Collecting Requirements and Writing Your Design Document

CS 470

Project Requirements/Design Document

• Document should contain
  – Overview / Hypothesis
  – Planning / Lifecycle Methodology
  – Requirements
  – Design
• Just as you should not immediately jump into writing code, you should not immediately jump into writing your design document
  – Planning and methodology described earlier should be elaborated
  – Next we generally collect requirements
Capturing the requirements

• Requirement: a feature of the system or a description of something the system is capable of doing in order to fulfill the system’s purpose
• Three kinds of requirements:
  – those that absolutely must be met
  – those that are highly desirable but not necessary
  – those that are possible but could be eliminated

Why are Requirements Important?

• 1994 Standish Group survey of 350 companies about 8000 software projects
• 31% canceled before completion
• % of projects on time and within budget
  – Large companies: 9%
  – Small companies: 16%
• Top factors for failed projects:
  – Incomplete requirements (13%), lack of user involvement (12%), lack of resources (11%), unrealistic expectations (10%), lack of executive support (9%), changing requirements and specs (9%), lack of planning (8%), system no longer needed (7%)
### Requirements documents

- These should be in your writeup
  - Requirements definition: complete listing of what the customer expects the system to do
    - English, Mock-Ups
  - Requirements specification: restates the definition in technical terms so that the designer can start on the design
    - English, UML, ER Diagram, Other diagrams
- Not explicitly required in writeup but useful for large projects
  - Configuration management: how to deal with change (e.g. version control, track project revisions)

### Types of Requirements

- **Physical Environment**
  - Where equipment will function
  - Any environmental restrictions
- **Interfaces**
  - Where is input/output going from/to?
  - Protocol definitions for passing any messages?
  - Format for data?
  - Medium for data?
- **Users and Human Factors**
  - Who will be the user?
  - Skill level, training required?
  - How easy to use the system?
Types of Requirements

• Functionality
  – What will the system do? When?
  – Ways to change or enhance the system?
  – Constraints on execution, response?
• Data
  – Format of data?
  – Precision?
  – Data flow?
  – Retention?
• Resources
  – Materials, personnel, other resources required?
  – Developer skills?
  – Cost?

Types of Requirements

• Security
  – Must access be controlled?
  – How will user data be isolated?
  – Backup?
• Quality Assurance
  – Reliability, availability, maintainability?
  – Maximum restart time after failure?
  – Efficiency measures?
Characteristics of requirements

- Are they correct?
- Are they consistent?
- Are they complete?
- Are they realistic?
- Does each describe something the customer needs?
- Are they verifiable?

User Centered Requirements

- User-Centered Design emphasizes the gathering of requirements from the user
- Would like to capture:
  - Domain Knowledge
    - What previous knowledge is required to complete the task? E.g. what faculty do for a faculty workload system
    - What knowledge is required to effectively use the system? E.g. knowledge of acronyms PPP, SMTP, POP, or processes
  - Levels of Computing Experience
    - How tech savvy is the user population? Will impact interface and functionality.
    - Capturing user experience can be helpful in adapting metaphors; e.g. shopping cart or file folders on a web page
    - Adapt to user’s past experiences
      - Can also give pointers to what problems have persisted for the target user population in the past
User Centered Requirements

- User Computing Environment
  - What environment is the target user on? All Windows, all Unix, mixture?
  - We’ll see the environment can affect usability
- Content
  - Type of content users are interested in and the organization of the content
  - Difficult to gather; next we’ll see some methods
- Benchmarking
  - Examine similar systems to assess features, usability

Methods for Gathering Requirements

- Once it is determined what requirements should be collected, the next step is to actually collect them
- Many methods for gathering requirements
  - Interviews
  - Surveys
  - Focus Groups
  - Indirect
- Use multiple methods if possible
  - One method may be biased; e.g. chatty user dominates interview, only tech-savvy complete online survey, etc
  - In our short time frame, you’ll probably just use interviews with the client
Gathering User Requirements

• Bottom Line: Involve users in some way to collect the requirements for the system.

• Don’t just come up with requirements yourself for what you think will solve the user’s problems!

Expressing Requirements

• Informal
  – English, Mock-ups, Diagrams, User Stories
  – Fine for this project, but more formal, unambiguous requirements may be better when possible

• Formal
  – ER Diagrams
  – Object-Oriented Specs
  – Unified Modeling Language
  – Finite State Automata and Transition Diagrams
English Example

• The store must be able to accept electronic cash in two ways:
  – Ship product first, then redeem e-cash
  – Redeem e-cash first, then ship product
• Users must be able to search by keyword or by product number

Mock-Up Examples

Search by keyword
GO

Search by product #
GO

Product #
Submit
Design

• At the end of the Requirements, we should know what the proposed system is supposed to do
  – E.g., requirements for a house may be: 2 bedrooms, kitchen, indoor water, electricity
• The purpose of Design is to describe the solution
  – E.g., architectural diagram, straw bale walls, septic vs. sewer, off the grid power system, etc.

Conceptual design

• Tells the customer what the system will do
• Answers:
  – Where will the data come from?
  – What will happen to the data in the system?
  – What will the system look like to users?
  – What choices will be offered to users?
  – What is the timing of events?
  – What will the reports and screens look like?
• Characteristics of good conceptual design
  – in customer language with no technical jargon
  – describes system functions
  – independent of implementation
  – linked to requirements
Technical design

- Tells the programmers what the system will do
- Includes:
  - major hardware components and their function
  - hierarchy and function of software components
  - classes and objects
  - data structures
  - structure charts
  - data flow diagrams
  - algorithm pseudocode

Desirable Design Characteristics

- Minimal complexity
  - Avoid “clever” designs that are hard to understand
- Ease of maintenance
- Loose coupling
- Extensibility
- Reusability
- High fan-in
- Low fan-out
- Leanness
- Stratification
  - Layers
- Standard techniques
General Design Levels

• Depending on the project, some are more applicable than others
  
• Architecture: associates system components with capabilities
• Code design: specifies algorithms and data structures for each component
• Executable design: lowest level of design, including memory allocation, data formats, bit patterns

Specific Levels of Design

1. Entire software system
2. Division into subsystems or packages
   • Focus should be here for the proposal/design document
3. Division into classes within package
   • Could have details here or lower if you wish but not required
4. Division into data and routines within classes
5. Internal routine design
Subsystems/Packages

- Common subsystems:
  - User Interface, Data Storage, Business Rules, System dependencies
- Avoid chaotic dependencies
- Simple, restricted dependencies among subsystems much easier to understand

Design Heuristics

- Covered in CS 401
  - Use inheritance if it simplifies the design
  - Hide secrets; information hiding
  - Use simple forms of coupling
    - Simple data types as parameters preferred over objects; avoid semantic coupling where modules indirectly related
  - Aim for strong cohesion
    - Code within a module should be closely related to support some central purpose
  - Build hierarchies
  - Use brute force if it meets requirements and is simpler to understand
Capturing Your Design Work

• Some tips to help capture your design
  – Insert design documentation into code itself
  – Capture discussions/decisions on a wiki or blog
  – Write email summaries
  – Save flip charts
  – Create UML diagrams

Expressing designs

• Can use more detailed version of previous tools for requirements
  – UML, ER Diagram, Data Flow
• General methods
  – Modular decomposition
  – Data-oriented decomposition
  – Event-oriented decomposition
  – Object-oriented design
System Architecture – Modular Decomposition, Website Director Pro

Example - More Detailed UML Diagram
Delivery Service Example – Process Model

A Data Flow Diagram (DFD)

Delivery Service Example – Detailed ER Diagram

More detailed diagram prior to implementation
Delivery Service Example
- Normalized Tables

A Set of Normalized Database Tables

<table>
<thead>
<tr>
<th>CUSTOMER</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cust_No</td>
<td>Name</td>
<td>Address</td>
</tr>
<tr>
<td></td>
<td>***</td>
<td>***</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRODUCT</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Prod_ID</td>
<td>Descript.</td>
<td>Unit Price</td>
</tr>
<tr>
<td></td>
<td></td>
<td>***</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INVOICE</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Inv_#</td>
<td>Date</td>
<td>Total Amt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>***</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ORDER</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ord_No</td>
<td>Date</td>
<td>Qty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>***</td>
</tr>
</tbody>
</table>

Pseudocode Example

Input: Opposition schedule
For each Television company name, create Opposition company.
   For each Opposition schedule,
      Locate the Episode where Episode schedule date = Opposition
transmission date AND Episode start time = Opposition transmission time
      Create instance of Opposition program
      Create the relationships Planning and Competing
Output: List of Opposition programs
What should be in my design document?

- The document is both a requirements and design document
  - As much detail as possible to nail down what your project will be and how you will know when you’re done
  - But not a giant comprehensive document covering all the little details like what you may have produced in CS 401

- Major Sections
  - Overview / Hypothesis / Background
  - Requirements
    - English or formal, mock-ups
  - Design
    - English or more formal, architecture, decomposition
  - Planning
    - Schedule with milestones and deliverables
  - References

Proposal Guidelines

- How long?
  - Depends; probably 5-10 pages, but be succinct

- Writing style
  - Formal document, okay to use “I”
    - Instead of: “You’ll probably do something like clicking a button or pressing enter, to trigger the login screen”
    - More formal: “Click the submit button to begin the login process”
  - Number each section, e.g. 2. Requirements, 2.1 Functional specifications, 2.2 Non-Functional specifications, etc.

- Spell check and proofread!