CSE 403
Software Engineering
Spring 2020

Requirements

April 08, 2020
Recap: Software development life cycle (SDLC)

**SDLC**: produce software through
- a series of stages,
- from conception to end-of-life,
- over months or even years.

**Goals of SDLC:**
- **Manage risks** and produce **high quality software**.
  (all models have advantages and drawbacks).

**Goals of each stage:**
- **What steps?** Define a clear set of steps to perform.
- **What outcome?** Produce a tangible item.
- **How to review?** Allow for review of work and progress.
- **What’s next?** Specify actions to perform in the next stage.
Recap: Life-cycle stages

Virtually all SDLC models have the following stages:
- Requirements
- Design
- Implementation
- Testing
- Maintenance

Traditional models:
- Waterfall, Prototyping, Spiral, etc.

Agile models:
- eXtreme Programming, Scrum, etc.
Today

Requirements
● What are requirements?
● How can we gather requirements?
● A first example.
● Common challenges and mistakes.
Requirements in one picture

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. How the customer explained it</td>
<td></td>
</tr>
<tr>
<td>2. How the Project Leader understood it</td>
<td></td>
</tr>
<tr>
<td>3. How the Analyst designed it</td>
<td></td>
</tr>
<tr>
<td>4. How the Engineer wrote it</td>
<td></td>
</tr>
<tr>
<td>5. How the Business Consultant described it</td>
<td></td>
</tr>
<tr>
<td>6. How the project was documented</td>
<td></td>
</tr>
<tr>
<td>7. What operations installed</td>
<td></td>
</tr>
<tr>
<td>8. How the customer was billed</td>
<td></td>
</tr>
<tr>
<td>9. How it was supported</td>
<td></td>
</tr>
<tr>
<td>10. What the customer really needed</td>
<td></td>
</tr>
</tbody>
</table>

Diagram: 9 panels of trees with various elements attached, each labeled with a step in the process.
Software requirements

Requirements specify what to build

- describe **what, not how**
- describe the problem, not the solution
- reflect system design, not software design
“What” vs. “how” is relative

One person’s **what** is another person’s **how**:

- **Input** file processing is the **what**, parsing is the **how**.
- **Parsing** is the **what**, a **stack** is the **how**.
- **Stack** is the **what**, a **linked list** is the **how**.
- A **linked list** is the **what**, **Node** is the **how**.
Why should you care and focus on requirements?

Benefits of eliciting requirements from customers:
- The #1 reason that projects succeed is user involvement [Standish Group survey of over 8000 projects].
- Easy access to end users is one of three critical success factors in rapid-development (agile) projects [Steve McConnell].

Benefits of working with customers:
- Good relations improve development speed.
- Improves perceived development speed.
- Customers don’t always know what they want.
- Customers do know what they want...it just changes over time.
How to elicit requirements?

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?
Requirements: Goals and roles

Goals when eliciting requirements:
- **Understand** precisely what is required of the software.
- **Communicate** this understanding precisely to all involved parties.
- **Control** production to ensure that system meets specification.

Roles of requirements:
- **Customers**: what should be delivered (contractual base).
- **Managers**: scheduling and monitoring (progress indicator).
- **Designers**: a spec to design the system.
- **Coders**: a range of acceptable implementations.
- **QA / Testers**: a basis for testing, verification, and validation.
How to elicit requirements?

Do:
● Talk to the users -- to learn how they work.
● Ask questions throughout the process -- "dig" for requirements.
● Think about why users do something in your app, not just what.
● Allow (and expect) requirements to change later.

Don't:
● Be too specific or detailed.
● Describe complex business logic or rules of the system.
● Describe the exact user interface used to implement a feature.
● Try to think of everything ahead of time. (You will fail!)
● Add unnecessary features not wanted by the customers.
Example project: smart fridge

Scenario (when this crisis is over):
● Dinner/party time.
● On the way home.
● Inviting a lot of friends.
● Is the fridge stocked?

Solution:
● DIY smart fridge.
● Realtime data.
● Mobile app.
Smart fridge requirements

Solution:
- DIY smart fridge.
- Realtime data.
- Mobile app.

Breakout rooms:
- Two teams, 3x5 minutes
  1. Team 1: customer; team 2: developer
  2. Team 2: customer; team 1: developer
  3. Group requirements

Instructions:
https://docs.google.com/document/d/12CTMEWtMGy7H5S0XK35NTF5risIJ1x6M_LdW1UeQ02w
Requirements engineering

The process of eliciting, analyzing, documenting, and maintaining requirements.

Classic way to classify requirements:

- **Functional requirements**
  - E.g., input-output behavior

- **Non-functional requirements**
  - E.g., security, privacy, scalability

- **Additional constraints**
  - E.g., programming language, frameworks, testing infrastructure
Challenges and mistakes

Common Challenges
● Unclear scope and unclear requirements.
● Changing/evolving requirements.
● Finding the right balance (depends on customer):
  ○ Comprehensible vs. detailed.
  ○ Graphics vs. tables and explicit and precise wording.
  ○ Short and timely vs. complete and late.

Common Mistakes
● Implementation details instead of requirements.
● Feature creep/bloat.
Feature creep/bloat

Feature creep:
- Gradual accumulation of features over time.
- Often has a negative overall effect on a large software project.

Why does feature creep happen? Because features are fun!
- Developers like to code them.
- Sales teams like to brag about them.
- Users (think they) want them.

Why is it bad?
- Too many options, more bugs, more delays, less testing, …
- "Boiled frog" analogy.

Can you think of any products that have had feature creep?