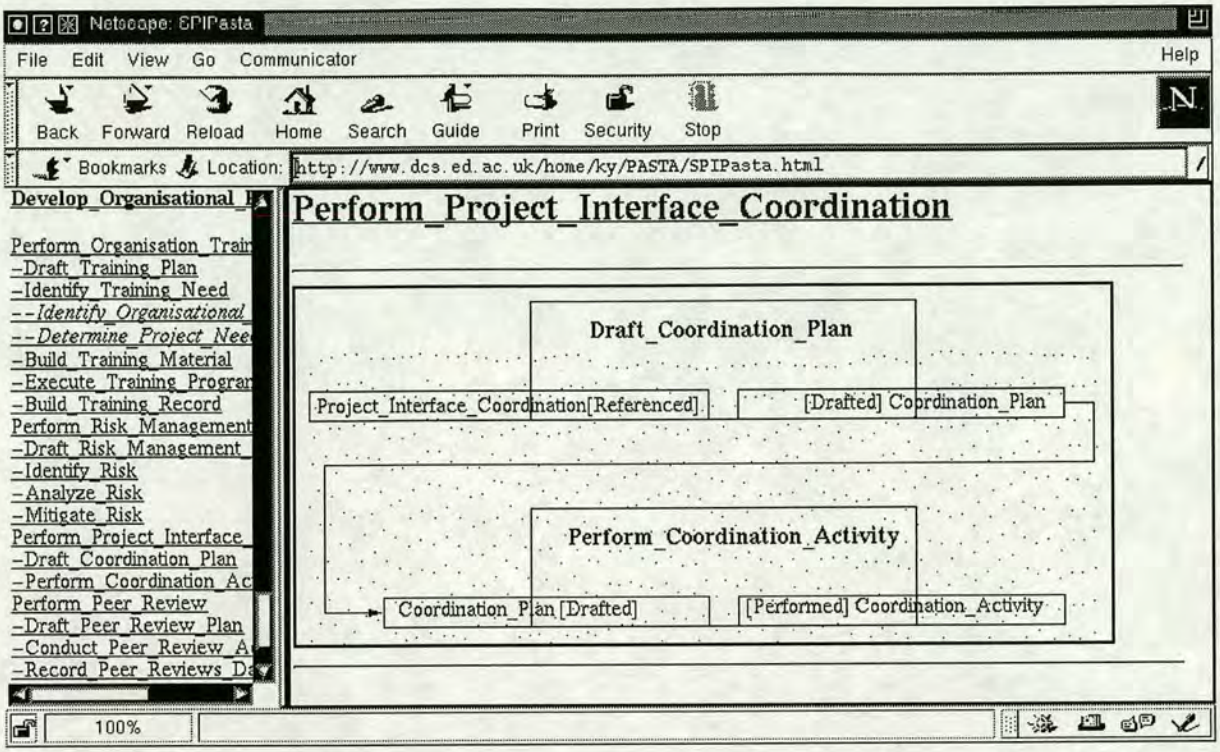

Figure 5.37: The P-State Diagram of Project Interface Coordination

5.2.4.1 Peer Reviews in the CMM

The underlying concept of peer reviews is that a small group of peers can detect more defects than the same number of people working alone. However, this key process area is quite different from the Software Project Control and the Software Quality Assurance key process areas. Software Project Control focuses on tracking software activities based on the software development plan. Software Quality Assurance concentrates on objectively reviewing the software project's activities and work products. Traditionally Software Quality Assurance is performed by an SQA group that is independent of the software project in order to keep objectivity. A risk is a potential problem. The purpose of risk management is to prevent the risk from becoming a problem or limit its impact if it does. Software testing is performed to demonstrate to the customer that the software system satisfies the customer requirements for the software project.

Defects inevitably occur through the software development life cycle and the later these defects are detected, the higher the cost of their repair. Peer reviews are used to detect the defects in software work products as early as possible. Performing peer reviews should involve a methodical examination of software work products by the producers' peers to identify defects. One of peer reviews methods is inspections. Inspections were first performed by Fagan at IBM. Fagan

defined a set of inspection process steps[Fag76]: Overview, Preparation, Inspection, Rework and Follow-up. Ebenau and Strauss[ES94] adopted this process and developed a wider inspection process for improving the quality of a variety of products. Ebenau's inspection process is defined as follows:

- **Planning:** During the planning stage, the necessary materials are collected and the inspection team is organised.
- **Overview:** A presentation explaining the material's functions and relationships should be held.
- **Preparation:** Preparation is an individual exercise performed by all the inspectors to allow them to become thoroughly familiar with the materials so that they can better find defects.
- **Inspection:** All inspectors formally examine distributed materials agreement on the inspection defect list.
- **Rework:** During rework, all defects should be revised.
- **Follow-up:** After the defects have been resolved, the inspectors follow-up to verify the defect resolution.

Basically, this key process area follows these steps to perform reviews. The specific software work products that will undergo a peer review are identified in the project's defined software process, which may include the software development plan, software estimates, requirements, design and codes. To effectively perform peer reviews, the successful completion of the peer review should be used as an exit criterion for the task associated with developing and maintaining the software work product.

5.2.4.2 Processes in Peer Reviews

Figure 5.38 shows the P-state tree of Peer Reviews. This P-state consists of three operations: Draft Peer Review Plan, Conduct Peer Review Activity and Record Peer Review Data.

Main Roles: The main roles participating in the operations are software product managers and reviewers who will conduct peer review activities.

Artifact List: The artifact list in this KPA is Peer Reviews which consists of three sub-artifacts: Peer Review Plan, Peer Review Activity and Peer Review Data. The artifacts in peer reviews are shown in Figure 5.39.

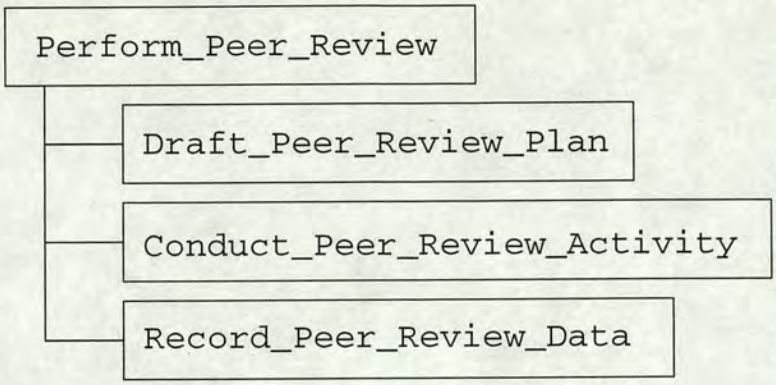

Figure 5.38: The P-State Tree of Peer Reviews

Information Artifacts: The project's defined software processes provides a guideline to perform the activities of peer reviews.

Entrance Condition: state-of(Peer_Reviews) = Referenced

When the need to perform peer reviews is identified, managers should conduct the activities of peer reviews. This need may be identified by the project's defined software process and defined in the software development plan.

Activities: Three operations are presented in the P-state diagram of Peer Reviews as shown in Figure 5.40. Firstly, managers should develop a plan for activities of peer reviews. The plan is a basis to guide the activities to perform peer reviews and is typically included in the project's software development plan. Any rework resulting from the peer reviews should be planned as part of the software development effort. This planning typically includes the specific software work products that will undergo peer review and those who will be invited to participate in the peer review of each software work product. In the next step, reviewers conduct the activities of peer reviews in accordance with the peer review plan. Reviewers should study the material to be reviewed and use the appropriate review checklists to help find defects. After identifying defects in software work products, the identified defects have to be corrected. Reviewers may conduct rereviews as necessary in order to verify the identified defects are corrected. Finally, data on the activities of peer reviews should be recorded for future reference and analysis.

Exit Condition: state-of(Peer_Reviews) = Performed

After completing the P-states, the managers must check whether the exit condition has been reached. This means that most of the defects of software work

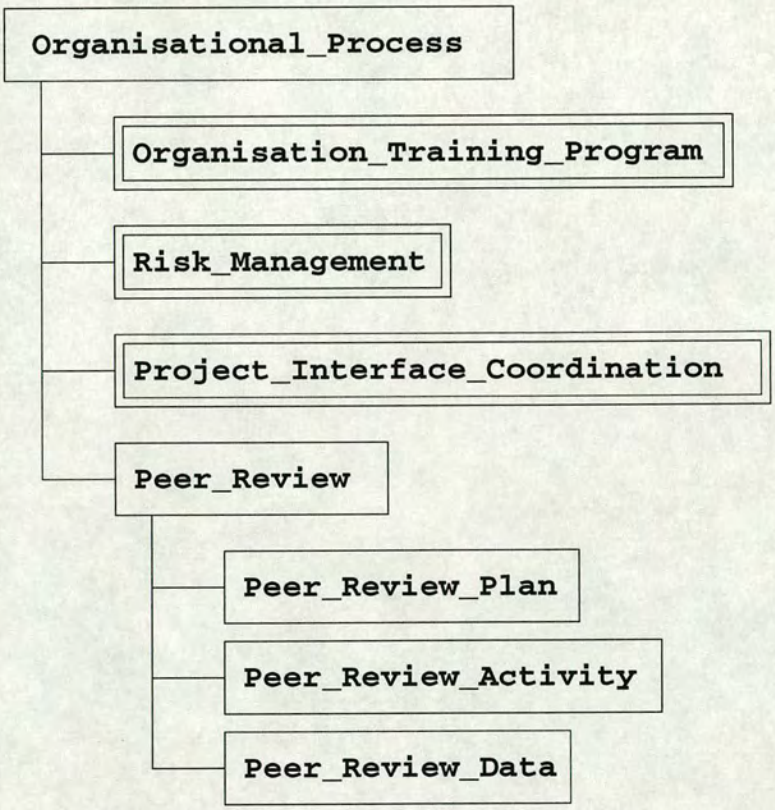

Figure 5.39: The Artifact Tree of Peer Reviews

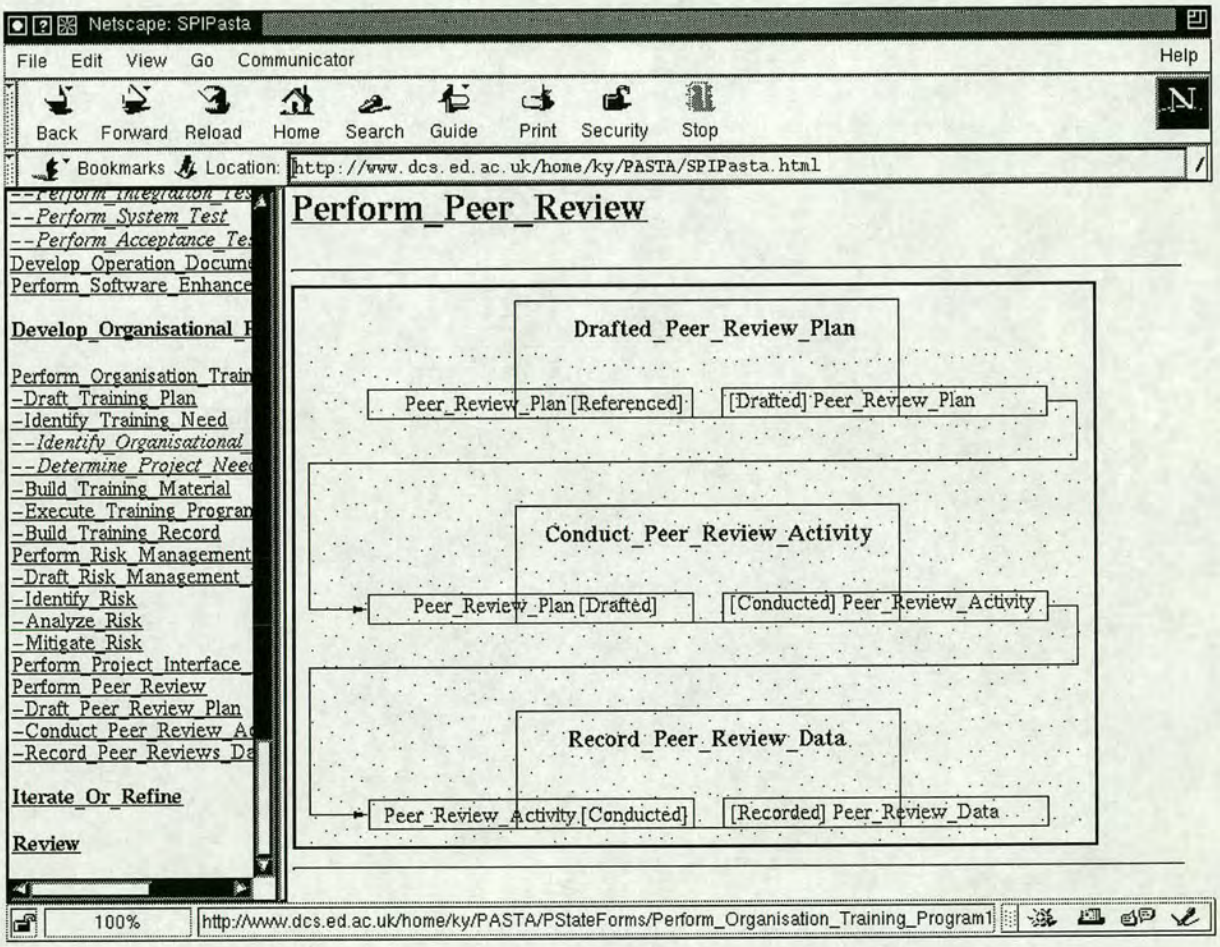

Figure 5.40: The P-State Diagram of Peer Reviews

products have been identified and removed and relevant data is appropriately recorded.

5.3 Summary

This chapter discussed how to develop software work products. We use MIL-STD-498 to comply with the CMM as software engineering processes. In the meantime, the UML is adopted at software requirement analysis and software design stages. With these standards, the software organisations exploit effective software engineering practices to develop their products. In addition, an appropriate training program is performed to ensure all staff can play their software roles. The communication between development teams is effectively conducted to share system-level requirements, objectives and issues. Furthermore, the software work products are reviewed by other team members in order to remove defects early and project's risk will be under control. Once these activities have been completed, both software engineering and management activities are stable and repeatable. Consequently, software organisations will develop their products under a common, organisation-wide understanding of the activities, roles, and responsibilities.

Chapter 6

Implementation and Assessment

SPI PASTA is an artifact-driven framework of controls for the development of software systems. It is independent of any particular set of tools and techniques and can be used with object-oriented methods, such as the UML. SPI PASTA is not a standard or a procedure. It does not provide a detailed step-by-step set of instructions describing how a particular software activity is to be carried out. However, SPI PASTA can provide insight into what artifacts should be developed at a particular time and provide guidance on what activities should be performed for the artifact. SPI PASTA consists of advice on best practice. The advice is the organisation's software process assets. It is expected that information in SPI PASTA will usually be followed by software developers. Furthermore, SPI PASTA can be sufficiently flexible to allow for the adaption of existing scenarios. For example, Objectory may be adapted in the P-state of software analysis and design. With connecting to CASE tools, the Objectory process presents experts' advice for software development.

6.1 Implementation of SPI PASTA

Widespread use of Internet/Intranet is giving an opportunity for collaborative applications that link users from a distributed environment. Using the WWW, users have an interface that can access information anywhere on the Internet/Intranet. SPI PASTA adopts the WWW as communication infrastructure making distributed projects feasible. SPI PASTA presents a unifying process model which guides the development of software projects. Without the process model, development team members have no common framework in which to interpret the activities and artifacts generated by these activities.

SPI PASTA consists of three parts, Artifacts, P-states and Roles. The artifacts specify the set of interrelated work products generated by following the P-states.

The P-states are integrated with a permissive ordering of activities, a set of tools and historical experiences to offer developers complete solutions to their software development. In addition to specifying artifacts and activities, the SPI PASTA also specifies the roles played by people involved in the software projects. As with artifacts and activities, different processes will specify different roles.

SPI PASTA defines a strategy for defining what the artifact will be produced for a given project. It provides a baseline for communication between and across team members, and between differing levels of management. On the technical side, SPI PASTA provides guidance on how to decide what sort of artifact it is necessary to control. From the management point of view, the project manager should firstly organise the development team. Then he/she points out the tasks and responsibility to all team members by using SPI PASTA.

Figure 6.1 shows the homepage of SPI PASTA that is split into two frames. Role tree, artifact tree and P-state tree are listed on the left-hand side. Users can extend any tree and see the relationship between items. Each item is linked to its definition form. The action frame, located on the right-hand side, provides an area for definition forms and diagrams.

The following example represents related activities to develop the software management process. Five P-states (as Figure 6.2) are listed on the P-state diagram of Develop_Software_Management_Process. There is a relationship between Derive_Software_Development_Plan and Perform_Software_Project_Control. This means Software Project Control must be performed after completing the software development plan. There is no relationship among other P-states. SPI PASTA permits a parallel process since the model provides processes for different development team members concurrently and independently. Therefore, system requirements, the software development plan and acquisition management might be developed concurrently. Each rectangle consists of the sub-P-states, entrance conditions and exit conditions. The entrance condition decides which P-state can be performed. Such as Derive_Software_Development_Plan, once two entrance conditions have been satisfied, users may go down to next layer, the P-state diagram of Derive_Software_Development_Plan. Before entering to next step, Users may click the entrance condition to fetch the A-state diagram (as Figure 6.3. The operation between two states can be connected to operation definition form as a guide in completing the software artifacts.

Furthermore, users may check the details presented at P-state definition form. P-state tree can link to the definition form, moreover the title of P-state diagram may also connect to the form. From the definition form, users should collect

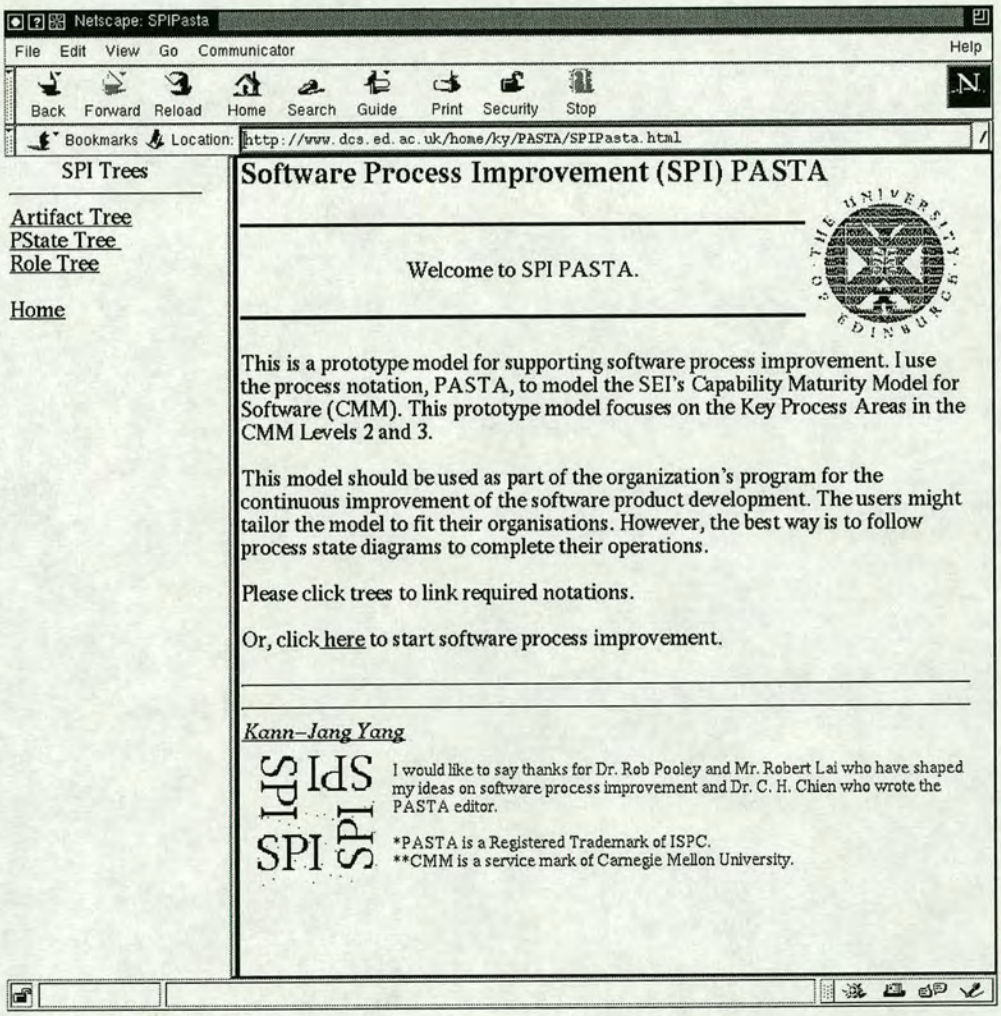

Figure 6.1: The Homepage of SPI PASTA

related information, such as what is the purpose of the P-state, who will perform these activities, what are the artifacts operated on for this P-state, the condition required for entry into this P-state and so on. In this example, the project's defined software process should be in place. When the need to derive the software development plan has been identified, users may perform next steps.

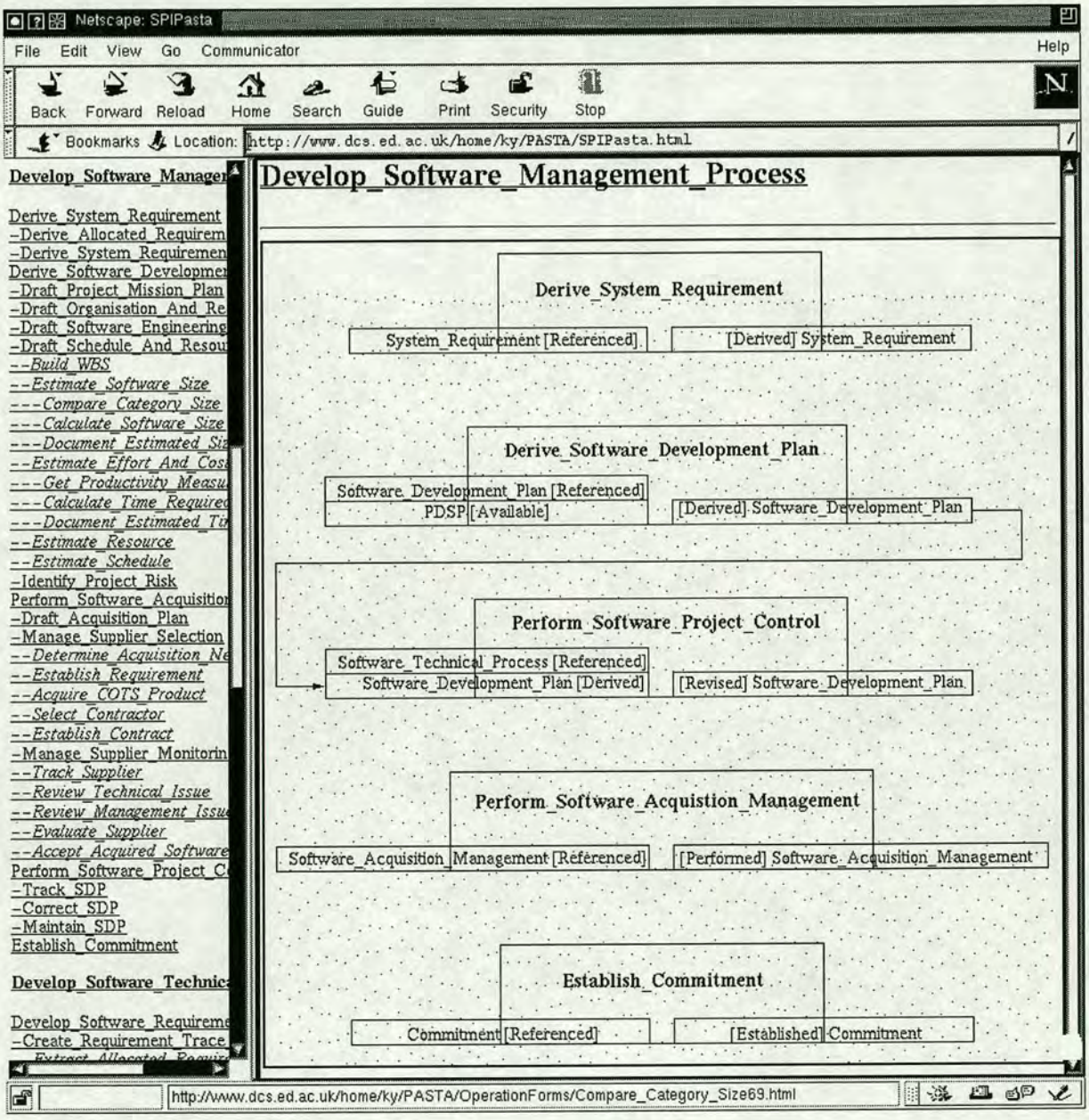

Figure 6.2: The P-State Diagram of Software Management Process

Users may click the rectangle of Derive_Software_Development_Plan to enter the sub-P-state (as Figure 6.4). Five operations stand on the P-state diagram and four of them are connected together. There is a sequence among them. Drafting the project mission plan defines the system overview. Drafting the organisation

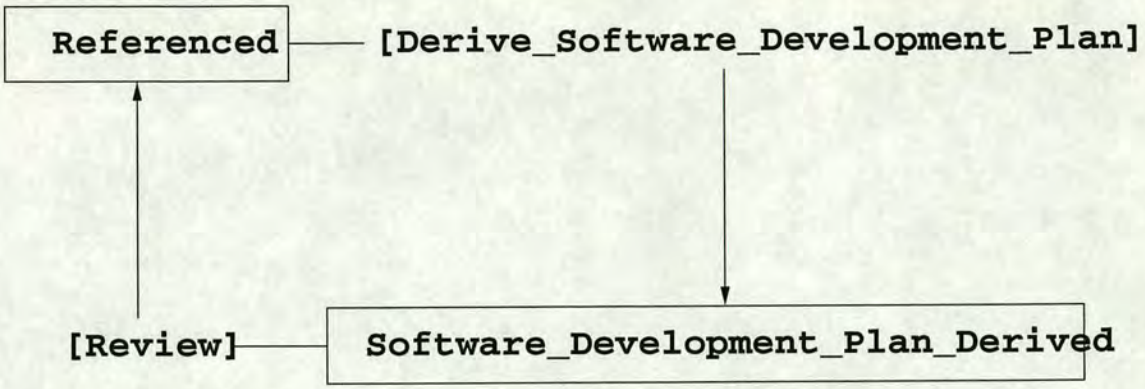

Figure 6.3: The A-State Diagram of Software Development Plan

and responsibility plan describes the organisational structure to be used on the project, and the authority and responsibility of each organisation for carrying out required activities. Drafting the software engineering activities defines the software development process to be used. Drafting the schedule and resource plan identifies the activity procedure and the resources to be applied to the project. The first three activities are operations that will be connected to the definition form. The definition form defines required information for users. The users may rely on the procedures to perform activities to complete the artifact. If necessary, the users can link to the artifact definition form to get required information about the artifact and even the sample unified by the organisation.

To draft the schedule and resource plan is a crucial task for software development. In the beginning of the project, the allocated requirements are not clear enough. The analysts have to build the work breakdown structure to estimate the software size in order to estimate the required effort and cost. Finally, the resources and schedule are defined and allocated. We have already defined operations for each activity. Users can find out the operation definition form and related artifact to perform drafting the schedule and resource plan.

Then, users should identify the project risk. This activity might perform anytime during deriving the software development plan.

Finally, all activities will be terminated until the exit condition of Derive_Software_Development_Plan has been satisfied. The exit condition of the primary P-states will be linked to the goals of related KPA. This is because the assessment of the CMM depends on the goals and KPAs. Once the related goals are reached, the exit condition of the P-state is satisfied.

Since SPI PASTA adopts MIL-STD-498 as a standard for the activities of software product engineering, we appropriately link these activities and data item descriptions to the P-state forms (See Table 6.1). Users may follow these activities

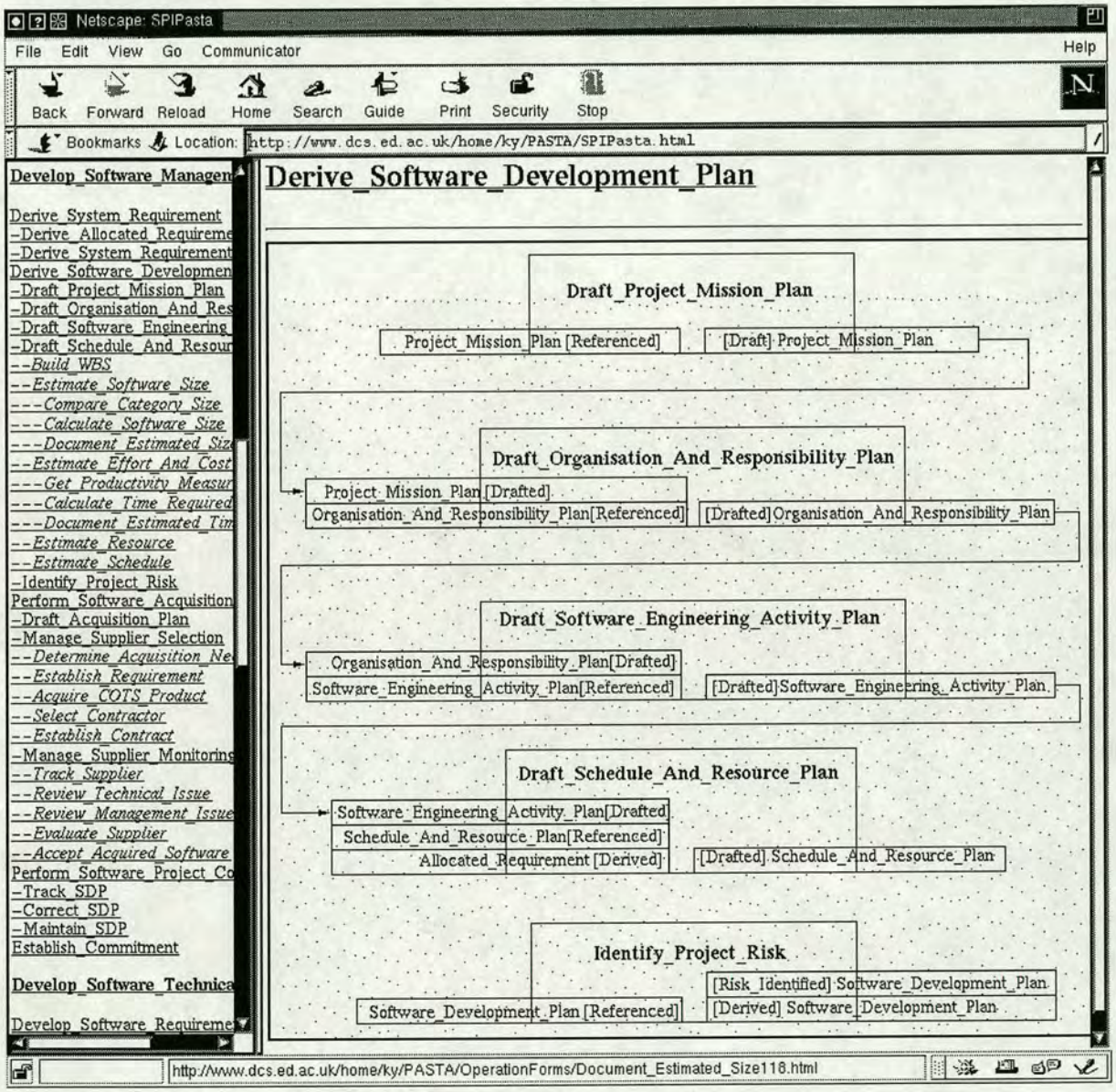

Figure 6.4: The P-State Diagram of Software Development Plan

Activities in MIL-STD-498	P-State Forms in SPI PASTA
5.1 Project planning and oversight	Derive Software Development Plan
5.2 Establishing a software development environment	Derive Software Development Plan
5.3 System requirements analysis	Derive System Requirement
5.4 System design	Derive System Requirement
5.5 Software requirements analysis	Develop Software Requirement
5.6 Software design	Develop Software Design
5.7 Software implementation and unit testing	Develop Software Code
5.8 Unit integration and testing	Perform Integration Test
5.9 CSCI qualification testing	Perform Integration Test
5.10 CSCI/HWCI integration and testing	Perform Integration Test
5.11 System qualification testing	Perform System Test
5.12 Preparing for software use	Develop Operation Documentation
5.13 Preparing for software transition	Develop Operation Documentation
5.14 Software configuration management	Perform Configuration Management
5.15 Software product evaluation	Perform Acceptance Test
5.16 Software quality assurance	Perform Software Quality Assurance
5.17 Corrective action	Perform Peer Reviews
5.18 Joint technical and management reviews	Perform Peer Reviews

Table 6.1: Mapping the Activities of MIL-STD-498 to P-State Forms of SPI PASTA

and link the relevant data item description. When these data item descriptions are completed, they will be viewed as the artifact for assessing the maturity level.

6.2 Assessment of SPI PASTA

6.2.1 The CMM Appraisal Framework

Assessment is one of the most essential issues in the CMM. Without assessing the defined software process, we do not know whether it will go to the right destination. In order to assess the software process performed in SPI PASTA, we must firstly study the appraisal method in the CMM.

The CMM appraisal framework (CAF), developed by SEI, is a framework for developing, defining, and using appraisal methods based on the CMM. The CAF provides a framework for rating the process maturity of an organisation against a generally accepted reference model through the use of an appraisal method. However, the CAF is not an appraisal method and does not directly assess the software process performed in the software organisation. The CAF identifies the requirements and desired characteristics of a CMM-based appraisal method in

order to improve consistency and reliability of methods and their results. Together the CMM and the CAF describe "what" must be accomplished by CAF compliant appraisal methods. The appraisal methods themselves detail "how" to transform an organisation's software process data into information of value to meeting an organisation's business needs[MB95].

Currently, the SEI has published two CAF-compliant methods: software process assessment and software capability evaluation.

- **Software process assessments** are used to determine the state of an organisation's current software process, to determine the high-priority software process-related issues facing an organisation, and to obtain the organisational support for software process improvement. The CMM-Based Appraisal for Internal Process Improvement (CBA IPI), developed by the SEI, is intended to be a diagnostic tool that enables an organisation to gain insight into its software development capability by identifying strengths and weaknesses of its current processes, to relate these strengths and weaknesses to the CMM, to prioritise software improvement plans, and to focus on software improvements that are most beneficial, given its current level of maturity and the business goals[DM96].

- **Software capability evaluations (SCE)** are used to gain insight into the software process capability of a supplier organisation and is intended to help decision makers make better acquisition decisions, improve subcontractor performance, and provide insight to a purchasing organisation. SCE version 3.0, published by the SEI, provides a CAF-compliant method for evaluating the software process of an organisation[BP96].

The basic difference between an assessment and an evaluation is that an assessment is an appraisal that an organisation does to and for itself, and an evaluation is an appraisal where an external group comes into an organisation and looks at the organisation's process capability in order to make a decision regarding future business[DM96].

The concept of building SPI PASTA is software process improvement. As a result, to assess SPI PASTA is an activity which gains insight into its software development capability by identifying strengths and weaknesses of its processes. With the CBA IPI, the status of each process in SPI PASTA will be assessed. SEPG and managers can recognise the strengths and weaknesses of the process and build an action plan for the process improvement program.

Before discussing software process assessment, we need to know the rating system in the CAF. Fundamentally, the assessment is rated by relying on the CMM structure (as Figure 6.5). Each maturity level is decomposed into several KPAs that indicate the areas an organisation should focus on to improve its software process. Each KPA identifies a cluster of related activities that achieve a set of goals considered important for enhancing process capability. Therefore, satisfaction of a key process area depends on satisfaction of the goals. This satisfaction can involve implementation of the key practices that map to that goal or implementation of an alternative set of practices that achieve the goal. When the assessment team members examine a specific key process area in SPI PASTA, all of the goals for the specific KPA must be satisfied in order for the KPA to be satisfied.

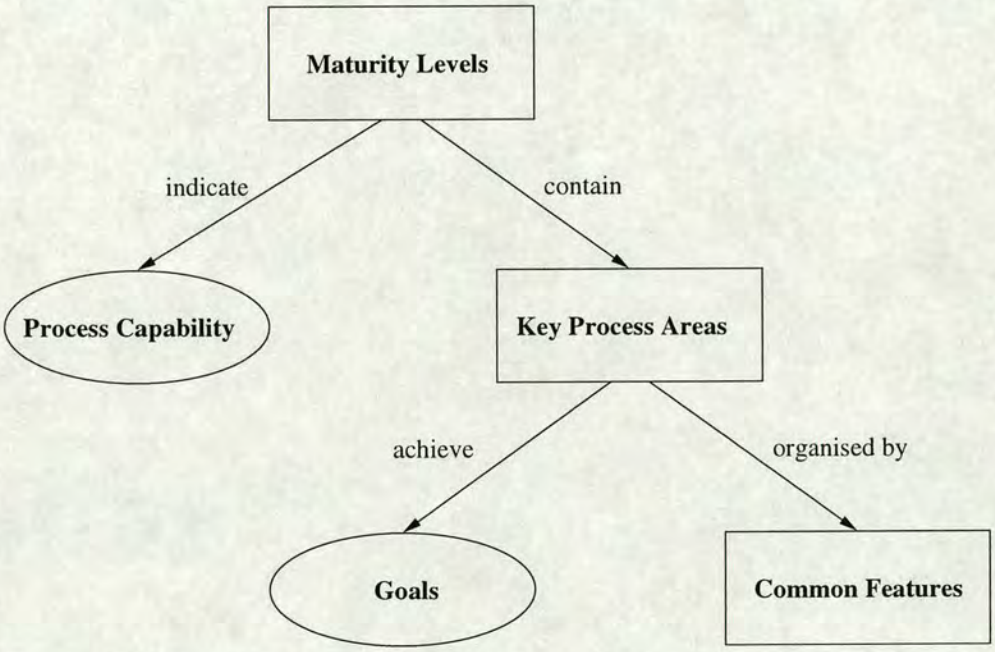

Figure 6.5: The CMM Structure

It is not necessary to assess all KPAs in SPI PASTA, since some KPAs, such as Software Acquisition Management, are not applicable in the software project. Therefore, the CAF defines rating values as following[MB95]:

- A KPA or goal is **satisfied** if this aspect of the CMM is implemented and institutionalised either as defined in the CMM, or with an adequate alternative.
- A KPA or goal is **unsatisfied** if there are significant weaknesses in the

appraised entity's implementation or institutionalisation of this aspect of the CMM, as defined, and no adequate alternative is in place.

- A KPA or goal is **not applicable** if the KPA is not applicable in the organisation's environment.

- A KPA or goal is **not rated** if the associated appraisal findings do not meet coverage criteria or if this aspect of the CMM falls outside the scope of the appraisal.

In this rating system, in addition to not applicable and not rate, KPAs and goals only have two choices, satisfied or unsatisfied. Kitson[Kit96] described that the CMM is a staged model. Overall process capability of the organisational unit assessed is a roll-up of individual KPA ratings. If one goal of the KPA is unsatisfied, this KPA will be unsatisfied and the organisational unit assessed will fail to reach this level's capability.

If we contrast ISO 15504, Kitson[Kit96] described it a continuous model. In ISO 15504, process capability is measured on a process-by-process basis. The approach to rating used in the ISO 15504 product set is use a four-point ordinal rating scale of adequacy for each process. This scale, different to the CMM's, includes Not adequate (N), Partially adequate (P), Largely adequate (L) and Fully adequate (F). An actual process capability level rating shall be determined for each process instance assessed by aggregating the generic practice adequacy ratings within each capability level. For each process instance, the actual process capability level ratings shall describes, for each capability level, the proportion of generic practices that were rated at each point on the generic practice adequacy scale in a clear and unambiguous way[SPI95]. Consequently, the process capability level rating can be represented as the following vector:

[% Fully, % Largely, % Partially, % Not Adequate]

The concept of ISO 15504's rating system is more flexible than CAF's. It provides a range idea to show how well the process is rather than just "on or off". For example, to complete the goal 2 in the Software Project Planning key process area, "Estimates of the software project's planning parameters are established and maintained," five key practices, AC.06, AC.07, AC.08, AC.10, AC.15 (Activities performed), must be satisfied. The appraisal team might assess the goal 2 for X project and make a rating such as AC.06 and AC.07 are largely adequate, AC.08 and AC.10 are partially adequate, AC.15 is fully adequate and none of key practices are not adequate. As a result, one of key practices is fully adequate -

20%, two of them are largely adequate - 40%, two of them are partially adequate - 40% and none of them are not adequacy. The adequacy rating of goal 2 can be represented as a vector in the form:

PP.GO.02 = [20%, 40%, 40%, 0]

In some circumstances it can even assign a weighting to the four points on the adequacy scale, for example 100% for fully adequacy, 75% for largely adequacy, 25% for partially adequacy and 0% for not adequacy. Any derived rating may be represented as a single value rather than as a vector. Therefore, the above vector should be calculated as following:

*20% * 100% + 40% * 75% + 40% * 25% + 0 * 0 = 60%*

This value might give the project manager a clear vision to show the project's condition. Furthermore, the appraisal team may point out the strengths and weaknesses of each key practice. The project manager can make a correction for the project in accordance with the assessment results.

In SPI PASTA, we adopted the SEI's rating system. However, users may change this rating system to ISO's if they prefer value presentation.

6.2.2 Software Process Assessment

An assessment method should contain three phases of appraisal execution. The first phase includes the activities necessary to plan and prepare for the assessment. The second phase consists of on-site activities for conducting assessment. The final phase is to report the results. All activities performed for assessment are shown in Figure 6.6.

6.2.2.1 Plan and Prepare for Assessment

The first phase, planning and preparation for assessment, is the key to success of assessment. As shown in Figure 6.6, this phase consists of analysing requirements, selecting and preparing the assessment team, selecting and preparing participants and developing the assessment plan.

Analyse Requirements Analysing the requirements for a particular assessment includes development of assessment goals, constraints and scope. The scope of assessment consists of the CMM scope and organisational scope. To assess a project, it may include one or more KPAs within the CMM. Then the appraised entity should be defined. An appraised entity might be any portion of an organisation, such as an organisation unit, a specific project and so on.

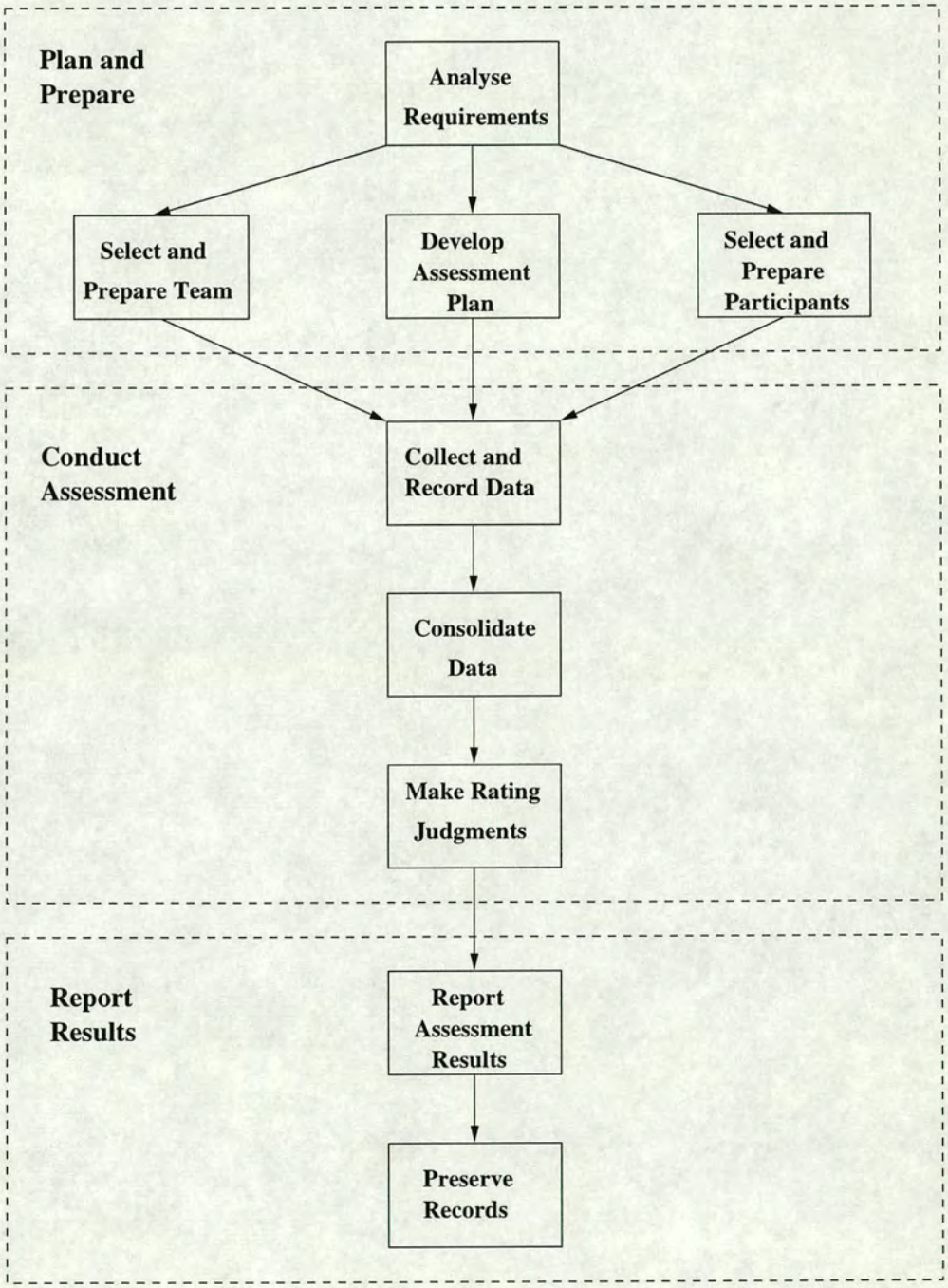

Figure 6.6: Activity Diagram for Assessment

Select and Prepare Team This activity includes selection and preparation of the assessment team. It contains identifying the assessment team leader, selecting each of the team members, and providing the team with training and orientation needed to prepare for the assessment.

Select and Prepare Participants Participants are those who provide the assessment team with data concerning the appraised entity's software process. The participants should be appropriately oriented in the assessment process to comply with the assessment team.

Develop Assessment Plan In the end of Phase one, an assessment plan should be developed. It will guide and define execution of the assessment. The assessment plan should include information on the following item[MB95]:

- Identifies the assessment goals.
- Identifies the assessment scope.
- Identifies the assessment activities.
- Provides a schedule for the activities.
- Identifies the people, resources and budget required to perform the activities.
- Identifies the assessment outputs and their anticipated use.
- Identifies anticipated follow-on activities.
- Documents any planned tailoring of the assessment method and associated trade-offs.
- Identifies risks associated with assessment execution.

6.2.2.2 Conduct Assessment

The second phase, conducting the assessment, shown in Figure 6.6, consists of collecting and recording information on the software process of the project, consolidating information collected, and making rating judgements.

Collect and Record Data Information about the organisation's software processes can be collected from four categories: instruments, presentations, interviews and documents. Data collection using instruments includes such activities as administering questionnaires and surveys and gathering their responses. Data collection using presentations can involve presentations by the assessment team or the assessment participants that include interaction between the two. Data collection through interviews involves assessment team members asking questions and engaging in discussions with assessment participants and recording their responses. Data collection using documents involves reviewing a lasting representation of information. Documents may exist in various hardcopy or electronic forms.

Consolidate Data Consolidation is the decision making activity in the iterative information gathering and decision making process. During consolidation, the assessment team organises the information obtained from data gathering sessions and combines it into a manageable summary of data. Then the assessment team validates the information to ensure that they accurately reflect the practices of the appraised entity.

Make Rating Judgements Goals and KPAs are two components of the CMM that can be rated. Maturity level ratings depend exclusively on the ratings of KPAs and KPA ratings depend on the ratings of the goals. The assessment team must come to consensus on the ratings which it provides to an appraised entity. Without this procedure, the rating could not be counted as a valid one.

6.2.2.3 Report Results

The final phase, Reporting results, shown in Figure 6.6, consists of reporting assessment results and preserving records.

Report Assessment Results Strengths and weaknesses of appraised entity are presented for each KPA within the assessment scope. These results will guide the areas in which an organisation focuses its software process improvement efforts.

Preserve Records Assessment records should be preserved for conducting a subsequent assessment.

6.2.3 Assessing SPI PASTA

Table 6.2 and 6.3[1] show that the key practices map to goals of KPAs at Level 2 and 3. Both tables provide a foundation to assess the development processes by assessment team. All goals within the assessment scope must be rated. Since each goal has its own key practices, to satisfy a goal, its key practices should be completely performed.

In SPI PASTA, we have allocated related key practices to the operation definition forms. For example, the goals of the Software Project Planning key process area are linked to the P-state diagram of Derive Software Development Plan. Users check the exit condition with call goals of the KPA (as Figure 6.7). Then they can concentrate on one goal by calling all related key practices (as Figure 6.8). Each key practice has its own selection menu. This selection menu has been defined to four grades, Satisfied, Unsatisfied, Not Applicable and Not Rated. Users can select the appropriate value for each key practice and click the "Assess" button, then data will be collected by assigned person (such as the project manager). The development team may use the data to check whether the goal is satisfied or how good key practices are performed.

Assessing SPI PASTA is not necessary to organise an assessment team. It will depend on what the purpose is. For the purpose of software process improvement inside organisation, the development team may perform assessment for its own software project. The project manager may assign one or two members as assessment team to trace all key practices of project's defined software process. This team can follow the assessment procedures to record strengths and weaknesses of the software process. This will be the basis to improve SPI PASTA.

6.2.4 Post Action

We would not say that SPI PASTA can well fit any type of software project. After all, a real-time system is quite different from a business information system. SPI PASTA need to be tailored in order to comply with the features of software projects. However, it does not mean that applying a tailored process can develop a good software product. Process modification can take place at any time during the software development period. It could be viewed as part of a process improvement effort. As mentioned in assessing SPI PASTA, users assess the key practices in order to decide project's maturity. In the meantime, they will comment the strength and weakness of the project, such as why a P-state or an operation

[1]CO: Commitment to perform, AB: Ability to perform, AC: Activities performed, ME: Measurement and analysis, VE: Verifying implementation

KPAs	Goal	Key Practices
Requirements Management	1	CO.01, AB.01, AB.02, AB.03, AB.04, AC.01, ME.01, VE.01, VE.02, VE.03, VE.04
	2	AC.02, AC.03
	3	AC.04
Software Project Planning	1	CO.01, AB.01, AB.02, AB.03, AB.04, AC.01, ME.01, VE.01, VE.02, VE.03, VE.04
	2	AC.06, AC.07, AC.08, AC.10, AC.15
	3	AC.12, AC.13
	4	AC.02, AC.03, AC.04, AC.05, AC.09, AC.11, AC.14
Software Project Control	1	CO.01, AB.01, AB.02, AB.03, AB.04, AC.01, ME.01, VE.01, VE.02, VE.03, VE.04
	2	AC.03, AC.04, AC.05, AC.07, AC.13
	3	AC.02, AC.06, AC.08, AC.09, AC.10
	4	AC.11, AC.12
Software Acquisition Management	1	CO.01, AB.01, AB.02, AB.03, AB.04, AC.01, ME.01, VE.01, VE.02, VE.03, VE.04
	2	AC.02, AC.04, AC.05
	3	AC.03, AC.06
	4	AC.07, AC.08, AC.09, AC.10, AC.11, AC.12
Software Quality Assurance	1	CO.01, AB.01, AB.02, AB.03, AB.04, AC.01, ME.01, VE.01, VE.02, VE.03, VE.04
	2	AC.02, AC.04
	3	AC.03, AC.04
	4	AC.04, AC.05
Software Configuration Management	1	CO.01, AB.01, AB.02, AB.03, AB.04, AC.01, ME.01, VE.01, VE.02, VE.03, VE.04
	2	AC.03, AC.09
	3	AC.02, AC.06, AC.07, AC.08, AC.09
	4	AC.04, AC.05, AC.07, AC.09

Table 6.2: Mapping the Key Practices to Goals at Level 2

KPAs	Goal	Key Practices
Organisation Process Focus	1	CO.01, CO.02, AB.01, AB.02, AB.03, AB.04, AC.01, ME.01, VE.01, VE.02, VE.03,
	2	AC.02
	3	AC.03, AC.04, AC.05, AC.06
Organisation Process Definition	1	CO.01, CO.02, AB.01, AB.02, AB.03, AB.04, AC.01, ME.01, VE.01, VE.02, VE.03,
	2	AC.02
	3	AC.03, AC.04, AC.05, AC.06
Organisation Training Program	1	CO.01, CO.02, AB.01, AB.02, AB.03, AB.04, AC.01, ME.01, VE.01, VE.02, VE.03,
	2	AC.02, AC.04, AC.05, AC.06
	3	AC.03, AC.05
Integrated Software Management	1	CO.01, AB.01, AB.02, AB.03, AB.04, AC.01, ME.01, VE.01, VE.02, VE.03, VE.04
	2	AC.02, AC.03
	3	AC.04, AC.05, AC.08
	4	AC.06, AC.07
Software Product Engineering	1	CO.01, AB.01, AB.02, AB.03, AB.04, AC.01, ME.01, VE.01, VE.02, VE.03, VE.04
	2	AC.02, AC.03, AC.04, AC.05, AC.09
	3	AC.02.AC.06, AC.07, AC.08
	4	AC.09, AC.10.AC.11, AC.12
Project Interface Coordination	1	CO.01, AB.01, AB.02, AB.03, AB.04, AC.01, ME.01, VE.01, VE.02, VE.03, VE.04
	2	AC.02, AC.03, AC.04, AC.05
	3	AC.03, AC.06, AC.07
Peer Reviews	1	CO.01, AB.01, AB.02, AB.03, AB.04, AC.01, ME.01, VE.01, VE.02, VE.03, VE.04
	2	AC.02, AC.03, AC.05
	3	AC.04, AC.05

Table 6.3: Mapping the Key Practices to Goals at Level 3

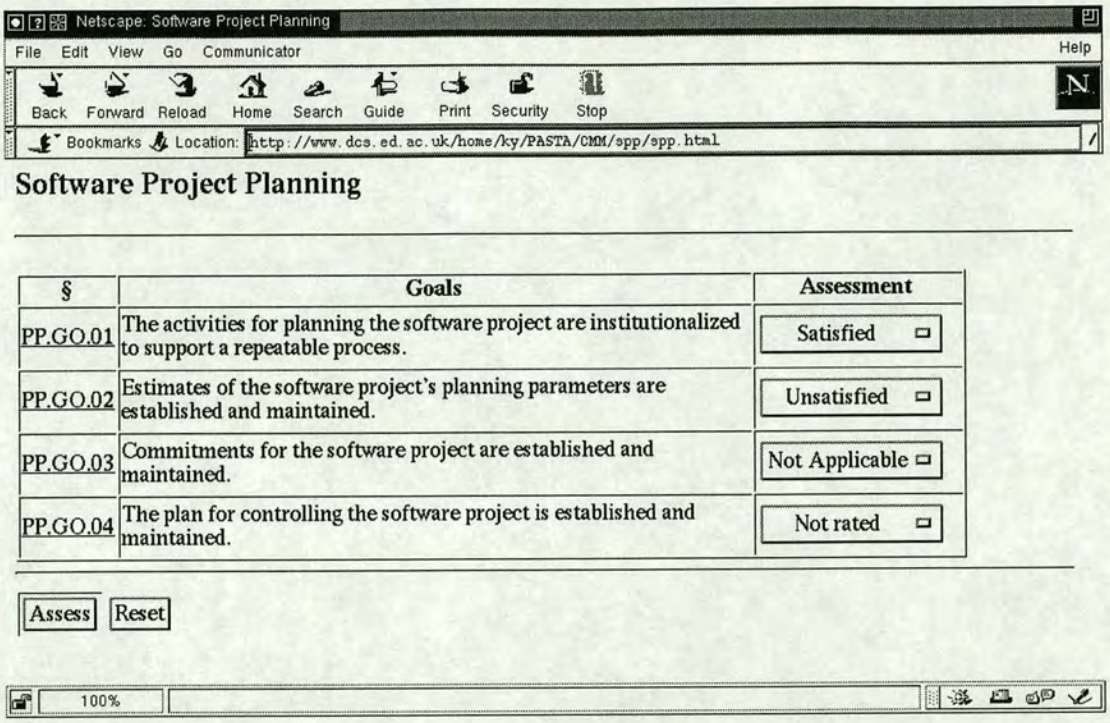

Figure 6.7: The Assessment of Software Project Planning

can not be appropriately performed. The project manager should rely on these comments to modify SPI PASTA. This effort could be done after developing the project. It is viewed as the organisation's software process assets to contribute to future software projects. Moreover, this effort could happen during developing the software project. It could be that the process might not be performed as expected or some critical facilities force process changes.

in environments ranging from the individual PC to global distributed systems.

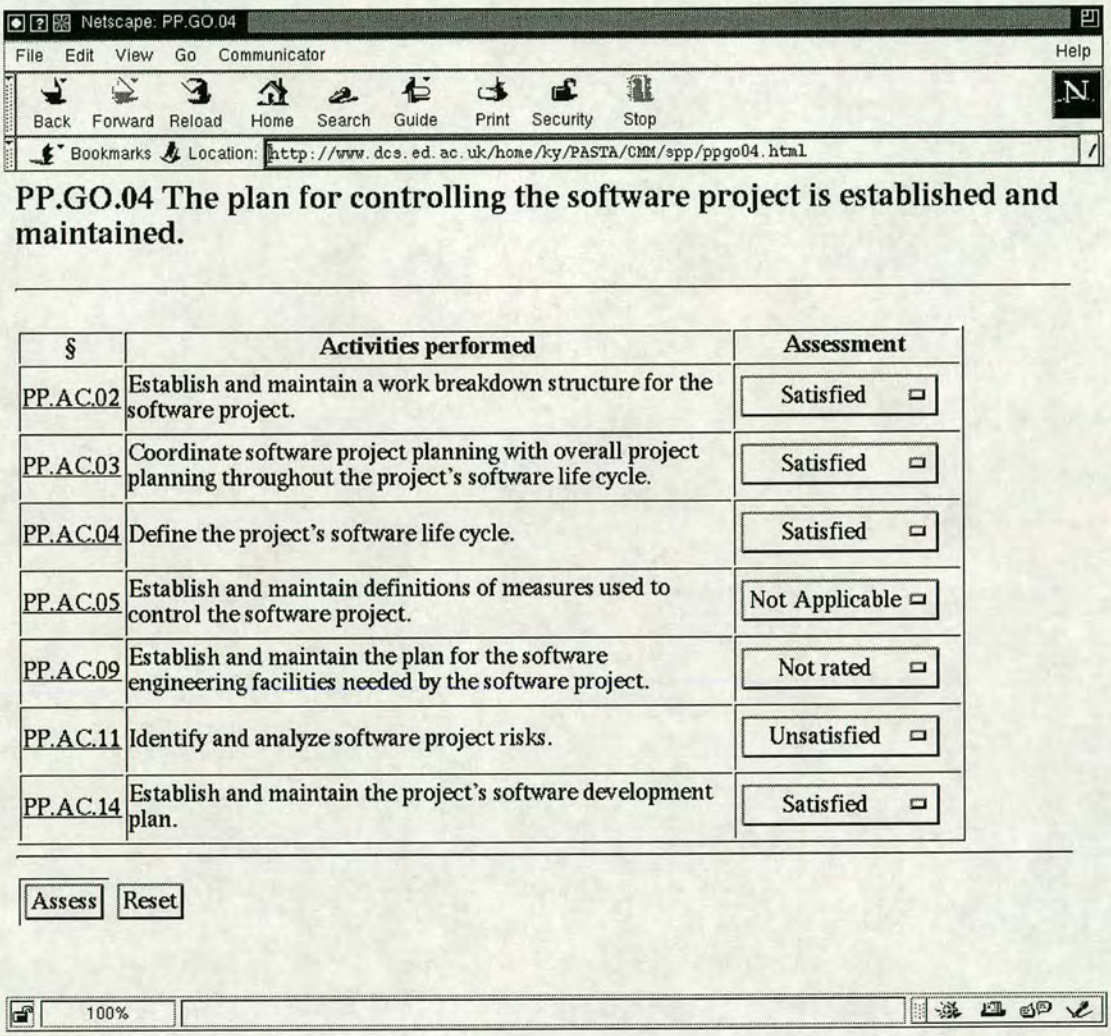

Figure 6.8: The Assessment of Establishing the Software Development Plan

Chapter 7

Conclusions and future work

7.1 Conclusions

This thesis has explored a variety of issues in process modelling for software development under the capability maturity model. The CMM focuses on the software process that can have an effect on the effectiveness of people in doing their work and the adoption of effective technologies, all of which will help the organisation attain its business objectives. However, many of limitations and weaknesses in the CMM have long been evident. Consequently, only 16% of assessed organisations have reached higher than maturity Level 3 for ten years. One of weaknesses in the CMM is that the CMM is actually a descriptive model in the sense that it describes essential attributes that would be expected to characterise an organisation at a particular maturity level. This means that the CMM describes what a process should address rather than how it should be implemented. As a result, there are many difficulties for software organisations using the CMM since the CMM contains so many key practices for developing and maintaining software. In spite of supplementary works, such as the Software Process Framework and the Trillium model, software organisations still have difficulty to follow the CMM.

In this thesis, an artifact-driven approach has been developed to support the definition of software processes. It corresponds to the CMM's KPAs complying with the relevant standards to develop software projects. The goal of this research is to provide a model to guide a continuous improvement program. Since the CMM is a descriptive model that does not specify how software development should be implemented, some standards and methodology, such as the UML, ISO 9001 and so on, had to be added in software processes. SPI PASTA has been developed from artifact and process views to help users develop software work products. We believe there are several reasons for using graphical process notations to model the software process:

- The software development activities are increasingly complex and interwoven. It is hard to handle a software project as a single person. Clearly, the software process should be recognised by all interested parties which may be a project manager, requirement analysts, designers, quality assurance staff and so on. A unified process model provides a common language to these process roles. This will lead to a significant progress for software development.

- SPI PASTA provides a framework in terms of artifacts which can integrate standards, CASE tools, and relevant activities to apply the CMM's KPAs. Software organisations may then more easily develop their software projects in correspondence with the CMM.

- An experience of the software development efforts can be recorded by the process notations. This will form a foundation of the organisation's software process assets in order to improve organisation's software process.

- The process modelling clearly presents what software process information and where the resources developers need to follow and apply. This will be helpful for those who just join the development team to avoid chaos in the beginning.

In chapter 3 we firstly addressed how to establish the organisation's set of standard software processes which covers the entire software process. We started out by selecting software life cycle, establishing the tailoring guidelines and creating the organisation's software measurement database. These activities form a foundation to establish the project's defined software process which is practically relied on to develop a software project.

Developing the software management processes is described in section 4.1. The most important task in the software management processes is to establish the software development plan. The software development team relies on the software development plan to perform and track software activities, communicate status and take corrective action. The allocated requirements should be derived in order to form the basis for planning, performing and tracking the software project's plans and activities. The software may be acquired from different sources, such as COTS products, external contractors and so on. Software acquisition management defines approaches to manage these sources. However, SPI PASTA does not focus on the topic for real-time and control systems. Currently, the system requirements allocated to hardware are not discussed in SPI PASTA.

Two support processes, software quality assurance and software configuration management, are described in section 4.2. The software project must be conducted under quality control. These activities ensure the software project will keep its pace on the right way. In the meantime, changes of software work products should be well controlled, especially in the complex and distributed environment.

In section 5.1 we addressed the technical processes. Since SPI PASTA adopted OO methodology, with MIL-STD-498 standard, the software work products are partitioned into several CSCIs and may be developed in several builds. The technical processes provide the guideline to conduct these activities throughout the software life cycle.

Several organisational processes are described in section 5.2. Relevant training should be organised in order to allow staff to perform those activities during software development. Furthermore, the potential problems must be discovered as early as possible and a solution found to tackle them. Software risk management defines the approach to identify, analyse and eliminate those software risks. Since developing a software project needs different roles to perform relevant activities, project coordination issues should receive appropriate attention. All groups of the software project must communicate well to ensure that everyone involved in the software project is appropriately aware of his/her status. Moreover, the activities to identify defects of software work products are performed by the developers' peers.

Finally, we addressed an appraisal method to assess SPI PASTA. By way of assessing each key practice, the goals of KPAs will be appropriately achieved and SPI PASTA will be increasingly improved.

7.2 Future Work

As we have mentioned in previous chapters, the CMM contains too much information for developing and maintaining software. People developing software products consistent with the CMM need a framework which helps them to analyse and implement these process information. In this thesis, we addressed a software process improvement issue by modelling the CMM's level 2 and 3. We believe that the key process areas described in the CMM's level 2 and 3 are the major barrier for software organisation. As a result, an infrastructure has been built to help software developers handle software processes.

In the future, some technical problems, such as tree navigation, might be modified in order to run smoothly. Some processes, such as OO testing strategies,

should be completely developed by the software organisation to comply with the whole processes. Moreover, SPI PASTA should be expanded into the key process areas at level 4 and 5. These areas focus on establishing a quantitative understanding of both the software process and the software work products being built, and implementing continual and measurable software process improvement. These processes must base on the infrastructure defined at level 2 and 3 and software organisations need time to build this infrastructure. Furthermore, the SEI has developed a number of other CMMs, such as the System Engineering CMM (SE-CMM), the People CMM (P-CMM) and the Software Acquisition CMM (SA-CMM). The SEI is developing an integrated framework for describing the current and intended relationships of existing and potential maturity models. Ideally the various CMMs should work together harmoniously for the benefit of organisations needing to efficiently apply more than one CMM to improve their product quality and productivity. However, we believe it will be helpful by using artifact and process abstraction to integrate CMMs. Consequently, the process of developing the real-time and control systems should be included in SPI PASTA in the future.

Finally, a measure of the actual results achieved by following a process should be appropriately collected and controlled. The common software measures for the organisation are comprised of process and product measures that summarise the software process performance achieved by the projects. In order to achieve this goal, the relevant tools must be integrated with SPI PASTA. This will provide a seamless connection between software processes and tools. With this integrated process environment, developers do their job by following SPI PASTA and the common data will be automatically collected by those tools. Once this step is achieved, it will be much easier to reach the final destination of software process improvement.

List of Figures

List of Tables

Bibliography

[AES95] Paul G. Arnold, William H. Ett, and S. Wayne Sherer. Software Process Framework: Tool to Determine Consistency With the CMM. *Software Technology Conference, Salt Lake City, UT*, April 1995.

[AG83] A. J. Albrecht and J. E. Gaffney. Software Function, Source Lines of Code, and Development Effort Prediction. *IEEE Transaction on Software Engineering*, pages 639–648, November 1983.

[Amb95] Scott Ambler. The Object Primer. *SIGS Books, New York,*, 1995.

[Aoy90] M Aoyama. Distributed Concurrent Development of Software System: An Object-Oriented Process Model. *Proc.Compsac, IEEE CS Press*, pages 330–337, 1990.

[Aoy93] M Aoyama. Concurrent-Development Process Model. *IEEE SOFTWARE*, pages 46–55, July 1993.

[Aoy97] M Aoyama. Agile Software Process Model. *COMPSAC'97, The Twenty-first Annual International Computer Software & Applications Conference, Washington D.C.*, pages 454–459, August 1997.

[Bac94] James Bach. The Immaturity of the CMM. *American Programmer*, September 1994.

[BBB95] David Bristow, Brian Bulat, and Roger Burton. Product-Line Process Development. *Software Technology Conference, Salt Lake City, UT*, April 1995.

[BC96] Lisa Brownsword and Paul Clements. A Case Study in Successful Product Line Development. *Technical Report, SEI-96-TR-016, SEI, Carnegie Mellon University, Pittsburgh, PA*, October 1996.

[BCKM97] Kathy Beckman, Neal Coulter, Soheil Khajenoori, and Nancy R. Mead. Collaborations: Closing the Industry-Academia Gap. *IEEE SOFTWARE*, pages 49–57, November 1997.

[Ber97] Klaus Berg. Component-Based Development: No Silver Bullet. *Object Magazine*, March 1997.

[BHS80] E. H. Bersoff, V. D. Henderson, and S. G. Siegel. *Software Configuration Management.* Prentice-Hall, 1980.

[BJR97] Grady Booch, Ivar Jacobson, and James Rumbaugh. The Unified Modelling Language. *Documentation Set Version 1.1, http://www.rational.com/, Rational Software Corporation, CA, USA*, September 1997.

[BKW95] Roger Bate, Dorothy Kuhn, and Curt Wells. A Systems Engineering Capability Maturity Model, Version 1.1. *Maturity Model, CMU/SEI-95-MM-003, SEI, Carnegie Mellon University, Pittsburgh, PA*, November 1995.

[BM91] Terry B. Bollinger and Clement McGowan. A Critical Look at Software Capability Evaluations. *IEEE SOFTWARE*, pages 25–41, July 1991.

[BNF96] Sergio Bandinelli, Elisabetta Di Nitto, and Alfonso Fuggetta. Supporting cooperation in the SPADE-1 Environment. *IEEE Transactions on Software Engineering*, 22(12), December 1996.

[Boe81] Barry W. Boehm. *Software Engineering Economics.* Prentice-Hall, 1981.

[Boe88] B. Boehm. A Spiral Model for Software Development and Enhancement. *Computer*, 21(5):61–72, May 1988.

[Boh97] Kathy Bohrer. Middleware Isolates Business Logic. *Object Magazine*, November 1997.

[Boo93] G. Booch. *Object-Oriented Analysis and Design with Application.* The Benjamin-Cummings Publishing Company, Inc., second edition, 1993.

[Boo96] G. Booch. *Object Solutions, Managing the Object-oriented Project.* Addison-Wesley Publishing Company, Inc., 1996.

[BP96] P. Byrnes and M. Phillips. Software Capability Evaluation Version 3.0 Method Description. *Technical Report, SEI-96-TR-002, SEI, Carnegie Mellon University, Pittsburgh, PA*, April 1996.

[BSK95] Israel Z. Ben-Shaul and G. E. Kaiser. An Interoperability Model for Process-Centered Software Engineering Environments and its Implementation in Oz. *Columbia University Department of Computer Science, CUCS-034-95*, December 1995.

[BSK96] Israel Z. Ben-Shaul and G. E. Kaiser. Integrating Groupware Activities into Workflow Management Systems. *7th Israeli Conference on Computer Based Systems and Software Engineering, Tel Aviv, Israel*, pages 140–149, June 1996.

[Car97] David Carney. Assembling Large System from COTS Components: Opportunities, Cautions, and Complexities. *SEI Monographs on COTS, SEI, Carnegie Mellon University, Pittsburgh, PA*, June 1997.

[CB89] Ward Cunningham and Kent Beck. A Laboratory For Teaching Object-Oriented Thinking . *Proceedings of the OOPSLA'89 Conference, ACM SIGPLAN Notices, New Orleans, Louisiana*, 24(10), October 1989.

[CFM+96] Sholom Cohen, Seymour Friedman, Lorraine Martin, Tom Royer, Nancy Solderitsch, and Robert Webster. Concept of Operations for the ESC Product Line Approach. *Technical Report, SEI-96-TR-018, SEI, Carnegie Mellon University, Pittsburgh, PA*, September 1996.

[CH95] Maribeth B. Carpenter and Harvey K. Hallman. Training Guidelines: Creating a Training Plan for a Software Organization. *Technical Report, SEI-95-TR-007, SEI, Carnegie Mellon University, Pittsburgh, PA*, September 1995.

[CHM95] Bill Curtis, William E. Hefley, and Sally Miller. People Capability Maturity Model . *Maturity Model, CMU/SEI-95-MM-02, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa.*, September 1995.

[CKM+93] Marvin J. Carr, Suresh L. Konda, Ira Monarch, F. Carol Ulrich, and Clay F. Walker. Taxonomy-Based Risk Identification. *Technical Report, SEI-93-TR-6, SEI, Carnegie Mellon University, Pittsburgh, PA*, June 1993.

[CLM+95] Reidar Conradi, Jens-Otto Larsen, Nguyen Ngoc Minh, Bjrn P. Munch, and Per H. Westby. Integrated Product and Process Man-

agement in EPOS. *Journal of Integrated CAE (special issue on Integrated Product and Process Modelling)*, 1995.

[CLM+97] A. Christie, L. Levine, E. J. Morris, B. Riddle, and D. Zubrow. Software Process Automation: Interviews, Survey, and Workshop Results. *Technical Report, SEI-97-TR-008, SEI, Carnegie Mellon University, Pittsburgh, PA*, October 1997.

[CLMZ96] A. Christie, L. Levine, E. J. Morris, and D. Zubrow. Software Process Automation: Experiences from the Trenches. *Technical Report, SEI-96-TR-013, SEI, Carnegie Mellon University, Pittsburgh, PA*, July 1996.

[CN96] P. C. Clements and L. M. Northrop. Software Architecture: An Executive Overview. *Technical Report, SEI-96-TR-003, SEI, Carnegie Mellon University, Pittsburgh, PA*, February 1996.

[CNFG96] Gianpaolo Cugola, Elisabetta Di Nitto, Alfonso Fuggetta, and Carlo Ghezzi. A framework for formalizing inconsistencies and deviations in human-centered systems. *ACM Transaction on Software Engineering and Methodology*, 5(3), July 1996.

[Con95] Reidar Conradi. PSEE Architecture: EPOS Process Models and Tools. *Workshop on Process-centered Software Engineering Environment Architecture, Milano*, March 1995.

[CS95] Michael A. Cusumano and R. W. Selby. *Microsoft Secrets.* The Free Press, New York, U.S., 1995.

[CS97] Michael A. Cusumano and R. W. Selby. How Microsoft Builds Software. *Communications of the ACM*, 40(6):30–40, June 1997.

[Cus91] Michael A. Cusumano. *Japan's Software Factories.* Oxford University Press, 1991.

[DL95] Alan M. Davis and Dean A. Leffingwell. Using Requirements Management to Speed Delivery of High Quality Applications. *Technical Report 0001, Rational Software Corporation, http://www.rational.com*, 1995.

[DM96] D. K. Dunaway and S. Masters. CMM-Based Appraisal for Internal Process Improvement(CBA IPI) : Method Description. *Technical*

Report, SEI-96-TR-007, SEI, Carnegie Mellon University, Pittsburgh, PA, April 1996.

[DOD94] DOD. Software Development and Documentation. *MIL-STD-498, Department of Defense, U.S.*, December 1994.

[DS97] Michael Diaz and Joseph Sligo. How Software Process Improvement Helped Motorola. *IEEE SOFTWARE*, pages 75–81, September 1997.

[DSD97] DSDM. DSDM, The Dynamic Systems Development Method. *The DSDM Consortium, http://www.dsdm.org/*, 1997.

[Ebe97] Christof Ebert. The Road to Maturity: Navigating Between Craft and Science. *IEEE SOFTWARE*, pages 77–82, November 1997.

[ES94] R. G. Ebenau and S. H. Strauss. *Software Inspection Process.* McGraw-Hill, Inc., 1994.

[Fag76] M. E. Fagan. Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, 15(3), 1976.

[FC96] Mohamed E. Fayad and Marshall Cline. Managing Object-Oriented Softwrae Development. *IEEE COMPUTER*, pages 26–31, September 1996.

[FHK+97] Pat Ferguson, Watts S. Humphrey, Soheil Khajenoori, Susan Macke, and Annette Matvya. Results of Applying the Personal Software Process. *IEEE SOFTWARE*, pages 24–31, May 1997.

[FKN94] A. Finkelstein, J. Kramer, and B. Nuseibeh. Software Process Modelling and Technology. *Research Studies Press LTD*, 1994.

[Fow97] Martin Fowler. *UML Distilled, Applying the Standard Object Modelling Language.* Addison-Wesley, 1997.

[FS97] M. E. Fayad and D. C. Schmidt. Object-Oriented Application Frameworks. *Communications of the ACM*, 40(10), October 1997.

[Gar98] Suzie Garcia. Evolving Improvement Paradigms: Capability Maturity Models & ISO/IEC 15504 (PDTR). *Software Process Improvement and Practice, volume 3, issue 1, Wiley/Gaulthier-Villars*, 1998.

[Gat97] Linda Parker Gates. How to Use the Software Process Framework . *Special Report, SEI-97-SR-009, SEI, Carnegie Mellon University, Pittsburgh, PA*, October 1997.

[GHJV94] E. Gamma, R. Help, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional Computing Series, 1994.

[GM97] Jennifer Gremba and Chuck Myers. The IDEAL Model: A Practical Guide for Improvement . *Bridge, Issue three, the Software Engineering Institute (SEI) publication*, 1997.

[GQ95] M. P. Ginsberg and L. H. Quinn. Process Tailoring and the Software Capability Maturity Model. *Technical Report, SEI-94-TR-024, SEI, Carnegie Mellon University, Pittsburgh, PA*, November 1995.

[GR95] Adele Goldberg and Kenneth S. Rubin. *Succeeding with Objects.* Addison-Wesley Publishing Company, 1995.

[Hal96] Thomas J. Haley. Software Process Improvement At Raytheon. *IEEE SOFTWARE*, pages 33–41, NOVEMBER 1996.

[HC91] Watts S. Humphrey and Bill Curtis. Comments on 'A Critical Look'. *IEEE SOFTWARE*, pages 42–46, July 1991.

[HGD+94] Ronald P. Higuera, David P. Gluch, Audrey J. Dorofee, Richard L. Murphy, Julie A. Walker, and Ray C. Williams. An Introduction to Team Risk Management (Version 1.0) . *Technical Report, SEI-94-TR-001, SEI, Carnegie Mellon University, Pittsburgh, PA*, May 1994.

[HH96] Ronald P. Higuera and Yacov Y. Haimes. Software Risk Management. *Technical Report, SEI-96-TR-012, SEI, Carnegie Mellon University, Pittsburgh, PA*, June 1996.

[HIW+95] T. Haley, B. Ireland, E. Wojtaszek, D. Nash, and R. Dion. Raytheon Electronic Systems Experience in Software Process Improvement . *Technical Report, SEI-95-TR-017, SEI, Carnegie Mellon University, Pittsburgh, PA*, November 1995.

[HO97] Will Hayes and James W. Over. The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual

Engineers. *Technical Report, SEI-97-TR-001, SEI, Carnegie Mellon University, Pittsburgh, PA*, December 1997.

[Hum88] W. S. Humphrey. Characterizing the Software Process. *IEEE SOFTWARE*, 5(2):73–79, March 1988.

[Hum95] W. S. Humphrey. *A Discipline for Software Engineering.* Addison-Wesley Publishing Company, 1995.

[IEE90] IEEE. IEEE Standard for Software Configuration Management Plans, IEEE 828-1990. *The Institute of Electrical and Electronics Engineers, Inc.*, September 1990.

[ISO96] ISO/IEC15504-9. Information Technology - Software Process Assessment Part 9: Vocabulary. *ISO/IEC JTC1/SC7 N1609*, 1996.

[Jac87] Ivar Jacobson. Object-oriented Development in an Industrial Environment. *Proceedings of OOPSLA'87, Orlando, FL*, pages 183–191, 1987.

[JCJO92] Ivar Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. *Object-Oriented Software Engineering, A Use Case Driven Approach.* Addison-Wesley, 1992.

[JEJ95] Ivar Jacobson, M. Ericsson, and A. Jacobson. *The Object Advantage, Business Process Reengineering with Object Technology.* Addison-Wesley, 1995.

[JGJ97] Ivar Jacobson, M. L. Griss, and P. Jonsson. *Software Reuse, Architecture, Process and Orhanisation for Business Success.* Addison-Wesley, 1997.

[Jon91] Capers Jones. *Applied Software Management: Assuring Productivity and Quality.* McGraw-Hill, New York, 1991.

[Jon96] Capers Jones. Conflict and Litigation Between Software Clients and Developers. *Software Productivity Research, Inc. One New England Executive Park Burlington, MA, http://www.spr.com/html/resources.htm*, March 1996.

[KCF+96] Mike Konrad, Mary Beth Chrissis, Jack Ferguson, Suzanne Garcia, Bill Hefley, Dave Kitson, and Mark C. Paulk. Capability Maturity

Modeling at SEI. *Software Process - Improvement and Practice*, 2:21–34, 1996.

[KDJY97] Gail E. Kaiser, Stephen E. Dossick, Wenyu Jiang, and Jack Jingshuang Yang. An Architecture for WWW-based Hypercode Environments. *1997 International Conference on Software Engineering: Pulling Together, Boston MA*, May 1997.

[Kit96] D. H. Kitson. Relating the SPICE Framework and the SEI Approach to Software Process Assessment. *The Fifth European Conference on Software Quality, Dublin, Ireland*, September 1996.

[Kit97] David H. Kitson. An Emerging International Standard for Software Process Assessment. *Proceedings of the Third International Software Engineering Standards Symposium and Forum, IEEE Computer Society, Walnut Creek, CA*, June 1997.

[KS95a] Carol Klingler and Dan Schwarting. A Practical Approach to Process Definition. *Software Technology Conference, Salt Lake City, UT*, April 1995.

[KS95b] R. E. Kraut and L. A. Streeter. Coordination in Software Development. *Communications of the ACM*, 38(3), March 1995.

[Lai91] Robert. C. T. Lai. Process Definition and Process Modeling Methods. *Technical Report, Software Productivity Consortium, SPC-91084-N*, September 1991.

[LIO96] J. L. LIONS. Ariane 501 - Presentation of Inquiry Board report. *Nr 33-96, European Space Agency, http://www.esrin.esa.it/tidc/Press/press96b.html*, July 1996.

[Man84] J. H. Manley. CASE: Foundation for Software Factories. *COMPCON Proceedings, IEEE*, pages 84–91, September 1984.

[Mat96] M. Mattsson. Object-Oriented Frameworks - A survey of methodological issues. *Licentiate Thesis,LU-CS-TR: 96-167, Department of Computer Science, Lund University*, 1996.

[Mau96] Frank Maurer. Computer Support in Project Coordination. *Proceedingd of the Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 96), Stanford University, California, USA*, June 1996.

[MB95] Steve Masters and Carol Bothwell. CMM Appraisal Framework, Version 1.0. *Technical Report, SEI-95-TR-001, SEI, Carnegie Mellon University, Pittsburgh, PA*, February 1995.

[McC97] Steve McConnell. Tool Support for Project Tracking. *IEEE SOFTWARE*, pages 119–120, September 1997.

[McF96] Bob McFeeley. IDEAL: A User's Guide for Software Process Improvement. *CMU/SEI-96-HB-001, Software Engineering Institute, Carnegie Mellon University*, Feburary 1996.

[Mon83] Yasuhiro Monden. *Toyota production system : practical approach to production management.* Industrial Engineering and Management Press, Institute of Industrial Engineers,, 1983.

[MPB94] Frank McGarry, Gerald Page, and Victor Basili. Software Process Improvement in the NASA Software Engineering Laboratory. *Technical Report, SEI-94-TR-022, SEI, Carnegie Mellon University, Pittsburgh, PA*, December 1994.

[MTC96] Nancy Mead, Lawrence Tobin, and Suzanne Couturiaux. Best Training Practices Within the Software Engineering Industry. *Technical Report, SEI-96-TR-034, SEI, Carnegie Mellon University, Pittsburgh, PA*, November 1996.

[NaC96] Minh N. Nguyen and Alf Inge Wang andReidar Conradi. Total Software Process Model Evolution in EPOS. *4th ICSP, Brigthon, UK*, December 1996.

[NAS90] NASA. Manager's Handbook for Software Development, Revision 1. *Software Engineering Laboratory Series SEL-84-101, NASA, U.S.*, November 1990.

[NF95] E. Di Nitto and A. Fuggetta. Integrating process technology and CSCW. *Proceedings of IV European Workshop on Software Process Technology, Leiden, The Nederland*, April 1995.

[NFK+97] K. Nakamura, Y. Fujii, Y. Kiyokane, M. Nakamura, K. Hinenoya, Yeo Hua Peck, and C. Siow. Distributed and Concurrent Development Environment via Sharing Design Information. *COMPSAC'97,*

The Twenty-first Annual International Computer Software & Applications Conference, Washington D.C., pages 274–279, August 1997.

[Obj97] Objectory. Rational Objectory Process - Introduction 4.1. *Rational Software Corporation, http://www.rational.com*, July 1997.

[OMG97] OMG. The Common Object Request Broker: Architecture and Specification (CORBA 2.1/IIOP). *http://www.omg.org/, Object Management Group, Framingham, MA U.S.A.*, August 1997.

[ORO94] T. G. Olson, N. R. Reizer, and J. W. Over. A Software Process Framework for the SEI Capability Maturity Model. *Handbook, SEI-94-HB-001, SEI, Carnegie Mellon University, Pittsburgh, PA*, September 1994.

[Par92] Robert E. Park. Software Size Measurement: A Framework for counting Source Statements . *Technical Report, CMU/SEI-92-TR-20, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa.*, September 1992.

[Pau94] Mark C. Paulk. A Comparision of ISO 9001 and the Capability Maturity Model for Software. *Technical Report, SEI-94-TR-012, SEI, Carnegie Mellon University, Pittsburgh, PA*, July 1994.

[Pau95] Mark C. Paulk. The Rational Planning of (Software) Projects. *Proceedings of the First World Congress for Software Quality, San Francisco, CA*, June 1995.

[Pau97] Mark C. Paulk. Capability Maturity Model for Software, Version 2.0, Draft C. *http://www.sei.cmu.edu/technology/cmm/, SEI, Carnegie Mellon University, Pittsburgh, PA*, September 1997.

[PCCW93a] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber. Capability Maturity Model, Version 1.1. *IEEE SOFTWARE*, 10(4):18–27, July 1993.

[PCCW93b] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber. Capability Maturity Model for Software, Version 1.1. *Technical Report, CMU/SEI-93-TR-024, SEI, Carnegie Mellon University, Pittsburgh, PA*, February 1993.

[PDHT97] Gerald M. Powell, Jorge L. Diaz-Herrera, and Dennis J. Turner. Achieving Synergy in Collaborative Education. *IEEE SOFTWARE*, pages 58–65, November 1997.

[PGC96] Mark C. Paulk, S. M. Garcia, and Mary Beth Chrissis. The Continuing Improvement of the CMM: Version 2. *Fifth European Conference on Software Quality, Dublin, Ireland*, September 1996.

[PM92] Lawrence H. Putnam and Ware Myers. *Measures for Excellence: Reliable Software on Time, within Budget.* Yourdon Press, Englewood Cliffs, NY, 1992.

[Pre94] R. S. Pressman. *Software Engineering, A Practitioner's Approach.* McGraw-Hill Book Company Europe, Berkshire, England, third edition, 1994.

[PRI97] PRINCE. PRINCE, Projects in Controlled Environments. *The Official PRINCE Home Page, http://www.ccta.gov.uk/prince/prince.htm*, February 1997.

[PW96] Rose Pajerski and Sharon Waligora. The Improvement Cycle: Analyzing Our Experience. *Twenty-First Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Maryland, USA*, December 1996.

[PWG+93] Mark C. Paulk, Charles V. Weber, Suzanne M. Garcia, Mary Beth Chrissis, and Marilyn W. Bush. Key Practices of the Capability Maturity Model, Version 1.1. *Technical Report, CMU/SEI-93-TR-025, SEI, Carnegie Mellon University, Pittsburgh, PA*, February 1993.

[RE95] Richard Randall and William Ett. Using Process to Integrate Software Engineering Environments. *Software Technology Conference, Salt Lake City, UT*, April 1995.

[Sco92] Roger L. Van Scoy. Software Development Risk: Opportunity, Not Problem. *Technical Report, SEI-92-TR-30, SEI, Carnegie Mellon University, Pittsburgh, PA*, September 1992.

[SEI97] SEI. Semi-Annual Process Maturity Profile of the Software Community Report. *http://www.sei.cmu.edu/technology/measurement/, SEI, Carnegie Mellon University, Pittsburgh, PA*, May 1997.

[SJ94] F. Sisti and S. Joseph. Software Risk Evaluation Method Version 1.0. *Technical Report, SEI-94-TR-019, SEI, Carnegie Mellon University, Pittsburgh, PA*, December 1994.

[SPI95] SPICE. Software Process Assessment Part 4: Guide To Conducting Assessments, Version1.0. *ISO/IEC JTC1/SC7 N944R*, 1995.

[Tri94] Trillium. Model for Telecom Product Development & Support Process Capability. *Bell Canada, Internet Edition, Release 3.0, http://ricis.cl.uh.edu/process_maturity/download.html*, December 1994.

[TS96] Scott R. Tilley and Dennis B. Smith. Coming Attractions in Program Understanding. *Technical Report, SEI-96-TR-019, SEI, Carnegie Mellon University, Pittsburgh, PA*, December 1996.

[TT84] D. Tajima and T.Matsubara. Inside the Japanese Software Factory. *Computer*, 17(3):34–43, March 1984.

[TW97] P. P. Texel and C. B. Williams. *Use Cases Combined with Booch, OMT, UML.* Prentice Hall PTR, 1997.

[Uzz96] Lyn Uzzle. SEEWeb: A Software Engineering Environment Information Interface Based on World Wide Web Technology. *Software Technology Conference (STC '96), Salt Lake City, UT*, April 1996.

[VK94] M. R. Vigder and A. W. Kark. Software Cost Estimation and Control. *Institute for Information Technology, National Research Council of Canada*, February 1994.

[Wit96] James Withey. Investment Analysis of Software Assets for Product Lines. *Technical Report, SEI-96-TR-010, SEI, Carnegie Mellon University, Pittsburgh, PA*, November 1996.

[You96] E. Yourdon. *Rise & Resurrection of the American Programmer.* PTR Prentice-Hall, Inc., 1996.

Appendix A

P-state Definition Forms

SPI PASTA has created more than 700 files which are interrelated with each other. Most of them are definition forms that describes required information to perform relevant activities. Among these files, the most important one is the P-state definition form which describes the artifacts process roles and relevant operations. All information can be found or connected in this form. Since it is impossible to present all files in this thesis, we would like to only list P-state definition forms. These forms are organised by alphabet. Readers can easily look for what they need and comply with SPI PASTA on the Web (http://www.dcs.ed.ac.uk/home/ky/PASTA/SPIPasta.html).

Process State Definition Form	
Name	Control_SQA_Issue
Synopsis	Objectively control the SQA issues which include software activities and software work products.
Main Role	Quality_Assurance_Staff
Entrance Condition	state-of(SQA_Plan)=Drafted
Artifact List	SQA_Issue
Information Artifacts	SQA_Plan
Operation List	
Name	Review_Activity
Synopsis	Objectively review designated software activities against the applicable requirements, process descriptions, standards, and procedures
Name	Review_Work_Product
Synopsis	Objectively review designated software work products against the applicable requirements and standards.
Exit Condition	state-of(SQA_Issue)=Controlled
Informal Specification	QA.AC.03 QA.AC.04
Formal Specification	

Process State Definition Form	
Name	Create_Class_Diagram
Synopsis	A class is drawn as a solid-outline rectangular box with three compartments, with the class name in the top compartment, a list of attributes in the middle compartment, and a list of operations in the bottom compartment. The activity to create a class diagram is associated with the use case model. From use case model, we collect the same structure, behaviour and relationship to create the class.
Main Role	Software_Designer, Requirement_Analyst, Software_Product_Manager
Entrance Condition	state-of(CRC_Card)=Created
Artifact List	Class_Diagram
Information Artifacts	CRC_Card, Use_Case_Diagram
Operation List	
Name	Identify_Key_Class
Synopsis	The key classes are identified from Use Case diagrams and CRC cards.
Name	Identify_Attribute_And_Operation
Synopsis	Identify attributes and operations of the Class.
Name	Identify_Relationship
Synopsis	Identify the relationship between the Classes.
Name	Add_Interface_Class
Synopsis	Add the GUI Classes to the model.
Name	Complete_Class_Specification
Synopsis	Complete all descriptions of the Class Specification.
Exit Condition	state-of(Class_Diagram)=Created

Informal Specification	Reviewing the software requirements to ensure that issues affecting the software design are identified and resolved. Adhering to applicable software design criteria and standards. Developing the software architecture early, within the constraints of the software life cycle and technology being used. Reviewing and getting agreement with affected parties on the software architecture, to ensure that architecture issues affecting the detailed software design are identified and resolved. Basing the detailed software design on the software architecture and the software requirements. Documenting the software design. Tracing the software design to the software requirements and documenting the traceability. Placing the traceability documentation for the software design under version control and change control.
Formal Specification	

Process State Definition Form	
Name	Create_Collaboration_Diagram
Synopsis	Create the collaboration diagram to offer another dynamic view of the syste.
Main Role	Software_Designer, Requirement_Analyst, Software_Product_Manager
Entrance Condition	state-of(Sequence_Diagram)=Created
Artifact List	Collaboration_Diagram
Information Artifacts	Sequence_Diagram, Use_Case_Diagram
Operation List	
Name	Convert_Sequence_Diagram
Synopsis	Use CASE tools to convert the sequence diagram to the collaboration diagram.
Name	Refine_Collaboration_Diagram
Synopsis	Refine the converted collaboration diagram to complete design.
Exit Condition	state-of(Collaboration_Diagram)=Created
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Create_Implementation_Diagram
Synopsis	Create the implementation diagrams which include component diagrams and deployment diagrams.
Main Role	Software_Designer, Requirement_Analyst, Software_Product_Manager
Entrance Condition	state-of(Implementation_Diagram)=Referenced
Artifact List	Implementation_Diagram
Information Artifacts	
Operation List	
Name	Create_Component_Diagram
Synopsis	Create the component diagram to show the dependencies among software components.
Name	Create_Deployment_Diagram
Synopsis	Create the deployment diagram to show the configuration of run-time processing elements.
Exit Condition	state-of(Implementation_Diagram)=Created
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Create_Interaction_Diagram
Synopsis	Create the interaction diagram which includes the sequence diagram and the collaboration diagram.
Main Role	Software_Designer, Requirement_Analyst, Software_Product_Manager
Entrance Condition	state-of(Interaction_Diagram)=Referenced
Artifact List	Interaction_Diagram
Information Artifacts	Use_Case_Diagram
Sub-P-State List	
Name	Create_Sequence_Diagram
Synopsis	Create the sequence diagram to offer the dynamic view for the system.
Name	Create_Collaboration_Diagram
Synopsis	Create the collaboration diagram to offer another dynamic view of the syste.
Exit Condition	state-of(Interaction_Diagram)=Created
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Create_Requirement_Trace_Matrix
Synopsis	Create the Requirement Trace Matrix to handle the allocated requirement.
Main Role	Customer, Requirement_Analyst, Software_Product_Manager
Entrance Condition	state-of(Allocated_Requirement)=Derived
Artifact List	Requirement_Trace_Matrix
Information Artifacts	Allocated_Requirement
Operation List	
Name	Extract_Allocated_Requirement
Synopsis	Produce an initial RTM that contains the entire set of sentence from the allocated requirement.
Name	Categorize_RTM
Synopsis	Categorize each entry in the RTM according to its "type".
Exit Condition	state-of(Requirement_Trace_Matrix)=Categorized
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Create_Sequence_Diagram
Synopsis	Create the sequence diagram to offer the dynamic view for the system.
Main Role	Software_Designer, Requirement_Analyst, Software_Product_Manager
Entrance Condition	state-of(Sequence_Diagram)=Referenced and state-of(Use_Case_Diagram)=Created
Artifact List	Sequence_Diagram
Information Artifacts	Use_Case_Diagram
Operation List	
Name	Identify_Object_Lifeline
Synopsis	Identify the object lifeline as completely as possible.
Name	Identify_Message
Synopsis	Identify the messages from the lifeline of one object to the lifeline of another object.
Exit Condition	state-of(Sequence_Diagram)=Created
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Create_Statechart_Diagram
Synopsis	Determine whether or not a Class has to create a statechart diagram, then, create a statechart diagram to present the individual behaviour of the class.
Main Role	Software_Designer, Requirement_Analyst, Software_Product_Manager
Entrance Condition	state-of(Class_Diagram)=Created
Artifact List	Statechart_Diagram
Information Artifacts	Class_Diagram, Use_Case_Diagram
Operation List	
Name	Identify_State
Synopsis	A state is a condition during the life of an object or an interaction during which it satisfies some condition, performs some action, or waits for some event.
Name	Identify_Internal_Transition
Synopsis	Examine all possible transitions within the state.
Name	Identify_External_Transition
Synopsis	Examine all possible transitions of states within the Class.
Exit Condition	state-of(Statechart_Diagram)=Created
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Create_Use_Case_Diagram
Synopsis	Create the use case diagram to present the allocated requirement.
Main Role	Customer, Requirement_Analyst, Software_Product_Manager
Entrance Condition	state-of(Requirement_Trace_Matrix)=Categorized
Artifact List	Use_Case_Diagram
Information Artifacts	Requirement_Trace_Matrix, Allocated_Requirement
Operation List	
Name	Identify_Use_Case
Synopsis	Extract the software requirements from the RTM and reformat these requirements into Use Case format.
Name	Develop_Scenario
Synopsis	Develop scenario to provide the operational concept behind a use case.
Name	Establish_Project_Package
Synopsis	Establish an initial Package which is a collection of logically related classes.
Name	Allocate_Use_Case
Synopsis	The purpose of assigning Use Cases to Packages is to allocate the responsibility for Use Case development to a Package.
Name	Draft_GUI_Sketch
Synopsis	To provide a draft of the GUI for the system, as envisioned during the development of the scenarios.
Exit Condition	state-of(Use_Case_Diagram)=Created
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Derive_Software_Development_Plan
Synopsis	There are two levels for deriving the software development plan: The low level for estimating and planning a software project is described in the Software Project Planning key process area in the CMM. The high level for managing the software development plan is described in the Integrated Software Process key process area.
Main Role	Senior_Manager, Quality_Assurance_Staff, SEPG, Testing_Staff, System_Engineer, Software_Product_Manager, Project_Manager
Entrance Condition	state-of(Software_Development_Plan)=Referenced and state-of(PDSP)=Established
Artifact List	Software_Development_Plan
Information Artifacts	Allocated_Requirement, PDSP
Sub-P-State List	
Name	Draft_Schedule_And_Resource_Plan
Synopsis	Establish the software engineering resources needed by the project and the project's software schedule.
Operation List	
Name	Draft_Project_Mission_Plan
Synopsis	Draft the project mission plan with those affected on the project mission plan.
Name	Draft_Organisation_And_Responsibility_Plan
Synopsis	Draft the organisation and responsibility plan with those affected on the plan.
Name	Draft_Software_Engineering_Activity_Plan
Synopsis	Drafting the softwrae enginnering activity plan means selecting software life cycle models for use in the organisation.
Name	Identify_Project_Risk
Synopsis	Identify and analyze software project risks.
Exit Condition	state-of(Software_Development_Plan)=Derived
Informal Specification	IM.AC.03 PP.AC.14 PP.AC.15 Project planning and oversight in MIL-STD-498 Establishing a software development environment in MIL-STD-498
Formal Specification	

Process State Definition Form	
Name	Derive_System_Requirement
Synopsis	This phase is to produce a complete system requirements for the software project. The starting point is usually from a set of customer requirements that describe the project or problem.
Main Role	Customer, System_Engineer, Software_Product_Manager, Project_Manager
Entrance Condition	state-of(System_Requirement)=Referenced
Artifact List	System_Requirement
Information Artifacts	
Operation List	
Name	Derive_Allocated_Requirement
Synopsis	This document is produced by the software engineering team as the key product of the requirements definition.
Name	Derive_System_Requirement_To_Hardware
Synopsis	This document is produced by the system engineering team as the key product of the requirements definition.
Exit Condition	state-of(System_Requirement)=Derived
Informal Specification	System requirements analysis in MIL-STD-498 System design in MIL-STD-498
Formal Specification	

Process State Definition Form	
Name	Develop_Organisational_Process
Synopsis	Develop Organisational Process to provide the ability to control the software project.
Main Role	Reviewer, Training_Staff, Software_Product_Manager, Project_Manager
Entrance Condition	state-of(Organisational_Process)=Referenced
Artifact List	Organisational_Process
Information Artifacts	PDSP
Sub-P-State List	
Name	Perform_Organisation_Training_Program
Synopsis	Perform the organisation training program to develop the skill and knowledge of the development team.
Name	Perform_Risk_Management
Synopsis	Software risk management involves identifying risks, analyzing their likelihood and potential impact, determining and evaluating risk contingencies, tracking risks, and proactively manageing the risks.
Name	Perform_Project_Interface_Coordination
Synopsis	The purpose of Project Interface Coordination is to ensure that software managers and staff effectively communicate, coordinate, and collaborate with other functions in the organisation to staisfy the customer's needs.
Name	Perform_Peer_Review
Synopsis	Perform the activities for peer reviews.
Exit Condition	state-of(Organisational_Process)=Developed
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Develop_Software_Code
Synopsis	Develop and maintain the software code.
Main Role	Software_Programmer, Configuration_Management_Staff
Entrance Condition	state-of(Software_Design)=Developed
Artifact List	Software_Code
Information Artifacts	Software_Design
Operation List	
Name	Generate_Body_Structure_Code
Synopsis	Use the CASE tools to generate the body structure of a the Class.
Name	Implement_Source_Code
Synopsis	Implement the class method by using behaviour diagrams.
Exit Condition	state-of(Software_Code)=Developed
Informal Specification	PE.AC.05 Establish_Productivity_Measure Establish_Quality_Measure Software implementation and unit testing in MIL-STD-498
Formal Specification	

Process State Definition Form	
Name	Develop_Software_Design
Synopsis	Develop the software design to form the framework for coding.
Main Role	Software_Designer, Requirement_Analyst, Software_Product_Manager, Configuration_Management_Staff
Entrance Condition	state-of(Software_Requirement)=Developed
Artifact List	Software_Design
Information Artifacts	Software_Requirement
Sub-P-State List	
Name	Create_Class_Diagram
Synopsis	A class is drawn as a solid-outline rectangular box with three compartments, with the class name in the top compartment, a list of attributes in the middle compartment, and a list of operations in the bottom compartment. The activity to create a class diagram is associated with the use case model. From use case model, we collect the same structure, behaviour and relationship to create the class.
Name	Create_Statechart_Diagram
Synopsis	Determine whether or not a Class has to create a statechart diagram, then, create a statechart diagram to present the individual behaviour of the class.
Name	Create_Interaction_Diagram
Synopsis	Create the interaction diagram which includes the sequence diagram and the collaboration diagram.
Name	Create_Implementation_Diagram
Synopsis	Create the implementation diagrams which include component diagrams and deployment diagrams.
Operation List	
Name	Create_Activity_Diagram
Synopsis	Create the activity diagram to depict the execution steps to be performed by a method.
Exit Condition	state-of(Software_Design)=Developed
Informal Specification	PE.AC.04 Software design in MIL-STD-498
Formal Specification	

Process State Definition Form	
Name	Develop_Software_Management_Process
Synopsis	Develop Software Management Processes to effectively and effeciently control the software project.
Main Role	Management_Group, Customer, Software_Product_Manager
Entrance Condition	state-of(Software_Management_Process)=Referenced
Artifact List	Software_Management_Process
Information Artifacts	PDSP
Sub-P-State List	
Name	Derive_System_Requirement
Synopsis	This phase is to produce a complete system requirements for the software project. The starting point is usually from a set of customer requirements that describe the project or problem.
Name	Derive_Software_Development_Plan
Synopsis	There are two levels for deriving the software development plan: The low level for estimating and planning a software project is described in the Software Project Planning key process area in the CMM. The high level for managing the software development plan is described in the Integrated Software Process key process area.
Name	Perform_Software_Acquisition_Management
Synopsis	Perform Software Acquisition Management to manage the acquisition of software from sources external to the software project.
Name	Perform_Software_Project_Control
Synopsis	Track software project performance against the software development plan, and take corrective actions.
Operation List	
Name	Establish_Commitment
Synopsis	Establish the software project's commitments.
Exit Condition	state-of(Software_Management_Process)=Developed
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Develop_Software_Process_Improvement
Synopsis	Developing Software Process Improvement is trying to develop a framework that describes the necessary roles, activities, and resources needed for a successful process improvement effort.
Main Role	Development_Team
Entrance Condition	state-of(Software_Process_Improvement)=Initiated
Artifact List	Software_Process_Improvement
Information Artifacts	
Sub-P-State List	
Name	Establish_Organisation_Process_Focus
Synopsis	The purpose of Organisation Process Focus is to establish and maintain an understanding of the organisation's software processes and coordinate the organisation's software process improvement activities.
Name	Establish_Organisation_Process_Definition
Synopsis	Establish a usable set of software process assets that improve process performance across the organisation and that provide a basis for cumulative, long-term benefits to the organisation.
Name	Establish_PDSP
Synopsis	Establish the project's defined software process tailoring from the organisation's set of standard software process. The main tasks establishing project's defined software process are to model the software processes by using PASTA.
Operation List	
Name	Iterate_Or_Refine
Synopsis	
Name	Review
Synopsis	
Exit Condition	
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Develop_Software_Requirement
Synopsis	Develop and maintain the software requirements.
Main Role	Requirement_Analyst, Software_Product_Manager, Configuration_Management_Staff, Project_Manager
Entrance Condition	state-of(Software_Management_Process)=Developed
Artifact List	Software_Requirement
Information Artifacts	Allocated_Requirement
Sub-P-State List	
Name	Create_Requirement_Trace_Matrix
Synopsis	Create the Requirement Trace Matrix to handle the allocated requirement.
Name	Create_Use_Case_Diagram
Synopsis	Create the use case diagram to present the allocated requirement.
Operation List	
Name	Create_CRC_Card
Synopsis	Create CRC cards to assist analysts and customers in mapping the collaborations among classes, defined by the responsibilities each has in the system being modelled.
Exit Condition	state-of(Software_Requirement)=Developed
Informal Specification	PE.AC.02 PE.AC.03 Software requirement analysis in MIL-STD-498
Formal Specification	

Process State Definition Form	
Name	Develop_Software_Support_Process
Synopsis	Develop Software Support Process to support the software project management.
Main Role	Quality_Assurance_Staff, Software_Product_Manager, Configuration_Management_Staff, Project_Manager
Entrance Condition	state-of(Software_Support_Process)=Referenced
Artifact List	Software_Support_Process
Information Artifacts	PDSP
Sub-P-State List	
Name	Perform_Configuration_Management
Synopsis	This process involves: -Identifying the configuration of the software at given points in time. -Controlling changes to configuration items. -Building software work products from the software configuration library. -Maintaining the integrity of software baselines throughout the software life cycle.
Name	Perform_Software_Quality_Assurance
Synopsis	This process involves: -reviewing the software activities and work products against the applicable requirements, process descriptions, standards, and procedures. -identifying and documenting noncompliance issues. -providing feedback to project staff and managers. -ensuring that noncompliance issues are addressed.
Exit Condition	state-of(Software_Support_Process)=Developed
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Develop_Software_Technical_Process
Synopsis	Develop the software technical process to perform the engineering tasks to specify, design, build, deliver, and maintain the software using the project's defined software process in order to verify and validate that the software products satisfy their requirements.
Main Role	Development_Group, Quality_Assurance_Staff, Configuration_Management_Staff
Entrance Condition	state-of(Software_Technical_Process)=Referenced
Artifact List	Software_Technical_Process
Information Artifacts	Allocated_Requirement
Sub-P-State List	
Name	Develop_Software_Requirement
Synopsis	Develop and maintain the software requirements.
Name	Develop_Software_Design
Synopsis	Develop the software design to form the framework for coding.
Name	Develop_Software_Code
Synopsis	Develop and maintain the software code.
Name	Develop_Software_Test
Synopsis	Develop the software test to validate that the system satisfies its requirements.
Operation List	
Name	Develop_Operation_Documentation
Synopsis	Develop the documentation that will be used to install, operate, and maintain the software.
Name	Perform_Software_Enhancement
Synopsis	Maintain the software to correct problems, adapt to new operating environment, or enhance the software.
Exit Condition	state-of(Software_Technical_Process)=Developed
Informal Specification	The technical software engineering tasks include: - eliciting and analyzing the customer requirements and system requirements allocated to software, - developing the software requirements, - designing the software, - coding the software, - integrating the software, - testing the software to verify that it satisfies its requirement, - preparing documentation to support installation, operation, and maintenance of the software, - delivering the software to the customer, - supporting the operation and use of the software, and - maintaining the software.
Formal Specification	

Process State Definition Form	
Name	Develop_Software_Test
Synopsis	Develop the software test to validate that the system satisfies its requirements.
Main Role	Customer, Testing_Staff, Software_Product_Manager, Configuration_Management_Staff, Project_Manager
Entrance Condition	state-of(Software_Code)=Developed
Artifact List	Software_Test
Information Artifacts	System_Requirement
Sub-P-State List	
Name	Perform_Software_Test
Synopsis	Perform the software test to validate that the system satisfies its requirements.
Operation List	
Name	Draft_Test_Plan
Synopsis	Draft the test plans to be followed while testing the end-to-end functionality of the completed system.
Exit Condition	state-of(Software_Test)=Developed
Informal Specification	Establish_Quality_Measure
Formal Specification	

Process State Definition Form	
Name	Draft_Schedule_And_Resource_Plan
Synopsis	Establish the software engineering resources needed by the project and the project's software schedule.
Main Role	Requirement_Analyst, Software_Product_Manager, Project_Manager
Entrance Condition	state-of(Schedule_And_Resource_Plan)=Referenced and state-of(Allocated_Requirement)=Derived or state-of(Software_Engineering_Activity_Plan)=Drafted
Artifact List	Schedule_And_Resource_Plan
Information Artifacts	Object_Size_Category, Allocated_Requirement, Use_Case_Diagram
Sub-P-State List	
Name	Estimate_Software_Size
Synopsis	Estimate the size of all major software work products.
Name	Estimate_Effort_And_Cost
Synopsis	Estimate the effort and cost for the software project.
Operation List	
Name	Build_WBS
Synopsis	Establish and maintain a work breakdown structure for the software project.
Name	Estimate_Resource
Synopsis	Estimate the software engineering resource needed by the software project.
Name	Estimate_Schedule
Synopsis	Establish the project's software schedule.
Exit Condition	state-of(Schedule_And_Resource_Plan)=Drafted
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Establish_Organisation_Process_Definition
Synopsis	Establish a usable set of software process assets that improve process performance across the organisation and that provide a basis for cumulative, long-term benefits to the organisation.
Main Role	Senior_Manager, SEPG, Software_Product_Manager, Project_Manager
Entrance Condition	state-of(Organisaation_process_Definition)=Referenced
Artifact List	Organisation_Process_Definition
Information Artifacts	Organisation_Process_Focus
Sub-P-State List	
Name	Establish_Software_Measurement_Database
Synopsis	Establish and maintain the organisation's software measurement database.
Operation List	
Name	Establish_SSSP
Synopsis	Establish and maintain the organisation's set of standard software processes.
Name	Approve_Software_Life_Cycle
Synopsis	Establish and maintain the descriptions of the software life cycles approved for use in the organisation.
Name	Establish_Tailoring_Guideline
Synopsis	Establish and maintain the tailoring guidelines for the organisation's standard software process family.
Name	Establish_Process-related_Documentation
Synopsis	Establish and maintain the organisation's library of software process-related documentation.
Exit Condition	state-of(Organisation_Process_Definition)=Established
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Establish_Organisation_Process_Focus
Synopsis	The purpose of Organisation Process Focus is to establish and maintain an understanding of the organisation's software processes and coordinate the organisation's software process improvement activities.
Main Role	Senior_Manager, SEPG
Entrance Condition	state-of(Organisation_Process_Focus)=Referenced
Artifact List	Organisation_Process_Focus
Information Artifacts	
Sub-P-State List	
Name	Perform_Organisation_Process_Focus
Synopsis	Perform the activities for software process improvement.
Operation List	
Name	Draft_Process_Improvement_Plan
Synopsis	Draft a software process improvement plan to manage the activities of organisation process focus.
Exit Condition	state-of(Organisation_Process_Focus)=Established
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Establish_PDSP
Synopsis	Establish the project's defined software process tailoring from the organisation's set of standard software process. The main tasks establishing project's defined software process are to model the software processes by using PASTA.
Main Role	Senior_Manager, SEPG, Software_Product_Manager, Project_Manager
Entrance Condition	state-of(PDSP)=Referenced and state-of(Organisation_Software_Process)=Established
Artifact List	Organisational_Process, Software_Support_Process, Software_Management_Process, Software_Technical_Process
Information Artifacts	Organisation_Process_Definition
Sub-P-State List	
Name	Develop_Software_Support_Process
Synopsis	Develop Software Support Process to support the software project management.
Name	Develop_Software_Management_Process
Synopsis	Develop Software Management Processes to effectively and effeciently control the software project.
Name	Develop_Software_Technical_Process
Synopsis	Develop the software technical process to perform the engineering tasks to specify, design, build, deliver, and maintain the software using the project's defined software process in order to verify and validate that the software products satisfy their requirements.
Name	Develop_Organisational_Process
Synopsis	Develop Organisational Process to provide the ability to control the software project.
Exit Condition	state-of(PDSP)=Established and state-of(SSSP)=Updated
Informal Specification	IM.AC.02 IM.AC.03 IM.AC.08
Formal Specification	

Process State Definition Form	
Name	Establish_Productivity_Measure
Synopsis	Establish the productivity measure to estimate the development time for each project task.
Main Role	Software_Product_Manager, Project_Manager
Entrance Condition	state-of(Software_Technical_Process)=Referenced
Artifact List	Productivity_Measure
Information Artifacts	Project_Plan_Summary, Time_Recording_Log, Size_Estimating_Template
Operation List	
Name	Build_Time_Recording_Log
Synopsis	Build the time recording log and calculate development time.
Name	Build_Size_Estimating_Template
Synopsis	Build the size estimating template to guide the size estimating process and to hold the estimate data.
Name	Build_Project_Plan_Summary
Synopsis	Build the project plan summary to hold the estimated and actual project data in a convenient and readily retrievable form.
Exit Condition	state-of(Productivity_Measure)=Established
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Establish_Quality_Measure
Synopsis	Establish the quality measure to improve developer's quality
Main Role	Software_Product_Manager, Project_Manager
Entrance Condition	state-of(Software_Technical_Process)=Referenced
Artifact List	Quality_Measure
Information Artifacts	Defect_Recording_log
Operation List	
Name	Build_Defect_Recording_Log
Synopsis	Build the defect recording log to hold the data on each defect as you find and correct.
Exit Condition	state-of(Quality_Measure)=Established
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Establish_Software_Measurement_Database
Synopsis	Establish and maintain the organisation's software measurement database.
Main Role	SEPG, Software_Product_Manager, Project_Manager
Entrance Condition	state-of(Software_Measurement_Database)=Referenced
Artifact List	Software_Measurement_Database
Information Artifacts	
Sub-P-State List	
Name	Establish_Productivity_Measure
Synopsis	Establish the productivity measure to estimate the development time for each project task.
Name	Establish_Quality_Measure
Synopsis	Establish the quality measure to improve developer's quality
Operation List	
Name	Establish_LOC_Counting_Standard
Synopsis	Establish the LOC counting standard to count the program size.
Name	Establish_Object_Size_Category
Synopsis	Establish object size categories to give organisation a feamework for judging the size of the new objects in the planned product.
Exit Condition	state-of(Software_Measurement_Database)=Established
Informal Specification	PD.AC.05
Formal Specification	

Process State Definition Form	
Name	Estimate_Effort_And_Cost
Synopsis	Estimate the effort and cost for the software project.
Main Role	Requirement_Analyst, Software_Product_Manager, Project_Manager
Entrance Condition	state-of(Software_Size)=Estimated
Artifact List	Effort_And_Cost
Information Artifacts	Project_Plan_Summary, Software_Size
Operation List	
Name	Get_Productivity_Measure
Synopsis	Get the developers' productivity to estimate ptoject's effort and costs.
Name	Calculate_Time_Required
Synopsis	Calculate required effort and costs from estimated program size and developer's productivity.
Name	Document_Estimated_Time
Synopsis	Document the estimated time in the software measurement database.
Exit Condition	state-of(Effort_And_Cost)=Estimated and state-of(Project_Plan_Summary)=Updated
Informal Specification	PP.AC.05 PP.AC.07
Formal Specification	

Process State Definition Form	
Name	Estimate_Software_Size
Synopsis	Estimate the size of all major software work products.
Main Role	Requirement_Analyst, Software_Product_Manager, Project_Manager
Entrance Condition	(state-of(Allocated_Requirement)=Derived or state-of(Use_Case_Diagram)=Created) and state-of(Object_Category_Size)=Defined
Artifact List	Software_Size
Information Artifacts	Allocated_Requirement, Class_Diagram
Operation List	
Name	Compare_Category_Size
Synopsis	Estimate the new object size by comparing the object category size.
Name	Calculate_Software_Size
Synopsis	Use the PROBE method to calculate the software size.
Name	Document_Estimated_Size
Synopsis	Document the software size data to make regression analysis.
Exit Condition	state-of(Software_Size)=Documented
Informal Specification	PP.AC.05 PP.AC.06
Formal Specification	

Process State Definition Form	
Name	Identify_Training_Need
Synopsis	Identify the training needs which include the organisational needs and the project needs.
Main Role	Training_Staff, Software_Product_Manager, Project_Manager
Entrance Condition	state-of(Training_Plan)=Drafted
Artifact List	Training_Need
Information Artifacts	SSSP, PDSP
Operation List	
Name	Identify_Organisational_Need
Synopsis	Identify the strategic software training needs of the organisation.
Name	Determine_Project_Need
Synopsis	Determine the organisational training support needed to address the specific training needs of software projects and support groups.
Exit Condition	state-of(Training_Need)=Identified
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Manage_Configuration_Item
Synopsis	The configuration items have been identified, controlled and recorded.
Main Role	Configuration_Management_Staff
Entrance Condition	state-of(CM_Library_System)=Built
Artifact List	Configuration_Item
Information Artifacts	CM_Plan
Operation List	
Name	Identify_Configuration_Item
Synopsis	Identify the configuration items that will be placed under software configuration management.
Name	Control_Configuration_Item
Synopsis	Control changes to the content of configuration items.
Name	Record_Configuration_Item
Synopsis	Establish records describing configuration items.
Exit Condition	state-of(Configuration_Item)=Managed
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Manage_Supplier_Monitoring
Synopsis	Monitor the software suppliers' performance and results to ensure that the software satisfies its requirements.
Main Role	Quality_Assurance_Staff, Contract_Management_Staff, Software_Product_Manager, Project_Manager
Entrance Condition	state-of(Supplier_Selection)=Managed
Artifact List	Supplier_Monitoring
Information Artifacts	Acquisition_Plan
Operation List	
Name	Track_Supplier
Synopsis	Track the software supplier's performance against the supplier agreement.
Name	Review_Technical_Issue
Synopsis	Review technical issues with the software supplier.
Name	Review_Management_Issue
Synopsis	Review management issues with the software supplier.
Name	Evaluate_Supplier
Synopsis	Periodically evaluate the performance of the software supplier.
Name	Accept_Acquired_Software
Synopsis	Conduct acceptance reviews and tests for the acquired software and associated work products prior to them being accepted.
Exit Condition	state-of(Supplier_Monitoring)=Managed
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Manage_Supplier_Selection
Synopsis	Select software suppliers and establish the agreements with the software suppliers.
Main Role	Requirement_Analyst, Contract_Management_Staff, Software_Product_Manager, Project_Manager
Entrance Condition	state-of(Supplier_Selection)=Referenced and state-of(Acquisition_Plan)=Drafted
Artifact List	Supplier_Selection
Information Artifacts	Acquisition_Plan
Operation List	
Name	Determine_Acquisition_Need
Synopsis	Determine the software acquisition needs for the software project.
Name	Establish_Requirement
Synopsis	Establish the requirements for the acquired software.
Name	Acquire_COTS_Product
Synopsis	Select off-the-self software products to satisfy the software project's needs.
Name	Select_Contractor
Synopsis	Select software contractors based on an evaluation of their ability to meet the specified software requiremet.
Name	Establish_Contract
Synopsis	Establish an agreement with the software supplier as the basis for managing the contractual relationship.
Exit Condition	state-of(Supplier_Selection)=Managed
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Perform_Configuration_Management
Synopsis	This process involves: -Identifying the configuration of the software at given points in time. -Controlling changes to configuration items. -Building software work products from the software configuration library. -Maintaining the integrity of software baselines throughout the software life cycle.
Main Role	Configuration_Management_Staff
Entrance Condition	state-of(Configuration_Management)=Initiated
Artifact List	Configuration_Management
Information Artifacts	
Sub-P-State List	
Name	Manage_Configuration_Item
Synopsis	The configuration items have been identified, controlled and recorded.
Operation List	
Name	Draft_CM_Plan
Synopsis	Establish the plan for performing software configuration management.
Name	Build_CM_Library_System
Synopsis	Build a software configuration library system for the software baselines.
Name	Verify_CM_system
Synopsis	Verify the status of software configuration management activities and contents.
Exit Condition	state-of(Configuration_Management)=Verified
Informal Specification	Software configuration management in MIL-STD-498
Formal Specification	

Process State Definition Form	
Name	Perform_Organisation_Process_Focus
Synopsis	Perform the activities for software process improvement.
Main Role	Senior_Manager, SEPG
Entrance Condition	state-of(Process_Improvement_Plan)=Drafted
Artifact List	Organisation_Process_Focus
Information Artifacts	Process_Improvement_Plan
Operation List	
Name	Appraise_Software_Process
Synopsis	Appraise the organisation's software processes to identify strengths and weaknesses periodically and as needed.
Name	Draft_Action_Plan
Synopsis	Draft an action plan to address the findings of the software process appraisals.
Name	Implement_Action_Plan
Synopsis	Coordinate implementation of software process action plans across the organisation.
Name	Deploy_Software_Process_Asset
Synopsis	Coordinate the deployment of the organisation's software process assets.
Name	Evaluate_Software_Process_Asset
Synopsis	Review and evaluate the organisation's software process assets.
Exit Condition	state-of(Organisation_Process_Focus)=Performed
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Perform_Organisation_Training_Program
Synopsis	Perform the organisation training program to develop the skill and knowledge of the development team.
Main Role	Training_Staff, Project_Manager
Entrance Condition	state-of(Organisation_Training_Program)=Referenced
Artifact List	Organisation_Training_Program
Information Artifacts	PDSP
Sub-P-State List	
Name	Identify_Training_Need
Synopsis	Identify the training needs which include the organisational needs and the project needs.
Operation List	
Name	Draft_Training_Plan
Synopsis	Draft the training plan for organisational software training.
Name	Build_Training_Material
Synopsis	Establish and maintain software training materials that address the needs of the organisation.
Name	Execute_Training_Program
Synopsis	Train people in the software skill needed to perform their their roles.
Name	Build_Training_Record
Synopsis	Establish and maintain training records for the organisation.
Exit Condition	state-of(Organisation_Training_Program)=Performed
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Perform_Peer_Review
Synopsis	Perform the activities for peer reviews.
Main Role	Reviewer, Software_Product_Manager, Project_Manager
Entrance Condition	state-of(Peer_Reviews)=Referenced
Artifact List	Peer_Reviews
Information Artifacts	PDSP
Operation List	
Name	Draft_Peer_Review_Plan
Synopsis	Draft the peer review plan to conduct the peer reviews activities.
Name	Conduct_Peer_Review_Activity
Synopsis	Conduct and implement peer review activities.
Name	Record_Peer_Reviews_Data
Synopsis	Record data on the preparation, conduct, and results of the peer reviews of the software work products.
Exit Condition	state-of(Peer_Review)=Performed
Informal Specification	Establish_Quality_Measure Joint technical and management reviews in MIL-STD-498 Corrective action in MIL-STD-498
Formal Specification	

Process State Definition Form	
Name	Perform_Project_Interface_Coordination
Synopsis	The purpose of Project Interface Coordination is to ensure that software managers and staff effectively communicate, coordinate, and collaborate with other functions in the organisation to staisfy the customer's needs.
Main Role	Development_Group, Management_Group
Entrance Condition	state-of(Project_Interface_Coordination)=Referenced
Artifact List	Project_Interface_Coordination
Information Artifacts	Software_Management_Process, Software_Technical_Process
Operation List	
Name	Draft_Coordination_Plan
Synopsis	Draft a coordination plan to conduct the activities of project interface coordination.
Name	Perform_Coordination_Activity
Synopsis	Perform related activities of project interface coordination.
Exit Condition	state-of(Project_Interface_Coordination)=Performed
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Perform_Risk_Management
Synopsis	Software risk management involves identifying risks, analyzing their likelihood and potential impact, determining and evaluating risk contingencies, tracking risks, and proactively manageing the risks.
Main Role	System_Engineer, Software_Product_Manager, Project_Manager
Entrance Condition	state-of(Risk_Management)=Referenced
Artifact List	Risk_Management
Information Artifacts	Software_Management_Process, Software_Technical_Process
Operation List	
Name	Draft_Risk_Management_Plan
Synopsis	Draft a risk management plan to conduct the activities of the software risk management.
Name	Identify_Risk
Synopsis	Identify and document software project risks.
Name	Analyze_Risk
Synopsis	Analyze identified software project risks to determine risk exposure and priority.
Name	Mitigate_Risk
Synopsis	Mitigate the software project risk.
Exit Condition	state-of(Risk_Management)=Performed
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Perform_Software_Acquisition_Management
Synopsis	Perform Software Acquisition Management to manage the acquisition of software from sources external to the software project.
Main Role	Software_Supplier, Requirement_Analyst, Contract_Management_Staff, Software_Product_Manager, Project_Manager
Entrance Condition	state-of(Software_Acquisition_Management)=Referenced
Artifact List	Software_Acquisition_Management
Information Artifacts	PDSP
Sub-P-State List	
Name	Manage_Supplier_Selection
Synopsis	Select software suppliers and establish the agreements with the software suppliers.
Name	Manage_Supplier_Monitoring
Synopsis	Monitor the software suppliers' performance and results to ensure that the software satisfies its requirements.
Operation List	
Name	Draft_Acquisition_Plan
Synopsis	Establish the acquisition plan for managing the acquisition of software.
Exit Condition	state-of(Software_Acquisition_Management)=Performed
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Perform_Software_Project_Control
Synopsis	Track software project performance against the software development plan, and take corrective actions.
Main Role	Development_Group, Customer, Project_Manager
Entrance Condition	state-of(Software_Development_Plan)=Derived and state-of(Software_Technical_Process)=Referenced
Artifact List	Software_Development_Plan
Information Artifacts	Software_Management_Process, Software_Technical_Process
Operation List	
Name	Track_SDP
Synopsis	Actual performance and results of the software project are tracked against the software develpment plan. These issues include the size of software work products, software costs and efforts, critical computer resources, software engineering facilities, the schedule, risks, commitments, and project reviews.
Name	Correct_SDP
Synopsis	Take corrective action as necessary when actual accomplishments and progress differ significantly from that planned.
Name	Maintain_SDP
Synopsis	Revise the software development plan to reflect accomplishments, progress, changes, and corrective actions as appropriate.
Exit Condition	state-of(Software_Development_Plan)=Revised
Informal Specification	
Formal Specification	

Process State Definition Form	
Name	Perform_Software_Quality_Assurance
Synopsis	This process involves: -reviewing the software activities and work products against the applicable requirements, process descriptions, standards, and procedures. -identifying and documenting noncompliance issues. -providing feedback to project staff and managers. -ensuring that noncompliance issues are addressed.
Main Role	Senior_Manager, Quality_Assurance_Staff
Entrance Condition	state-of(Software_Quality_Assuranced)=Initiated
Artifact List	Software_Quality_Assurance
Information Artifacts	
Sub-P-State List	
Name	Control_SQA_Issue
Synopsis	Objectively control the SQA issues which include software activities and software work products.
Operation List	
Name	Draft_SQA_Plan
Synopsis	Draft the plan for software quality assurance in the early stage of the overall project planning.
Name	Reprot_SQA_Result
Synopsis	Report the results of the software quality assurance activities and address noncompliance issues.
Exit Condition	state-of(Software_Quality_Assurance)=Reported
Informal Specification	Software quality assurance in MIL-STD-498
Formal Specification	

Process State Definition Form	
Name	Perform_Software_Test
Synopsis	Perform the software test to validate that the system satisfies its requirements.
Main Role	Customer, Requirement_Analyst, Testing_Staff, Software_Product_Manager, Project_Manager
Entrance Condition	state-of(Test_Plan)=Drafted and state-of(Software_Code)=Developed
Artifact List	Software_Test
Information Artifacts	Test_Plan
Operation List	
Name	Perform_Integration_Test
Synopsis	Perform integration test to ensure that the software components interact correctly when combined.
Name	Perform_System_Test
Synopsis	Perform system test to validate the software satisfies the allocated requirements.
Name	Perform_Acceptance_Test
Synopsis	Perform acceptance test to demonstarte to the customer that the software system satisfies the customer requirements for the software project.
Exit Condition	state-of(Software_Test)=Performed
Informal Specification	Establish_Quality_Measure
Formal Specification	