

## Software Process and Requirements Analysis

## Announcements

- HW9: Do commit the map if you have cropped it. Your repo will be fine

Fall 15 CSCI 2600, A Milanova

2

## Announcements

- CHECK YOUR GRADES!
  - HW1-6, Exam1-2, Quiz1-10 in the LMS
  - HW7 feedback and grade in Homework Server
- HW9 due on Friday, Dec 11<sup>th</sup>
- Final exam: Mon, Dec 21, 3-6pm in DCC 318
- Final exam review on Friday
  - I'll post back tests and review slides by Friday
  - No regular office hours during the Week of Dec 14<sup>th</sup>. I'll post revised office hours schedule

Fall 15 CSCI 2600, A Milanova

3

## Today's Lecture Outline

- Some catch up, Usability
- Software process, overview
- Requirements analysis
- Software processes

Fall 15 CSCI 2600, A Milanova

4

## Usability

- Design principles for **learnability**
  - Consistency: internal, external, metaphorical
  - Use simple words, not tech jargon
  - Recognition, not recall
- Design principles for **visibility**
  - Make system state visible
  - Give prompt feedback
- **Simplicity!**

Fall 15 CSCI 2600, A Milanova

5

## Usability

- Design principles for **efficiency**
  - Human motor processor, Fitts's law and Steering law:
  - Make important targets big and nearby
  - Avoid steering tasks
  - Provide shortcuts
- Design principles for **safety (error handling)**
  - Avoid mode errors
  - Use confirmation windows sparingly

## Question: Which Menubar is Easier to Access and Why?

- Mac: Menubar at the very top of the screen



- Windows: Menubar separated from the top of the screen by a window title bar

Fall 15 CSCI 2600, A Milanova

7

## Document Your System

- Write the user manual
  - Program and UI metaphors
  - Key functionality
  - Do not include: exhaustive list of all menus
- What is hard to do?
- Who is your target audience?
  - Power users need a manual
  - Casual users might not
- Piecemeal online help is no substitute

Fall 15 CSCI 2600, A Milanova. Slide from Michael Ernst

8

## Outline

- Usability
- Iterative Design
  - Design
    - Design principles
  - Implement
    - Low-fidelity prototypes
  - Evaluate
    - User testing

Fall 15 CSCI 2600, A Milanova

9

## Low-fidelity Prototype

- Paper is a very fast and effective prototyping tool
  - Sketch windows, menus, dialogs, widgets
  - Crank out lots of designs and evaluate them
- Hand-sketching is OK --- even preferable
  - Focus on behavior & interactions, not fonts & colors
  - Similar to design of your ADTs and classes
- Paper prototypes can even be executed!
  - Use pieces to represent windows, dialogs, menus
  - Simulate computer's responses by moving pieces around and writing on them

Fall 15 CSCI 2600, A Milanova. Slide due to Michael Ernst

10

## User Testing

- Start with a prototype
- Write up a few representative tasks
  - Short but non-trivial
    - E.g., "add this meeting to calendar",
    - E.g., "type this letter and print it"
- Find a few representative users
  - 3 is often enough to find obvious problems
- Watch them do tasks with the prototype

Fall 15 CSCI 2600, A Milanova. Slide due to Michael Ernst

11

## How to Watch Users

- Brief the user first
  - "I'm testing the system, not testing you"
  - "If you have trouble, it's the system's fault"
  - "Feel free to quit at any time"
  - Ethical issues: informed consent
- Ask user to think aloud
- Be quiet!
  - Don't help, don't explain, don't point out mistakes
  - Two exceptions: prod user to think aloud, and move on to the next task when stuck
- Take lots of notes

12

## Watch for Critical Incidents

- Critical incidents: events that strongly affect task performance or satisfaction
- Usually negative
  - Errors
  - Repeated attempts
  - Curses
- Can also be positive
  - "Cool!"
  - "Oh, now I see."

Fall 15 CSCI 2600, A Milanova. Slide due to Michael Ernst

13

## More Formal User Testing

- Empirical methods collect evidence about how users interact with UI
  - These methods **quantify** UI designs
- E.g., lab experiments, field studies, surveys
- Controlled experiments quantify UI usability
  - Hypothesis: e.g., Mac bar faster than Windows bar
  - Independent variable: e.g., y-coordinate of menu bar
  - Dependent variable: e.g., #errors, #tasks done
  - Statistical methods
- **Ethics!**

14

## Usability, Summary

- You are not the user
- Keep human capabilities and UI design principles in mind
- Iterate over your design
- Write documentation
- Make cheap, throw-away prototypes
- Evaluate prototypes with users

Fall 15 CSCI 2600, A Milanova. Slide due to Michael Ernst

15

## Software Process

- **Software lifecycle** activities:
  - Requirements analysis
  - Design
  - Implementation
  - Integration + Testing and verification
  - Deployment and maintenance
    - Maintenance is costly. The later a problem is found, the costlier it is to fix
- **Software process** puts these together
  - How do we combine these activities?
  - In what order?

16

## Software Lifecycle

- Activities from inception to end-of-life
- Can take months or years
- Each activity has specific goals
  - Defines a clear set of steps
  - Produces an artifact (i.e., tangible item)
  - Allows for review
  - Specifies actions to perform in next activity

Fall 15 CSCI 2600, A Milanova (based on slides by Michael Ernst)

17

## Activities and Their Artifacts

- Requirements analysis produces "requirements documents"
  - Use-case model, supplementary specifications
- Design produces "design models"
  - Class diagrams, interaction diagrams, ADT specs, other
- Implementation produces, well, ... obviously code
  - + specs for classes and individual methods, AFs and RIs
  - **Readability** of code is crucial!
- Testing produces
  - Test suites

Fall 15 CSCI 2600, A Milanova

18

## Overview Example, Step 1: Requirements, Use-Case Model

### ■ A Dice Game

### ■ Use case: Play Dice Game

Main success scenario:

Player picks up the two dice and rolls them.

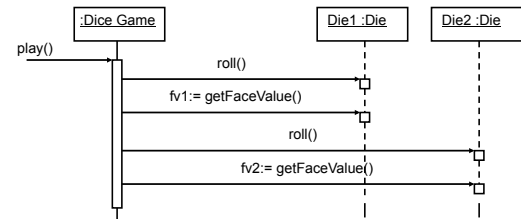
If face value is 7 then they win...

Else ...

19

## Example, Step 2: Design: Interaction Diagram

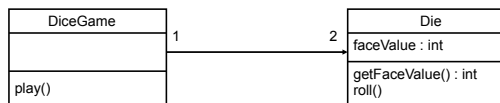
### ■ Dynamic object design



20

## Example, Step 2: Design: Class Diagram

### ■ Static design



21

## Example, Step 3: Code + specs

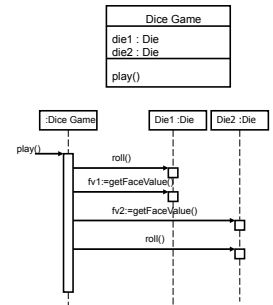
```

/* a mutable class ... */
class DiceGame {
    private Die die1 = new Die();
    private Die die2 = new Die();
}
    
```

// Abstraction Function: ...  
// Rep invariant: ...

```

// @modifies:
// @effects:
public void play() {
    die1.roll();
    int fv1 = die1.getFaceValue();
    ...
}
    
```



## Analogy with Process in Other Engineering Disciplines

### ■ Civil engineering

- Requirements: architectural design
- Design: engineering blueprints
- Implementation: construction
- Testing: inspections throughout construction and after completion

Fall 15 CSCI 2600, A. Milanova

23

## Main Differences with other Engineering Disciplines

- Requirements changes do not creep into construction
  - In contrast, in software, requirements change constantly even when implementation is well underway
- Majority of engineering design and construction uses tried and true materials and techniques
  - Software constantly innovates --- new languages, new frameworks, new uses
- Construction projects are mostly on time
  - But it's a myth that construction projects are never late. E.g., The Big Dig

Fall 15 CSCI 2600, A. Milanova

24

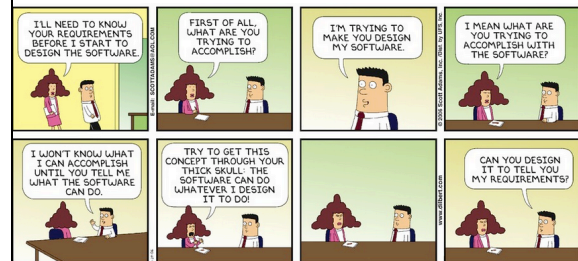
## Today's Lecture Outline

- Some catch up
- Software process, overview
- Requirements analysis
- Software processes

Fall 15 CSCI 2600, A Milanova

25

## Requirements Analysis...



Fall 15 CSCI 2600, A Milanova

26

## Requirements Analysis is Hard

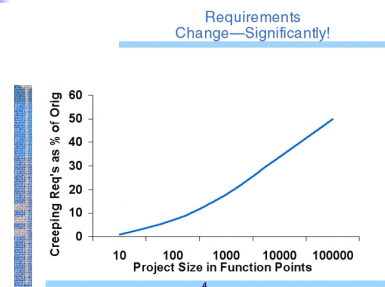
- Major causes of project failure
  - Poor user input
  - Incomplete requirements
  - Changing requirements
- Essential tools
  - Classification of requirements
  - Use cases

Fall 15 CSCI 2600, A Milanova

27

## Requirements Change

Function points is a measure of software complexity. Roughly, it measures the number of interactions of the system with the user.



Fall 15 CSCI 2600, A Milanova (Source: Craig Larman, 2003)

28

## Aside: Measures of Complexity

- Function points
  - Roughly, the number of interactions of user with the system
- Cyclomatic complexity
  - Roughly, the number of predicate nodes in CFG
- Lines of code

Fall 15 CSCI 2600, A Milanova

29

## Classification of Requirements

- FURPS+ model
- The FURPS:
  - Functionality, Usability, Reliability, Performance, Supportability
- The +:
  - Design constraints, implementation requirements (e.g., must use Java), other

Fall 15 CSCI 2600, A Milanova

30

## Requirements Analysis Artifacts

- Requirements analysis produces:
  - **Use-case model**
    - A set of use cases
    - Specifies the **functional requirements** (behavior, features) of the system
  - **Supplementary specification**
    - Specifies non-functional requirements (-ilities: **usability**, **reliability**, **performance**, **supportability**)

Fall 15 CSCI 2600, A. Milanova

31

## Use Cases

- Describe the interaction of the user with the system as TEXT stories
- The most widely used approach to requirements analysis in modern software practice
  - Requirements are discovered and recorded through use cases
  - All other activities influenced by use cases!

Fall 15 CSCI 2600, A. Milanova

32

## Example Use Case

- Point-of-sale (POS) system
- **Process Sale**: A **customer** arrives at checkout with items to buy. The **cashier** uses the **POS system** to record each purchased item. The **system** presents a running total and line-item details. The **customer** enters payment information, which the **system** validates and records. The **system** updates inventory. The **customer** receives a receipt.
- The use case is a collection of scenarios: **main success scenario + scenario variations**

Fall 15 CSCI 2600, A. Milanova

33

## Another Example Use Case Scenario

- RPI Campus Paths
- **Shortest Path**: Campus Paths **system** displays the campus map. The **user** selects two buildings. The system draws the shortest path on the map.

Fall 15 CSCI 2600, A. Milanova

34

## Brief, Casual and Fully Dressed formats

- **Brief**: one-paragraph summary, usually for the main success scenario
- **Casual**: multiple paragraphs that cover various scenarios
- **Fully dressed**: all steps and variations are written down
  - Developed iteratively

Fall 15 CSCI 2600, A. Milanova

35

## Example Use Case

### Handle Returns use case

Main success scenario: A **customer** arrives with items to return. The **cashier** uses the **POS system** to ...

### Alternative Scenarios

- If they paid by credit card, but reimbursement transactions to their credit card are rejected, pay by cash
- If the system detects a failure in the external accounting system, ...
- ...

Fall 15 CSCI 2600, A. Milanova

36

## Use Cases vs. Feature Lists

- Low-level feature lists --- common in the past:

ID	Feature
FEAT1.9	System should accept entry of item identifiers.
...	...
FEAT2.4	System should log credit payments to the accounts receivable system.

- Are use cases the better way to discover and record requirements?

Fall 15 CSCI 2600, A Milanova

37

## Today's Lecture Outline

- Some catch up
- Software process, overview
- Requirements analysis
- Software processes**

Fall 15 CSCI 2600, A Milanova

38

## Software Process

- Software lifecycle** activities:
  - Requirements analysis
  - Design
  - Implementation
  - Testing
  - Deployment and maintenance
- Software process** puts these activities together
- Software process** forces attention to these activities and their artifacts

39

## Some Software Processes

- Code-and-fix** (ad-hoc): write some code, make up some inputs, debug
- Waterfall**: 1<sup>st</sup>: requirements analysis, 2<sup>nd</sup>: design, 3<sup>rd</sup>: implementation, 4<sup>th</sup>: testing
- Iterative** (Unified process, Agile, Scrum) repeat activities: (a small chunk of requirements, design, implementation, testing)<sup>+</sup>
- Other**

Fall 15 CSCI 2600, A Milanova

40

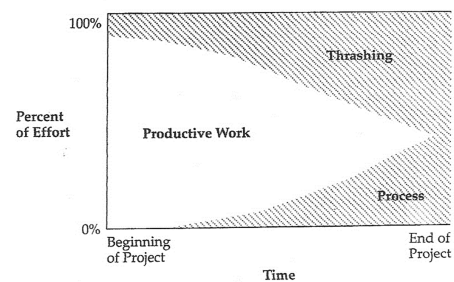
## Benefits of Software Process

- A framework to work within
- A management tool
- Forces attention to important activities and their artifacts
  - Won't forget requirements analysis or requirements documents, or design, or testing, etc.
- Drawbacks?**

Fall 15 CSCI 2600, A Milanova

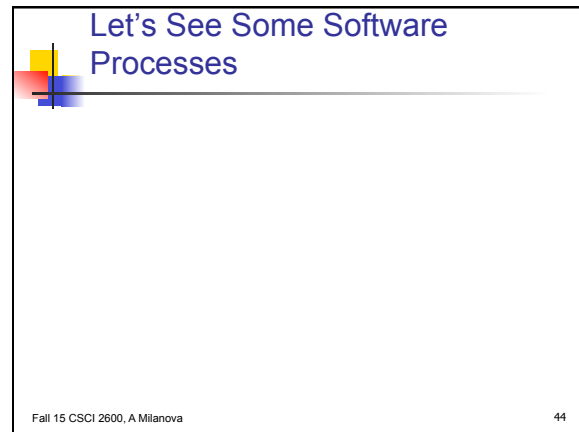
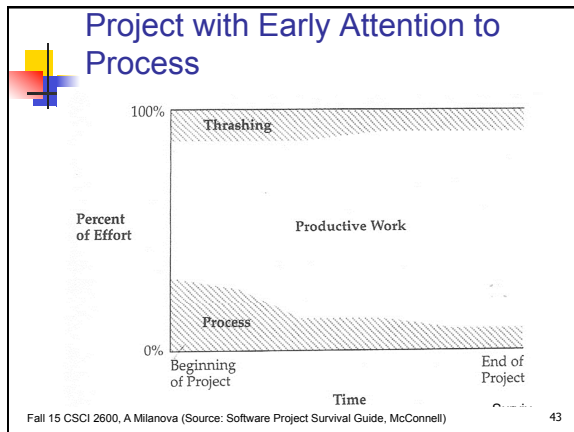
41

## Project with Little Early Attention to Software Process

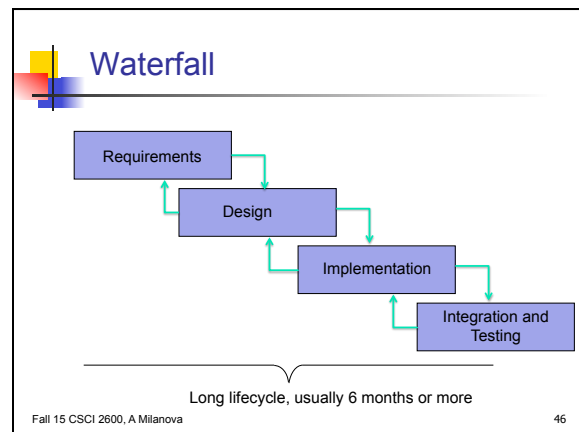


Fall 15 CSCI 2600, A Milanova (Source: Software Project Survival Guide, McConnell)

42



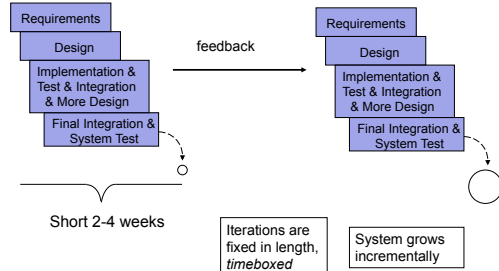
- ### Code-and-fix
- Advantages
    - Little or no overhead. Just dive in and see progress quickly
    - Applicable for very small, very short-lived projects
  - Dangerous for most projects
    - May ignore important tasks and artifacts (design, testing)
    - Not clear when to start or stop an activity
    - Hard to review
    - Scales poorly to multiple people
- Fall 15 CSCI 2600, A Milanova (based on a slide by Michael Ernst)



- ### Advantages of Waterfall
- Can work well for projects with very well understood requirements
    - Tackles all planning upfront
  - Orderly, easy to follow, **sequential** process
  - Stages are well-defined, easy to perform reviews at each stage to determine if the product is ready to advance
  - Ideal for experienced teams
- Fall 15 CSCI 2600, A Milanova

- ### Waterfall Limitations
- Requirements change
    - Waterfall requires a lot of planning upfront
    - Waterfall **assumes** requirements are clear and well-understood
  - Rigid, sequential; does not embrace change
    - Costly to "swim upstream" back to an earlier phase
  - No sense of progress until the very end
  - Integration occurs at the very end
    - Defies "integrate early and often" rule
    - Inflexible, no feedback until the very end
    - Product may not match customer's need
  - Reviews are massive affairs (inertia)
- Fall 15 CSCI 2600, A Milanova (based on slide by Michael Ernst)

## Iterative Processes Work in Short Iterations



Fall 15 CSCI 2600, A Milanova

49

## Advantages of Iterative Process

- Accommodate, embrace change
- Provides constant feedback, problems are visible early
- Appropriate at the beginning of the project when requirements are still fluid
- Always address the biggest risk first
  - As costs increase, risks decrease!

Fall 15 CSCI 2600, A Milanova

50

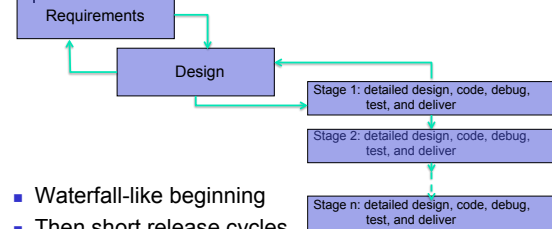
## Disadvantages of Iterative Process

- A lot of planning and management
- Frequent change of task
- Requires customer and contract flexibility
- Developers must be able to assess risk
  - Must address most important issues

Fall 15 CSCI 2600, A Milanova

51

## Staged Delivery



- Waterfall-like beginning
- Then short release cycles
  - Plan, design, implement, test, deliver
- Delivery possible at the end of any cycle

Fall 15 CSCI 2600, A Milanova (based on a slide by Michael Ernst)

52

## Staged Delivery Advantages

- Can ship at the end of any release
  - Looks like success to customers, even if not original goal
- Intermediate deliveries show progress, customers are happy, lead to feedback
- Problems are visible early
- Facilitates shorter more predictable release cycles
- **Practical, widely used and successful**

Fall 15 CSCI 2600, A Milanova (based on slide by Michael Ernst)

53

## Next time

- Review
- Final on December 21<sup>st</sup>

Fall 15 CSCI 2600, A Milanova

54