CSC 440/540: Software Engineering

Software Development Methodologies
Topics

1. The Waterfall Model
2. Agile Software Development
3. The Unified Process
4. Object-Oriented Analysis and Design
5. The Inception Phase
Software Life Cycles

Waterfall Model

- Early model of software engineering derived from hardware manufacturing
- Each stage produces a product, which is consumed by next stage
- No going backwards
Waterfall Model

- Requirements
- Design
- Implementation
- Testing
- Maintenance
The Waterfall Model, cont’d

Make sure each phase is **100% complete and absolutely correct** before proceeding to the next phase.

**Big Design Up Front (BDUF)**

- Set requirements in stone before starting the design.
- Otherwise, you might waste design work on “incorrect” requirements.
- Spend extra time at the beginning to **make sure that the requirements and design are absolutely correct**.

Source: Wikipedia article on “Waterfall Model”
Waterfall Model Problems

- Faulty Assumptions
  - Requirements can be fairly accurate
  - Requirements are stable
  - Design can be done before coding
- Large steps
- No feedback
- Heavy process
  - Many artifacts created to rigidly predict and control software development in a bureaucratic environment
We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Source: http://agilemanifesto.org/
Agile Software Development

- Iterative and incremental development.

- Each iteration is a “mini waterfall”:
  - plan (with new requirements)
  - refine design
  - add new code
  - unit and integration testing

- Iterations are short: weeks rather than months.

- Iterations are sometimes called “sprints”.
  - We do sprints, not marathons!
Agile Software Development

- The initial iteration produces a conceptual design and a prototype.

- Subsequent iterations refine the design and incrementally build the actual product.

- Each subsequent iteration may also include a prototype that is quickly produced (rapid prototyping).
The initial iteration Prototyping and iterative development are the foundation for Rapid Application Development (RAD) tools.

- Is Ruby on Rails RAD?

Agile methodologies range from Extreme Programming (XP) to the Rational Unified Process (RUP).
Project Phases, cont’d
Development is a series of iterations.
Each iteration is a “mini waterfall” consisting of design, code (implementation), and test.
Extreme programmers say: design, test, code
The Unified Process

Iterative and incremental development process

- Each iteration includes requirements, design, implementation, and test activities
- Each iteration produces a working partial system
- Customer feedback on partial system alters requirements, so final system meets customer needs
- Iterations are timeboxed: 2-8 weeks
- Address high risk aspects in early iterations

Created by Rational Software (now IBM)
UP Iterations

4 weeks (for example)

Iterations are fixed in length, or timeboxed

Feedback from iteration N leads to refinement and adaptation of the requirements and design in iteration N+1.

The system grows incrementally.
UP Schedule and Phases

Schedule organized into 4 phases

- Each phase consists of one or more iterations
- Phases different from Waterfall Model Phases

UP Phases

- Inception
- Elaboration
- Construction
- Transition
UP Schedule

- **iteration**: Increment, Elaboration, Construction, Transition
- **phase**: Development Cycle
- **milestone**: An iteration end-point when some significant decision or evaluation occurs.
- **release**: A stable executable subset of the final product. The end of each iteration is a minor release.
- **increment**: The difference (delta) between the releases of 2 subsequent iterations.
- **final production release**: At this point, the system is released for production use.
Inception Phase

Entry Criteria
- Need for project: original vision, legacy system, mock-up

Activities
- Formulate scope of project
- Develop risk mitigation strategy
- Develop initial project plan with cost/schedule tradeoffs
- Prepare project environment

Exit Criteria
- Product vision, containing core requirements
- Success criteria
- Initial risk assessment
- Estimate of required resources for elaboration phase
Elaboration Phase

Entry Criteria

- Exit criteria artifacts from last iteration
- Plan approved by project management and funding authority
- Required resources allocated

Activities

- Define the core architecture
- Validate the architecture
- Baseline the architecture, addressing high risk items
- Refine requirements

Exit Criteria

- Construction Phase plan, with number and contents of each iteration
- Development environment and tools
- Domain analysis model
- Executable architecture baseline
Construction Phase

Entry Criteria
- Exit criteria artifacts and products from last iteration
- Risks being mitigated during this iteration
- Defects being fixed during this iteration

Activities
- Develop and test software components to satisfy use cases and nonfunctional requirements
- Manage resources and control process
- Assess the iteration

Exit Criteria
- Updated versions of artifacts
- Iterative implementation of remaining lower risk elements
- Test cases and test results for iteration’s products
- Iteration plan for next iteration
- Measurable evaluation criteria for assessing next iteration’s results
Transition Phase

Entry Criteria
- Exit criteria artifacts from last iteration
- A software product mature enough to beta test

Activities
- Test software product in customer environment
- Fine tune product based on customer feedback
- Deliver final product to customer
- Finalize end-user support documents

Exit Criteria
- Update of previous documents as necessary
- “Post-mortem” analysis of project relative to original and revised success criteria
UP Disciplines

Discipline: set of activities and artifacts in one subject area

- **Requirements**: requirements analysis, including writing use cases and identifying non-functional requirements
- **Design**: software architecture, responsibility assignment to objects, database design, etc.
- **Implementation**: programming and building the system
- **Test**: unit tests for individual classes, integration tests for putting classes together, and acceptance tests by customers
- **Environment**: setting up tool and process environment for software development
Disciplines across Iterations

Sample UP Disciplines

- Business Modeling
- Requirements
- Design
- Implementation
- Test
- Deployment
- Configuration & Change Management
- Project Management
- Environment

A four-week iteration (for example). A mini-project that includes work in most disciplines, ending in a stable executable.

Note that although an iteration includes work in most disciplines, the relative effort and emphasis change over time. This example is suggestive, not literal.
UP Variants

Agile Unified Process (AUP)
- Lightweight variant
- Succeeded by Disciplined Agile Delivery (DAD)

Open Unified Process (OpenUP)
- Used to develop Eclipse Framework

Oracle Unified Method (OUM)
- Oracle development process

Rational Unified Process (RUP)
- IBM/Rational software development process
Object-Oriented Analysis & Design

Analysis

- Investigation of the problem and requirements, with an emphasis on finding and describing the objects (or concepts) in the problem domain
  - ex: a library information system would include domain objects like *Book*, *Library*, or *Patron*

Design

- Creating a conceptual solution that fulfills the requirements by defining software objects and how they collaborate to fulfill the requirements
  - ex: the *Book* object may have a *title* attribute and *getChapter* method
Object-Oriented Analysis & Design

Requirements Analysis

- Before OOA/D, write use cases to capture requirements

Assigning Responsibilities

- The most important skill in OOA/D is assigning responsibilities to objects

Design Patterns

- Named problem-solution formulas that codify best practices for object-oriented design

UML

- Standard diagramming notation used to visually represent objects and their interactions
OOA/D Example: Dice Game

Dice Game Example

- Player rolls two dice
- If total is 7, player wins
- Otherwise, player loses
Use Cases

Use Case: Play a Dice Game

- A player picks up and rolls two dice.
- If the dice face value total is 7, player wins.
- Otherwise, player loses.
Domain Model

OO Analysis

- Create description of problem domain in terms of classifications of objects
- Identify important concepts, attributes, and associations

```
Player
  | name
  | 1

Die
  | faceValue
  | 2

Rolls
  | 2

Plays
  | 1

DiceGame
  | Includes
  | 1
```
Interaction Diagram

**OO Design**

- Defining software objects and their collaborations
- Note that while in the real world a *Player* rolls the dice, in the design, the *DiceGame* object rolls the dice by sending messages to *Die* objects
- Object design is **not** generally a direct model of the real world
Design Class Diagram

Interaction diagram provides a *dynamic* view of collaborating objects

Design class diagram provides a *static* view

- Attributes
- Methods
- Containment
Architectural Layers

Interface

application logic and domain object layer

technical services layer

- minor focus
  - explore how to connect to other layers

- primary focus of case study
  - explore how to design objects

- secondary focus
  - explore how to design objects

- Item ID
- Quantity
- Enter Item
- And so on...
- Sale
- Payment
- Log
- PersistenceFacade
Iterative Learning & Development

- Book organized in three iterations.
- Each iteration delivers a product to the customer.
  - First iteration has core features.
  - Second and third elaborate.
- Each iteration of book introduces analysis and design skills relevant to current iteration of software product.
Problem: Do the stakeholders have basic agreement on the vision of the project, and is it worth investing in serious investigation?

Envision product scope, vision, and business case.
Inception Phase

Brief: few weeks at most
High level: order of magnitude estimates

Artifacts

- Vision
- Use Case Model
- Supplementary Specification
- Glossary
- Risk Plan
- Prototypes
- Iteration Plan
Vision

Describes the high level goals and constraints from stakeholder’s point of view, focusing on key stakeholder needs.

- Describe the problem to be solved.
- Identify stakeholders and what they need.
- Describe what the system does.
- Describe major features.
- Note major non-functional requirements.
- State any design constraints.
Use Case Model

Describes the functional requirements

- Functional requirements provide features to user

Inception Phase To Do

- Preliminary Use Case Model consisting of:
  - Names of use cases
  - Names of actors
  - ~10% of named use cases are detailed
Supplementary Specification

Describes non-functional requirements that do not appear in the Use Case model

- Non-functional requirements don’t provide features
- Examples: performance, reliability, security

Inception Phase To Do

- High level supplementary specification
Glossary

Defines key domain terms

- Aliases
- Descriptions or definitions
- Relationships to other terms
- Units, data type, or other format information
- Range of values
- Validation rules

Inception Phase To Do

- Define key terms, like UPC for POS example
Risk Plan

List of known and expected risks

- Business
- Technical
- Resource
- Schedule

Ideas for mitigation or response to each risk
Prototypes

Develop prototypes to clarify vision

- User-interface prototypes
- Technical tests to assess risks

Inception Phase:

- TBD
Iteration Plan

Plan activities for first iteration of Elaboration Phase.
UP Requirement Artifacts

Functional
- Vision
- Use Cases

Non-functional
- Supplementary Specification
- Glossary
1. Software development methodologies: waterfall vs agile.
2. The RUP is an agile, iterative process for developing software products.
3. Write use cases to capture requirements.
4. The most important skill in OOA/D is assigning responsibilities to objects.
5. Design patterns are named problem-solution formulas that codify best practices for object-oriented design.
6. UML is the standard visual notation for object modeling.
7. Inception Phase activities for your project.
Acknowledgements

- Some slides in this presentation are adapted from slides created by Professor Ron Mak at SJSU.