Process Modelling to Support Object-Oriented Software Production

Kann-Jang Yang, Rob Pooley
Department of Computer Science
Edinburgh University
JCMB King's Buildings, Mayfield Road
Edinburgh EH9 3JZ, UK
{ky,rjp}@dcs.ed.ac.uk

Abstract

Recently, the concepts of software process modelling and object-oriented methodology have been widely discussed in the literature to tackle the problems of software development. In this article, we will concentrate on the method that uses formal specification techniques to model the software process. PASTA (Process and Artifact State Transition Abstraction) is a bridge between guidance and automation of the software development process. By using PASTA to model the Booch method, a double advantage can be obtained in software development. Furthermore, the rapid advancement of workstation and communication technologies has accelerated the spread of the distributed development paradigm for software development. A process-centred software environment with these advances could result in a shortening of the development cycle and potentially an improvement in productivity and quality.

Keywords: software process, process modelling, object-oriented methodology, software factory

Submission type: Research paper

1 Introduction

The software industry has experienced a dramatic change in last few years. As The Economist reported[1]: In 1990 just a few academics had heard of the Internet; now, anything up to 50m people use it. In a year's time, that figure could be 100m. At the moment, no one can deny the WEB has changed some things for the software industry. As a consequence, software development is not only a local business but a team work with international competition.

Lai [2] described the steps of the software industry. The first wave of software was developed using the waterfall model, introduced in 1970. Today we are in the midst of a second wave, a maturity movement, as we attempt to formally define the development process and the best ways to continuously improve it. The third wave will involve the mechanisation of the software industry, characterised by the mass production of uniformly high quality products.

Since time-to-market is one of the most critical issues in the software industry, the pressure of developing software is getting higher for the project manager. To develop software projects quickly, the software factory concept, analogous to the mass production line of the automobile factory, would be a good solution. The software industry would likely evolve in two directions. Firstly, a software component industry would supply fine-grained components that would eventually end up in enormous numbers of software products. Secondly, a CASE tool would offer process control and automatically generate the relevant codes. As a consequence, the traditional software industry would change dramatically. Software development would not be a hand-craft industry any more, but assemble software products from purchased components with good process control.

However, some experts have different opinions. In his book Pressman [3] discussed software characteristics. He said that software is developed or engineered, it is not manufactured in the classical sense and concluded that software is a logical rather than a physical system element. Quintas[4] and Humphrey[5] also suggested that software is designed and developed rather than being manufactured. Much of software activity is non-routine, and has traditionally been highly dependent on the skills of the programmer. This implies that it is impossible to develop software in the same manner as physical components.

Standardised designs and interchangeable components are the key points in mass production for traditional industries. By means of scheduling of component production and arranging factory layouts, the products are produced quickly and uniformly. Everybody must agree that software development is different from traditional industry. It is highly dependent on the skills of large numbers of IT professions: analysts, designers, programmers, project managers and so on. They are labour-intensive activities but, in contrast to the traditional industry, there are more communication processes and social interactions within the developers' community.

Cusumano[6] found that with standardised but specialised skills, procedures, and tools controlling the work process, software factories are allowed, by increased flexibility, to adapt to different customer needs or changes in the work flow. Consequently, enacting the software process to ensure software quality and productivity has been a very active research area in the software development environment communities. In the EPOS[7, 8] project, the process models are expressed in SPELL, a process modelling language. The internal process model is a network of activity descriptions, being linked to descriptions of other tasks, products, tools, and roles. The activities interact with each other and with tools and humans. In the SPADE[9, 10] project, a process language, SLANG, has been defined, which is a fully reflective language built over a high-level extension of Petri nets. It provides execution mechanisms to cope with the evolution problem. What others have done is to develop integration mechanisms. A European research effort funded under the Eureka program, the ESF (Eureka Software Factory)[11, 12, 13] is a project that aims at a Factory Support Environment (FSE). The FSE architecture, built around standard interfaces between subsystems, will enable a high degree of independence from any particular set of software engineering tools or computer systems. The communication between the systems in the FSE takes place over a communications channel called the Software Bus. The same efforts have been made on the ISPW[14] and SPMS[15] projects. However, previous work focused on project management and group coordination processes. There are few studies that have been done on basic processes, such as analysis and design. The software process model must be able to control not only coordination and cooperation between groups but also basic processes.

This paper presents some preliminary results from a project which is examining support for the software enginering process in the context of efforts to produce "software factories". Its objectives are to use the benefits of conventional factory style production organisation in the context of software production. It takes the object oriented view of software as a basis for a component based software production method.

The rest of this paper is organised as follows. Section 2 clarifies the motivation in writing this paper. Section 3 outlines the environment used to experiment with these ideas. Section 4 presents methods and mechanisms in managing process model. Finally, some conclusions and further works are outlined in section 5.

2 Motivation

In the 60's and 70's, public recognition of a 'software crisis' came from people who had been working on the largest and most complex systems development projects. This problem is concerned with a failure to meet the demand for applications and to deliver systems on time and within agreed costs. Industry and government organisations had realized that without the ability to manage the software process, it would be difficult to control a software project even if there are good methods and tools. To control software process, the Capability Maturity Model(CMM)[16], developed by the Software Engineering Institute(SEI) in Carnegie Mellon University, has become a standard to improve the software development process for many organisations. Moreover, the government has used the SEI's Software Capability Evaluation to select a contractor[17]. The CMM was designed to help developers select process-improvement strategies by determining their current process maturity and identifying the most critical issues to improve their software quality. However, the CMM does not specify how you must perform software development or management activities. That means that the CMM represents a destination, not a road map. Unfortunately, there are still lots of misconceptions in the CMM[18].

The CMM for software provides software organisations with guidance on how to gain control of their processes for developing and maintaining software. As SEI defined[19]:

> A software process can be defined as a set of activities, methods, practices, and transformations that people used to developed and maintain software and the associated products.

Thus, for developing a complex software product, a well-defined software process is essential. With a standard process, the members in the developing team know their roles and responsibilities. It is easy to follow the procedure to develop a software project. Moreover, object-oriented analysis and design have much in common. The same object-oriented concepts, techniques, and notations used in analysis apply equally well in design. As a consequence, the same development tools can be used to support both activities. Often, these similarities make it hard to tell which activity is being carried out[20]. As Booch[21] mentioned, a well-defined process must provide guidance as to the order of a team's activities. Processes would give OOA and OOD a good framework to develop a software project.

Moreover, the rapid advancement of telecommunication technologies and software development techniques have accelerated the spread of the distributed development paradigm for software development. Aoyama [22, 23] suggested that the software development process should be changed from centralised, serial development to distributed, concurrent development. How can we do that? Basicly, the Web would become the ideal platform for collaborative work. By means of integrating software process with Web, developers can track the process on the net. It can hook developers with a network via the Internet.

3 Experiment environment

3.1 PASTA

Just as software is a description based on an abstracted model of a real-world problem, software process description is based on an abstracted model of a software development process. It provides benefits because it:

- provides a precise, unambiguous description of the process.
- provides a basis for building and integrating process tools.
- allows all parties concerned (technologists, developers, managers) to agree (standardise) on the process.
- provides a basis for process improvement/process evolution.
- provides information for process and project management to reason about the status of a project.

PASTA (Process and Artifact State Transition Abstraction) is a bridge between guidance and automation of the software development process. It enables a process engineer to model any process to any level of detail [24].

By building on the simple paradigm that people (roles) and resources perform activities that lead to products (artifacts), and by providing a rich set of relationships among those basic ingredients of a process, it is possible to model the process and to maintain state tables that contain information about the completeness of the product. Therefore, it is possible to know, at any given time, the state of completeness of all artifacts, and hence the state of completeness of the project. This provides the project manager with great insight and control over the events in the project.

3.1.1 Primary Elements of a Generic Model

- Artifact and A-State

  A software development process is a sequence of decision-making activities. Artifacts capture the decisions made during the software development process. Examples of artifacts include modules, objects, packages, or subroutines. To characterise the state of a software development process, the engineer must characterise the state of the artifacts produced during the software process. The states belong to the low level in the PASTA model; such states are called artifact state (A-states).

  However, merely characterising the state of the artifacts is insufficient to describe a complete software process. The process modeler 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. For example, activities that the engineer might perform on an interface specification of a software module are creating the specification, checking it for completeness and consistency, and conducting a formal review of it. It is necessary to define the upper level state model in PASTA.

- Process and P-State

  Because A-states alone are insufficient to describe the software process completely, descriptions of activities, operations on artifacts, analyses that the engineer can perform on artifacts within the state, and the roles of the people involved augment the A-states. These augmented states are called process states (P-states).

  Whether it is possible to perform an activity depends on the state of the artifacts. At any point in time, the set of performable activities represent the choice of artifacts on which the developers may work. Those that are not performable represent the artifacts on which he may not work. PASTA prescribes a permissive ordering of activities by specifying which activities are performable and which are not at any point. So, the process modeler specifies a permissive ordering to support a realistic process. In addition to specifying artifacts and activities, the process modeler also specifies the roles played by people involved in software development.

  By augmenting state descriptions with activities and other process-related information, the process modeler can analyse the needs of the developers at any point in the process. The goal of the analysis is to present to the developers the information that they need at the time they need it. It also helps focus on the concerns of what guidance to give to the developers and what analyses the process modeler can perform on the software at any time. Finally, it helps separate the concern of what information the process modeler should present to the developers from how the process modeler should present information to them.

3.1.2 The Process Notation

For a software process to be fully described under the PASTA process model, a notation is necessary for the process modeler to precisely define the process in his mind. The developers can use this notation to represent the necessary process-centred software environment for a project. The process modeler can do mapping by interpreting the software process description in a formal process notation and generating the activities coordination program to integrate tools. There are three forms of notation in PASTA: Form, Textual Language, and Graphical Diagrams.

- Form

  The process notation is in terms of a set of forms. Each form represents a template which asks for some data to be provided to describe the process. So far, there are nine form templates in PASTA model:

  - Artifact definition
  - Relation definition
  - Process state definition
  - Operation definition
  - Analysis definition
  - Action definition
  - Role definition
  - Resource definition

- Communication definition

In these forms, some of the data is formal. It is exactly the same as the data for the textual language. Some of the data is informal, i.e., an English explanation for an item.

- Textual Language

  Textual language is the centrepiece of PASTA's generic formal process notation. Together the form and graphical diagrams should contain a full set of information that the textual language will cover. The process modeler generates the textual language representation of the process model on the back end of the form/diagram browser/editor for the process notation. This textual representation is passed to the compiler/translator/automatic-instantiator to generate the process-centred software environment.

- Graphical Diagrams

  - P-State Transition Diagram

    Figure 1 shows an example of a P-state transition diagram. A P-state is represented as a rectangular box with two kinds of boxes that can intersect with it. They are the entrance and exit conditions. A set of cells divides the condition box. Each cell represents one artifact in one A-state which composes the entrance/exit conditions. The intersection lines between the condition box and the P-state box divides the cells into two parts. The parts outside of the P-state contain the name of the artifacts, and the parts inside of the P-state represent the A-state of the artifact. Each artifact cell may have more than one line connected to it. This means that an A-state change of an artifact may affect more than one P-state.

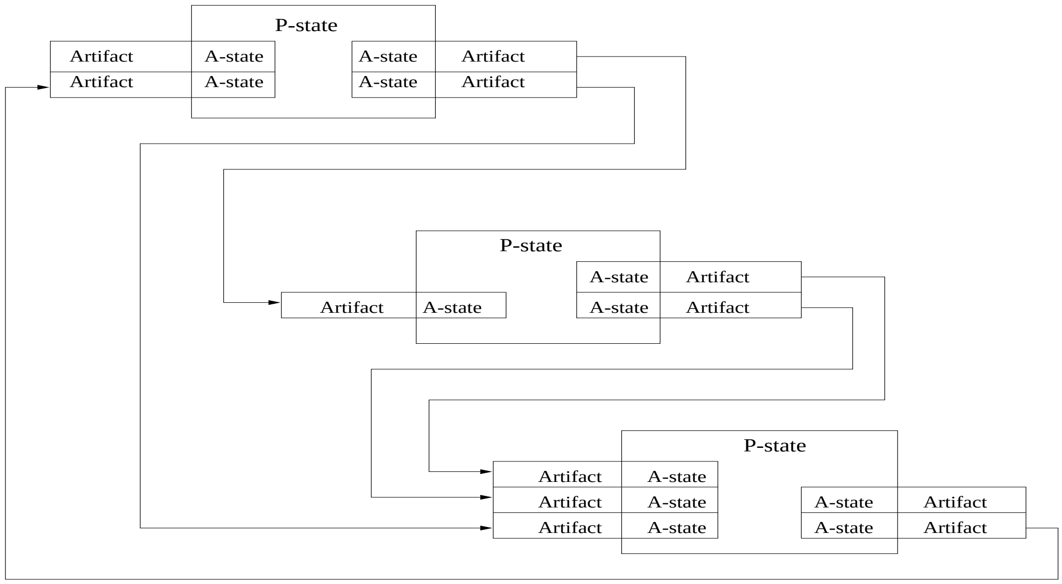

Figure 1: P-state Transition Diagram Example

  - A-State Transition Diagram

    Figure 2 shows an example of an A-state transition diagram. An A-state transition diagram, similar to a conventional state transition diagram, represents a view of the dynamic model of an artifact. By using the A-state transition diagram, we can follow the event-ordered behaviour of the artifact. There are two essential elements in the A-state transition diagram. The node represents the A-state and the arc represents the operation which makes the A-state change.

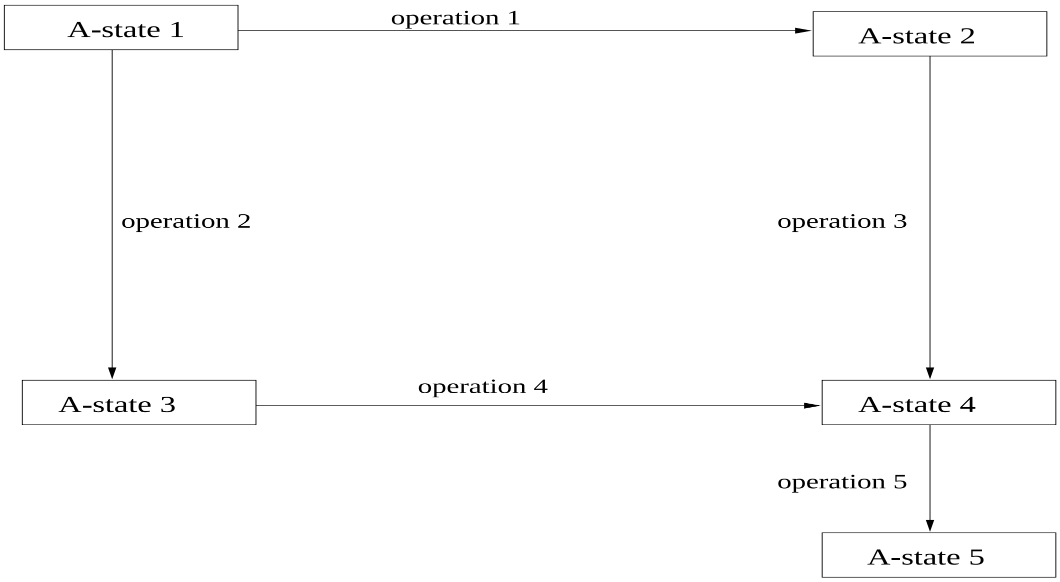

Figure 2: A-state Transition Diagram Example

3.2 Booch's object-oriented methodology

The Booch method is one of the most well-known and widely-used object-oriented analysis and design methods. This method provides a handful of notations which cover the important logical elements of an object-oriented architecture. Moreover, it adopts an iterative and incremental development which is composed of macro and micro process. The macro process, the primary concern of the software management team, is closely related to the traditional waterfall life cycle, and serves as the controlling framework for the micro process which is the primary concern of the individual developer or a small group of developers. In this section, we present only a brief description. The details can be found in Booch's books [25].

- Macro process
  - Conceptualization: Bracket the project's risks by building a proof of concept.
  - Analysis: Develop a common vocabulary and a common understanding of the system's desired behaviour by exploring scenarios with end users and domain expert.
  - Design: Establish a skeleton for the solution and lay down tactical policies for implementation by drafting the system's architecture.
  - Evolution: Refine the architecture and package releases for deployment; this phase typically requires further analysis, design, and implementation.
  - Maintenance: Continue the system's evolution in the face of newly-defined requirements.

- Micro process
  - Identifying classes and objects
  - Identifying the semantics of classes and objects
  - Identifying relationships among classes and objects
  - Implementing classes and objects

As mentioned previously, the Booch method offers a handful of notations to capture all important strategic and tactical decisions. When related notations are woven together they build an architecture for the software development. The notations are as follows:

- Class diagram
- State transition diagram
- Object diagram
- Interaction Diagram
- Module Diagram
- Process diagram

4 Process Model Creation Using PASTA

In the object-oriented paradigm, object-oriented analysis and design have much in common. These similarities make it hard to tell which activity is being carried out. For CASE tools, a class diagram is just a class diagram. No one knows what part of the class diagram is from analysis or design. This is why we need to model the software process. To produce a model of a software process we use the process notations to describe and manipulate the process artifacts. Following the process notations, we would be able to specify what artifacts should be developed and direct the tasks of individual developers as a whole, so that it can ensure that the process evolves according to desirable rules and procedures. The process model is a prescription for the artifacts to be used, the activities to be performed and their sequencing, and the roles that people play.

4.1 The Notations

4.1.1 The Artifact Definition

To model the Booch method by using PASTA, firstly, we must define the artifacts of the Booch method. The clearest artifacts would be the artifact which contains data objects related to software products. Booch in his book[25] defined that the analysis of a system will include sets of object diagrams, class diagrams, and state transition diagram, and the design of a system will include sets of class diagrams, object diagrams, module diagrams, and process diagrams. These would give us a direction to follow even though it is still not clear for class and object diagrams in analysis and design. However, these kinds of diagrams would be the basic artifacts in the process. Secondly, we must identify dependency and composition relations among the artifacts. In practice, we must further define the subartifacts of these basic notations and find their relations. Each artifact might be divided into many subartifacts. The more subartifacts you can find, the more detailed the model you can define. Finally, we use the A-state model to identify a set of states for each artifact. The A-state can be thought of as defining a state machine for each artifact. The developer traverses the states to complete the artifact by following the operations. Figure 3 shows the A-state transition diagram of artifact Analysis_Model. Combining the artifact definition form of artifact Analysis_Model(Figure 4), they give developers a clear process to follow.

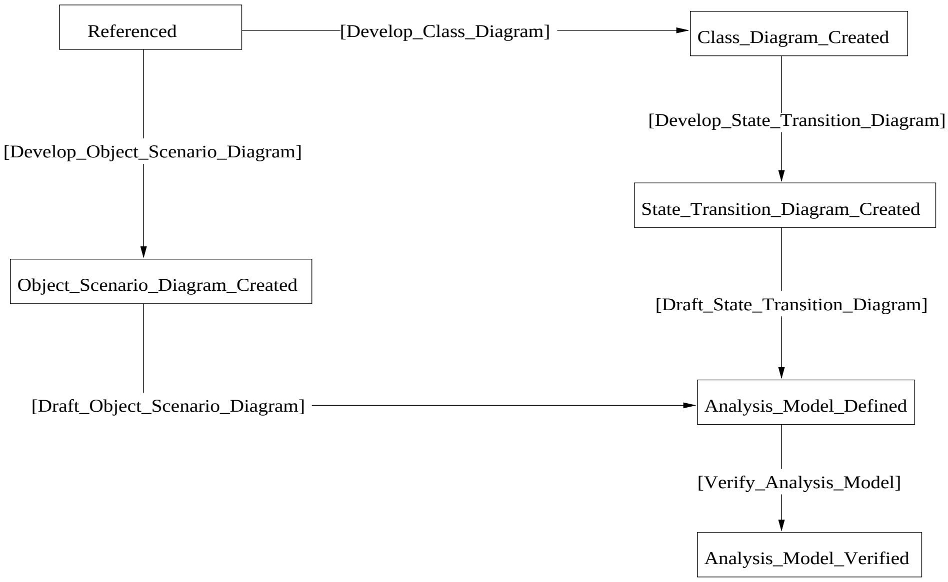

Figure 3: A-state transition diagram of artifact Analysis_Model

4.1.2 The Process State Definition

To reconcile the creative needs of individual programmers with management's needs for stability and predictability, Booch suggested useing the macro and the micro process. These processes provide a hierarchical structure for the set of P-states. From the macro process, we might represent the design as a single P-state at the top level of the model. Then, we might decompose the design P-state into a hierarchy of P-states, as the micro process. This gives us an architecture for the process state definition. In terms of the architecture, firstly, we must identify a set of operations for each artifact and identify entrance and exit conditions for each operation. Since operations are the actions for accomplishing artifacts, developers by ways of performing an action when the entrance condition for the operation is true would connect to an existing tool or a function library to complete artifacts. Figure 5 shows the P-state transition diagram of Develop_Analysis_Model. Its details are described in the process state definition form(Figure 6).

4.2 Implementation for Process Notation

A process model must supply an architecture providing a fully integrated set of tools for sharing information and developing application. In PASTA, the model specifies the data upon which the tools must operate, the operations to be performed by the tools, and the effects of those operations. From notations, the implementation generates the software environment in terms of tool integration and activities coordination. For each operation in each P-state, a set of commercially available tools that we can use to perform the operations can be selected. Meanwhile, to maintain artifact states, the notations would prevent forbidden transitions and suggest possible transitions.

In the Process Model World, people only pay attention to the salient features of the process such as the operations that they need to complete and the artifacts that they need to generate. Everything is an artifact or an operation. In the example of Develop_Analysis_Model, while the need to develop an analysis model has been identified, we enter two P-states, Develop_Class_Diagram and Develop_Object_Scenario_Diagram. From zooming in the P-states, the process state definition form would be traced. The artifact list provides the artifacts which we need to generate and the operation list gives us the actions which we need to do and connects to the tool which we can use to perform the operation. The sequence of doing something is always governed by pre- and post-conditions. The modeler can easily use process model to integrate different software development tools to develop a software project.

5 Conclusion

In this paper we have presented our work on using PASTA to model the Booch method. A defined process enables people to work more effectively by exploiting tools and experience and predictably to control schedules, budgets, and product quality. By using PASTA to the model Booch method, we standardise the software process, and provide a consistent environment based on an organisation-wide understanding of the activities, roles, and responsibilities in software industry. It is certainly not possible completing the whole model in a step. However, we can modify it as we need from time to time. Once it is completed, the environment would be generated for developing the software project.

Since the Internet is getting more popular nowadays, the Web has been viewed as a good information infrastructure for collaborative engineering. A suite of collaboration tools which are integrated with the Web enable the development team to share ideas and documents. Moreover, for the security's sake, the booming intranet concept gives us a good environment for the software industry. In the future, the PASTA model will be connected with the intranet or private Web-based Internet networks. The definition forms and diagrams would be presented by using HTML. If powerful CASE tools are made available as services on the Web with the PASTA model, developers can follow the P-state diagrams, A-state transition diagrams, and definition forms to develop the software product with integrating CASE tools. By sharing design information across the development team, developer expertise with those tools can be preserved. Furthermore, via the distributed, world-wide environment, the development team could be in different buildings, different cities, even different countries. As a consequence, for time being or budget, the idea would give us a good direction for the software development.

References

[1] "Why the Net should grow up," *The Economist*, October 19th 1996.

[2] R. Lai, "The Move to Mature Processes," *IEEE SOFTWARE*, pp. 14–17, July 1993.

[3] R. S. Pressman, *Software Engineering, A Practitioner's Approach.* McGrew-Hill Book Company Europe, european ed., 1994.

[4] P. Quintas, *Social Dimeensions of Systems Engineering.* ELLIS HORWOOD Limited, 1993.

[5] W. S. Humphrey, *Managing the Software Process.* Addison-Wesley Publishing Company, 1989.

[6] M. A. Cusumano, *Japan's Software Factories.* Oxford University Press, 1991.

[7] M. L. Jaccheri and R. Conradi, "Techniques for Process Model Evolution in EPOS," *IEEE Transactions on Software Engineering*, vol. 19, pp. 1145–1156, December 1993.

[8] C. Liu and R. Conradi, "Automatic Replanning of Task Networks for Process Model Evolution in EPOS," *4th European Software Engineering Conference*, pp. 434–450, September 1993.

[9] S. C. Bandinelli, A. Fuggetta, and C. Ghezzi, "Software Process Model Evolution in the SPADE Environment," *IEEE Transactions on Software Engineering*, vol. 19, pp. 1128–1144, December 1993.

[10] S. C. Bandinelli, M. Braga, A. Fuggetta, and L. Lavazza, "The Architecture of the SPADE-1 Process-Centered SEE," *Software Process Technology, Third European Workshop, EWSPT '94*, pp. 15–30, February 1994.

[11] C. Fernstrom, K. Narfelt, and L. Ohlsson, "Software Factory Principles, Architecture, and Experiments," *IEEE Software*, pp. 36–44, March 1992.

[12] R. Adomeit, W. Deiters, B. Holtkamp, F. Schulke, and H. Weber, "K/2R: A Kernel for the ESF Software Factory Support Environment," *ICSI '92 Proceedings of the Second International Conference on Systems Integration IEEE*, pp. 325–336, June 1992.

[13] C. Fernstrom, "The Eureka Software Factory: Concepts and Accomplishments," *3rd European Software Engineering Conference, ESEC '91*, pp. 23–36, October 1991.

[14] I. robertson, "An Implementation of the ISPW-6 Process Example," *Software Process Technology Third European Worrkshop, EWSPT '94*, pp. 187–206, February 1994.

[15] H. Krasner, J. Terrel, A. Linehan, P. Arnold, and W. H. Ett, "Lessons Learned from a Software Process Modeling System," *Communications of the ACM*, vol. 35, pp. 91–100, September 1992.

[16] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber, "Capability Maturity Model, Version 1.1," *IEEE SOFTWARE*, pp. 18–27, July 1993.

[17] D. Rugg, “Using Capability Evaluation to Select a Contractor,” *IEEE SOFTWARE*, pp. 36–45, July 1993.

[18] K. Wiegers, “Misconceptions of the CMM,” *Software Development*, pp. 57–64, November 1996.

[19] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber, “Capability Maturity Model for Software, Version 1.1,” *Technical Report, CMU/SEI-93-TR-024*, February 1993.

[20] A. Goldberg and K. S. Rubin, *Succeeding with Objects.* Addison-Wesley Publishing Company, 1995.

[21] G. Booch, *Object Solutions, Managing the Object-oriented Project.* Addison-Wesley Publishing Company, Inc., 1996.

[22] M. Aoyama, “Concurrent-Development Process Model,” *IEEE SOFTWARE*, pp. 46–55, July 1993.

[23] M. Aoyama, “Distributed Concurrent Development of Software System: An Object-Oriented Process Model,” *Proc.Compsac, IEEE CS Press*, pp. 330–337, 1990.

[24] R. Lai, “Process Definition and Process Modeling Methods,” *Technical Report, Software Productivity Consortium, SPC-91084*, September 1991.

[25] G. Booch, *Object-Oriented Analysis and Design with Application.* The BenjaminCummings Publishing Company, Inc., second ed., 1993.

Artifact Definition Form	
Name	Analysis_Model
Synopsis	The analysis model is the process of defining a precise, concise, and object-oriented model of that part of the real-world enterprise that is relevant to the system. It is through this process that the developers gain the detailed knowledge of the problem domain needed to create a system capable of carrying out the required functions.
Complexity Type	Composite
Data Type	Diagram
A-State List	
Name	Analysis_Model_Verified
Synopsis	The analysis model which includes class diagrams, object scenario diagrams, and state transition diagrams has been verified.
Condition	state-of(Analysis_Model)=Analysis_Model_Verified
Name	Referenced
Synopsis	The need to develop an analysis model for a software project has been identified.
Condition	state-of(Analysis_Model)=Referenced
Name	Class_Diagram_Created
Synopsis	The class diagrams that are used to indicate the common roles and responsibilities have been created.
Condition	state-of(Class_Diagram)=Class_Diagram_Created
Name	Analysis_Model_Defined
Synopsis	The analysis model has been defined.
Condition	state-of(Class_Diagram)=Class_Diagram_Created and state-of(Object_Scenario_Diagram)=Object_Scenario_Diagram_Created state-of(State_Transition_Diagram) =State_Transition_Diagram_Created
Name	Object_Scenario_Diagram_Created
Synopsis	The object scenario diagrams that are used to provide a trace of the system's behaviour have been created.
Condition	state-of(Object_Scenario_Diagrm) =Object_Scenario_Diagram_Created
Name	State_Transition_Diagram_Created
Synopsis	The state transition diagrams that are used to indicate the dynamic behaviour of the system have been created.
Condition	state-of(State_Transition_Diagram)=Created
A-State List	
Name	Class_Diagram
Synopsis	A class diagram is used to show the existence of classes and their relationships in the logical view of a system. During analysis, we use class diagrams to indicate the common roles and responsibilities of the entities that provide the system's behavior.
Name	State_Transition_Diagram
Synopsis	A state transition diagram is used to show the state space of a given class, the events that cause a transition from one state to another. and the actions that result from a state change. A single state transition diagram represents a view of the dynamic model of a single class or of the entire system. During analysis, we use state transition diagrams to indicate the dynamic behavior of the system.
Name	Object_Scenario_Diagram
Synopsis	An object scenario diagram is used to show the existence of objects and their relationships in the logical design of a system. During analysis, we use object scenario diagrams to indicate the semantics of primary and secondary scenarios that provide a trace of the system's behavior.

1

Figure 4: Definition form of artifact Analysis_Model

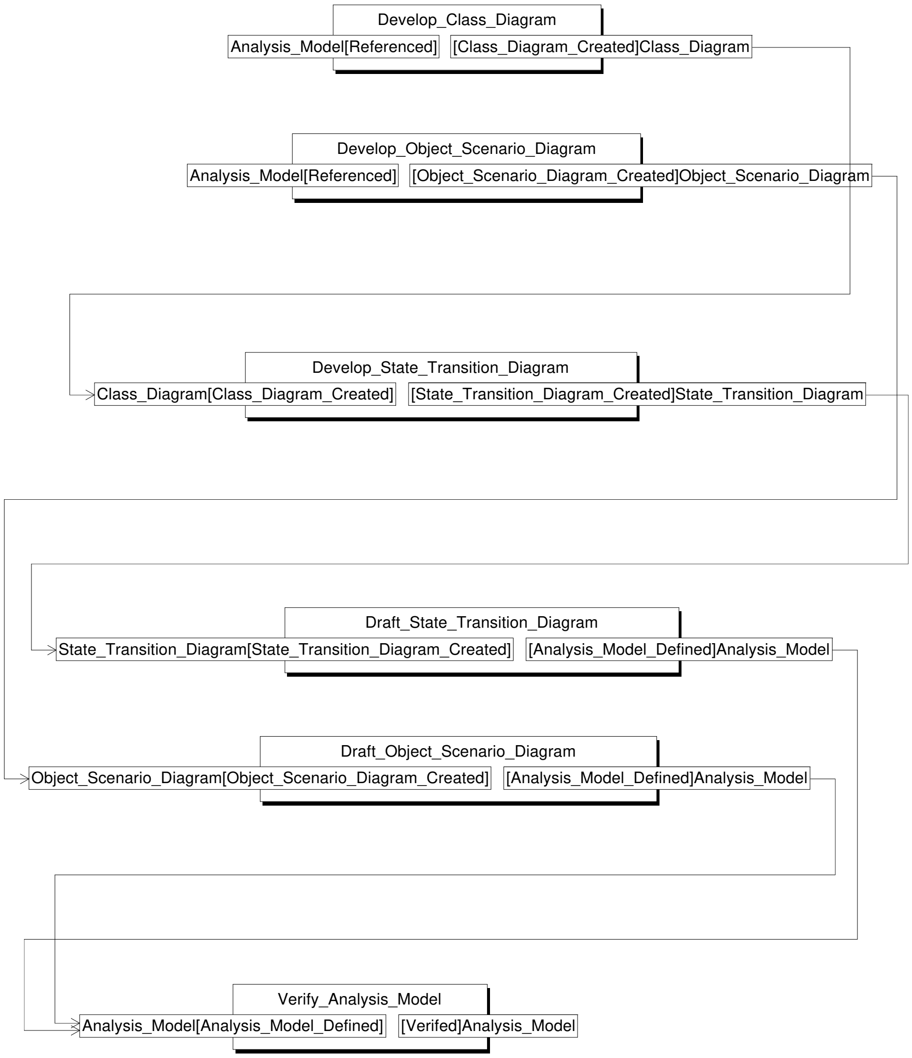

Figure 5: P-state transition diagram of Develop_Analysis_Model

Process State Definition Form	
Name	Develop_Analysis_Model
Synopsis	This step must identify three diagrams: class diagram, state tranisition diagram, and object scenario diagram, to establish the boundaries of the problem at hand.
Main Role	Analyst, Project Manager
Entrance Condition	state-of(Analysis_Model)=Referenced
Artifact List	State_Transition_Diagram, Object_Scenario_Diagram, Class_Diagram
Information Artifacts	Use_Case
Sub-P-State List	
Name	Develop_Class_Diagram
Synopsis	This step would develop a class diagram by identifying key classes, building data dictionary, defining relationship, defining cardinality of relationship, defining attribuute, and defining inheritance.
Name	Develop_State_Transition_Diagram
Synopsis	This step would develop state transition diagrams which is used to show the state space, events, and actions. The state transition diagram is only for those classes that exhibit event-order behaviour.
Name	Develop_Object_Scenario_Diagram
Synopsis	This step would develop object scenario diagrams which is used to show the existence of objects and their relationships.
Operation List	
Name	Draft_State_Transition_Diagram
Synopsis	The state transition diagram has been drafted.
Name	Draft_Object_Scenario_Diagram
Synopsis	The object scenario diagram has been drafted.
Name	Verify_Analysis_Model
Synopsis	The analysis model included class diagrams, state transition diagrams, and object scenario diagrams has been verified.
Exit Condition	((state-of(Class_Diagram)=Identified)) and (state-of(State_Transition_Diagram_in_Design)=Identified) and (state-of(Object_Scenario_Diagram)=Identified))
Informal Specification	Developing analysis model involves creating class diagrams, object scenario diagrams and state transition diagram.
Formal Specification	

1

Figure 6: Definition form of Develop_Analysis_Model