A Survey of System Development Process Models

CTG.MFA - 003

Models for Action Project: Developing Practical Approaches to Electronic Records Management and Preservation

This material is based upon work supported in part by the National Historical Publications and Records Commission under Grant No. 96023

Center for Technology in Government
University at Albany / SUNY

Introduction

This document provides an overview of the more common system development *Process Model*s, used to guide the analysis, design, development, and maintenance of information systems. There are many different methods and techniques used to direct the life cycle of a software development project and most real-world models are customized adaptations of the generic models. While each is designed for a specific purpose or reason, most have similar goals and share many common tasks. This paper will explore the similarities and differences among these various models and will also discuss how different approaches are chosen and combined to address practical situations.

Typical Tasks in the Development Process Life Cycle

Professional system developers and the customers they serve share a common goal of building information systems that effectively support business process 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. Typical activities performed include the following:[1]

- System conceptualization
- System requirements and benefits analysis
- Project adoption and project scoping
- System design
- Specification of software requirements
- Architectural design
- Detailed design
- Unit development
- Software integration & testing
- System integration & testing
- Installation at site
- Site testing and acceptance
- Training and documentation
- Implementation
- Maintenance

[1] Kal Toth, Intellitech Consulting Inc. and Simon Fraser University; list is partially created from lecture notes: Software Engineering Best Practices, 1997.

Process Model/Life-Cycle Variations

While nearly all system development efforts engage in some combination of the above tasks, they can be differentiated by the *feedback* and *control methods* employed during development and the *timing of activities*. Most system development *Process Models* in use today have evolved from three primary approaches: *Ad-hoc Development*, *Waterfall Model*, and the *Iterative* process.

Ad-hoc Development

Early systems development often took place in a rather chaotic and haphazard manner, relying entirely on the skills and experience of the individual staff members performing the work. Today, many organizations still practice *Ad-hoc Development* either entirely or for a certain subset of their development (e.g. small projects).

The Software Engineering Institute at Carnegie Mellon University[2] 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. Schedules, budgets, functionality, and product quality are generally (inconsistent). Performance depends on the capabilities of individuals and varies with their innate skills, knowledge, and motivations. There are few stable software processes in evidence, and performance can be predicted only by individual rather than organizational capability." [3]

Figure 1. Ad-hoc Development

"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. Success that

[2] Information on the Software Engineering Institute can be found at http://www.sei.cmu.edu.
[3] 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," Software Engineering Institute, February 1993, p 1.

rests solely on the availability of specific individuals provides no basis for long-term productivity and quality improvement throughout an organization."[4]

The Waterfall Model

The *Waterfall Model* is the earliest method of structured system development. Although it has come under attack in recent years for being too rigid and unrealistic when it comes to quickly meeting customer's needs, the *Waterfall Model* is still widely used. It is attributed with providing the theoretical basis for other *Process Models*, because it most closely resembles a "generic" model for software development.

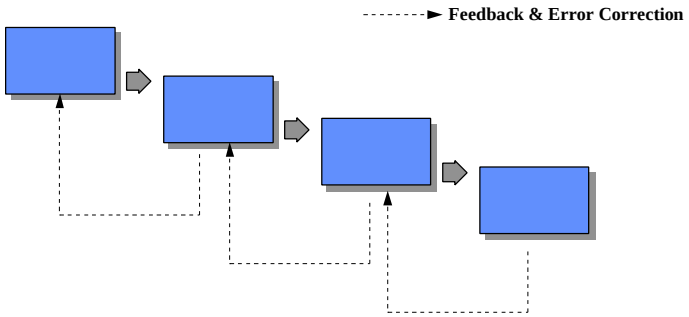

Figure 2. Waterfall Model

The *Waterfall Model* consists of the following steps:

- **System Conceptualization.** System Conceptualization refers to the consideration of all aspects of the targeted business function or process, 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 requirements (what information will be processed in the system?), the software construction (how will the application be constructed?), and the interface construction (what will the system look like? What standards will be followed?).
- **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 is generally

[4] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, "Capability Maturity Model for Software, Version 1.1," Software Engineering Institute, February 1993, p 18.

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. Testing can be a labor-intensive process, due to its iterative nature.

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. In recent years it has come under attack, due to its rigid design and inflexible procedure. 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, and it is therefore difficult for customers to identify these criteria on a detailed level. 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.

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. In a variation of this model, the software products which are produced at the end of each step (or series of steps) can go into production immediately as incremental releases.

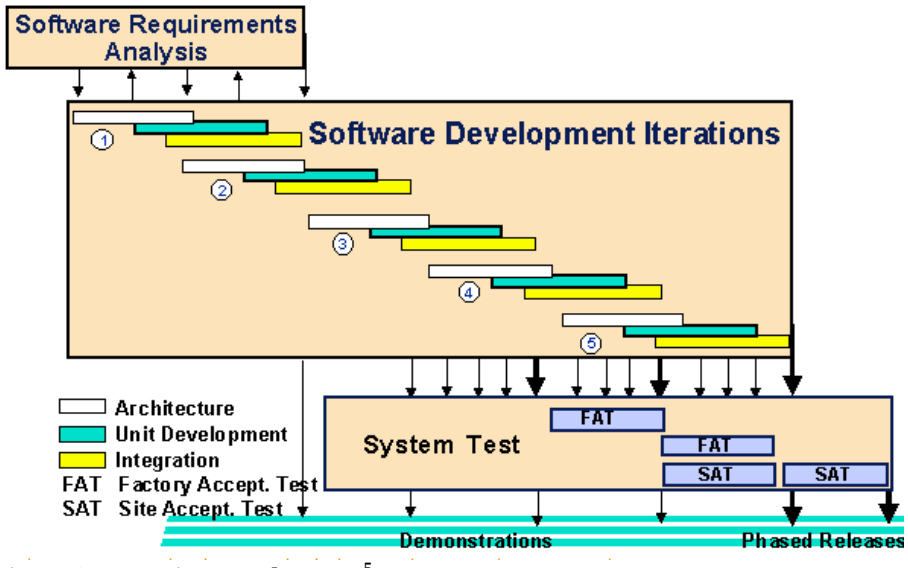

Figure 3. Iterative Development[5]

[5] Kal Toth, Intellitech Consulting Inc. and Simon Fraser University, from lecture notes: Software Engineering Best Practices, 1997.

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.
- Communication and coordination skills take center stage in project development.
- Informal requests for improvement after each phase may lead to confusion -- a 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.

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. Typically, users know many of the objectives that they wish to address with a system, but they do not know all the nuances of the data, nor do they know the details of the system features and capabilities. 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. Often, the prototype code is thrown away and entirely new programs are developed once requirements are identified.

There are a few different approaches that may be followed when using the *Prototyping Model*:

- creation of the major user interfaces without any substantive coding in the background in order to give the users a "feel" for what the system will look like,
- development of an abbreviated version of the system that performs a limited subset of functions; development of a paper system (depicting proposed screens, reports, relationships etc.), or
- 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.** Similar to the Conceptualization phase of the *Waterfall Model*, but not as comprehensive. 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.
- **Prototype Creation/Modification.** The information from the design is rapidly rolled into a prototype. This may mean the creation/modification of paper information, new coding, or modifications to existing coding.
- **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. The developer revises the prototype to make it more effective and efficient.
- **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:

- **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. More specifically, when using the *Prototyping Model*, the pre-implementation versions of a system are really nothing more than one-dimensional structures. The necessary, behind-the-scenes work such as database normalization, documentation, testing, and reviews for efficiency have not been done. Thus the necessary underpinnings for the system are not in place.
- **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. While initial software development is often built to be a "throwaway, " attempting to retroactively produce a solid system design can sometimes be problematic.

Variation of the Prototyping Model
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.

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. Validation is based on adequacy of the end result and not on its adherence to pre-conceived requirements.

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 and/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.

The Spiral Model

The *Spiral Model* was designed to include the best features from the *Waterfall* and *Prototyping Models*, and introduces a new component - risk-assessment. The term "spiral" is used to describe the process that is followed as the development of the system takes place. Similar to the *Prototyping Model*, an initial version of the system is developed, and then repetitively modified based on input received from customer evaluations. Unlike the *Prototyping Model*, however, the development of each version of the system is carefully designed using the steps involved in the

Waterfall Model. With each iteration around the spiral (beginning at the center and working outward), progressively more complete versions of the system are built.[6]

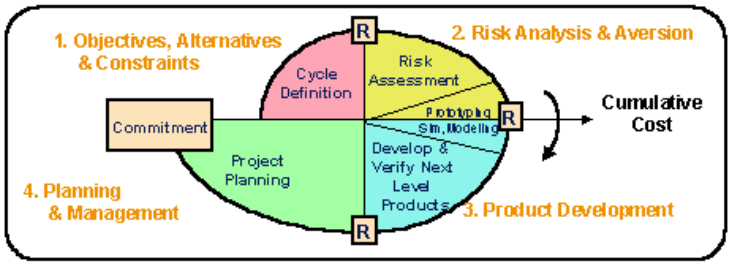

Figure 4. Spiral Model[7]

Risk assessment is included as a step in the development process as a means of evaluating each version of the system to determine whether or not development should continue. If the customer decides that any identified risks are too great, the project may be halted. For example, if a substantial increase in cost or project completion time is identified during one phase of risk assessment, the customer or the developer may decide that it does not make sense to continue with the project, since the increased cost or lengthened timeframe may make continuation of the project impractical or unfeasible.

The *Spiral Model* is made up of the following steps:

- **Project Objectives.** Similar to the system conception phase of the *Waterfall Model*. Objectives are determined, possible obstacles are identified and alternative approaches are weighed.
- **Risk Assessment.** Possible alternatives are examined by the developer, and associated risks/problems are identified. Resolutions of the risks are evaluated and weighed in the consideration of project continuation. Sometimes prototyping is used to clarify needs.
- **Engineering & Production.** Detailed requirements are determined and the software piece is developed.
- **Planning and Management.** The customer is given an opportunity to analyze the results of the version created in the Engineering step and to offer feedback to the developer.

Problems/Challenges associated with the Spiral Model

Due to the relative newness of the *Spiral Model*, it is difficult to assess its strengths and weaknesses. However, the risk assessment component of the *Spiral Model* provides both developers and customers with a measuring tool that earlier *Process Model*s do not have. The measurement of risk is a feature that occurs everyday in real-life situations, but (unfortunately) not as often in the system development industry. The practical nature of this tool helps to make the *Spiral Model* a more realistic *Process Model* than some of its predecessors.

The Reuse Model

[6] Linda Spence, University of Sutherland, "Software Engineering," available at http://osiris.sunderland.ac.uk/rif/linda_spence/HTML/contents.html

[7] Kal Toth, Intellitech Consulting Inc. and Simon Fraser University, from lecture notes: Software Engineering Best Practices, 1997.

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.
- **Requirements Refinement.** 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.

Creating and Combining Models

In many cases, parts and procedures from various *Process Models* are integrated to support system development. This occurs because most models were designed to provide a framework for achieving success only under a certain set of circumstances. When the circumstances change beyond the limits of the model, the results from using it are no longer predictable. When this

situation occurs it is sometimes necessary to alter the existing model to accommodate the change in circumstances, or adopt or combine different models to accommodate the new circumstances.

The selection of an appropriate *Process Model* hinges primarily on two factors: organizational environment and the nature of the application. Frank Land, from the London School of Economics, suggests that suitable approaches to system analysis, design, development, and implementation be based on the relationship between the information system and its organizational environment.[8] Four categories of relationships are identified:

- **The Unchanging Environment.** Information requirements are unchanging for the lifetime of the system (e.g. those depending on scientific algorithms). Requirements can be stated unambiguously and comprehensively. A high degree of accuracy is essential. In this environment, formal methods (such as the *Waterfall* or *Spiral Models*) would provide the completeness and precision required by the system.

- **The Turbulent Environment.** The organization is undergoing constant change and system requirements are always changing. A system developed on the basis of the conventional *Waterfall Model* would be, in part; already obsolete by the time it is implemented. Many business systems fall into this category. Successful methods would include those, which incorporate rapid development, some throwaway code (such as in *Prototyping*), the maximum use of reusable code, and a highly modular design.

- **The Uncertain Environment.** The requirements of the system are unknown or uncertain. It is not possible to define requirements accurately ahead of time because the situation is new or the system being employed is highly innovative. Here, the development methods must emphasize learning. Experimental *Process Model*s, which take advantage of prototyping and rapid development, are most appropriate.

- **The Adaptive Environment.** The environment may change in reaction to the system being developed, thus initiating a changed set of requirements. Teaching systems and expert systems fall into this category. For these systems, adaptation is key, and the methodology must allow for a straightforward introduction of new rules.

Summary

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. These rapid

[8] Frank Kand, "A Contingency Based Approach to Requirements Elicitation and Systems Development," London School of Economics, J. Systems Software 1998; 40: pp. 3-6.

and numerous changes in the system development environment simultaneously spawned the development of more practical new *Process Model*s and the demise of older models that were no longer useful.

References

Bassett, Paul "Partnerships take out the garbage", *Software Magazine*, Nov 1994 v14 n11 p96(2).

S Bell and A T Wood-Harper, "Rapid Information Systems Development - a Non-Specialists Guide to Analysis and Design in an Imperfect World", McGraw-Hill, Maidenhead, UK, 1992.

W Cotterman and J Senn, "Challenges and Opportunities for Research into Systems Development", John Wiley, Chichester, 1992.

A L Friedman (with D.S. Cornford), "Computer Systems Development: History, Organization and Implementation", John Wiley, Chichester, 1989.

Frank Kand, " A Contingency Based Approach to Requirements Elicitation and Systems Development," London School of Economics, J. Systems Software 1998.

Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, "Capability Maturity Model for Software, Version 1.1", Software Engineering Institute, February 1993

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", Software Engineering Institute, February 1993.

Linda Spence, University of Sutherland, "Software Engineering," available at http://osiris.sunderland.ac.uk/rif/linda_spence/HTML/contents.html

Kal Toth, Intellitech Consulting Inc. and Simon Fraser University; lecture notes: Software Engineering Best Practices, 1997.

G Walsham "Interpreting Information Systems in Organizations", John Wiley, Chichester, 1993.

Center for Technology in Government
University at Albany / SUNY
1535 Western Avenue
Albany, NY 12203
Phone: 518-442-3892
Fax: 518-442-3886
info@ctg.albany.edu
www.ctg.albany.edu