ISSN:0975-9646

Survey on Different Process Models Used In Software Development

Anu P Salim. , Chithra P. , Sreeja S.

M Tech CSE, Sree Buddha College of Engineering, Alappuzha , India

Abstract-**Professional system developers and the customers they serve share a common goal of building information systems that effectively support their objectives. In order to ensure that cost-effective, quality systems are developed which address an organization's business needs, developers employ some kind of system development Process Model to direct the project's life cycle. A software process model is actually an abstract representation of a Process which often represent a networked sequence of activities, objects, transformations, and events that embody strategies for accomplishing software evolution .There are a variety of process models in software development and the purpose of this paper is to perform a survey on different process models used in software development.**

I. INTRODUCTION

The process of developing and supporting software requires many distinct tasks to be performed by different people in some related sequences. When software engineers left to perform tasks based on their own experience, background and values they do not necessarily perceive and perform the task the same way or in the same order. They sometimes do not even perform the same task. This inconsistency causes projects to take a longer time with poor end products and in worst situations total project failure. Software process models give guidance for systematically coordinating and controlling the tasks that must be performed in order to achieve the end product and the project Subjectives. It presents a description of a process from some particular perspective as:

1. Specification.
2. Design.
3. Validation.
4. Evolution.

II.PRIMARY APPROACHES

Most system development *Process Models* in use today have evolved from three primary approaches: *Ad-hoc Development*, *Waterfall Model*, and the *Iterative* process.

1. Ad-hoc Development

Early systems development often took place in a rather chaotic and accidental manner, relying entirely on the skills and experience of the individual staff members performing the work .The Software Engineering Institute at Carnegie Mellon University points out that with *Ad-hoc Process Models*, "process capability is unpredictable because the software process is constantly changed or modified as the work progresses. Performance depends on the capabilities of individuals and varies with their innate skills, knowledge, and motivations .

Even in undisciplined organizations, however, some individual software projects produce excellent results. When such projects succeed, it is generally through the heroic efforts of a dedicated team, rather than through repeating the proven methods of an organization with a mature software process. In the absence of an organization-wide software process, repeating results depends entirely on having the same individuals available for the next project.

2. Waterfall Model

The waterfall model is the classical model of software engineering. This model is one of the oldest models and is widely used in government projects and in many major companies. As this model emphasizes planning in early stages, it ensures design flaws before they develop. In addition, its intensive document and planning make it work well for projects in which quality control is a major concern. The pure waterfall lifecycle consists of several non overlapping Stages. It is attributed with providing the theoretical basis for other *Process Models*, because it most closely resembles a "generic" model for software development.It consists of the following steps:

System Conceptualization: System on conceptualization refers to the consideration of all aspects of the targeted business function or process,

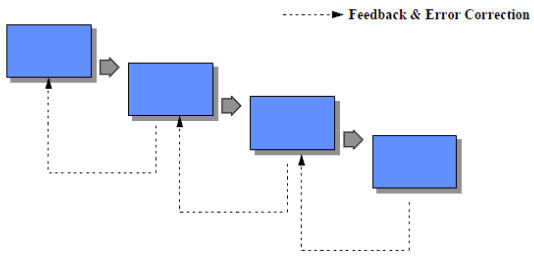

Fig.1.The *Waterfall Model*

with the goals of determining how each of those aspects relates with one another, and which aspects will be incorporated into the system.

Systems Analysis: This step refers to the gathering of system requirements, with the goal of determining how these requirements will be accommodated in the system. Extensive communication between the customer and the developer is essential.

System Design : Once the requirements have been collected and analyzed, it is necessary to identify in detail how the system will be constructed to perform necessary tasks. More specifically, the System Design phase is focused on the data requirement, the software construction ,and the interface construction

Coding: Also known as programming, this step involves the creation of the system software. Requirements and systems specifications from the System Design step are translated into machine readable computer code.

Testing. As the software is created and added to the developing system, testing is performed to ensure that it is working correctly and efficiently. Testing focused on two areas: internal efficiency and external effectiveness. The goal of external effectiveness testing is to verify that the software is functioning according to system design, and that it is performing all necessary functions or sub-functions. The goal of internal testing is to make sure that the computer code is efficient, standardized, and well documented.

Problems/Challenges associated with the Waterfall Model

Although the *Waterfall Model* has been used extensively over the years in the production of many quality systems, it is not without its problems. Criticisms fall into the following categories:

- Real projects rarely follow the sequential flow that the model proposes. At the beginning of most projects there is often a great deal of uncertainty about requirements and goals. The model does not accommodate this natural uncertainty very well.
- Developing a system using the *Waterfall Model* can be a long, painstaking process that does not yield a working version of the system until late in the process

Process Model

Just like the waterfall model, the V-Shaped model is a sequential path of execution of processes. Each phase must be completed before the next phase begins. The testing procedures are developed early in the life cycle before any coding is done, of the phases preceding implementation. The high-level design phase focuses on system architecture and design. An integration test plan is created in this phase in order to test the pieces of the software systems ability to work together. However, the low-level design phase lies where the actual software components are designed, and unit tests are created in this phase as well.

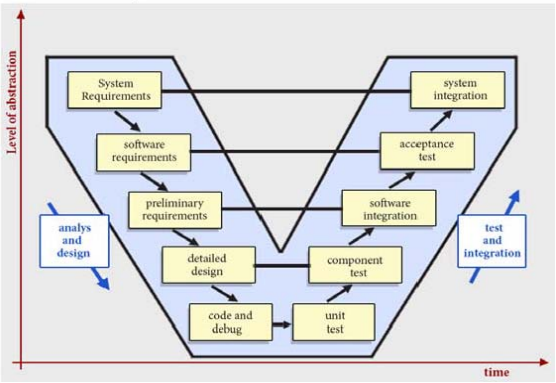

Fig.2. V-model

The implementation phase is, again, where all coding takes place. Once coding is complete, the path of execution continues up the right side of the V where the test plans developed earlier are now put to use.

Advantages

1. Simple and easy to use.
2. Each phase has specific deliverables.
3. Higher chance of success over the waterfall model due to the early development of test plans during the life cycle.
4. Works well for small projects where requirements are easily understood.

3. **Iterative Development**

The problems with the *Waterfall Model* created a demand for a new method of developing systems which could provide faster results, require less up-front information, and offer greater flexibility. With *Iterative Development*, the project is divided into small parts. This allows the development team to demonstrate results earlier on in the process and obtain valuable feedback from system users. Often, each iteration is actually a mini-*Waterfall* process with the feedback from one phase providing vital information for the design of the next phase.

Problems/Challenges associated with the Iterative Model

While the *Iterative Model* addresses many of the problems associated with the *Waterfall Model*, it does present new challenges. The user community needs to be actively involved throughout the project. While this involvement is a positive for the project, it is demanding on the time of the staff and can add project delay.Informal requests for improvement after each phase may lead to confusion - controlled mechanism for handling substantive requests needs to be developed.

The *Iterative Model* can lead to "scope creep," since user feedback following each phase may lead to increased customer demands. As users see the system develop, they may realize the potential of other system capabilities which would enhance their work.

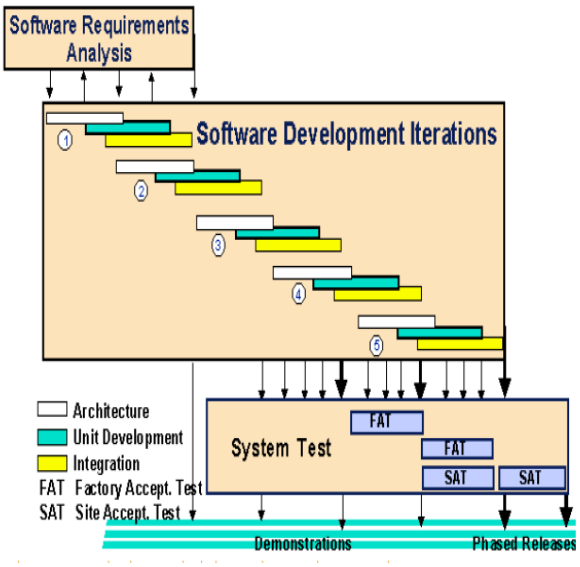

Fig.3. Iterative development model

III. VARIATIONS ON ITERATIVE DEVELOPMENT

A number of *Process Models* have evolved from the *Iterative* approach. All of these methods produce some demonstrable software product early on in the process in order to obtain valuable feedback from system users or other members of the project team. Several of these methods are described below.

Prototyping

The *Prototyping Model* was developed on the assumption that it is often difficult to know all of your requirements at the beginning of a project. The *Prototyping Model* allows for these conditions, and offers a development approach that yields results without first requiring all information up-front .When using the *Prototyping Model*, the developer builds a simplified version of the proposed system and presents it to the customer for consideration as part of the development process. The customer in turn provides feedback to the developer, who goes back to refine the system requirements to incorporate the additional information. There are a few different approaches that may be followed when using the *Prototyping model:*

1. Development of an abbreviated version of the system that performs a limited subset of functions.
2. Use of an existing system or system components to demonstrate some functions that will be included in the developed system.

Prototyping is comprised of the following steps:

Requirements Definition/Collection: The information collected is usually limited to a subset of the complete system requirements.

Design: Once the initial layer of requirements information is collected, or new information is gathered, it is rapidly integrated into a new or existing design so that it may be folded into the prototype.

PrototypeCreation/Modification: The information from the design is rapidly rolled into a prototype. **Assessment:** The prototype is presented to the customer for review. Comments and suggestions are collected from the customer.

Prototype Refinement: Information collected from the customer is digested and the prototype is refined.

System Implementation: In most cases, the system is rewritten once requirements are understood. Sometimes, the *Iterative* process eventually produces a working system that can be the cornserstone for the fully functional system.

Problems/Challenges associated with the Prototyping Model

Criticisms of the *Prototyping Model* generally fall into the following categories:

1. Prototyping can lead to false expectations.

Prototyping often creates a situation where the customer mistakenly believes that the system is “finished” when in fact it is not.

2. **Prototyping can lead to poorly designed systems.**

Because the primary goal of *Prototyping* is rapid development, the design of the system can sometimes suffer because the system is built in a series of “layers” without a global consideration of the integration of all other components.

IV.MODERN SOFTWARE DEVELOPMENT CYCLES

1.Rapid Application Development (RAD) And Rapid Prototyping

A popular variation of the *Prototyping Model* is called **Rapid Application Development (RAD)**.RAD introduces strict time limits on each development phase and relies heavily on rapid application tools which allow for quick development.

Rapid application development (RAD) is an iterative process that relies heavily on user involvement throughout development. In RAD, the entire team meets at the beginning of the process to determine requirements and a fundamental project design.RAD teams break out of this cycle by producing, reviewing, and refining a fundamental prototype during the design phase. Once the project requirements are defined, the developers model the structure and interaction of the objects needed to implement the requirements .

After the analysis and design is complete, the team implements the design in a series of iterations. Each iteration typically lasts several weeks, and implements the subset of features that the team agreed to implement for that iteration .The features implemented are almost always based

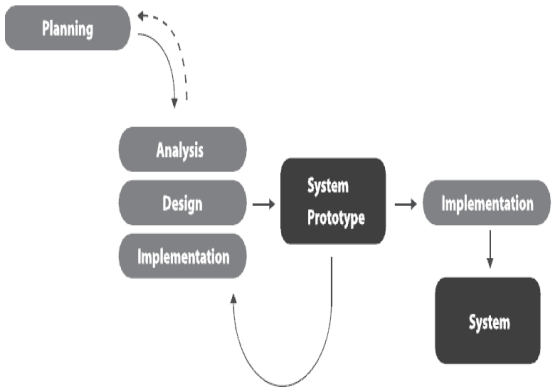

Fig.4. RAD Model

on requirements set forth in the design phase; there is some flexibility for refining existing requirements and adding new ones, but only when the modifications will fit within the original design.

After one iteration is completed, the customer can use the product and if necessary then suggest any necessary refinements. The next iteration will begin, and then the team will continue cycling though iterations until the initial design has been completely implemented.

RAD’s main advantages stem from its insistence on a design phase. Because developers and customers agree on a design before implementation begins, developers are aware of the “big picture” while they are coding. Having a design phase also helps the team to estimate project deadlines and budgets. With RAD, the team always knows the overall project parameters and milestones that must be met before the project is considered complete.

RAD is used for a project that is very dynamic or whose scope is difficult to define upfront, problems are likely because RAD is not designed to accommodate substantial change.

2.Incremental Development Model

The incremental process is well-suited for situations where there is some general project direction, but the product is continually redefined via the addition of different features. This approach is well-suited to innovative projects because their short iteration times allow the team to quickly show the customer the results of the latest request. This allows for rapid feedback on the success of the most recent iteration and the direction of the next. Such frequent and rapid feedback is not necessary for traditional projects (such as building a standard accounting or database program for internal use), but is critical for more innovative projects

3.Spiral model

The *spiral software development* model proposed by Boehm in 1988 was based mostly on experience from working with large projects using the waterfall approach. This risk analysis–oriented approach is depicted in Figure .

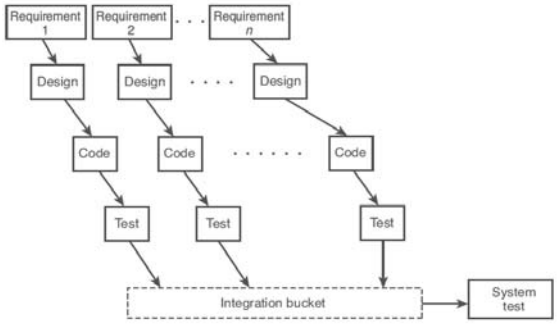

Fig.5.Incremental development model

The radial dimension of this figure represent cumulative cost incurred in accomplishing the steps to date; the angular dimension represents the progress made in completing each cycle of the spiral.Each cycle starts with a requirements phase followed by risk analysis and prototyping, then modelling , coding, testing, and finally deployment.

Areas of uncertainty that are significant sources of project risk are identified and re-evaluated at the beginning of each cycle. Then, a strategy for resolving such risks is proposed based on prototyping,simulation, or benchmarking. Next, phase-specifi c tasks are accomplished; these tasks include requirements specification and validation, software design and design validation,detailed design, coding, followed by unit, integration, and acceptance testing, and culminating in the product deployment (which Boehm called implementation).

The risk-driven approach in the spiral model accommodates a project specific mix of software development strategies . The important feature of the spiral model is that each cycle is completed by a user review, whose major objective is to ensure that users are satisfied with the current progress and committed to the next phase.

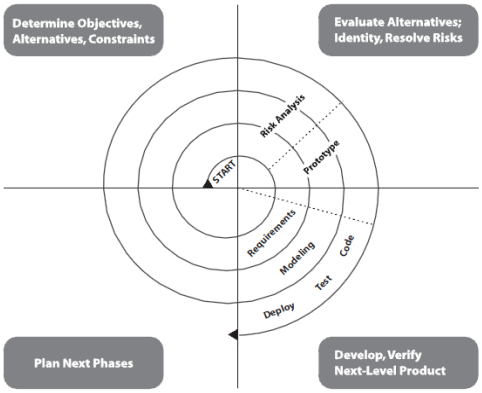

Fig.6. Spiral model

The spiral model has been used successfully in large software projects. Its risk-adverse approach, which is its primary advantage, eliminates many potential problems and accommodates the best features of more development specific methodologies.

4.The Exploratory Model

In some situations it is very difficult, if not impossible, to identify any of the requirements for a system at the beginning of the project. Theoretical areas such as Artificial Intelligence are candidates for using the *Exploratory Model*, because much of the research in these areas is based on guess-work, estimation, and hypothesis. In these cases, an assumption is made as to how the system might work and then rapid iterations are used to quickly incorporate suggested changes and build a usable system. A distinguishing characteristic of the *Exploratory Model* is the absence of precise specifications .The *Exploratory Model* is extremely simple in its construction; it is composed of the following steps:

Initial Specification development: Using whatever information is immediately available, a brief System Specification is created to provide a rudimentary starting point.

System construction/Modification: A system is created or modified according to whatever information is available.

System Test: The system is tested to see what it does, what can be learned from it, and how it may be improved.

System Implementation: After many iterations of the previous two steps produce satisfactory results, the system is dubbed as “finished” and implemented.

Problems/Challenges associated with the Exploratory Model

There are numerous criticisms of the *Exploratory Model*:

- It is limited to use with very high-level languages that allow for rapid development, such as LISP.
- It is difficult to measure or predict its cost-effectiveness.
- As with the *Prototyping Model*, the use of the *Exploratory Model* often yields inefficient or crudely designed systems, since no forethought is given as to how to produce a streamlined system.

5.The Reuse Model

The basic premise behind the *Reuse Model* is that systems should be built using existing components, as opposed to custom-building new components. The *Reuse Model* is clearly suited to Object-Oriented computing environments, which have become one of the premiere technologies in today's system development industry.Within the *Reuse Model*, libraries of software modules are maintained that can be copied for use in any system. These components are of two types: procedural modules and database modules. When building a new system, the developer will "borrow" a copy of a module from the system library and then plug it into a function or procedure. If the needed module is not available, the developer will build it, and store a copy in the system library for future usage. If the modules are well engineered, the developer with minimal changes can implement them.

The *Reuse Model* consists of the following steps:

- **Definition of Requirements.** Initial system requirements are collected. These requirements are usually a subset of complete system requirements.
- **Definition of Objects.** The objects, which can support the necessary system components, are identified.
- **Collection of Objects.** The system libraries are scanned to determine whether or not the needed objects are available. Copies of the needed objects are downloaded from the system.
- **Creation of Customized Objects.** Objects that have been identified as needed, but that are not available in the library are created.
- **Prototype Assembly.** A prototype version of the system is created and/or modified using the necessary objects.
- **Prototype Evaluation.** The prototype is evaluated to determine if it adequately addresses customer needs and requirements.
- **RequirementsRefinement.** Requirements are further refined as a more detailed version of the prototype is created.
- **Objects Refinement.** Objects are refined to reflect the changes in the requirements.

Problems/Challenges Associated with the Reuse Model

A general criticism of the *Reuse Model* is that it is limited for use in object-oriented development environments. Although this environment is rapidly growing in popularity, it is currently used in only a minority of system development applications.

6.Object-Oriented Unified Process

By the time that object-oriented methods had gained widespread acceptance in the software engineering community in the early 1990s.To complement UML, a *unified process* model was developed . (The unified process is sometimes called Rational Unified Process (RUP) after the Rational Corporation, primary builder of the original software tools to support UML and the unified process.).The unified process utilizes iterative and incremental development, and it builds on the best features of other models, emphasizing communication with customers and the implementation of *use cases* as the best methods for describing the customer's view of a system. It also takes an "architecture-first" approach, stressing the importance of a high-level system design capable of adapting to future changes and reuse.Phases of the unified process include inception, elaboration, construction,transition, and production. The *inception* phase encompasses both customer communication and planning activities. The *elaboration* phase consists of planning and modeling. This phase refines the preliminary use cases defined

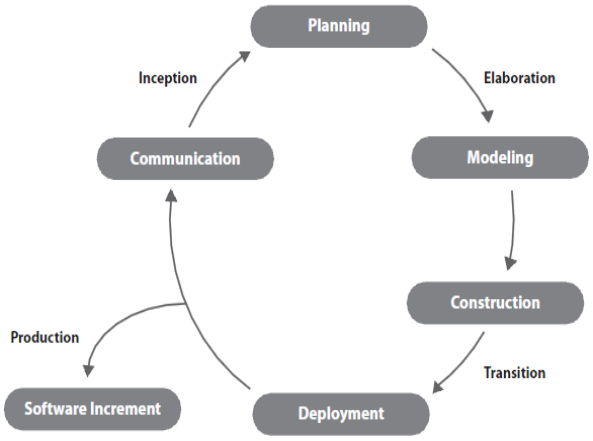

Fig.7.Object oriented unified process

In the *construction* phase, all the necessary features of the software (or its increment, or release), as defined by the use cases, are implemented.

In addition, unit tests, integration tests, and acceptance tests driven by use cases are conducted. The *transition* phase includes *beta testing* by users, which results in user feedback on defects and functionality problems. At the end of this phase, software release takes place. During the phase following the release, referred to as the *production* phase of the unified process, the functioning software is monitored to eliminate possible defects and make requested changes. An important feature of the unified process is that in addition to being iterative and incremental, it employs a high degree of parallel development, with its stages being staggered rather than sequential.

7.Extreme And Agile Programming

Extreme programming (XP) assumes that product requirements will change,so the application is designed and developed incrementally in a series of brief design-coding-testing iterations .During each iteration's design phase, the user provides a list of "stories" she would like implemented, the developer estimates the time required to implement each story, and then the user decides what to implement in the current iteration. The team then determines how to divide the work. During the implementation phase, the developers might work in pairs (*paired programming*): each pair writes unit tests before they code each unit, then writes the unit with one developer coding and the other watching for design flaws, lgorithmic

errors, and general coding problems. They then verify whether the unit is correct by running all relevant tests the team has accumulated. On a daily basis, each pair integrates their thoroughly tested code into the application.
At the end of each iteration, the customer has a working (but not fullfeatured)product to use. The customer provides feedback on the current iteration as the team begins the next design phase. The new iteration might implement features previously requested, incorporate features that satisfy new business requirements, or refine existing features. Code quality improves when XP is used because XP promotes defect prevention in the following ways:

1.Testing occurs throughout the development process, instead of just at the end.

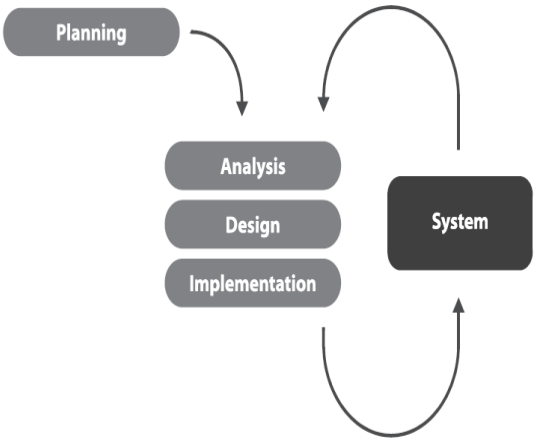

Fig.8.Extreme and Agile programming

2. By utilizing paired programming, XP prompts developers to engage in perpetual code review, and it encourages developers to follow coding standards
that avoid confusing and dangerous coding constructs.
Other advantages of XP include:

- It has the ability to accommodate the frequent changes and unexpected requirements common in today's development environments.
- Its insistence on producing correct code and frequently integrating it into the application means that the product can almost always be shown to the customers in a workable state.

V.CONCLUSION

The evolution of system development *Process Model*s has reflected the changing needs of computer customers. As customers demanded faster results, more involvement in the development process, and the inclusion of measures to determine risks and effectiveness, the methods for developing systems changed. In addition, the software and hardware tools used in the industry changed (and continue to change) substantially. Faster networks and hardware supported the use of smarter and faster operating systems that paved the way for new languages and databases, and applications that were far more powerful than any predecessors.
New models for software development enabled by the Internet group facilitation and distant coordination within open source software communities, and shifting business imperatives in response to these conditions are giving rise to a new generation of software processes and process models. These new models provide a view of software development and evolution that is incremental, iterative, ongoing, interactive, and sensitive to social and organizational circumstances, while at the same time, increasingly amenable to automated support, facilitation, and collaboration over the distances of space and time.

REFERENCES

[1] Martin, J., *Rapid Application Development*. Prentice-Hall, 1991.
[2] Boehm, B., A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy, Using the WinWin Spiral Model: A Case Study, *Computer*, 31(7), 33-44, 1998.
4] Boehm, B.W., "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, Vol. 21, No. 5, May 1988, pp. 61–72.
[5] Jacobson, I., Booch, G., and Rumbaugh, J., *The Unifi ed Software Development Process*. Addison-Wesley, 1999.
[6] Bolcer, G.A., R.N. Taylor, Advanced workflow management technologies, *Software Process--Improvement and Practice*, 4,3, 125-171, 1998.
[7] Chatters, B.W., M.M. Lehman, J.F. Ramil, and P. Werwick, Modeling a Software Evolution Process: A Long-Term Case Study, Software Process Improvement and Practice, 5(2-3), 91 102,2000.
[8] B. Curtis, H. Krasner, V. Shen, and N. Iscoe, On Building Software Process Models Under the Lamppost, *Proc. 9th. Intern. Conf. Software Engineering*, IEEE Computer Society, Monterey, CA, 96-103, 1987.
[9] Curtis, B., H. Krasner, and N. Iscoe, A Field Study of the Software Design Process for Large Systems, *Communications ACM*, 31, 11, 1268-1287, November, 1988.
[10] Cusumano, M. and D. Yoffie, Software Development on Internet Time, *Computer*, 32(10), 60-69, 1999.
[11] Walt Scacchi, Institute for Software Research, University of California, Irvine,Process Models Engineering,February 2001.