Software Development Processes & Process Models - Review

Eunjee Song
Dept. of Computer Science
Baylor University

Slide Sources: OBJECT ORIENTED & CLASSICAL SOFTWARE ENGINEERING
by S. Schach and Slides by Dr. D. Simmonds

Outline
- Software engineering process
- Process characteristics
- Software process models
  - Waterfall
  - Iterative/incremental
  - Rapid Application Development (RAD)
  - More RAD flavors: Evolutionary, Scrum
  - Component-based Software Development (CBSD)
  - The Unified Process (UP)
- Process maturity
- Process benefits

What is a software process?
- A set of activities whose goal is to develop or evolve software.
- A series of steps involving activities, constraints, and resources that produce an intended output.
- A sequence of steps required to develop or maintain software. (Watts Humphrey)
- Generic activities in all software processes are:
  - Specification - what the system should do and its development constraints
  - Development - production of the software system
  - Validation - checking that the software is what the customer wants
  - Evolution - changing the software in response to changing demands

Generic software process activities
- Generic activities in all software processes are lifecycle activities -
  - Workflow perspective represents inputs, outputs and dependencies
  - Data-flow perspective represents data transformation activities
  - Role/action perspective represents the roles/activities of the people involved in software process

A process model?
- An abstract representation of a process presented from a specific perspective.
  - e.g., The Circulatory System perspective
    - Flow of blood in CS
    - Blood vessels of the CS
    - Role of circulatory system in digestion.

Software process model?
- An abstract representation of a software process presented from a specific perspective
  - Workflow perspective represents inputs, outputs and dependencies
  - Data-flow perspective represents data transformation activities
  - Role/action perspective represents the roles/activities of the people involved in software process
Generic Software Process Models
- **Code and Fix**
- **The waterfall model**
  - Separate and distinct phases of specification and development
- **Incremental & iterative models**
  - Specification and development are interleaved
- **Evolutionary models**
- **Formal models**
  - A mathematical system model is formally transformed to an implementation
- **Component-based software development**
  - The system is assembled from existing components

“Code and Fix” Model
- **Process**
  - code, test, debug
- **Problems**
  - requirements analysis and design ignored
  - code does not evolve gracefully
  - debugging is difficult (why?)

Waterfall Model
- Proposed by Royce in response to desire to structure software development process.
- Phases similar to phases used to develop hardware.
- Boehm introduced iteration and validation at each phase

Problems with Waterfall Model
- Lack of feedback between phases
- Hard dates on phases
- Lack of completion criteria for analysis and design
- No insight given into how each activity transforms an intermediate product into another.
- Provides little guidance on how to handle changes to products and activities.

Problems with Waterfall Model (cont.)
- Treats development as a manufacturing process
  - Model is optimized for hardware development.
  - Does not take into consideration problem-solving nature of development.
  - Does not support design a little, code a little problem solving strategy.
- **Summary**
  - The difficulty of accommodating change after the process is underway
  - Takes static view of requirements

Waterfall model usage
- Model is appropriate when the requirements are well-understood

Waterfall model describes a process of stepwise refinement
- Based on hardware engineering models
- Widely used in military and aerospace industries
### Incremental and Iterative Models

#### DEPARTMENTS

- Developers
- Users

#### BUILD

- Build 1
- Build 2
- Build 3

#### RELEASE

- Release 1
- Release 2
- Release 3

---

### Incremental/Iterative Development

- **Incremental Development**
  - Add new features each iteration

- **Iterative Development**
  - Edit, modify, refactor, even if no new features are added

---

### The Incremental Model

#### Process

- Communication
- Planning
- Modeling
- Construction
- Deployment
- Delivery
- Feedback
- Analysis
- Design
- Code
- Test

#### Timeline

- Increment #1: 1st increment
- Increment #2: 2nd increment
- Increment #3: Deliveries

---

### Incremental Development

- **User requirements are prioritised** and the highest priority requirements are included in early increments.

- **Once the development of an increment is started**, the requirements are frozen though requirements for later increments can continue to evolve.

---

### Benefits of Iterative/Incremental Development

- **Incremental development**: Each build focuses on a subset of functionality, thus simplifying development.

- **Opportunity to learn about problem as design is developed**.
Benefits of Iterative/Incremental Development (continued)

- Users get a product early and can provide feedback for future builds.
- Unanticipated problems can be fixed globally in future releases.
- Different releases can focus on enhancements that require specialized expertise.
- Lower risk of overall project failure

Problems

- If builds are not properly planned, evolution can result in complex system.
- Backward compatibility can limit elegance of future releases.
- An error in the basic architecture of the system may require changes to basic structure that invalidate earlier releases.

The RAD Model

Evolutionary development

Evolutionary Models: Prototyping

Evolutionary Models: The Spiral Model
**Software Development Lifecycle**

**Spiral Model**
- Determine objectives, alternatives, constraints
- Evaluate alternatives, identify, resolve risks, develop prototypes
- Plan next phases
- Develop, verify next-level product

**Spiral development**
- Process is represented as a spiral rather than as a sequence of activities with backtracking
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required
- Risks are explicitly assessed and resolved throughout the process

**Evolutionary development**
- **Problems**
  - Lack of process visibility
  - Systems are often poorly structured
  - Special skills (e.g., in languages for rapid prototyping) may be required
- **Applicability**
  - For small or medium-size interactive systems
  - For parts of large systems (e.g., the user interface)
  - For short-lifetime systems

**SCRUM Framework**
- **Main roles in Scrum**
  - Scrum Master: maintains the processes (typically in lieu of a project manager)
  - Product Owner: represents the stakeholders and the business
  - Team: a cross-functional group of about 7 people who do the actual analysis, design, implementation, testing, etc.

  Scrum References: [http://members.cox.net/risingl1/Articles/IEEEScrum.pdf](http://members.cox.net/risingl1/Articles/IEEEScrum.pdf), [http://scrummethodology.com](http://scrummethodology.com), [http://www.scrumalliance.org](http://www.scrumalliance.org)

**SCRUM Framework (cont.)**
- A product owner creates a prioritized wish list called a product backlog.
- During sprint planning, the team pulls a small chunk from the top of that wish list, a sprint backlog, and decides how to implement those pieces.
- The team has a certain amount of time, a sprint, to complete its work - usually two to four weeks - but meets each day to assess its progress (daily scrum).
- Along the way, the Scrum Master keeps the team focused on its goal.

- At the end of the sprint, the work should be potentially shippable, as in ready to hand to a customer, put on a store shelf, or show to a stakeholder.
- The sprint ends with a sprint review and retrospective.
- As the next sprint begins, the team chooses another chunk of the product backlog and begins working again.
- The cycle repeats until enough items in the product backlog have been completed, the budget is depleted, or a deadline arrives.
SCRUM Framework (cont.)

- **Pros**
  - Improvement in productivity in teams by less-heavy process
  - Daily measured progress and communications
  - Utilization of backlog for completing items in a series of short iterations or sprints.

- **Cons**
  - Reliance on facilitation by a master who may lack the political clout to remove impediments and deliver the sprint goal.
  - Due to its reliance on self-organizing teams and the rejection of the traditional centralized "process control", internal power struggles may paralyze the team.

Component-based Software Development (CBSD)

- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems

- **Process stages**
  - Component analysis
  - Requirements modification
  - System design with reuse
  - Development and integration

  This approach is becoming more important but still limited experience with it.

Component-related Technologies

- Component-oriented programming
  - Enterprise JavaBeans (EJB) from Sun Microsystems

- Component-based domain-specific S/W framework
  - Earth System Modeling Framework (ESMF)

- Compound document technologies
  - Object linking and embedding (OLE)
  - OpenDoc

The Unified Process (UP)

- **Workflows**
  - Requirements
  - Analysis
  - Design
  - Implementation
  - Test
  - Support

- **UP Phases**
  - Inception
  - Elaboration
  - Construction
  - Transition
  - Production
**UP Work Products**

**Inception phase**
- Vision document
- Model use-case model
- Propose project strategy
- List learning cases
- Draft project plan
- Stakeholder analysis
- Business model
- If necessary: one or more prototypes

**Elaboration phase**
- Use-case model
- Supplementary requirements
- Initial use-case model
- Requirements analysis
- Design model
- Architecture model
- Functional and non-functional requirements
- Use-case model
- Software product
- Development plan
- Iteration plan
- draft work products

**Construction phase**
- Design model
- Software components
- Integration
- Test plans and procedures
- Test plans and procedures
- Deliver software increments
- Beta test reports
- General user feedback

**Transition phase**
- Requirements statement
- Design
- Implementation

---

**Impedance mismatches**

- As Management requested it.
- As the Project Leader defined it.
- As Systems designed it.
- As Programming developed it.
- As Operations installed it.
- What the user wanted.

---

**Essence of Software Development**

- **Requirements Statement**
- **Design**
- **Implementation**

---

**Formal systems development**

- Based on the transformation of a mathematical specification through different representations to an executable program
- Transformations are ‘correctness-preserving’ so it is straightforward to show that the program conforms to its specification

---

**Problems**
- Need for specialisation skills and training to apply the technique
- Difficult to formally specify some aspects of the system such as the user interface

**Applicability**
- Critical systems especially those where a safety or security case must be made before the system is put into operation

---

**Other Process Models**

- Transformational
  - formal models mechanically transformed to code
- Operational
  - building of a requirements specification that is executable
Process Models Overview

- Software process models can be classified along a spectrum ranging from:
  - radical: all activities carried out in parallel
  - conservative: all activities carried out in a strict sequence with no overlap.
- None of the extremes are viable.

Choosing a Model

- Choice depends on nature of project:
  - Are requirements clearly defined and stable?
  - Is there pressure to produce a working product quickly?
  - Are the consequences of operational errors serious?

Radical vs. Conservative Models

- More radical models suitable when:
  - quick results are needed
  - requirements are fuzzy or unstable
- More conservative models suitable when:
  - consequences of errors are very serious
  - requirements are well-understood and stable

Process Maturity

- Processes should be tailored to development environment.
- The software process maturity framework (SEI) allows organizations to determine the capabilities of their current processes and establish priorities for improvement.

Capability Maturity Model (CMM)

- A framework to determine process maturity
- Initial: Few defined processes; success depends on individual effort.
- Repeatable: Cost, schedule, and product tracking processes in place.
- Defined: Standard processes are defined and used.
- Managed: Defined processes and product qualities are meaningfully measured.
- Optimizing: Measures used to improve process/product.

An Immaturity Model!

Defined by A. Finkelstein:
- Foolish (0): failure to allow successful development process.
- Stupid (-1): Counter-productive process is imposed.
- Lunatic (-2): Contempt for good software engineering practices is institutionalized.
Elements of Process Descriptions

- Each activity has an entry and exit criteria.
- Activities are organized in a sequence.
- Every process has a set of guiding principles that explain goals of each activity.
- A process may be composed of subprocesses.

What are the benefits of a defined software processes?

A defined process:
- facilitates communication about the process
- provides a basis for process automation
- facilitates process reuse
- facilitates process evolution
- aid process management

Summary

- What did we discuss in this class period?
  - Software engineering process
  - Process characteristics
  - Software process models
    - Waterfall
    - Iterative/incremental
    - Rapid Application Development (RAD)
    - More RAD flavors
      - Evolutionary: Prototyping/Spiral
      - Scrum
    - Component-based Software Development (CBSD)
    - The Unified Process (UP)
  - Process maturity
  - Process benefits