Development Process Model

Software Engineering: A Practitioner's Approach, 7/e
by Roger S. Pressman

These slides are designed and adapted from slides provided by *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009) by Roger Pressman and *Software Engineering 9/e* Addison Wesley 2011 by Ian Sommerville

Social Learning Process

- Software adalah perwujudn informasi yang awalnya tersebar, tersembunyi dan tidak lengkap.
- Dalam rangka untuk mengubah pengetahuan ke dalam perangkat lunak, dialog diperlukan antara pengguna dan desainer, antara desainer dan tools untuk membawa informasi ke dalam perangkat lunak.
- Pengembangan perangkat lunak pada dasarnya adalah sebuah proses pembelajaran social yang berulang

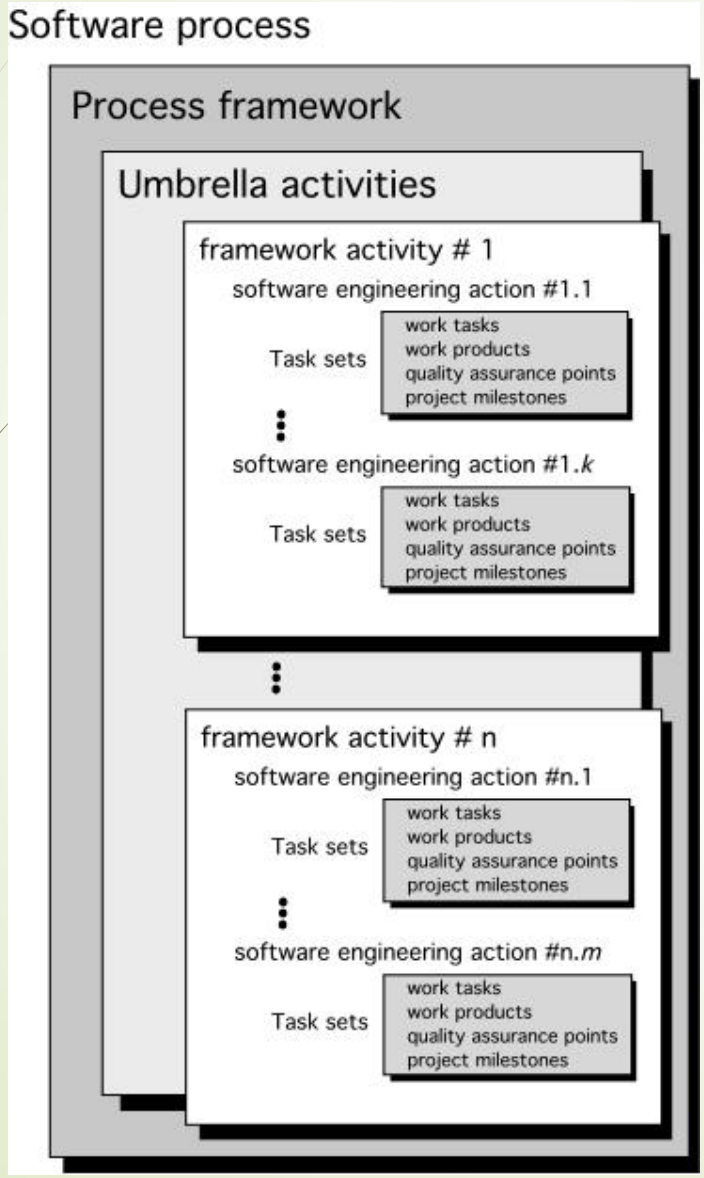
What / who / why is Process Models?

- **What**: Go through a series of predictable steps--- a road map that helps you create a timely, high-quality results.
- **Who**: Software engineers and their managers, clients also. People adapt the process to their needs and follow it.
- **Why**: Provides stability, control, and organization to an activity that can if left uncontrolled, become quite chaotic. However, modern software engineering approaches must be agile and demand ONLY those activities, controls and work products that are appropriate.
- **What Work products**: Programs, documents, and data
- **What are the steps:** The process you adopt depends on the software that you are building. One process might be good for aircraft avionic system, while an entirely different process would be used for website creation.
- **How to ensure right**: A number of software process assessment mechanisms that enable us to determine the maturity of the software process. However, the quality, timeliness and long-term viability of the software are the best indicators of the efficacy of the process you use.

Definition of Software Process

- A **framework** for the activities, actions, and tasks that are required to build high-quality software.
- SP defines the approach that is taken as software is engineered.
- Is not equal to software engineering, which also encompasses **technologies** that populate the process– technical methods and automated tools.

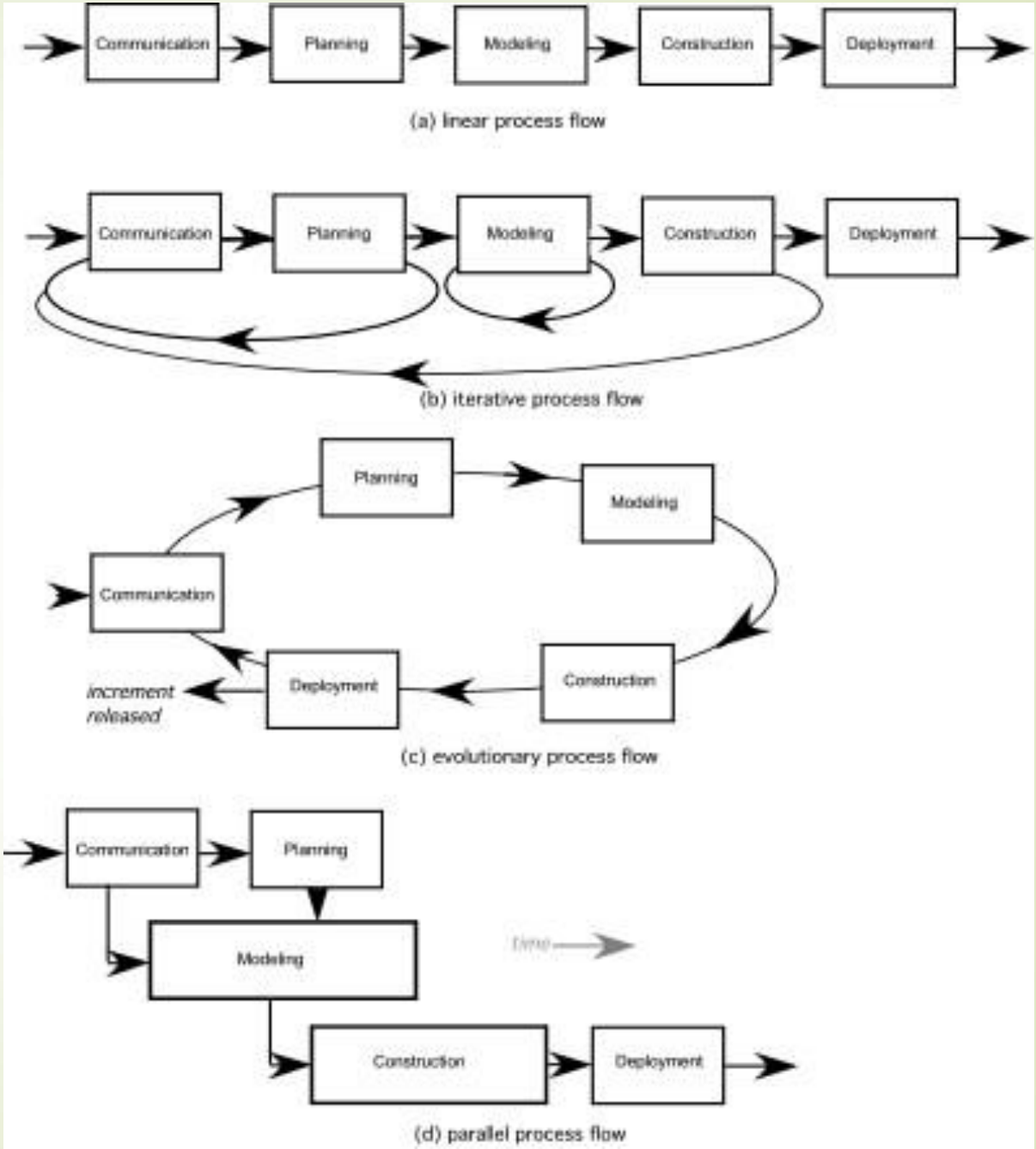
A Generic Process Model



A Generic Process Model

- As we discussed before, a generic process framework for software engineering defines five framework activities-communication, planning, modeling, construction, and deployment.
- In addition, a set of umbrella activities- project tracking and control, risk management, quality assurance, configuration management, technical reviews, and others are applied throughout the process.
- Next question is: how the framework activities and the actions and tasks that occur within each activity are organized with respect to sequence and time? See the **process flow** for answer.

Process Flow



Process Flow

- Linear process flow executes each of the five activities in sequence.
- An iterative process flow repeats one or more of the activities before proceeding to the next.
- An evolutionary process flow executes the activities in a circular manner. Each circuit leads to a more complete version of the software.
- A parallel process flow executes one or more activities in parallel with other activities (modeling for one aspect of the software in parallel with construction of another aspect of the software.

Identifying a Task Set

- Before you can proceed with the process model, a key question: what **actions** are appropriate for a framework activity given the nature of the problem, the characteristics of the people and the stakeholders?
- A task set defines the actual work to be done to accomplish the objectives of a software engineering action.
  - A list of the task to be accomplished
  - A list of the work products to be produced
  - A list of the quality assurance filters to be applied

Identifying a Task Set

- For example, a small software project requested by one person with simple requirements, the communication activity might encompass little more than a phone all with the stakeholder. Therefore, the only necessary action is phone conversation, the work tasks of this action 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.

Example of a Task Set for Elicitation

- The task sets for Requirements gathering action for a **simple** project may include:
  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.

Example of a Task Set for Elicitation

- 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.
  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.

- Both do the same work with different depth and formality. Choose the task sets that achieve the goal and still maintain quality and agility.

Process Patterns

- A *process pattern*
  - describes a process-related problem that is encountered during software engineering work,
  - identifies the environment in which the problem has been encountered, and
  - suggests one or more proven solutions to the problem.

Process Pattern Types

- *Stage patterns*—defines a problem associated with a framework activity for the process. It includes multiple task patterns as well. For example, EstablishingCommunication would incorporate the task pattern RequirementsGathering and others.
- *Task patterns*—defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice
- *Phase patterns*—define the sequence of framework activities that occur with the process, even when the overall flow of activities is iterative in nature. Example includes SprialModel or Prototyping.

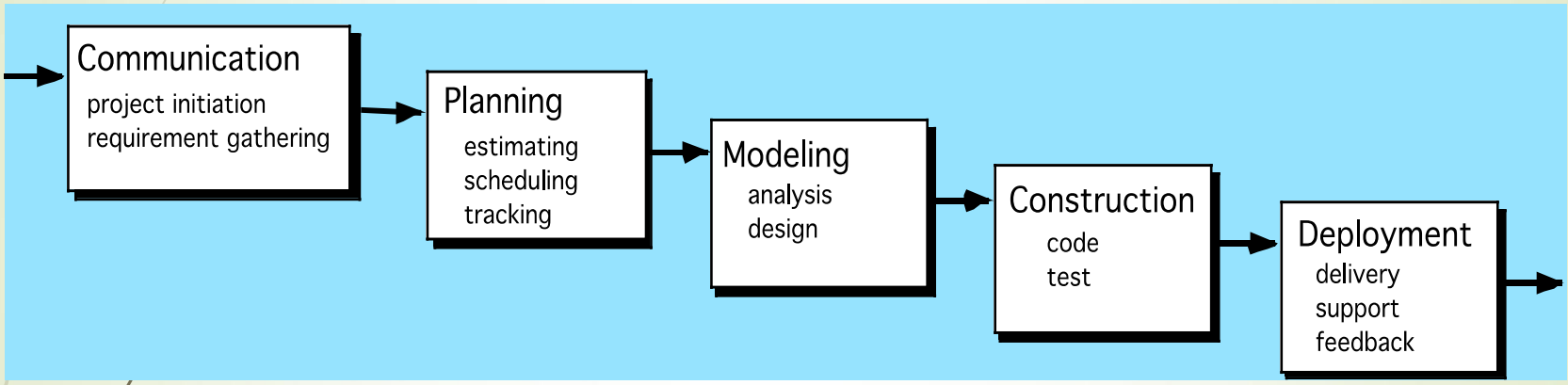
An Example of Process Pattern

- Describes an approach that may be applicable when stakeholders have a general idea of what must be done but are unsure of specific software requirements.
- **Pattern name.** RequiremetnsUnclear
- **Intent.** This pattern describes an approach for building a model that can be assessed iteratively by stakeholders in an effort to identify or solidify software requirements.
- **Type.** Phase pattern
- **Initial context.** Conditions must be met (1) stakeholders have been identified; (2) a mode of communication between stakeholders and the software team has been established; (3) the overriding software problem to be solved has been identified by stakeholders ; (4) an initial understanding of project scope, basic business requirements and project constraints has been developed.
- **Problem**. Requirements are hazy or nonexistent. stakeholders are unsure of what they want.
- **Solution.** A description of the prototyping process would be presented here.
- **Resulting context**. A software prototype that identifies basic requirements. (modes of interaction, computational features, processing functions) is approved by stakeholders. Following this, 1. This prototype may evolve through a series of increments to become the production software or 2. the prototype may be discarded.
- **Related patterns.** CustomerCommunication, IterativeDesign, IterativeDevelopment, CustomerAssessment, RequirementExtraction.

Process Assessment and Improvement

SP cannot guarantee that software will be delivered on time, meet the needs, or has the desired technical 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.

The Waterfall Model

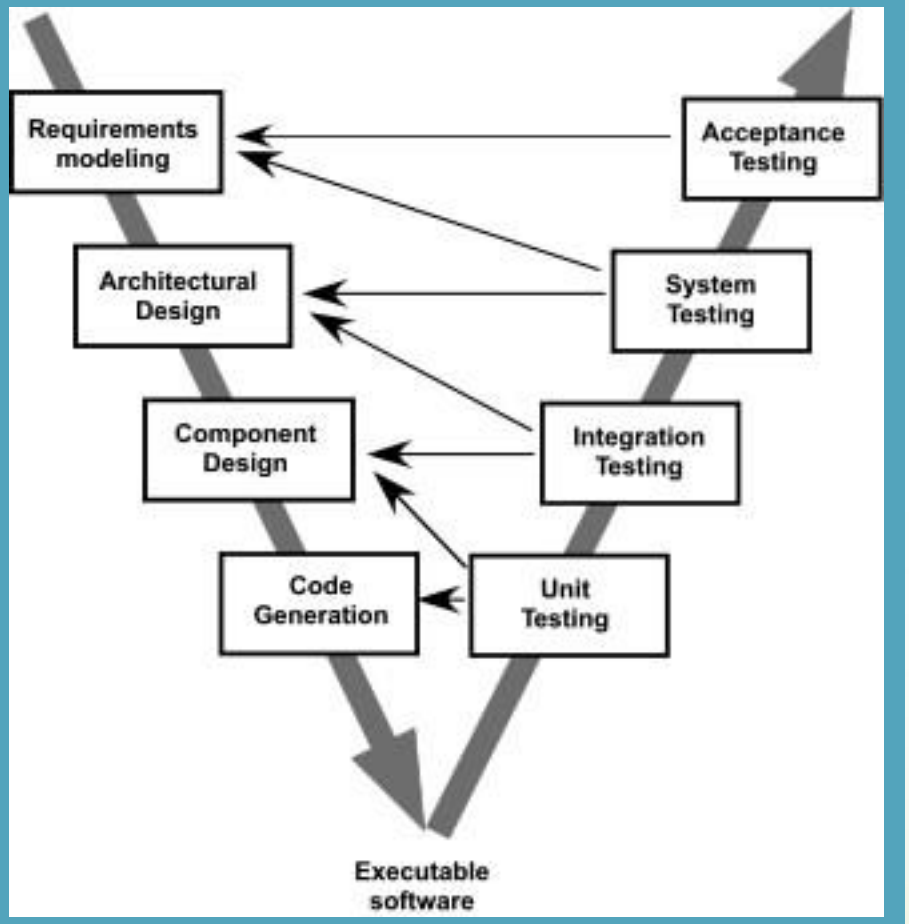


It is the oldest paradigm for SE. When requirements are well defined and reasonably stable, it leads to a linear fashion.

(problems: 1. rarely linear, iteration needed. 2. hard to state all requirements explicitly. Blocking state. 3. code will not be released until very late.)

The classic life cycle suggests a systematic, sequential approach to software development.

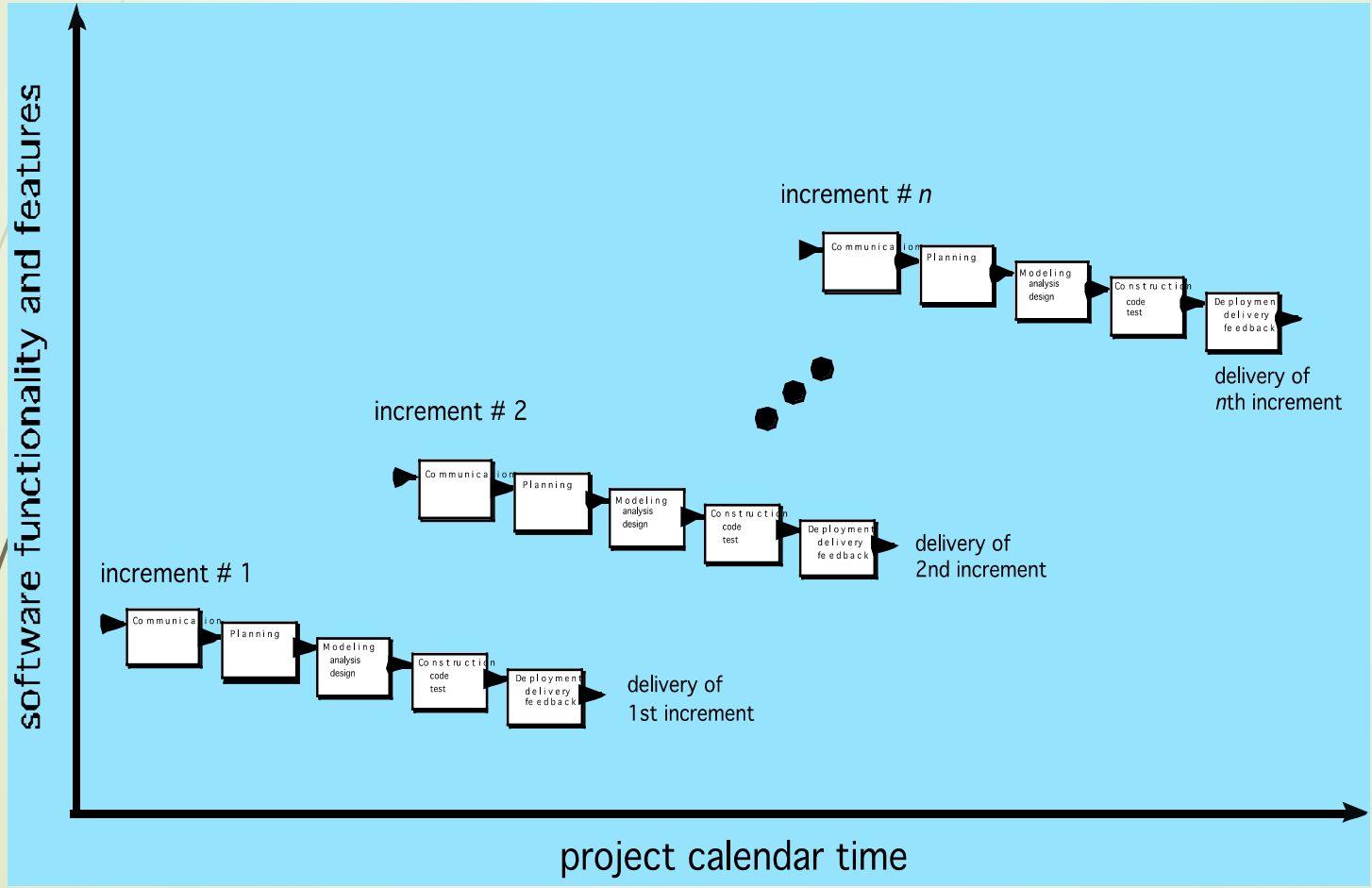
The V-Model



A variation of waterfall model depicts the relationship of quality assurance actions to the actions associated with communication, modeling 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 validate each of the models created as the team moved down the left side.

The Incremental Model



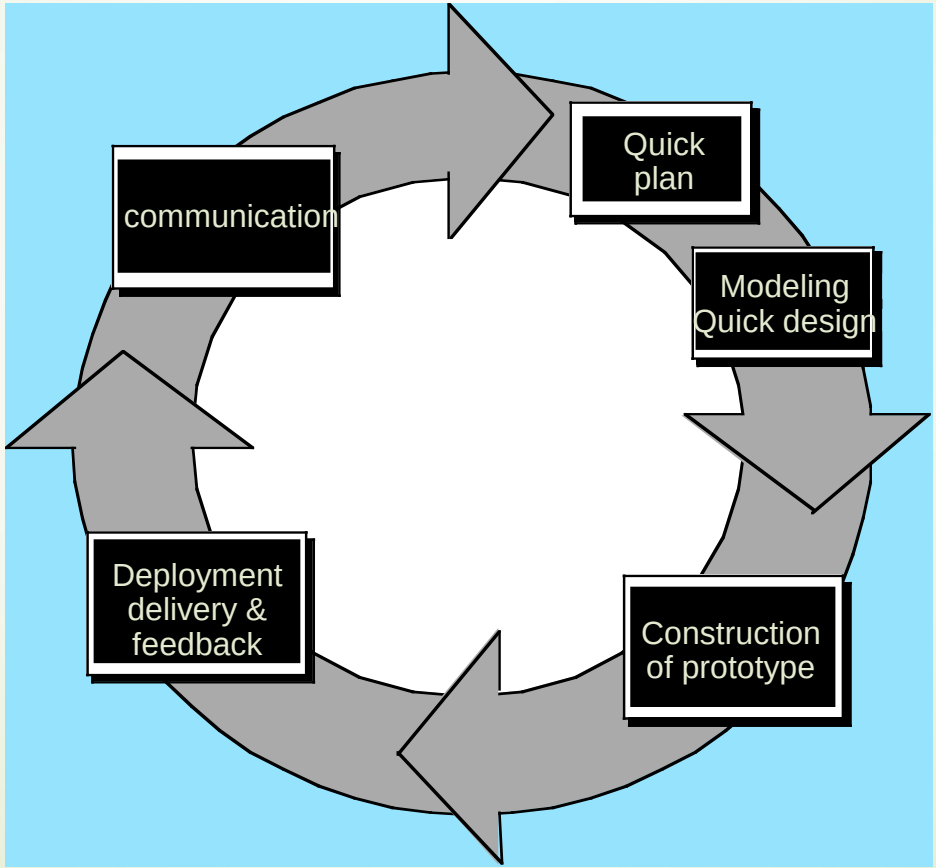
The Incremental Model

- When initial requirements are reasonably well defined, but the overall scope of the development effort precludes a purely linear process. A compelling need to expand a limited set of new functions to a later system release.
- It combines elements of linear and parallel process flows. Each linear sequence produces deliverable increments of the software.
- The first increment is often a core product with many supplementary features. Users use it and evaluate it with more modifications to better meet the needs.

Evolutionary Models: Prototyping

- When to use: Customer defines a set of general objectives but does not identify detailed requirements for functions and features. Or Developer may be unsure of the efficiency of an algorithm, the form that human computer interaction should take.
- 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. 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.

Evolutionary Models: Prototyping



What is Agility?

- Effective (rapid and adaptive) response to change (team members, new technology, requirements)
- Effective communication in structure and attitudes among all team members, technological and business people, software engineers and managers
- Drawing the customer into the team. Eliminate "us and them" attitude. Planning in an uncertain world has its limits and plan must be flexible. !
- Organizing a team so that it is in control of the work performed!
- Eliminate all but the most essential work products and keep them lean.!
- Emphasize an incremental delivery strategy as opposed to intermediate
- products that gets working software to the customer as rapidly as feasible. !

Why and what steps are "Agility" importants

- Why? The modern business environment is fast-paced and ever-changing. It represents a reasonable alternative to conventional software engineering for certain classes of software projects. It has been demonstrated to deliver successful systems quickly. !
- What? May be termed as "software engineering lite" The basic activities- communication, planning, modeling, construction and deployment remain. But they morph into a minimal task set that push the team toward construction and delivery sooner. !
- The only really important work product is an operational "software increment" that is delivered. !

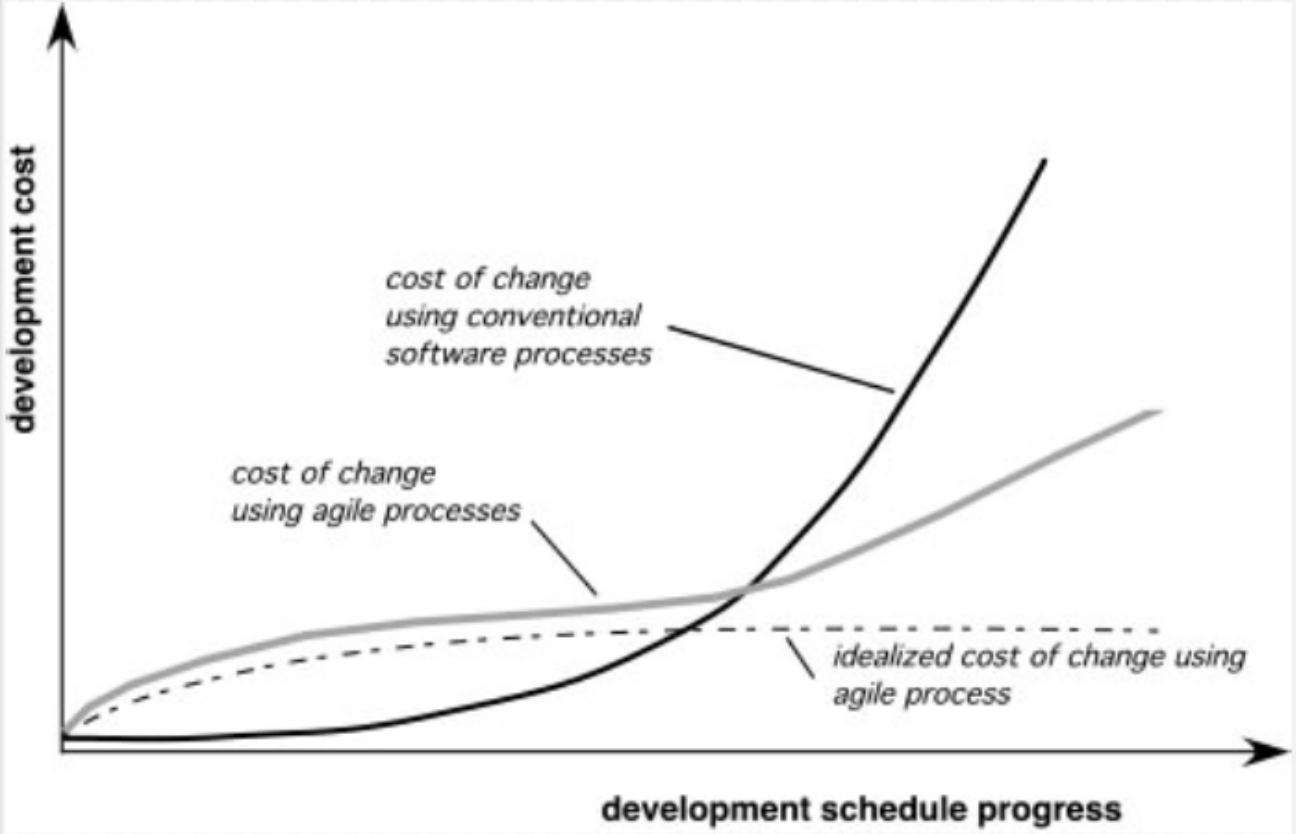
Agility and the cost of change

- Conventional wisdom is that the cost of change increases nonlinearly as a project progresses. It is relatively easy to accommodate a change when a team is gathering requirements early in a project. If there are any changes, the costs of doing this work are minimal. But if the middle of validation testing, a stakeholder is requesting a major functional change. Then the change requires a modification to the architectural design, construction of new components, changes to other existing components, new testing and so on. Costs escalate quickly.

Agility and the cost of change (2)

- A well-designed agile process may "flatten" the cost of change curve by coupling incremental delivery with agile practices such as continuous unit testing and pair programming. Thus team can accommodate changes late in the software project without dramatic cost and time impact.

Agility and the Cost of Change



An Agile Process

- Is driven by customer descriptions of what is required (scenarios). Some assumptions:

1. Recognizes that plans are short-lived (some requirements will persist, some will change. Customer priorities will change)
2. Develops software iteratively with a heavy emphasis on construction activities (design and construction are interleaved, hard to say how much design is necessary before construction. Design models are proven as they are created.)
3. Analysis, design, construction and testing are not predictable.

- Thus has to Adapt as changes occur due to unpredictability!
- Delivers multiple 'software increments', deliver an operational prototype or portion of an OS to collect customer feedback for adaption.

Agile Principles (1)

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face–to–face conversation.

Agile Principles (2)

- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity – the art of maximizing the amount of work not done – is essential.
- The best architectures, requirements, and designs emerge from self–organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

SCRUM

- A software development method Originally proposed by Schwaber and Beedle (an activity occurs during a rugby match) in early 1990.
- Scrum—distinguishing features

1. Development work is partitioned into "packets"
2. Testing and documentation are on-going as the product is constructed
3. Work units occurs in "sprints" and is derived from a "backlog" of existing changing prioritized requirements
4. Changes are not introduced in sprints (short term but stable) but in backlog.

SCRUM (2)

5. Meetings are very short (15 minutes daily) and sometimes conducted without chairs (what did you do since last meeting? What obstacles are you encountering? What do you plan to accomplish by next meeting?)!
6. "demos" are delivered to the customer with the time-box allocated. May not contain all functionalities. So customers can evaluate and give feedbacks.