Universitas Esa Unggul

Smart, Creative and Entrepreneurial

www.esaunggul.ac.id

CCR210-REKAYASA PERANGKAT LUNAK
Pertemuan Ke 09
Oleh : MALABAY
Prodi : Teknik Informatika/Sistem Informasi

Prescriptive Process Models

Sumber dari : https://www2.southeastern.edu/Academics/Faculty/galkadi/285/notes/Chapter3.doc

Prescriptive Process Models

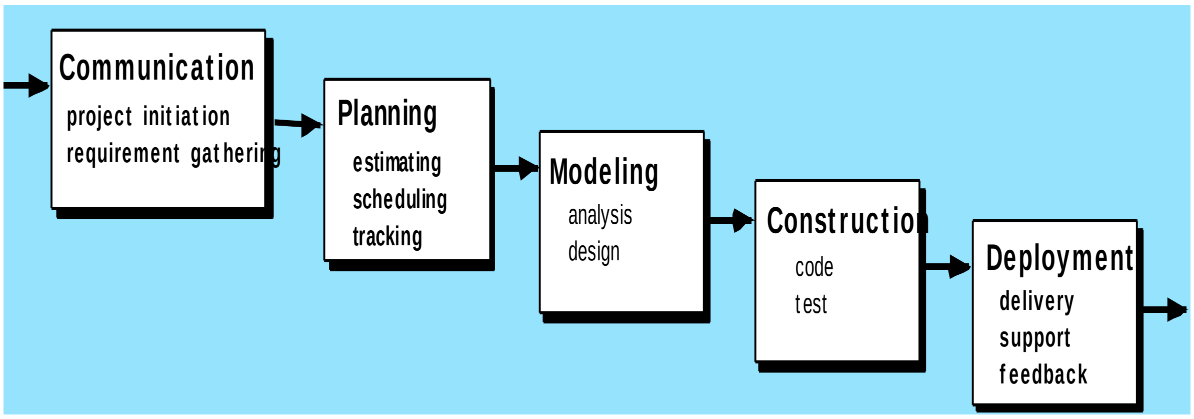
CHAPTER OVERVIEW AND COMMENTS

This intent of this chapter is to present process models used by professional software developers to manage large-scale software projects. No software process works well for every project. However, every project needs to conform to some systematic process in order to manage software development activities that can easily get out of control. Software processes help to organize the work products that need to be produced during a software development project. Ultimately the best indicator of how well a software process has worked is the quality of the deliverables produced. A well-managed process will produce high quality products on time and within budget.

3.1 Prescriptive Models

Many people (and not a few professors) believe that prescriptive models are "old school"—ponderous, bureaucratic document-producing machines.

3.2 The Waterfall Model



Many people dismiss the waterfall as obsolete and it certainly does have problems. But this model can still be used in some situations.

Among the problems that are sometimes encountered when the *waterfall* model is applied are:

- A Real project rarely follows the sequential flow that the model proposes. Change can cause confusion as the project proceeds.
- It is difficult for the customer to state all the requirements explicitly. The waterfall model requires such demand.
- The customer must have patience. A working of the program will not be available until late in the project time-span.

3.3 Incremental Process Models

The process models in this category tend to be among the most widely used (and effective) in the industry.

3.3.1 The Incremental Model

The *incremental model* combines elements of the *waterfall* model applied in an iterative fashion. The model applies linear sequences in a staggered fashion as calendar time progresses.

Each linear sequence produces deliverable "increments" of the software. (Ex: a Word Processor delivers basic file mgmt., editing, in the first increment; more sophisticated editing, document production capabilities in the 2nd increment; spelling and grammar checking in the 3rd increment.

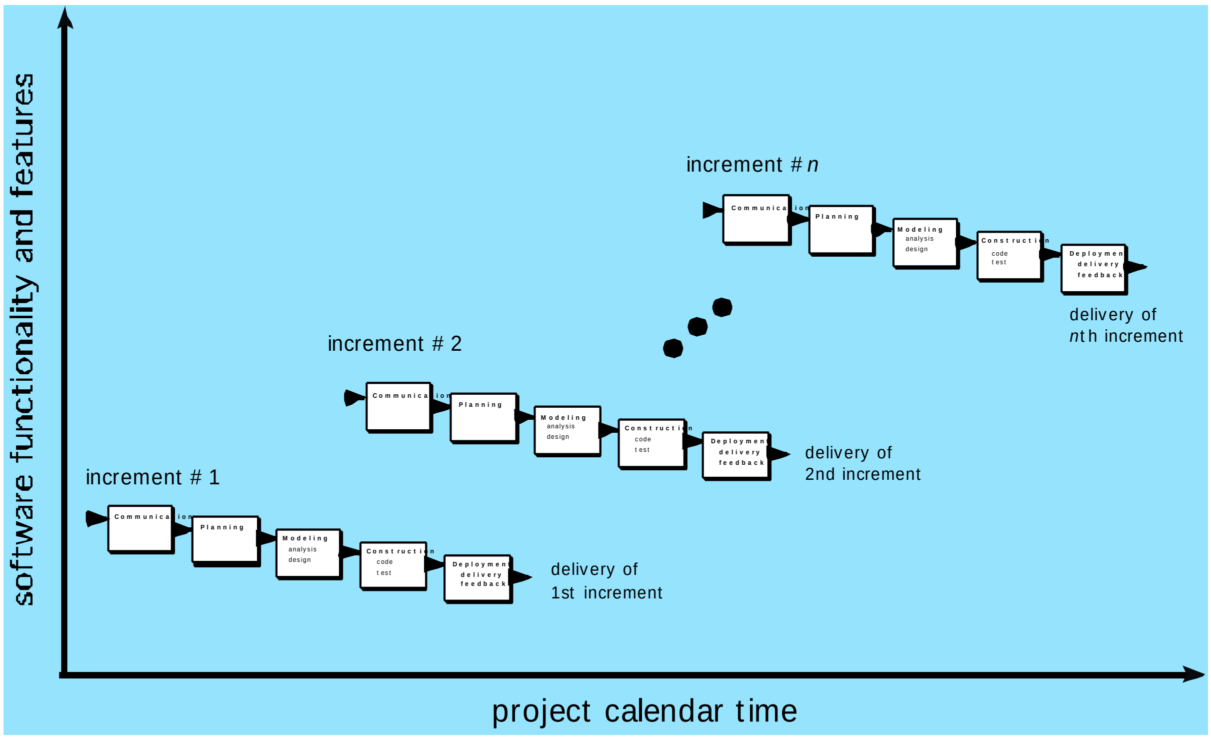
When an increment model is used, the 1st increment is often a *core product*. The core product is used by the customer.

As a result of use and / or evaluation, a plan is developed for the next increment.

The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality.

The process is repeated following the delivery of each increment, until the complete product is produced.

If the customer demands delivery by a date that is impossible to meet, suggest delivering one or more increments by that date and the rest of the Software later.


software functionality and features
project calendar time
increment # 1
increment # 2
increment # n
Communication
Planning
Modeling analysis design
Construction code test
Deployment delivery feedback
delivery of 1st increment
delivery of 2nd increment
delivery of nth increment

3.3.1 The RAD Model

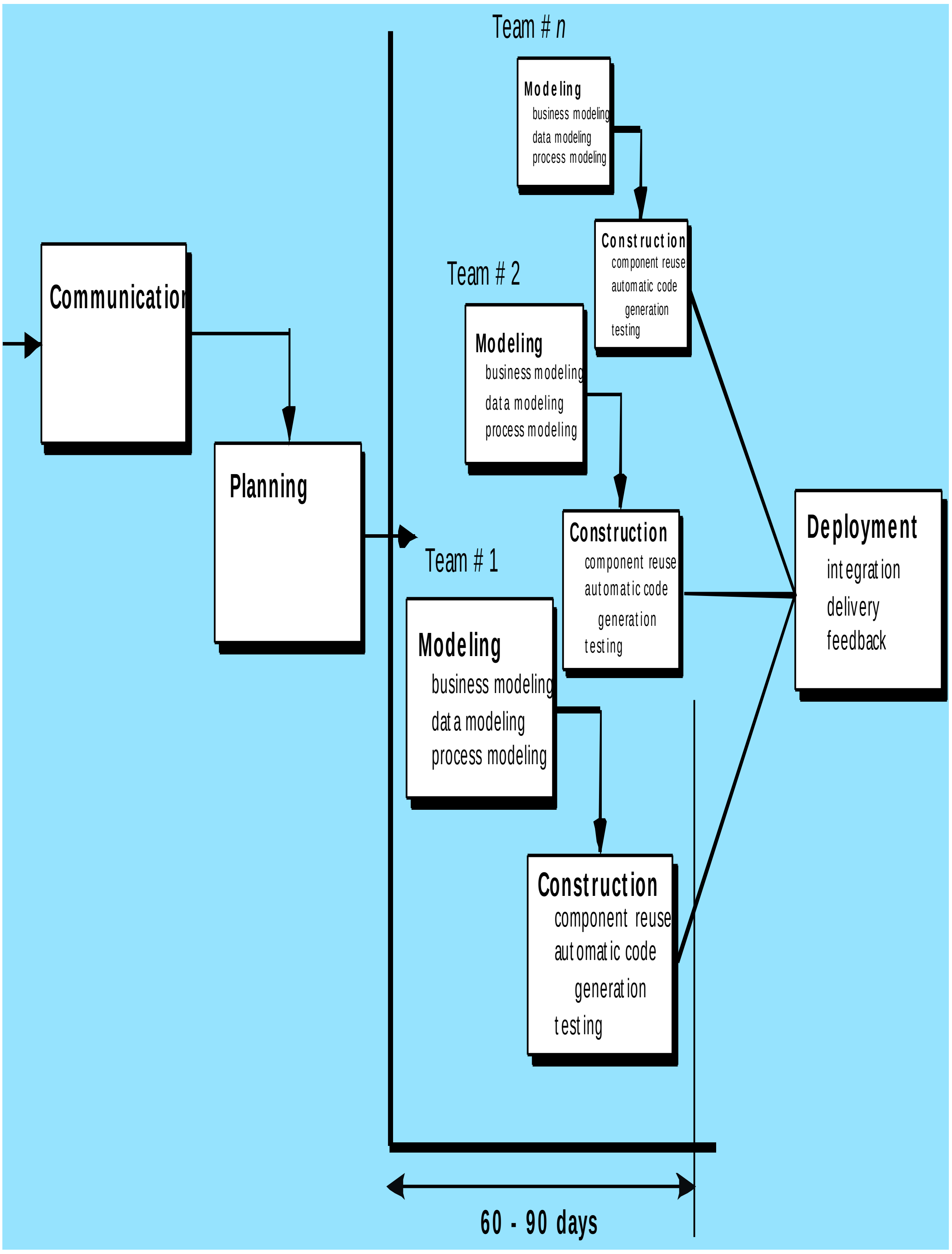
Rapid Application Development (*RAD*) is an incremental software process model that emphasizes a short development cycle.

RAD is a "high-speed" adaptation of the waterfall model, in which rapid development is achieved by using a component based construction approach.

If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a fully functional system within a short period of time.

What are the drawbacks of the RAD model?

1. For large, but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.
2. If developers and customers are not committed to the rapid-fire activities necessary to complete the system in a much abbreviated time frame, RAD project will fail.
3. If a system cannot properly be modularized, building the components necessary for RAD will be problematic.


Team # n
Modeling
business modeling
data modeling
process modeling
Construction
component reuse
automatic code generation
testing
Team # 2
Modeling
business modeling
data modeling
process modeling
Construction
component reuse
automatic code generation
testing
Communication
Planning
Team # 1
Modeling
business modeling
data modeling
process modeling
Construction
component reuse
automatic code generation
testing
Deployment
integration
delivery
feedback
60 - 90 days

3.4 Evolutionary Process Models

Software evolves over a period of time; business and product requirements often change as development proceeds, making a straight-line path to an end product unrealistic.

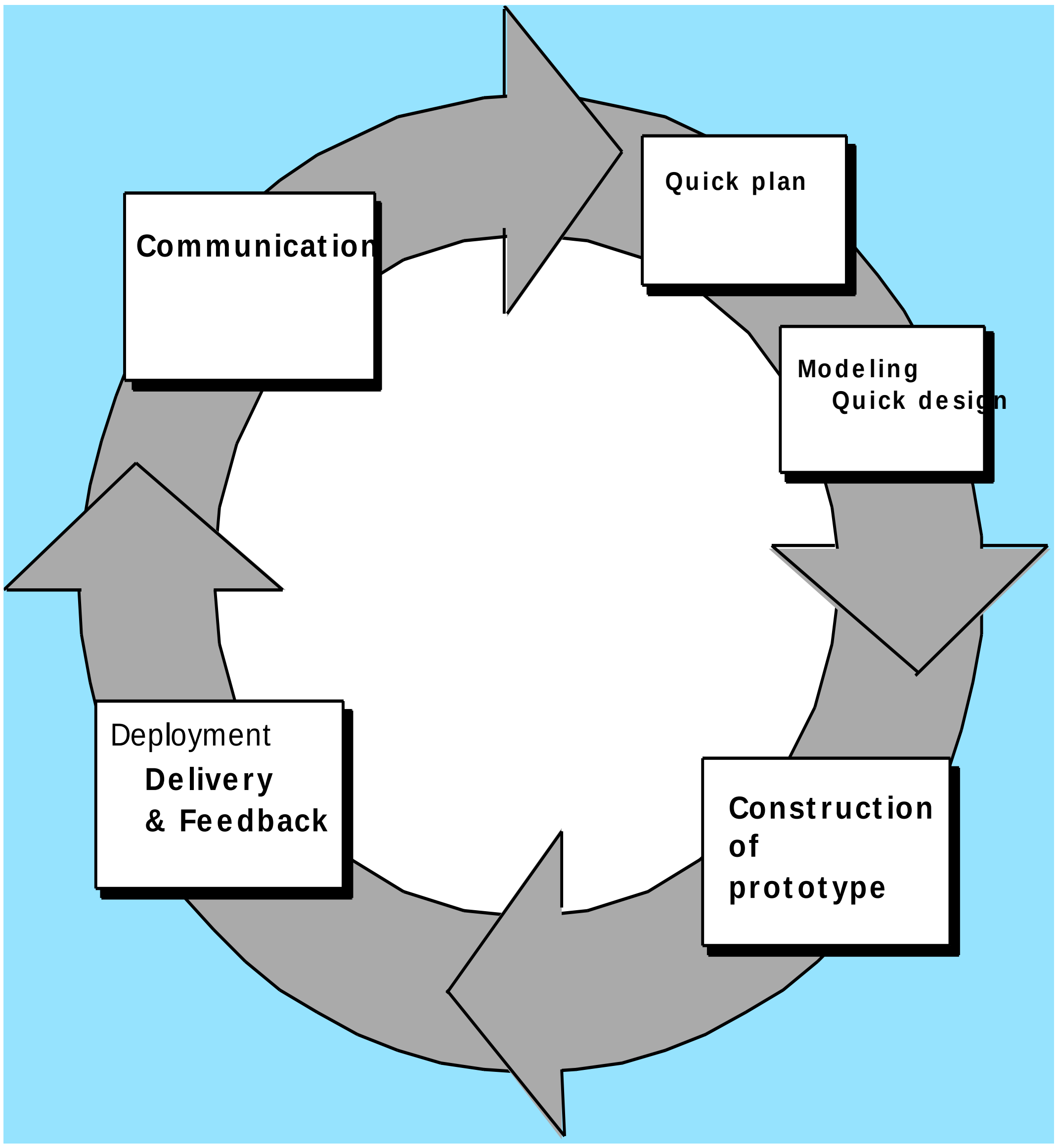
Software Engineering needs a process model that has been explicitly designed to accommodate a product that evolves over time.

Evolutionary process models are iterative. They produce increasingly more complete versions of the Software with each iteration.

3.4.1 Prototyping

Customers often define a set of general objectives for Software, but doesn't identify detailed input, processing, or input requirements.

Prototyping paradigm assists the Software engineering and the customer to better understand what is to be built when requirements are fuzzy.


Quick plan
Communication
Modeling
Quick design
Construction of prototype
Deployment
Delivery & Feedback

The prototyping paradigm begins with *communication* where requirements and goals of Software are defined.

Prototyping iteration is *planned* quickly and modeling in the form of quick design occurs.

The *quick design* focuses on a representation of those aspects of the Software that will be visible to the customer "Human interface".

The quick design leads to the *Construction of the Prototype*.

The prototype is *deployed* and then *evaluated* by the customer.

Feedback is used to refine requirements for the Software.

Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while enabling the developer to better understand what needs to be done.

The prototype can serve as the "first system". Both customers and developers like the prototyping paradigm as users get a feel for the actual system, and developers get to build Software immediately. Yet, prototyping can be problematic:

1. The customer sees what appears to be a working version of the Software, unaware that the prototype is held together "with chewing gum. "Quality, long-term maintainability." When informed that the product is a prototype, the customer cries foul and demands that few fixes be applied to make it a working product. Too often, Software development management relents.
2. The developer makes implementation compromises in order to get a prototype working quickly. An inappropriate O/S or programming language used simply b/c it's available and known. After a time, the developer may become comfortable with these choices and forget all the reasons why they were inappropriate.

The key is to define the rules of the game at the beginning. The customer and the developer must both agree that the prototype is built to serve as a mechanism for defining requirements.

3.4.2 The Spiral Model

The spiral model is an evolutionary Software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model.

It has two distinguishing features:

a. A cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk.
b. A set of *anchor point milestones* for ensuring stakeholder commitment to feasible and mutually satisfactory solutions.

Using the spiral model, Software is developed in a series of evolutionary releases.

During early stages, the release might be a paper model or prototype.

During later iterations, increasingly more complete versions of the engineered system are produced.

A spiral model is divided into a set of framework activities divided by the Software engineering team.

As this evolutionary process begins, the Software team performs activities that are implied by a circuit around the spiral in a clockwise direction, beginning at the center.

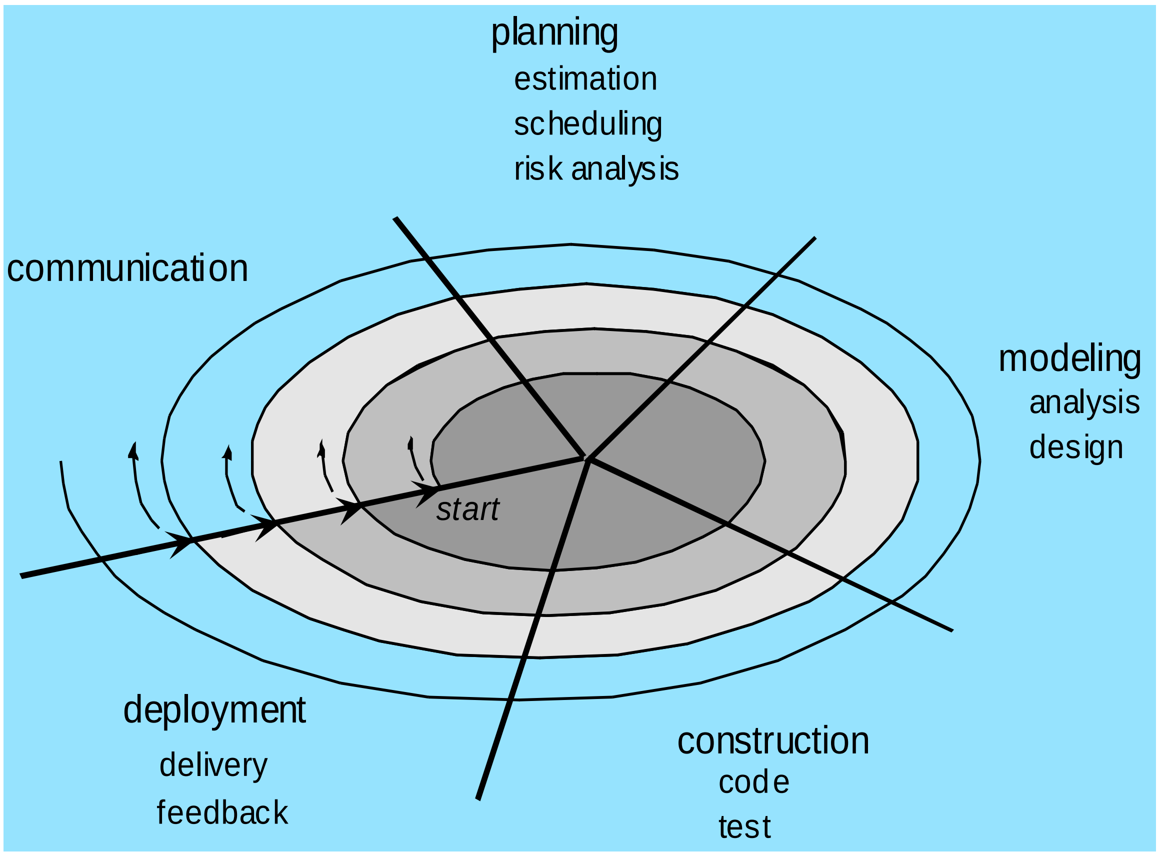
Risk is considered as each revolution is made.

Anchor-point milestones – a combination of work products and conditions that are attained along the path of the spiral- are noted for each evolutionary pass.

The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the Software.

Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from the customer after delivery.

Unlike other process models that end when Software is delivered, the spiral model can be adapted to apply throughout the life of the Software.


planning
estimation
scheduling
risk analysis
communication
modeling
analysis
design
start
deployment
delivery
feedback
construction
code
test

3.4.3 The concurrent Development Model

The *concurrent development model*, sometimes called *concurrent engineering*, can be represented schematically as a series of framework activities, Software engineering actions of tasks, and their associated states.

The concurrent model is often more appropriate for system engineering projects where different engineering teams are involved.

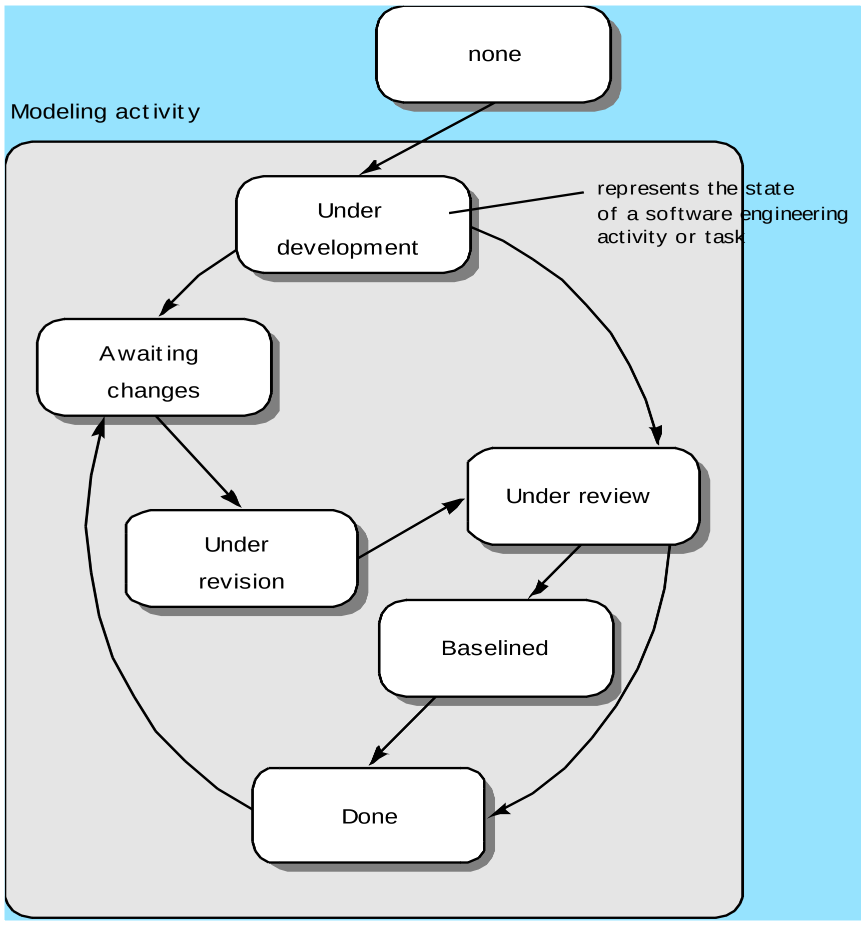


Figure above provides a schematic representation of one Software engineering task within the modeling activity for the concurrent process model. The activity – modeling- may be in any one of the states noted at any given time.

All activities exist concurrently but reside in different states.

For example, early in the project the *communication* activity has completed its first iteration and exists in the **awaiting changes** state. The *modeling* activity which existed in the **none** state while initial communication was completed now makes a transition into **underdevelopment** state.

If, however, the customer indicates the changes in requirements must be made, the *modeling* activity moves from the **under development** state into the **awaiting changes** state.

The concurrent process model defines a series of events that will trigger transitions from state to state for each of the Software engineering activities, actions, or tasks.

Specialized Process Models

3.5.1 Component Based Development

Commercial off-the-shelf (COTS) Software components, developed by vendors who offer them as products, can be used when Software is to be built. These components provide targeted functionality with well-defined interfaces that enable the component to be integrated into the Software.

The *component-based development* model incorporates many of the characteristics of the spiral model.

The *component-based development* model incorporates the following steps:

- Available component-based products are researched and evaluated for the application domain in question.
- Component integration issues are considered.
- Software architecture is designed to accommodate the components.
- Components are integrated into the architecture.
- Comprehensive testing is conducted to ensure proper functionality.

The *component-based development* model leads to Software reuse, and reusability provides Software engineers with a number of measurable benefits.

3.5.1 The Formal Methods Model

The Formal Methods Model encompasses a set of activities that leads to formal mathematical specifications of Software.

Formal methods enable a Software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation.

A variation of this approach, called *clean-room* Software *engineering* is currently applied by some software development organizations.

http://www.sei.cmu.edu/str/descriptions/cleanroom.html

Although not a mainstream approach, the formal methods model offers the promise of defect-free Software. Yet, concern about its applicability in a business environment has been voiced:

- The development of formal models is currently quite time-consuming and expensive.
- B/C few software developers have the necessary background to apply formal methods, extensive training is required.
- It is difficult to use the methods as a communication mechanism for technically unsophisticated customers.

3.5.3 Aspect-Oriented Software Development

Regardless of the software process that is chosen, the builders of complex software invariably implement a set of localized features, functions, and information content. For further information read book page 61.

3.6 The Unified Process

A "use-case driven, architecture-centric, iterative and incremental" software process closely aligned with the Unified Modeling Language (UML).

http://www.jeckle.de/files/uniproc.pdf, **http://www-01.ibm.com/software/awdtools/rup/**

The UP is an attempt to draw on the best features and characteristics of conventional software process models, but characterize them in a way that implements many of the best principles of agile software development.

The UP recognizes the importance of customer communication and streamlined methods for describing the customer's view of a system.

It emphasizes the important role of software architecture and "helps the architect focus on the right goals, such as understandability, reliance to future changes, and reuse."

UML provides the necessary technology to support Object Oriented Software Engineering practice, but it doesn't provide the process framework to guide project teams in their application of the technology.

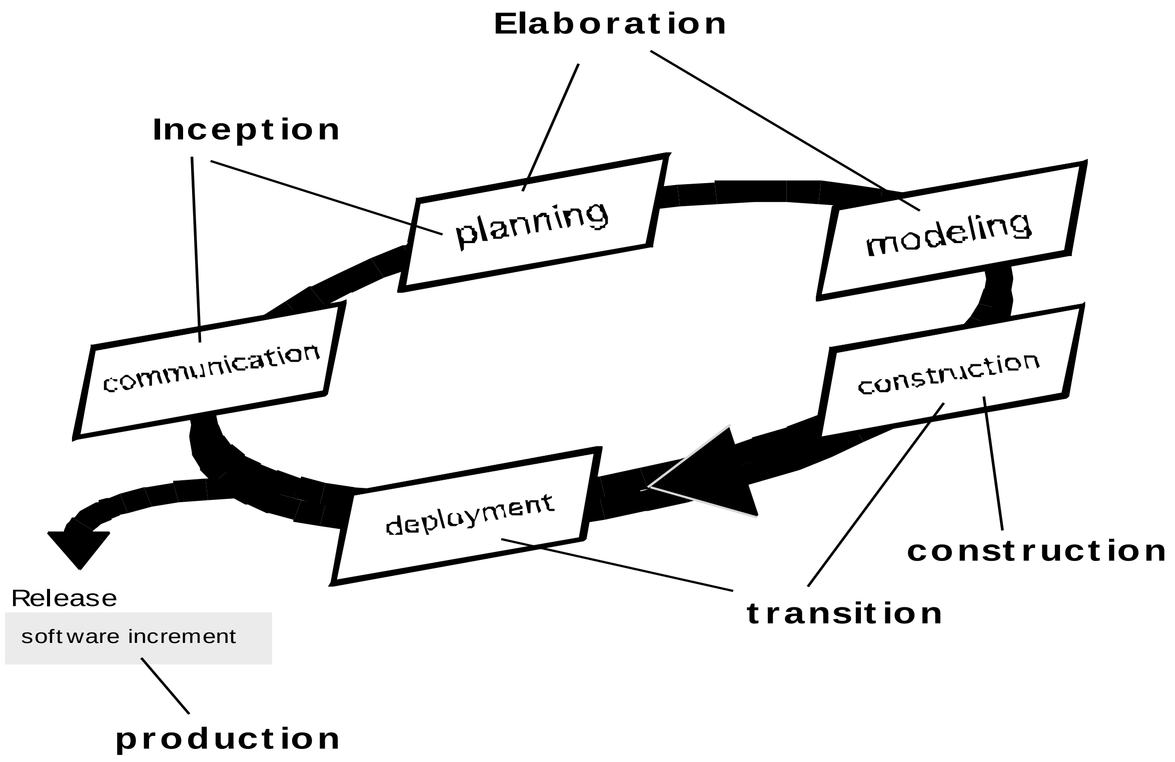
The UML developers developed the *Unified Process*, a framework Object Oriented Software Engineering using UML.

3.6.2 Phases of the Unified Process

The figure below depicts the phases of the UP and relates them to the generic activities.

The *Inception* phase of the UP encompasses both customer communication and planning activities.

By collaborating with the customer and end-users, business requirements for the software are identified, a rough architecture for the system is proposed, and a plan for the iterative, incremental nature of the ensuing project is developed.



A use-case describes a sequence of actions that are performed by an *actor* (person, machine, another system) as the actor interacts with the Software.

The *elaboration* phase encompasses the customer communication and modeling activities of the generic process model. Elaboration refines and expands the preliminary use-cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software - the use-case model, the analysis model, the design model, the implementation model, and the deployment model.

The *construction* phase of the UP is identical to the construction activity defined for the generic software process.

Using the architectural model as input, the construction phase develops or acquires the software components that will make each use-case operational for end-users.

The *transition* phase of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment activity.

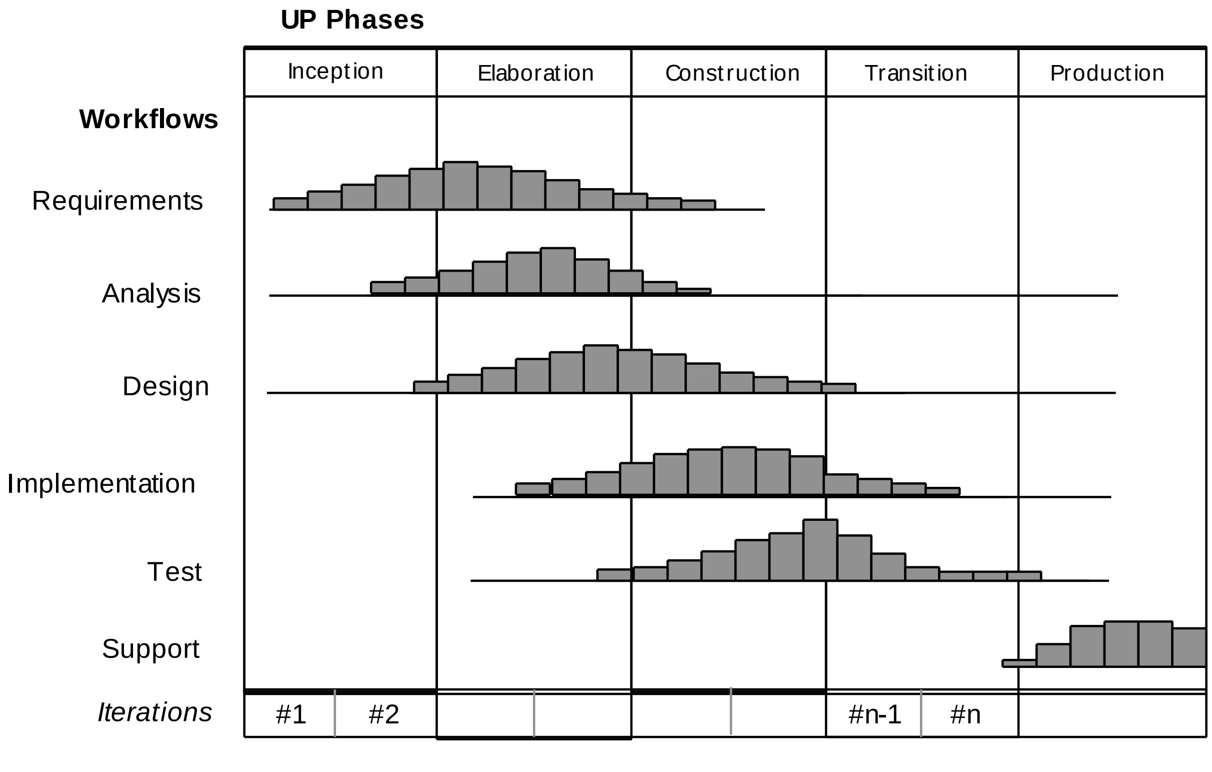
Software is given to end-users for beta testing, and user feedback reports both defects and necessary changes.

At the conclusion of the transition phase, the software increment becomes a usable software release "user manuals, trouble-shooting guides, and installation procedures.)

The *production* phase of the UP coincides with the development activity of the generic process.

The on-going use of the software is monitored, support for the operating environment is provided and defect reports and requests for changes are submitted and evaluated.

A Software Engineering workflow is distributed across all UP phases.



http://www.uml.org/ is the UML® Resource Page. Please link to it and familiarize yourself with it.

Sumber dari : http://www.eecs.qmul.ac.uk/~norman/SE-pages/Supporting%20documents/Pressman/software%20engineering%20paradigms.doc.

·· Essential Software Engineering

- planning--tasks required to define resources, timelines, and other project-related information
- risk assessment--tasks required to assess both technical and management risks
- engineering--tasks required to build one or more repre-sentations of the application
- installation-tasks required to test, install, and provideuser support (e.g., documentation and training)
- customer evaluation--tasks required to obtain customer feedback based on evaluation of the software representa-tions created during the engineering stage and implement-ed during the installation stage.

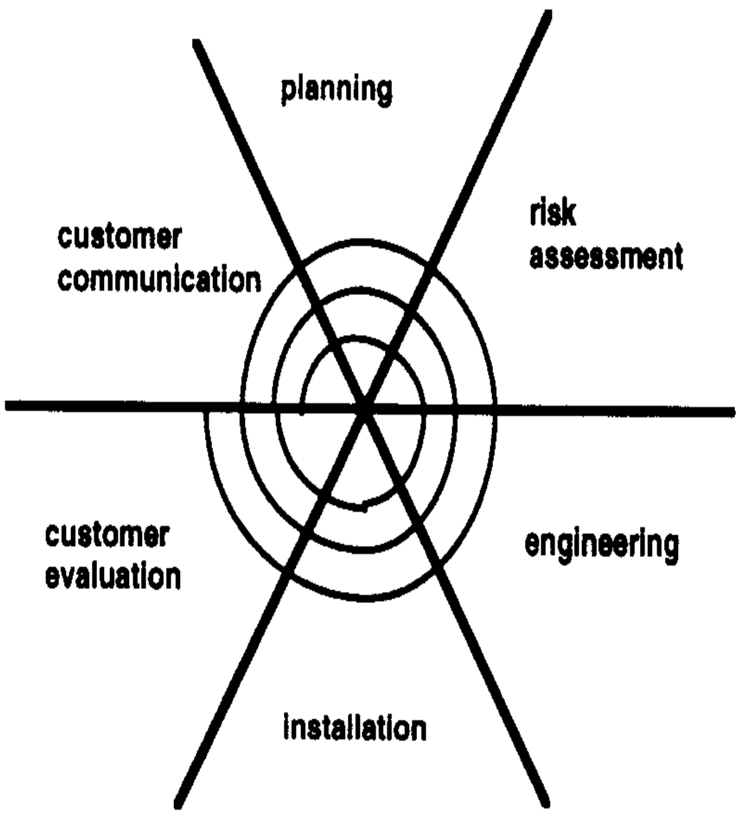


Figure 1. A Six-Segment Evolutionary Model

Management and technical tasks are defined for each of thetask regions. To accommodate the need for an adaptiveprocess (e.g., one that adapts itself to the characteristics of the project at hand), the evolutionary model should define a number of task sets. Each task set contains software engi-

neering tasks, milestones, and deliverables that have been chosen to meet the needs of different types of projects.

Each task set must provide enough discipline to achieve high software quality. But at the same time, it must not burden the project team with unnecessary work. Although any number of task sets can be suggested,the following are typical:

Casual. The process model does not apply to the project, but selected tasks may be applied informally and basic principles of software engineering must still be followed.

Disciplined. The process model will be applied for the project with a degree of discipline that will ensure high quality and good application maintainability.

Rigorous. All process model tasks, documents, and mile- stones will be applied to the project. High quality, good documentation, and long maintainability are paramount.

Quick reaction. The process model will be applied for the project, but because of extremely tight time constraints, only those tasks essential to maintaining good quality will be applied. When necessary, "back-filling" (e.g., develop-ing a complete set of documentation) will be accomplished after the application is delivered to the customer.