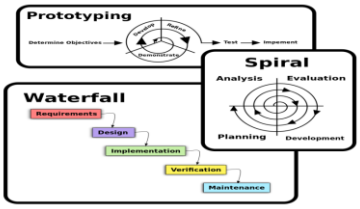
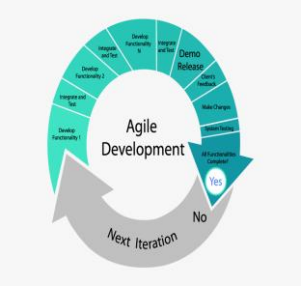
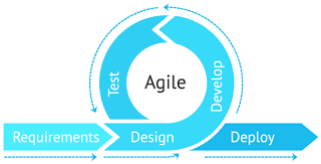


Software Engineering Process Models





Outline

- Process Models
- A Generic Process Model
- Process Assessment and Improvement
- Overview of Prescriptive Process Models:
  - Waterfall
  - Incremental
  - Evolutionary
  - Concurrent
- Overview of Specialized Process Models:
  - Component-Based
  - Formal Methods
  - Aspect-Oriented Software Development
- Unified Process
- Personal and Team Process Models.

Process Model

- The process is a dialogue in which the knowledge that must become the software is brought together and embodied in the software.
- The process provides interaction between users and designers, between users and evolving tools, and between designers and evolving tools [technology].
- It is an iterative process in which the evolving tool itself serves as the medium for communication, with each new round of the dialogue for more useful knowledge from the people involved.
- So, building computer software is an iterative social learning process, and the outcome, something called "software capital,"

3

Software Process

- A **process** is a collection of **activities**, **actions** and **tasks** that are performed when some work product is to be created
- A process is not a **rigid prescription** for how to build the software, rather it is **adaptable approach** that enables the people doing the work to **pick** and **choose** the **appropriate set of work actions** and tasks
- An **activity** try to **achieve** a **broad objective** (e.g., communication with stakeholders)
- An **activity** is **applied** regardless of the **application domain**, **size of the project**, **complexity of the effort**, or **degree of accuracy** with which software engineering is to be applied.
- An **action** (e.g., architectural design) **encompasses** a **set of tasks** that **produce** a major **work product** (e.g., an architectural design model).

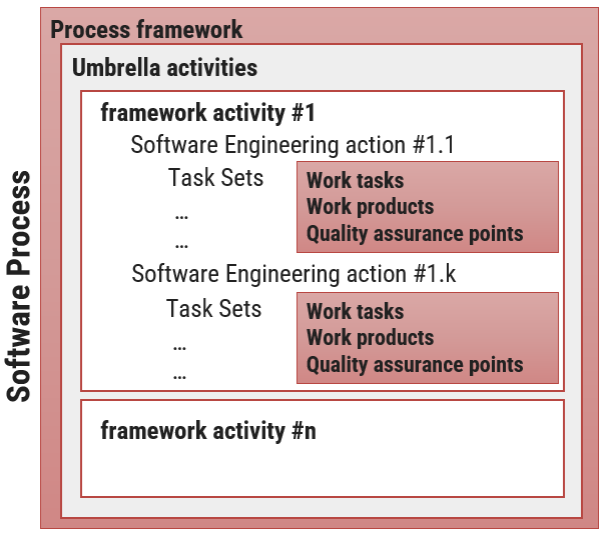
4

Software Process

- A **task** **focuses** on a **small, but well-defined objective** (e.g., conducting a unit test) that **produces** a **noticeable outcome**.
- **Each of these** activities, actions & tasks **reside within** a **framework** or model

5

Software Process Framework



6

Process Framework Activities (CPMCD)

- A process framework establishes the foundation for complete software engineering process, it encompasses five activities

Communication

Communication with Customers / stackeholders to understand project requirements for defining software features

Planning

Software Project Plan which defines workflow that is to follow.
It describes technical task, risks, resources, product to be produced & work schedule

7

Process Framework Activities (CPMCD)

Modeling

Creating models to understand requirements and shows design of software to achieve requirements

Construction

Code Generation
(manual or automated)
&
Testing
(to uncover errors in the code)

Deployment

Deliver Software to Customer
Collect feedback from customer based on evaluation
Software Support

8

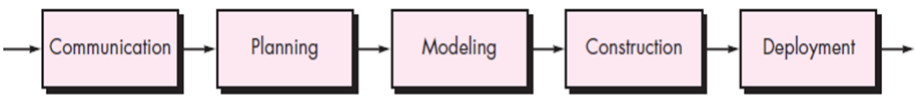
The Process Framework

- These five generic framework activities can be used during the development of small, simple programs, the creation of large Web applications, and for the engineering of large, complex computer-based systems. The details of the software process will be quite different in each case, but the framework activities remain the same.
- For many software projects, framework activities are applied iteratively as a project progresses. That is, communication, planning, modeling, construction, and deployment are applied repeatedly through a number of project iterations.
- Each project iteration produces a software increment that provides stakeholders with a subset of overall software features and functionality.
- As each increment is produced, the software becomes more and more complete.

9

The Process Flow

- Process flow—describes how the framework activities and the actions and tasks that occur within each framework activity are organized with respect to sequence and time which is illustrated in Figure

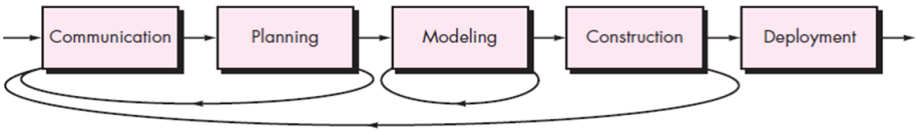


(a) Linear process flow

- A linear process flow executes each of the five framework activities in sequence, beginning with communication and ends with deployment

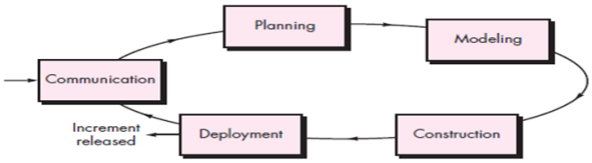
10

The Process Flow



(b) Iterative process flow

- An iterative process flow repeats one or more of the activities before proceeding to the next

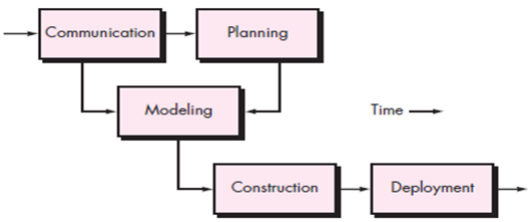


(c) Evolutionary process flow

- An evolutionary process flow executes the activities in a "circular" manner. Each circuit through the five activities leads to a more complete version of the software

11

The Process Flow



(d) Parallel process flow

- A parallel process flow executes one or more activities in parallel with other activities (e.g., modeling for one aspect of the software might be executed in parallel with construction of another aspect of the software).

12

The Framework

- How does a framework activity change as the nature of the project changes?
- For a small software project requested by one person (at a remote location) with simple, straightforward requirements, the communication activity might encompass little more than a phone call with the appropriate stakeholder. Therefore, the only necessary action is phone conversation, and the work tasks (the task set) that this action encompasses are:

1. Make contact with stakeholder via telephone.
2. Discuss requirements and take notes.
3. Organize notes into a brief written statement of requirements.
4. E-mail to stakeholder for review and approval.

13

The Framework

- If the project was considerably more complex with many stakeholders, each with a different set of (sometime conflicting) requirements, the communication activity might have six distinct actions : **inception, elicitation, elaboration, negotiation, specification, and validation.**

14

Identifying a Task Set

- A task set defines the actual work to be done to accomplish the objectives of a software engineering action.!
- For example, elicitation (more commonly called “requirements gathering”) is an important software engineering action that occurs during the communication activity.
- The goal of requirements gathering is to understand what various stakeholders want from the software that is to be built.
- For a small, relatively simple project, the task set for requirements gathering might look like this:

1. Make a list of stakeholders for the project.!
2. Invite all stakeholders to an informal meeting.!
3. Ask each stakeholder to make a list of features and functions required.!
4. Discuss requirements and build a final list.!
5. Prioritize requirements.!
6. Note areas of uncertainty. !

15

Identifying a Task Set

- The task sets for Requirements gathering action for a big project may include:!

1. Make a list of stakeholders for the project.!
2. Interview each stakeholders separately to determine overall wants and needs. !
3. Build a preliminary list of functions and features based on stakeholder input.!
4. Schedule a series of facilitated application specification meetings.!
5. Conduct meetings.!
6. Produce informal user scenarios as part of each meeting. !
7. Refine user scenarios based on stakeholder feedback.!
8. Build a revised list of stakeholder requirements.!

16

Identifying a Task Set

9. Use quality function deployment techniques to prioritize requirements.!
10. Package requirements so that they can be delivered incrementally.!
11. Note constraints and restrictions that will be placed on the system.!
12. Discuss methods for validating the system. !

17

Process Assessment and Improvement

- Assessment attempts to understand the current state of the software process with the intent of improving it.
- The existence of a software process is no guarantee that software will be delivered on time, that it will meet the customer's needs, or that it will exhibit the technical characteristics that will lead to long-term quality characteristics.
- However, the process can be assessed to ensure that it meets a set of basic process criteria that have been shown to be essential for a successful software engineering.
- A number of different approaches to software process assessment and improvement have been proposed over the past few decades:

18

Process Assessment and Improvement

- **Standard CMMI Assessment Method for Process Improvement (SCAMPI)** — provides a five step process assessment model that incorporates five phases: initiating, diagnosing, establishing, acting and learning.
- **CMM-Based Appraisal for Internal Process Improvement (CBA IPI)**—provides a diagnostic technique for assessing the relative maturity of a software organization; uses the SEI CMM as the basis for the assessment.
- **SPICE**—The SPICE (ISO/IEC15504) standard defines a set of requirements for software process assessment. The intent of the standard is to assist organizations in developing an objective evaluation of the efficacy of any defined software process.

19

Process Assessment and Improvement

- **ISO 9001:2000 for Software**—a generic standard that applies to any organization that wants to improve the overall quality of the products, systems, or services that it provides.
- Therefore, the standard is directly applicable to software organizations and companies.

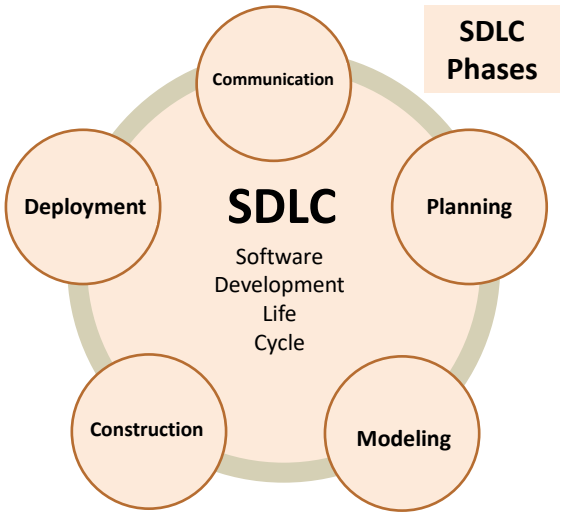
20

Prescriptive Process Models

- Also known as **Software development life cycle (SDLC)** or Application development life cycle Models.
- Process models **prescribe** a distinct set of **activities, actions, tasks and milestones (deliverables)** required to engineer high quality software.
- Process **models are not perfect,** but **provide roadmap** for software engineering work.
- Software models provide stability, control and organization to a process that if not managed can easily get out of control.
- Software process models are adapted (adjusted) to meet the needs of software engineers and managers for a specific project.
- The **process model** is the abstract representation of process.
- Each process model also prescribes a process flow (also called a workflow)—that is, the manner in which the process elements are interrelated to one another.

21

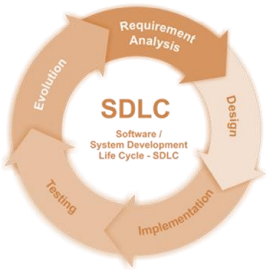
Prescriptive Process Models



22

Different Process Models

- Process model is selected based on different parameters
  - Type of the project & people
  - Complexity of the project
  - Size of team
  - Expertise of people in team
  - Working environment of team
  - Software delivery deadline

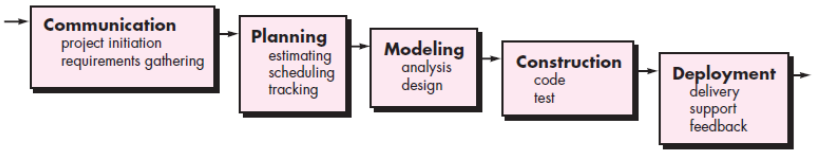


- **Process Models**
  - Waterfall Model (Linear Sequential Model)
  - Incremental Process Model
  - Prototyping Model
  - The Spiral Model
  - Rapid Application Development Model
  - Agile Model

23

Waterfall Models

Classic life cycle or **linear sequential model**



- When **requirements** for a problems are **well understood** then this model is used in which **work flow** from communication to deployment is **linear**.

When to use ?

- **Requirements** are very well **known**, **clear** and **fixed**
- Product **definition** is **stable**
- **Technology** is **understood**
- There are **no ambiguous** (unclear) **requirements**
- Ample (**sufficient**) **resources** with required **expertise** are **available** freely
- The **project** is **short**

24

Waterfall Models

Classic life cycle or linear sequential model

Advantage

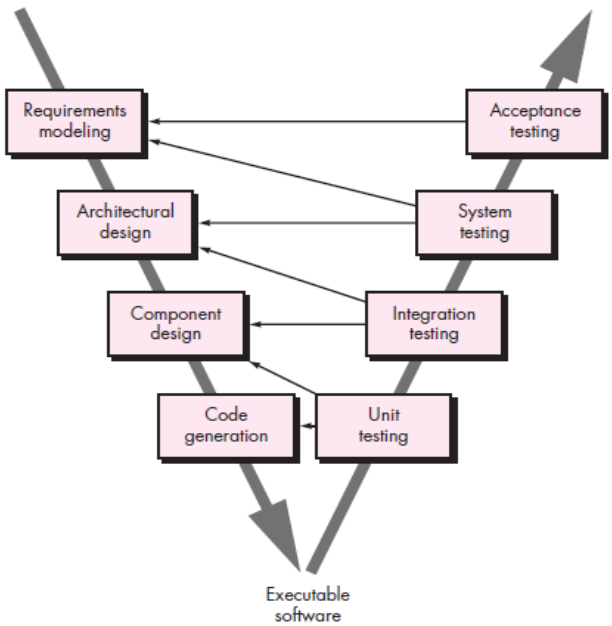
- **Simple to implement** and manage

Drawback

- **Unable to accommodate changes** at **later stages**, that is required in most of the cases.
- **Working version** is **not available** during development. Which can lead the development with major mistakes.
- **Deadlock can occur** due to delay in any step.
- **Not suitable** for **large projects**.

25

V Models



- quality assurance actions
- Once code has been generated, the team moves up the right side of the V

26

V-Models

- A variation of waterfall model depicts the relationship of quality assurance actions to the actions associated with communication, modelling and early code construction activates.
- Team first moves down the left side of the V to refine the problem requirements. Once code is generated, the team moves up the right side of the V, performing a series of tests that validates each of the models created as the team moved down the left side.
- In reality, there is no fundamental difference between the classic life cycle and the V-model. The V-model provides a way of visualizing how verification and validation actions are applied to earlier engineering work

27

Incremental Models

- The incremental model **combines** elements of **linear** and **parallel** process flows.
- This model applies linear sequence in a iterative manner.
- Initially **core working product** is **delivered**.
- **Each** linear **sequence** produces deliverable **"increments"** of the software.
- e.g. **word-processing software** developed **using** the **incremental model**
  - It might deliver basic file management, editing and document production functions in the first increment
  - more complex editing in the second increment;
  - spelling and grammar checking in the third increment; and
  - advanced page layout capability in the fourth increment.

28

Incremental Models

When to use ?

- When the **requirements** of the **complete** system are clearly **defined** and understood but **staffing is unavailable** for a **complete implementation** by the business deadline.

Advantage

- Generates **working software quickly** and early during the software life cycle.
- It is **easier to test** and debug during a smaller iteration.
- **Customer** can **respond** to each built.
- **Lowers initial** delivery **cost**.
- **Easier** to **manage risk** because risky pieces are identified and handled during iteration.

29

Evolutionary Process Models

- Software system evolves over time as requirements often change as development proceeds. Thus, a straight line to a complete end product is not possible.
- However, a limited version must be delivered to meet competitive pressure.
- When a set of **core product** or system requirements is **well understood** but the **details of product** or system extensions have **yet to be defined**.
- In this situation there is **a need of process model** which specially designed to accommodate **product** that **evolve with time**.
- **Evolutionary Process Models** are specially meant for that which produce an increasingly more complete version of the software with each iteration.
- Evolutionary Models are **iterative**.

30

Evolutionary Process Models

- They are characterized in a manner that enables you to develop **increasingly more complete versions of the software**.
- Evolutionary models are
  - **Prototyping Model**
  - **Spiral Model**

31

Prototyping Models

When to use ?

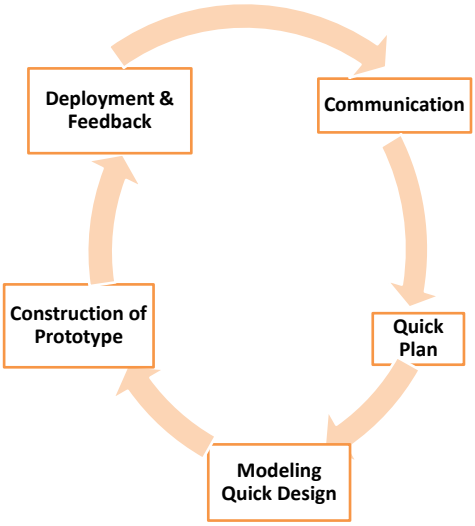
- **Customers have** general **objectives of software** but **do not have detailed requirements** for functions & features.
- **Developers** are **not sure** about **efficiency of an algorithm** & **technical feasibilities**.

- It serves as a **mechanism** for **identifying software requirements**.
- Prototype can be serve as "**the first system**".
- Both stakeholders and software engineers like prototyping model
  - Users get feel for the actual system
  - Developers get to build something immediately

32

Prototyping Models

It works as follows



- **Communicate** with stockholders & **define objective** of Software
- **Identify requirements** & design **quick plan**
- **Model** a quick **design** (focuses on visible part of software)
- **Construct Prototype** & deploy
- Stakeholders **evaluate** this **prototype** and provides **feedback**
- Iteration occurs and **prototype** is **tuned** based on **feedback**

33

Prototyping Models

What step?

- Begins with communication by meeting with stakeholders to define the objective, identify whatever requirements are known, outline areas where further definition is mandatory.
- A quick plan for prototyping and modeling (quick design) occur.
- Quick design focuses on a representation of those aspects the software that will be visible to end users. (interface and output). Design leads to the construction of a prototype which will be deployed and evaluated. Stakeholder's comments will be used to refine requirements.
- Both stakeholders and software engineers like the prototyping paradigm. Users get a feel for the actual system, and developers get to build something immediately.

34

Prototyping Models

What step?

- However, engineers may make compromises in order to get a prototype working quickly. The less than- ideal choice may be adopted forever after you get used to it.
- Iteration occurs as the prototype is tuned to satisfy the needs of various stakeholders, while at the same time enabling you to better understand what needs to be done.
- Although prototyping can be used as a stand-alone process model, it is more commonly used as a technique that can be implemented within the context of any one of the process models.
- Regardless of the manner in which it is applied, the prototyping paradigm assists you and other stakeholders to better understand what is to be built when requirements are fuzzy.

35

Prototyping Models

What step?

- According to Brooks - In most projects, the first system built is barely usable. It may be too slow, too big, awkward in use or all three. There is no alternative but to start again, smarting but smarter, and build a redesigned version in which these problems are solved.

Problem areas

- **Customer demand** that "**a few fixes**" be applied to **make** the **prototype a working product**, due to that software quality suffers as a result.
- **Developer** often makes **implementation** in order to get a prototype working quickly; **without considering other factors** in mind like OS, Programming language, etc.

36

Prototyping Models

Advantages

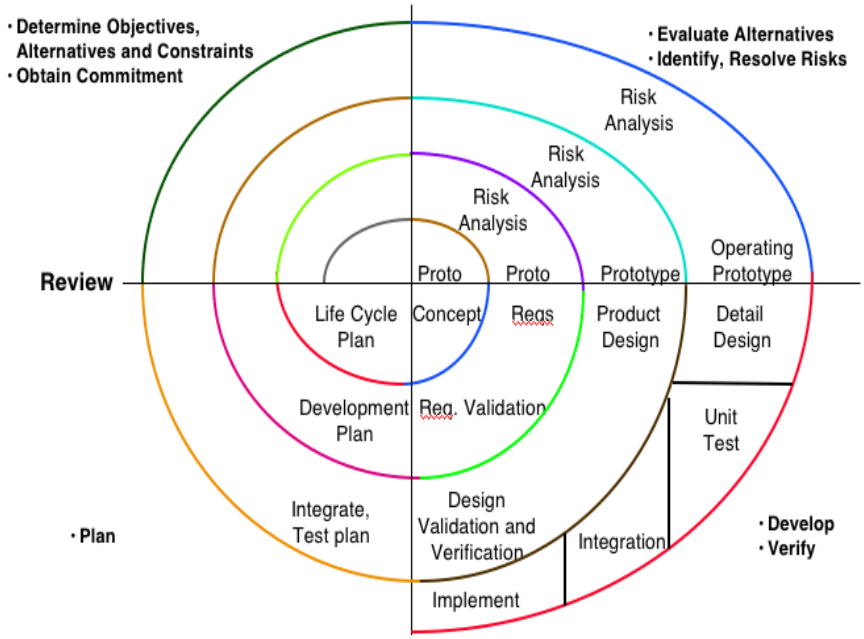
- **Users** are actively **involved** in the **development.**
- Since in this methodology a working model of the system is provided, the **users get a better understanding** of the **system** being developed.
- **Errors** can be **detected** much **earlier**

37

Spiral Models

- It provides the **potential** for **rapid development**.
- Software is developed in a series of evolutionary releases.
- **Early iteration** release might be **prototype** but **later iterations** provides more **complete version of software**.
- It is divided into framework activities (C,P,M,C,D). Each activity represent one segment of the spiral
- **Each pass** through the **planning** region results in **adjustments** to
  - the **project plan.**
  - **Cost** & **schedule** based on feedback

38



Spiral Models

- It couples the **iterative** nature of prototyping with the controlled and systematic aspects of the **waterfall model** and is a **risk-driven** process model generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems.
- Two main distinguishing features:
  - one is cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk.
  - The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.
- A series of evolutionary releases are delivered. During the early iterations, the release might be a model or prototype. During later iterations, increasingly more complete version of the engineered system are produced.

Spiral Models

- The first circuit in the clockwise direction might result in the 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 results in adjustments to the project plan. Cost and schedule are adjusted based on feedback. Also, the number of iterations will be adjusted by project manager.
- Good to develop large-scale system as software evolves as the process progresses and risk should be understood and properly reacted to. Prototyping is used to reduce risk.
- However, it may be difficult to convince customers that it is controllable as it demands considerable risk assessment expertise.

41

Spiral Models

When to use :

- For development of **large scale / high-risk projects**.
- When costs and **risk evaluation is important**.
- Users are **unsure** of their **needs**.
- **Requirements** are **complex**.
- New product line.
- Significant (**considerable**) **changes** are expected.

Advantage

- High amount of risk analysis hence, **avoidance of Risk** is enhanced.
- **Strong approval** and **documentation** control.
- **Additional functionality** can be **added** at a later date.
- **Software** is **produced early** in the Software Life Cycle.

42

Spiral Models

Disadvantage

- Can be **a costly model** to use.
- Risk analysis **requires highly specific expertise**.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

43

Three Concerns on Evolutionary Processes

1. First concern is that prototyping poses a problem to project planning because of the uncertain number of cycles required to construct the product.
2. Second, it does not establish the maximum speed of the evolution. If the evolution occur too fast, without a period of relaxation, it is certain that the process will fall into chaos. On the other hand if the speed is too slow then productivity could be affected.
3. Third, software processes should be focused on flexibility and extensibility rather than on high quality. We should prioritize the speed of the development over zero defects.

- Extending the development in order to reach high quality could result in a late delivery of the product when the opportunity niche has disappeared.

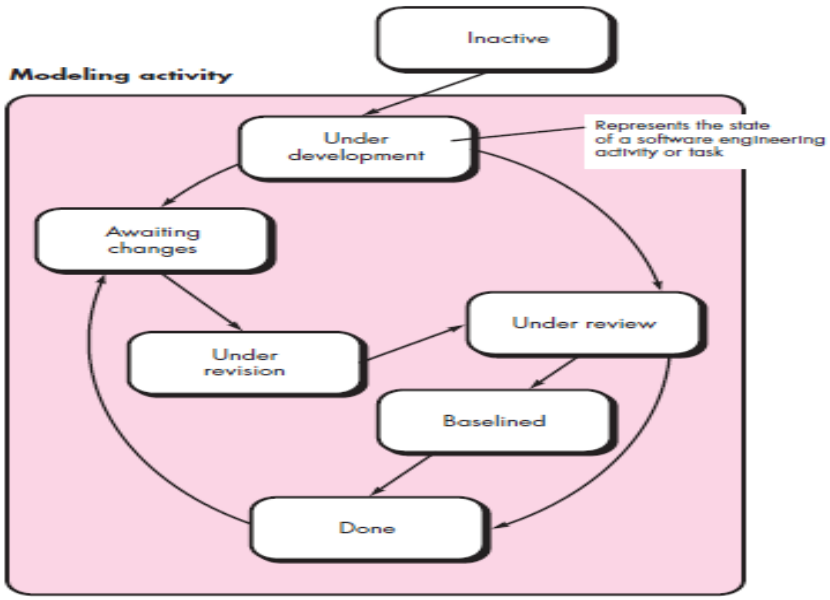
44

Concurrent Models

- Allow a software team to represent iterative and concurrent elements of any of the process models.
- For example, the modeling activity defined for the spiral model is accomplished by invoking one or more of the following actions: prototyping, analysis and design.
- The Figure shows modeling may be in any one of the states at any given time. For example, communication activity has completed its first iteration and in the awaiting changes state.

45

Concurrent Models



Concurrent Models

- The modeling activity was in inactive state, now makes a transition into the under development state. If customer indicates changes in requirements, the modeling activity moves from the under development state into the awaiting changes state.
- Concurrent modeling is applicable to all types of software development and provides an accurate picture of the current state of a project.
- Rather than confining software engineering activities, actions and tasks to a sequence of events, it defines a process network.
- Each activity, action or task on the network exists simultaneously with other activities, actions or tasks.
- Events generated at one point trigger transitions among the states.

47

Specialized Process Models

- Specialized process models (SP models) take on many of the characteristics of one or more of the traditional models.
- However, these models tend to be applied when a specialized or narrowly defined software engineering approach is chosen.

48

Component based development (SP Models)

- Component based development—the process to apply when **reuse is a development objective.**
- **Commercial off the shelf (COTS)** software **components** are offered **as product**.
- **COTS** provides **set of functionality** with **well defined interfaces** that enables component to be integrated into software.
- The component based development model **incorporates** many **characteristics** of the **spiral model**.
- It is **evolutionary** in **nature**.
- Component based development model **constructs** applications from **prepackaged** software **components,** either as conventional software modules or object-oriented classes or packages of classes.
- **Modeling** and **construction** activities begin with the **identification of components**.

49

Component based development (SP Models)

- **Component based development incorporates the following steps :**

1. Available **component-based products** are **researched** & **evaluated** for software development.
2. Component **integration issues** are **considered**.
3. A **software architecture** is **designed** to accommodate the components.
4. Components are **integrated** into the **architecture**.
5. **Testing** is conducted to insure proper functionality.

Advantage

- It leads to **software reuse**.
- It **reduces development** cycle **time**.
- **Reduction** in project **cost**.

50

Formal Methods (SP Models)

- **Formal methods**—emphasizes the mathematical specification of requirements (easy to discover and eliminate ambiguity, incompleteness and inconsistency)
- Formal methods enable you to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation.
- A variation on this approach, called **cleanroom software engineering** is currently applied by some software development organizations.
- Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily—not through ad hoc review, but through the application of mathematical analysis.
- When formal methods are used during design, they serve as a basis for program verification and therefore enable you to discover and correct errors that might otherwise go undetected.

51

Formal Methods (SP Models)

- 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.
  - Because few software developers have the necessary background to apply formal methods, extensive training is required.
  - It is difficult to use the models as a communication mechanism for technically un-sophisticated customers.
- These concerns notwithstanding, the formal methods approach has gained adherents among software developers who must build safety-critical software and among developers that would suffer severe economic hardship should software errors occur.

52

Aspect-Oriented software (SP Models)

- Aspect Oriented software development (AOSD)— provides a process and methodological approach for defining, specifying, designing, and constructing aspects.
- Regardless of the software process that is chosen, the builders of complex software invariably implement a set of localized features, functions, and information content.
- These localized software characteristics are modeled as components (e.g., object oriented classes) and then constructed within the context of a system architecture.
- As modern computer-based systems become more sophisticated (and complex), certain concerns span the entire architecture.
- Some concerns are high-level properties of a system.
- Other concerns affect functions, while others are systemic.

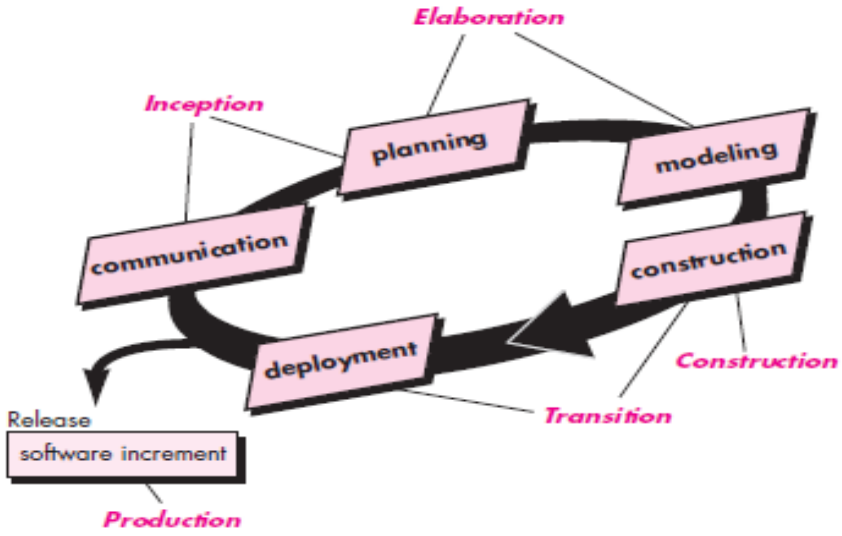
53

Unified Process

- UML provided the necessary technology to support object-oriented software engineering practice, but it did not provide the process framework to guide project teams in their application of the technology.
- Over the next few years, Jacobson, Rumbaugh, and Booch developed the Unified Process, a framework for object-oriented software engineering using UML.
- Today, the Unified Process (UP) and UML are widely used on object-oriented projects of all kinds.
- The iterative, incremental model proposed by the UP can and should be adapted to meet specific project needs.
- Five generic framework activities and argued that they may be used to describe any software process model. The Unified Process is no exception.

54

Unified Process



55

Unified Process

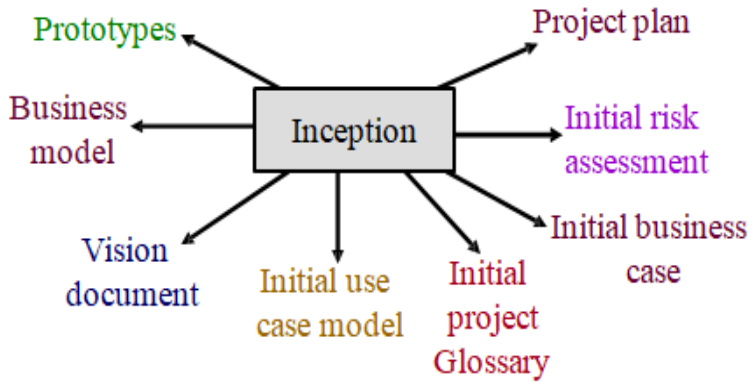
Inception phase

- The inception phase of the UP encompasses both customer communication and planning activities.
- By collaborating with stakeholders, 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.
- Fundamental business requirements are described through a set of preliminary use cases that describe which features and functions each major class of users desires.
- Architecture at this point is nothing more than a tentative outline of major subsystems and the function and features that populate them.

56

Unified Process

Inception phase Outcome



57

Unified Process

Inception phase

- Later, the architecture will be refined and expanded into a set of models that will represent different views of the system.
- Planning identifies resources, assesses major risks, defines a schedule, and establishes a basis for the phases that are to be applied as the software increment is developed.

Elaboration phase

- The elaboration phase encompasses the communication and modeling activities of the generic process model.
- It 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 requirements model, the design model, the implementation model, and the deployment model.**

58

Unified Process

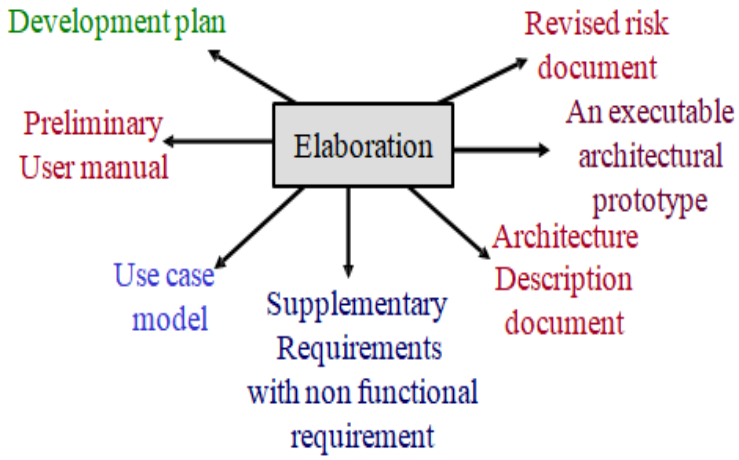
Elaboration phase

- In some cases, elaboration creates an "executable architectural baseline" that represents a "first cut" executable system. [not prototype]
- The architectural baseline demonstrates the viability of the architecture but does not provide all features and functions required to use the system.
- In addition, the plan is carefully reviewed at the culmination of the elaboration phase to ensure that scope, risks, and delivery dates remain reasonable.
- Modifications to the plan are often made at this time.

59

Unified Process

Elaboration phase Outcome



60

Unified Process

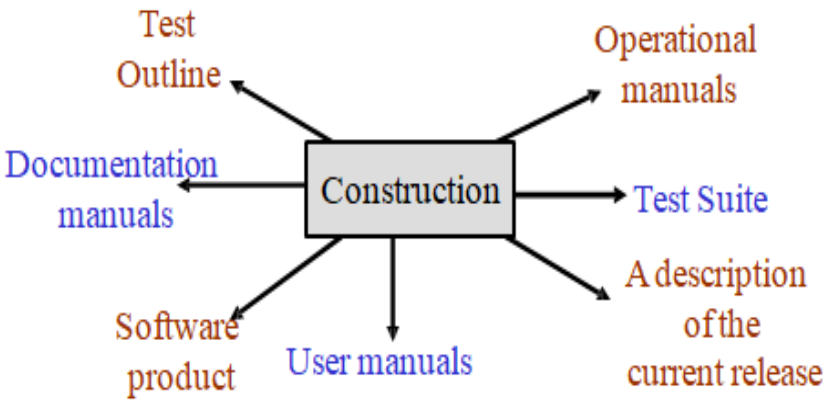
construction phase

- 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.
- To accomplish this, requirements and design models that were started during the elaboration phase are completed to reflect the final version of the software increment.
- As components are being implemented, unit tests are designed and executed for each.
- In addition, integration [cases] and acceptance [cases] activities are conducted.

61

Unified Process

Construction phase Outcome



62

Unified Process

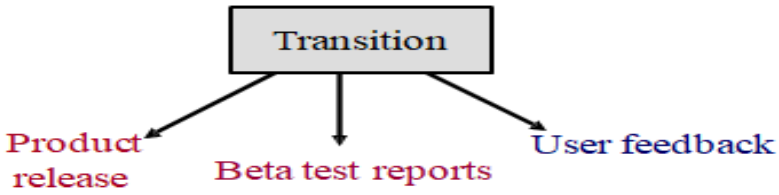
Transition phase

- The transition phase of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment (delivery and feedback) activity.
- Software is given to end users for beta testing and user feedback reports both defects and necessary changes.
- In addition, the software team creates the necessary support information (e.g., user manuals, troubleshooting guides, installation procedures) that is required for the release.
- At the conclusion of the transition phase, the software increment becomes a usable software release.

63

Unified Process

Transition phase Outcome



64

Unified Process

Production phase

- The production phase of the UP coincides with the deployment activity of the generic process.
- During this phase, the ongoing use of the software is monitored,
- support for the operating environment (infrastructure) is provided,
- and defect reports and requests for changes are submitted and evaluated.

65

Unified Process

- It is likely that at the same time the construction, transition, and production phases are being conducted, work may have already begun on the next software increment.
- This means that the five UP phases do not occur in a sequence, but rather with staggered concurrency.
- A software engineering workflow is distributed across all UP phases. In the context of UP, a workflow is analogous to a task set.
- That is, a workflow identifies the tasks required to accomplish an important software engineering action and the work products that are produced as a consequence of successfully completing the tasks.
- It should be noted that not every task identified for a UP workflow is conducted for every software project.
- The team adapts the process (actions, tasks, subtasks, and work products) to meet its needs.

66

Personal and Team Process Model

- The best software process is one that is close to the people who will be doing the work.
- If a software process model has been developed at a corporate or organizational level, it can be effective only if it is amenable to significant adaptation to meet the needs of the project team that is actually doing software engineering work.
- In an ideal setting, you would create a process that best fits your needs, and at the same time, meets the broader needs of the team and the organization.
- Alternatively, the team itself can create its own process, and at the same time meet the narrower needs of individuals and the broader needs of the organization.
- Watts Humphrey argues that it is possible to create a "personal software process" and/or a "team software process."
- Both require hard work, training, and coordination, but both are achievable.

67

Personal Software Process (PSP)

- Every developer uses some process to build computer software.
- The process may be haphazard or ad hoc; may change on a daily basis; may not be efficient, effective, or even successful; but a "process" does exist.
- Watts Humphrey suggests that in order to change an ineffective personal process, an individual must move through four phases, each requiring training and careful instrumentation.
- PSP emphasizes personal measurement of both the work product that is produced and the resultant quality of the work product.
- In addition PSP makes the practitioner responsible for project planning (e.g., estimating and scheduling) and empowers the practitioner to control the quality of all software work products that are developed.

68

Personal Software Process (PSP)

- Planning: This activity isolates requirements and develops both size and resource estimates.
- In addition, a defect estimate (the number of defects projected for the work) is made.
- All metrics are recorded on worksheets or templates.
- Finally, development tasks are identified and a project schedule is created.!
- High-level design: External specifications for each component to be constructed are developed and a component design is created.
- Prototypes are built when uncertainty exists. All issues are recorded and tracked.!
- High-level design review: Formal verification methods are applied to uncover errors in the design. Metrics are maintained for all important tasks and work results.!

69

Personal Software Process (PSP)

- Development: The component level design is refined and reviewed. Code is generated, reviewed, compiled, and tested. Metrics are maintained for all important tasks and work results.!
- Postmortem: Using the measures and metrics collected (this is a substantial amount of data that should be analyzed statistically), the effectiveness of the process is determined. Measures and metrics should provide guidance for modifying the process to improve its effectiveness.!
- PSP represents a disciplined, metrics-based approach to software engineering that may lead to culture shock for many practitioners.
- However, when PSP is properly introduced to software engineers, the resulting improvement in software engineering productivity and software quality are significant.

70

Personal Software Process (PSP)

- However, PSP has not been widely adopted throughout the industry.
- The reasons, sadly, have more to do with human nature and organizational inertia than they do with the strengths and weaknesses of the PSP approach.
- PSP is intellectually challenging and demands a level of commitment (by practitioners and their managers) that is not always possible to obtain.
- Training is relatively lengthy, and training costs are high.
- The required level of measurement is culturally difficult for many software people.

71

Team Software Process (TSP)

- Build self-directed teams that plan and track their work, establish goals, and own their processes and plans.
- These can be pure software teams or integrated product teams (IPT) of three to about 20 engineers. !
- Show managers how to coach and motivate their teams and how to help them sustain peak performance. !
- Accelerate software process improvement by making CMM Level 5 behavior normal and expected. !
  - The Capability Maturity Model (CMM), a measure of the effectiveness of a software process, is discussed later.
  - Provide improvement guidance to high-maturity organizations. !
  - Facilitate university teaching of industrial-grade team skills.!

72

Team Software Process (TSP)

- A self-directed team has a consistent understanding of its overall goals and objectives;
  - defines roles and responsibilities for each team member;
  - tracks quantitative project data (about productivity and quality);
  - identifies a team process that is appropriate for the project and a strategy for implementing the process;
  - defines local standards that are applicable to the team's software engineering work; continually assesses risk and reacts to it;
  - and tracks, manages, and reports project status.

73

Team Software Process (TSP)

- TSP defines the following framework activities:
  - project launch,
  - high-level design,
  - implementation,
  - integration and test, and
  - postmortem.
- Like their counterparts in PSP, these activities enable the team to plan, design, and construct software in a disciplined manner while at the same time quantitatively measuring the process and the product.
- The postmortem sets the stage for process improvements.
- TSP makes use of a wide variety of scripts, forms, and standards that serve to guide team members in their work. "Scripts" define specific process activities and other more detailed work functions that are part of the team process.

74

Team Software Process (TSP)

- Like PSP, TSP is a rigorous approach to software engineering that provides distinct and quantifiable benefits in productivity and quality.
- The team must make a full commitment to the process and must undergo thorough training to ensure that the approach is properly applied.

75

76