Assignment 8

• Due Friday, November 15, 2013
Software Process
Phases of the Software Process

• Requirements
• Design
• Implementation
• Testing
• Maintenance
Software Process Models
Waterfall Model

Phases (activities)

Requirements → Design → Implementation → Testing → Maintenance

Time

Figure from [Braude & Bernstein]
Waterfall with Feedback

- Figure 3.4 on page 37

Figure from [Braude & Bernstein]

Phases (activities)
# Waterfall

**Pros**

- Simple & easy to use
- Practiced for many years
- Easy to manage
- Facilitates allocation of resources
- Works well for smaller projects where requirements are very well understood

**Cons**

- Requirements must be known up front
- Hard to estimate reliably
- No stakeholder feedback until after testing
- Major problems not discovered until late in process
- Lack of parallelism
- Inefficient use of resources
Iterative & Incremental
Iterative

Figure from [Braude & Bernstein]
Figure from [Braude & Bernstein]
Incremental

Figure from [Pressman]
Requirements
Why should we care about requirements?
“There’s no sense being exact about something if you don’t even know what you’re talking about.”

- John von Neumann
“The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is so difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.”

- Fred Brooks
Cost of Defects

- Requirements: 1
- Design: 5
- Implementation: 20
- Test: 50
- Maintenance: 100

Northeastern University
College of Computer and Information Science
Reasons for Project Failure

- Incomplete requirements 13.1%
- Lack of user involvement 12.4%
- Lack of resources 10.6%
- Unrealistic expectations 9.9%
- Lack of executive support 9.3%
- Changing requirements/specification 8.7%
- Lack of planning 8.1%
- Didn’t need it any longer 7.5%

Project Success Factors

- User involvement: 15.9%
- Management support: 13.9%
- Clear statement of requirements: 13.0%
- Proper planning: 9.6%
- Realistic expectations: 8.2%
- Smaller milestones: 7.7%
- Competent staff: 7.2%
- Ownership: 5.3%

Sources of requirements

- Stakeholders
- Documents
- Books
Types of Requirements

• Functional
  - expresses a function that an application must perform

• Nonfunctional
  - does not involve specific functionality
  - qualifies services or functionalities
  - should be specific, quantifiable, and testable
Examples

• Functional
  - The application allows clerks to check out DVDs.

• Nonfunctional
  - The application shall display a customer’s account status in less than two seconds.
Analysis & Specification

• Analysis: process of understanding the problem and the requirements for a solution

• Specification: process of describing what a system will do

• Analysis leads to Specification – they are not the same
Output

• Software Requirements Specification (SRS)

• All stakeholders can agree on set of requirements
User of a Requirements Document

[Kotonya & Sommerville]

- **System customers**
  - Specify the requirements & read them to check that they meet their needs.
  - Specify changes to the requirements.

- **Managers**
  - Use the requirements document to plan a bid for the system & to plan the system development process

- **System engineers**
  - Use the requirements to understand what system is to be developed.

- **System test engineers**
  - Use the requirements to develop validation tests for the system.

- **System maintenance engineers**
  - Use the requirements to understand the system & the relationships between its parts
Requirements Document

• The official statement of what is required of the system developers should include both a definition and a specification of requirements

• It is NOT a design document

• As far as possible, it should set of WHAT the system should do rather than HOW it should do it
Format of Requirements

[Hull, Jackson, and Dick]
Functional Requirements

• The <stakeholder type> shall be able to <capability/function>.

• The system shall allow the <stakeholder type> to <capability/function>.

• The system shall <capability/function>.
Constraints
[Hull, Jackson, and Dick]

- **Performance/capability**: The <system> shall be able to <function> <object> not less than <performance> times per <units>.
- **Performance/capability**: The <system> shall be able to <function> <object> of type <qualification> within <performance> <units>.
- **Performance/capacity**: The <system> shall be able to <function> not less than <quantity> <object>.
- **Performance/timeliness**: The <system> shall be able to <function> <object> within <performance> <units> from <event>.
- **Performance/periodicity**: The <system> shall be able to <function> not less than <quantity> <object> within <performance> <units>.
- **Interoperability/capacity**: The <system> shall be able to <function> <object> composed of not less than <performance> <units> with <external entity>.
- **Sustainability/periodicity**: The <system> shall be able to <function> <object> for <performance> <units> every <performance> <units>.
- **Environmental/operability**: The <system> shall be able to <function> <object> while <operational condition>. 

Quality of Requirements

[Braude & Bernstein]

- Comprehensiveness
  - How comprehensive is the SRS?
  - Does the SRS include all of the customer’s wants and needs?
- Consistency
  - How consistent is the SRS?
  - Are there contradictions in the SRS?
- Self-Completeness
  - How self-complete is the SRS?
  - Does the SRS contain all of the necessary parts?
- Understandability
  - How understandable is each requirement?
- Unambiguity
  - How unambiguous is each requirement?
  - Can the requirement be interpreted in only one way?
- Prioritization
  - How effectively prioritized are the requirements?
  - Is there an order for implementation?
- Testability
  - How testable is each requirement?
  - Is it possible to validate the requirement in the finished product?
- Traceability
  - How traceable is each requirement?
  - Is the requirement traceable forward? Backward?
Traceability

- Track each requirement to design and code that implements it
- Track each part of code back to corresponding elements of design and requirements
- Critical in requirements engineering to link the requirements for the system and other important entities of the software
Traceability

- requirement
- design element

- test
- code

- inspection

Figure from [Braude & Bernstein]
Benefits of Traceability

[Hull, Jackson, and Dick]

• Greater confidence in meeting objectives.
• Ability to assess the impact of change.
• Improved accountability of subordinate organizations.
• Ability to track progress.
• Ability to balance cost against benefit.
Stories
User Stories
[Cohn]

• Describe software functionality that will be important for the users

• Allow software engineers to gain an understanding of what the software may include
Aspects in User Stories  [Cohn]

• A written description of the story used for planning and as a reminder

• Conversations about the story that serve to flesh out the details of the story

• Tests that convey and document details and that can be used to determine when a story is complete
Use Cases

- Use case: sequence of actions taken by the actor and the application
- Actor (person or system)
- Scenario: list of numbered steps
  - Main success scenario
  - Extensions or error conditions
Use Case: Retrieve a File

1. User clicks *File* menu
2. System displays options *New* and *Open*
3. User clicks *Open*
4. System displays file window
5. User enters directory and file name
6. User hits *Open* button
7. System retrieves referenced file into word processor window Extensions
Use Case: Retrieve a File

Main Success Scenario
1. User clicks File menu
2. System displays options New and Open
3. User clicks Open
4. System displays file window
5. User enters directory and file name
6. User hits Open button
7. System retrieves referenced file into word processor window

Extensions
7a System displays error indicating file could not be opened
Use Case Format

• Name: describes the task
• Associated requirement
• Actor: who is the user
• Priority
• Main success scenario
• Extensions
Use Case Exercise
Rent a Movie

1. User presses “Rent a Movie”
2. System displays movies that are for rent
3. User selects a movie
4. System displays selected movie information
5. User presses “Rent”
6. System prompts for credit card payment
7. User swipes credit card
8. System ejects movie
Rent a Movie

1. User presses “Rent a Movie”
   a. The user shall be able to press “Rent a Movie” from the Main screen when she wants to rent a movie.

2. System displays movies that are for rent
   a. The system shall display the list of possible movies to rent when the user presses “Rent a Movie” from the Main screen.

3. User selects a movie
   a. From the Movie Selection screen, the user shall be able to select a movie to rent.
   b. The system shall only display available movies on the Movie Selection screen.

4. System displays selected movie information
   a. The system shall display movie information on the selected movie when the user selects the movie from the Movie Selection screen.
Rent a Movie

5. User presses “Rent”
   a. The user shall be able to press “Rent” from the Movie screen when she wants to rent the selected movie.

6. System prompts for credit card payment
   a. The system shall prompt the user for credit card payment via the Payment screen when the user indicates she wants to rent the selected movie.

7. User swipes credit card
   a. The user shall be able to swipe her credit card to pay for the selected movie.

8. System ejects movie
   a. The system shall process the credit card payment when the user has selected a movie and swiped her credit card.
   b. The system shall eject the movie once the payment has been processed.
Stories for Online Bookstore

• Books can be searched by author, title, or ISBN.

• Books can be viewed detailed information by number of pages, publication date, and a brief description.

• Book can be placed in “shopping cart” and bought.

• Books can be rated and reviewed.
Requirements Engineering is difficult

Despite the clear benefits of getting requirements complete, right, and error-free early, they are the hardest part of the system development lifecycle to do right because:

- we don’t always understand everything about the real world that we need to know,
- we may understand a lot, but we cannot express everything that we know
- we may think we understand a lot, but our understanding may be wrong,
- requirements change as client’s needs change,
- requirements change as clients and users think of new things they want, and
- requirements of a system change as a direct result of deploying the system.
YOUR REQUIREMENTS DOCUMENT IS THE BIGGEST I'VE EVER SEEN.

IT'S TOO BIG TO READ, BUT I CAN GUESS FROM ITS WEIGHT WHAT MUST BE IN THERE.

YOU KNOW IT'S A MULTI-USER, GLOBAL SYSTEM, RIGHT?

NO, I'M NOT GETTING THAT.
I'll need to know your requirements before I start to design the software.

First of all, what are you trying to accomplish?

I'm trying to make you design my software.

I mean what are you trying to accomplish with the software?

I won't know what I can accomplish until you tell me what the software can do.

Try to get this concept through your thick skull: the software can do whatever I design it to do!

Can you design it to tell you my requirements?
Other Approaches to Balanced Trees

• AVL Trees
• 2-3 Trees
AVL Trees
[Levitin]

An AVL tree is a binary search tree in which the balance factor of every node, which is defined as the difference between the heights of the node’s left and right subtrees, is either 0 or +1 or -1. (The height of the empty tree is defined as -1.)
AVL Tree

Not AVL Tree

Figures from Levitin
**Figure 6.3** Four rotation types for AVL trees with three nodes. (a) Single $R$-rotation. (b) Single $L$-rotation. (c) Double $LR$-rotation. (d) Double $RL$-rotation.
2-3 Trees
[Levitin]

- A 2-3 tree is a tree that can have nodes of two kinds: 2-nodes and 3-nodes.
- A 2-node contains a single key $K$ and has two children.
- A 3-node contains two ordered keys $K_1$ and $K_2$ ($K_1 < K_2$) and has three children.
- A 2-3 tree is always height-balanced
Figure from Levitin
Inserting into 2-3 Trees
[Levitin]

• Always insert a new key K at a leaf except for the empty tree.
• The appropriate leaf is found by performing a search of K.
• If the leaf in question is a 2-node, we insert K there as with the first or the second key, depending on whether K is smaller or larger than the node’s old key.
• If the leaf is a 3-node, we split the leaf in two: the smallest of the three keys (two old ones and the new key) is put in the first leaf, the largest is put in the second leaf, while the middle key is promoted to the old leaf’s parent. (If the leaf happens to be the tree’s root, a new root is created to accept the middle key.)
• Note that promotion of a middle key to its parent can cause the parent’s overflow (if it was a 3-node) and hence can lead to several node splits along the chain of the leaf’s ancestors.