This Week ...

Start 3rd sprint (first “real” sprint)

- Follow the Scrum process … sprint planning meeting
  (select stories for sprint backlog), sprint review (w/ sponsor), sprint retrospective
- Base on where you are in the project, e.g., some dev work, more design work, some tool/setup work

Due next Tuesday:

- Results of your planning meeting (including sprint stories)
- Draft of Project Plan Section 3
- Be sure to read over HW-5
Types of Requirements ...

**Business objectives (external objectives)**
- desired outcomes of the product for the company (client)
- increase profits, save time/money, attract investors, ...

**Business rules**
- constraints imposed on the system due to nature of business
- e.g., policies, regulations, processes, ...
Types of Requirements

Functional
● Needed functions (or features) of the system

Nonfunctional
● characteristic (attribute) of the system or a feature
● fast, secure, reliable, easy-to-use, …
● should be “measurable” (quantified)

Types of Requirements

System constraints
● hardware or software constraints
● e.g., must run on ios device, must be written in C#

Beware of unnecessary constraints!
● make sure it is really a constraint and not a suggestion

Solution ideas
● requirements that mix what must be done with how to do it
● these require clarification by the user …
● may suggest other (hidden) requirements:
  ○ e.g., instead of a distributed DB server … some of the features may just have a time constraint
Types of Requirements

Stretch goals (not “requirements”)

- What to do next if requirements are finished (e.g., after MVP)
- Good to identify and prioritize (helps define MVP)

“Good” Requirement Statements

Complete (hard!)

- fully describes function or characteristic
- all information needed to start designing/implementing

Correct

- accurately describes function or characteristic
- captures feature needed by customer/users

Feasible

- possible to implement
- in given time frame and/or with technologies
“Good” Requirement Statements

Necessary (!)
● customer actually needs it (not just “nice to have”)
● good to trace back to requester (who needs it)

Unambiguous
● all readers should agree
● a single, consistent interpretation

Focused
● about one feature
● e.g., check for “feature1, feature2, and feature2” in stories
● if you have these, split up into separate stories

Specific
● should be a concrete function
● try to break into at most one week’s worth of work (more later)

Verifiable
● can check software satisfies requirement
● e.g., as an “acceptance test”

Show deposits in transaction history.
1. Login
2. Make a deposit
3. Go to "transactions"
4. Check that new deposit added
“Good” Requirement Specifications

Complete (hard!)
- no missing requirements
- warning: hard (if not impossible) to do

Consistent
- no conflicting requirements

Modifiable (bec. they’ll change)
- give them unique labels/names
- best to pick meaningful names
- traditional: hierarchical numbering (R.1, R.2.1, …)
- or mix: ED-1.1, ED-2.1, … ED stands for “editor functions”

“Good” Requirement Specifications

Prioritized (!)
- requirements ordered by priority
- highest to lowest (1, 2, 3, … or 10, 20, 30, …)
- High, medium, low

Note: sometimes prioritizing is hard
- customer may not understand … they want everything!
- estimation can help (more later)

Tricks to help (there are numerous prioritization approaches):
- must, should, could, won’t have
- assign {urgent, not-urgent} and {important, not-important}
- use value, cost, risk scores (focus high value, low risk & cost)
Tips for writing good requirements

General suggestions

- use terms consistently (may need a glossary)
- break vague requirements into smaller, specific ones
- use consistent wording (e.g., user story templates)
- use “must” or “shall”, not “should”, “may”, “might”
- include user roles (who is this really for?)

Tips for writing good requirements

Words to avoid in requirements

- acceptable, adequate
- as much as possible
- efficient, improved, better
- including, ‘and so on’, ‘etc.’
- normally, ideally
- optionally
Architecture Diagrams

How I generally think of architecture vs (detailed) design

- **architecture** focused on subsystems, technology choices
- **detailed design** focused on classes, interfaces, interactions

Organizing into subsystems:

- Names for major components / responsibilities
- Basic flow of data and/or delegation
- What specific technologies will be used

Context → Architecture → Detailed Design
Architecture Diagrams

Just like UI design, good to sketch architectures

- saves time
- prevents “over architecting”
- lets you quickly converge as a team

Like any diagram or model, presents an abstraction …

- helps team have a shared understanding
- use same terminology
- start to break up into well-defined areas of work

Architecture Diagrams

Can start with sticky notes or 3x5 cards …

- identify subsystems (e.g., major responsibilities)
- think about creating logical boundaries
- think about how they interact / what they need
- think about how external services / technologies fit
Examples

Examples
Exercise: In your team, identify subsystems

- what are candidates?
- what are their major responsibilities?
- what technology decisions (APIs, DBMSs, Formats, …)?

Sketch an architecture diagram