An Overview of Software Process
Objectives

- To introduce the general phases of the **software development life cycle (SDLC)**
- To describe various generic **software process models** and discuss their pros and cons
- To introduce some specific **software processes**
- To discuss software process **assessment and improvement**
Generalizing the Software Development Life Cycle (SDLC)

- Specify system scope
- Elicit and specify system services
- Elicit and specify system constraints
- Begin designing the user interface (isn’t this design?!)
- Establish deliverables
- Discuss open issues
- Document
- Verify
Generalizing the Software Development Life Cycle (SDLC)

- Requirements
  - system services and constraints
  - “What”
  - intended system structure
  - “How”

- Design
  - Overall architectural design
  - Component design
  - Component interface design
  - Algorithm design
  - Data structure design
  - Hardware and software decisions
  - Discuss open issues
  - Document
  - Verify
Generalizing the Software Development Life Cycle (SDLC)

- **Requirements**
  - system services and constraints
  - “What”

- **Design**
  - intended system structure
  - “How”

- **Implementation**
  - coding
  - successful compilation of code units
  - unit testing
  - code inspection
  - document
Generalizing the Software Development Life Cycle (SDLC)

- Requirements
  - system services and constraints
  - “What”

- Design

- Implementation
  - intended system structure
  - “How”

- Testing
  - code

- final product

- Component testing
- Integration testing
  - Subsystem testing
  - System testing
- Acceptance testing
- Document
- Deployment (actually its own phase)
Generalizing the Software Development Life Cycle (SDLC)

- Requirements: system services and constraints
- Design: “What”
- Implementation: intended system structure
  “How”
- Testing: code

Final product

Maintenance

- Bug fixes
- Refactoring
- Upgrades
- Document
Software Process *Models*

- An abstract representation of how the SDLC phases can be addressed

- Major models:
  - Waterfall
  - Spiral
  - Iterative and Incremental Development (IID)
  - Prototyping
    - Evolutionary
    - Throwaway
Waterfall Model

Winston Royce, 1970
Observations

- Contains all phases of the SDLC
- May have to return to the previous phase
- Still widely used, especially on very large projects
Spiral Model

Determine objectives, alternatives and constraints

Evaluate alternatives, identify, resolve risks

Risk analysis

Prototype 1

Prototype 2

Prototype 3

Operational prototype

Simulations, models, benchmarks

Product design

Detailed design

Code

Unit test

Integration test

Develop, verify next-level product

Concept of Operation

S/W requirements

Risk analysis

Risk analysis

Requirement validation

Design V&V

Acceptance test

Integration and test plan

Service

Plan next phase

Requirements plan

Life-cycle plan

REVIEW

Review Barry Boehm, 1988
Observations

- Each loop in the spiral represents a phase in the process.
- Is **iterative**
- **Risks** are explicitly assessed and resolved throughout the process.
- Uses **prototyping**
Iterative and Incremental Development (IID)

Requirements

Design

Implementation

Testing

Maintenance

Determine the “pieces”

Develop each “piece,” adding to the previous ones

Final system emerges
Observations

- Contains all phases of the SDLC
- Development and delivery is broken down into functional increments ("pieces")
- The increments are prioritized
- Is an **iterative, incremental** process
- Common to deploy at the end of each iteration
Prototyping

Requirements

Design

Implementation

Testing

Throw prototype away?

Design

Implementation

Testing

Maintenance

Final System Development (waterfall, spiral, IID, etc.)

Prototyping (waterfall, spiral, IID, etc.)

CMSC 345, Version 1/12
S. Mitchell
Observations

- Contains all phases of the SDLC
- Terrific requirements elicitation and validation technique
- There is always a “working” model (prototype) of the final system
- Is an **iterative** process
- Prototype can be thrown away (**throwaway prototyping**) or evolved into the final system (**evolutionary prototyping**)

![Diagram of the prototyping process](image-url)
Software Processes

- Rational Unified Process (RUP) (’90’s)
- Agile processes (late ’90’s)
  - Scrum
  - Extreme Programming (XP)
- Customized
Rational Unified Process (RUP)

- Rational Software Corporation, now owned by IBM
  - “Three Amigos”
    - Grady Booch
    - James Rumbaugh
    - Ivar Jacobson

- A popular type of Unified Process (UP)
Rational Unified Process (1)
Rational Unified Process (UP) (2)

- Set of activities (workflows), artifacts (e.g., documents, diagrams, code), and roles (e.g., architect, code reviewer, tester)
- Customizable generic process framework
- Characteristics
  - Use case driven (functional requirements)
  - Architecture-centric (system structure)
  - Iterative (cycles through “workflows”)
  - Incremental (incremental deliveries of a specified set of use cases)
- Makes extensive use of the Unified Modeling Language (UML)
Agile Processes

- Agile Manifesto (2001)
  - Emphasizes “lightweight” processes
  - Values
    - Individuals and interactions over processes and tools
    - Working software over comprehensive documentation
    - Customer collaboration over contract negotiation
    - Responding to change over following a plan
  - [www.agilemanifesto.org](http://www.agilemanifesto.org)

- Some agile processes
  - Scrum
  - Extreme Programming (XP) (Is it a process?)
Scrum (1)

Rugby – A way of restarting the game after an infringement or after the ball goes out of play
Scrum (2)  

“Scrum is superimposed on and encapsulates whatever engineering practices already exist.”

Roles

- **Scrum Master**
  - Responsible for ensuring that Scrum values, practices, and rules are enacted and enforced
  - Represents management and the team to each other
  - Responsible for the success of the Scrum

- **Product Owner**
  - Solely controls the Product Backlog

- **Scrum Team**
  - Commits to achieving a Sprint goal
  - Accorded full authority to do whatever it decides is necessary to achieve the goal
  - Responsible for doing all of the analysis, design, coding, testing, and user documentation
  - Self-organizing, cross-functional

- **Stakeholders**
  - Customers, vendors, others
Scrum (3)

- Some Tasks
  - Daily Scrums
    - What the team has accomplished since the last meeting
    - What it is going to do before the next meeting
    - What obstacles are in its way
  - 30-day Sprints
    - Sprint planning meeting
    - Sprint goal
    - End-of-Sprint review

- Some Artifacts
  - Product Backlog
    - An evolving, prioritized queue of business and technical functionality that needs to be developed into a system.
  - Release Backlog
    - The subset of the Product Backlog that is selected for a release.
  - Sprint Backlog
    - Tasks that the Scrum Team has devised for a Sprint.
Extreme Programming (XP) (1)

- Basic principles (Beck)
  - Rapid feedback
  - Assume simplicity
  - Incremental change
  - Embracing change
  - Quality work
Extreme Programming (XP) (2)

- Practices
  - The planning game
  - Small releases
  - Metaphor
  - Simple design
  - Testing
  - Refactoring
  - Pair programming
  - Collective ownership
  - Continuous integration
  - 40-hour week
  - On-site customer
  - Coding standards
Customized Processes

- Sometimes (usually?) it’s best to “pick and choose”
- Questions to ask:
  - Is there a required process?
  - Are the requirements well-understood?
  - What else? (Think about this on your own.)
Assessing Process (1)

- Software “crisis” in the 1960’s, ’70’s, ’80’s
  - Over budget
  - Over schedule
  - Poor quality

- Software Engineering Institute (SEI)
  - Carnegie Mellon University
  - Federally-funded, non-profit research and development center
  - Consortium of academia, government, and industry
  - Mission: to “advance the practice of software engineering” (from www.sei.cmu.org)
Assessing Process (2)

- **SEI Capability Maturity Model (CMM), 1991**
  - Provides guidance for software process improvement
  - Also a method for assessing the maturity of an organization’s software process

- **Capability Maturity Model Integration (CMMI), 2002**
  - Successor to CMM
  - Version 1.2, released August 2006
  - Five levels of process “maturity”
    - Incomplete
    - Initial (ad hoc)
    - Managed (can repeat earlier successes)
    - Defined (standardized and documented process)
    - Quantitatively Managed (software process metrics gathered)
    - Optimizing (continuous process improvement)
  - Is not a specific process
  - Is process-independent
Assessing Process (3)

- Some government agencies and other organizations require contractors to have achieved a specific minimal CMMI level

- Other standards and certifications:
  - ISO 9000 (International Organization for Standardization)
    - A family of standards
    - Can be certified as “ISO 9000 compliant”
  - Six Sigma
    - Originally developed by Motorola
    - Origins in quality (defect) control in manufacturing
    - Various certifications
CMSC 345 Process (1)

- Linear process. Why?
  - First time through the entire life cycle
  - Semester is very short
  - I must give you hard deadlines
- Probably will have to integrate some iteration into the process
- Prototyping strongly recommended
  - For requirements elicitation
  - Keep your customer informed (and happy!)
References (1)

References (2)

- Software Engineering Institute (SEI), www.sei.cmu.edu
- Software Engineering Institute CMMI Website, http://www.sei.cmu.edu/cmmi/