Today

- Quiz 5
- Requirements (wrap up)
- Estimation

Reading

- Ch 2: 48-67
Requirements Analysis

Customers rarely provide succinct, complete, well-organized lists of requirements

So, as requirements analysts:

- it is our job to analyze and organize the information we get
- to create a useful set of requirement statements (specification)

Types of Requirements

Business requirements

Business Objectives

Business rules

- constraints imposed on the system due to the nature of the business
- e.g., when certain tasks must happen, policies must be followed

Functional requirements

features

- feature (function) of the system
- e.g., user stories (although less “formal”)

Non-Functional requirements

characteristics

- characteristic or attribute of the system (or of a feature)
- fast, secure, reliable, easy-to-use, ...

S. Bowers
System **Constraints**

- hardware or software constraints
- e.g., must run on a phone
- or must be a windows application

**Beware unnecessary constraints!**

- make sure it is really a *constraint* and not just a *suggestion*

**Solution Ideas**

- requirements that mix *what* must be done with *how* to do it
- these require *clarification* by the user ...
- solution may also suggest other needs implicitly
  - e.g., instead of a parallel DB system ... some of the features have a time constraint (e.g., < 1/2 sec. response time)
Writing Good Requirements

General Suggestions

• use terms consistently (glossary can help)

• **break** vague requirements into smaller, specific ones

• use **consistent** wording (e.g., user-story mad lib)

• use “must” or “shall” instead of “should”, “may”, “might” ... Why?

• use specific user role when possible

• use good grammar, spelling, complete sentences

• avoid vague and ambiguous words

Examples of words to avoid in requirement statements

Q: Why should these be avoided?

• acceptable, adequate verify?

• as much as possible vague

• efficient, improved, better verify?

• including, and so on, etc. incomplete

• normally, ideally verify, necessary?

• optionally necessary?
Avoid “Analysis Paralysis”

- goal is to write “good enough” requirements (Agile/Scrum)
  - get started early on design and development
  - but minimize level of (rework) risk
  - and provide initial estimates
- expect requirements to change

Prioritizing Requirements

Ranking importance of features is often hard

- especially after just coming up with a long list of features
  “I need all these features, just make it happen somehow.”

Estimation can help!

- if you determine it will take 12 months to deliver all features
- but customer needs product in 6 months
- priorities now become an issue

In iterative models (e.g., Scrum):

- each iteration is a fixed length (e.g., one-two weeks)
- only a few stories can be developed per iteration
- need to select which ones to work on first
• then again before each iteration ... possibly involving re-prioritization

In more “traditional” models

• prioritization often done once

• as part of requirements gathering

Our book uses the priority scale:

<table>
<thead>
<tr>
<th>Scale</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>most important</td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>least important</td>
</tr>
</tbody>
</table>

Multiples of ten on purpose:

• group features instead of a total order

• can squeeze features between groups later if needed (e.g., 25)

• need to be aware of (and track) dependencies between features
Tricks for when it is hard to assign priorities

Classify by **urgency and importance**

- classify each feature as either URGENT or NOT-URGENT
- and as either IMPORTANT or NOT-IMPORTANT
- focus just on the URGENT & IMPORTANT features
- might consider NON-URGENT & IMPORTANT
- beware of URGENT & NOT-IMPORTANT features!

Assign a **value**, **cost**, and **risk** value

- higher value and lower risk & cost have **higher priority**
- useful to sort out “middle of the pack” requirements
Estimation via Planning Poker

In the textbook, to estimate, just follow this simple formula:

\[
\text{estimate} = \sum_{i=1}^{n} \text{estimate of user story}_i
\]

And estimate user stories using planning poker

• involves the developers ... they do the work!

• estimates based on “consensus” model (convergence)

• a kind of voting model ... avoids takeover by one person

How planning poker works ...

Everyone has a deck of planning poker cards

• each card has a number of person days (developer days)

• this is the number of days one person would need to finish the story

• assumes:
  – developer is working “full time” (40 hours/wk ... more later)
  – developer is only working on the story
  – finishing the story includes design, code, test, and deliver
• ? … not sure
• Coffee cup … take a break
• 0 … already done
The planning poker “rules”

1. Dealer places a user story on the table
2. Each player chooses an estimate card for the story
3. Each player lays their estimate card face down on the table
4. Each player turns their estimate card over at the same time
5. Players analyse the result:
   - What is the spread spread across the estimates?
     - dealer writes these down
     - the larger the difference in the spread ...
     - the less confident the team is in the estimate
   - What are the assumptions?
     - each player describes their assumptions leading to estimate
     - dealer writes these down
6. Agree on a consensus estimate of the story
   - a single “point” estimate for the story (more later)
   - along with a list of assumptions for the consensus estimate
   - may need to clarify assumptions (e.g., with customer)
   - may need to break up story into smaller ones and replay
7. Goal is to end up with estimates the team is confident about
What is a “good” estimate?

We want estimates that we are confident are accurate

“The process is called estimation, not exactimation” (Phillip Armour)

While we often give “single-point” numbers of developer-days ...

- it is hard to be 100% confident in such a number
- estimates have inherent uncertainty
- are more natural to express as ranges or using confidence levels (probabilities)
- Let’s say we want “90% confidence” in our estimates ...