1 Project Overview

1.1 Project Summary

- **Project “why”:** problem being solved
- **Project “what”:** how the product solves the problem

1.2 Project Objectives

- Main “business objectives” of the project

1.3 Project Stakeholders

- Everyone with a stake/interest in project
- Team, sponsor, user groups, advisor, DAB, ...

1.4 Project Deliverables

- Software product and documentation (user and/or technical)
- Other possibilities: TOS, performance eval, maintenance plan, ...
Business Objective (Requirement)

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

Good business objectives are **specific** and **measurable**
- Measurable = quantified with definite time period (by when)

Some examples:
- Increase online sales by 15% by end of next year
- Improve customer response time by 10% in first 6 months
- 1000 downloads within first six months of release

Business Objective (cont)

Why worry about defining the business objectives?
- Informs your team about the expectations of the product
- Can help define the MVP
- Can signal unrealistic expectations

Some mistakes I see:
- Business objectives are not deliverables or features
- May be difficult to get to quantified objectives (but try!)
  - Sometimes won’t make sense or artificial to quantify
- Can’t create these without help from sponsor / client
Exercise: Business objectives

1. Individually: Define two business objectives for your project
   - Make sure they are measurable!

2. Discuss them all as a group, pick & improve on two

Introduction to Sprints

Work on your milestones as a series of iterations

- called “sprints” in scrum
- usually 1-4 weeks (for your project 2 weeks)

Each sprint has

- a goal (what functionality are you tackling?)
- something new to show customer at end of sprint (visibility)
- review with customer/users & plan next sprint
2. **Sprint planning**

   - **Sprint Backlog**
     - Focus of sprint

   - **Sprint**
     - 2 weeks

---

1. **Initial backlog** (before 1st sprint)

2. **Sprint planning** (sprint backlog)

3. **Carryout sprint** (taskboard model)

4. **Sprint Review**
   - Demo/feedback

5. **Sprint Retrospective**
   - Process improvement

---

**Scrum Process (cont)**

---

**Example Backlog**

**Scrum Alliance Webpage**

(https://www.mountaingoatsoftware.com/agile/scrum/scrum-tools/product-backlog/example)

*Note: Without priorities …*
The plan

- “user stories” (agile/scrum technique)
- requirements in general

Requirement Statement (or just requirement)

- a single feature or characteristic of the system
- needed by the customer to make the system useful

Requirement Specification

- a set of requirement statements
- together make up the requirements of the product

… captured through the product backlog in scrum

User Stories

A specific kind of requirement statement

- describes one thing the software must do
- written in “language the customer understands”
- written “by the customer” (not literally …)
  - driven solely by customer need … even if dev wrote it
- short and sweet!
  - usually 1 sentence
  - should fit on one side of a 3x5 card
- Meant to start discussion about functionality & features
User Stories

Properties of “Good” User Stories (from Intro to Agile)

- **Independent** … stand alone (as much as possible)
- **Negotiable** … spur discussion on best solution
- **Valuable** … to business, highest value highest priority
- **Estimatable** … not too vague or big to understand complexity
- **Small** … can complete in one sprint
- **Testable** … can be tested to ensure works as expected

User Stories

User stories contain ...

1. a meaningful **title** (short & descriptive)

2. a short **description** (see template)

3. **acceptance tests**

   *what needs to happen for story to be “completed”*
   
   - conditions to check for
   - context/pre-post conditions (e.g., first select x, then …)
   - starts on back of 3x5 card
User Stories

User stories **should not** ...

1. use technical terms unfamiliar to customer

2. mention specific (implementation) technologies
   - unless absolutely required by customer (more later)

3. be unnecessarily detailed

User Story Templates

It can help to have consistently worded stories …

- both for reading and writing stories

One way to achieve this is to use a template:

   “As a *(role)* I can *(achieve function)* so that *(reason)*.”

For example:

   “As a **student** I can **purchase a parking pass** so that **I can drive to school**.”

Good practice to include roles and reasons (benefits)

- if obvious though, don’t need them
Some Distinctions ....

A “**function**” is something a product can do for a user

- what can be done with the software
- e.g., make a phone call

A “**feature**” is how the product helps enable a function

- e.g., a keypad to enter the number, a dial button to initiate the call

_Sometimes_: “As a (role) I can (feature) so that (reason).”

Exercise

1. Individually, define your product’s user **role**s …
   - who are the users?
   - will they perform different tasks?
   - brainstorm the various roles of your users

2. Share and improve with your team
Exercise

1. Individually come up with at least two user stories …
   - use the template:
     
     As a (role) I can (do function) so that (reason)
   - Consider stories for each different role

2. Share and improve stories with your team

Some Distinctions ....

User stories can be at varying levels of detail

“Epics” are large stories
   - Covers a large amount of functionality
   - Requires multiple sprints to complete

“Themes” are smaller than epics but larger than stories
   - An epic usually is broken into a set of themes
   - Each theme consists of a set of stories

So Epics ⇒ Themes ⇒ Stories
What is the “right size” for a story?

- Provides concrete value, describes (smallest) action a user wants to do, done in one sprint

When to decompose stories into smaller ones:

- Look for “x, y, and z” (3 stories) or “u or v” (2 stories)
- If the story is vague ...
  - Can you (easily) estimate how much work involved?
  - Do you know the corresponding features?
  - Do you have testable acceptance criteria?
  - Still big questions about it?
- Too big for a sprint

A “spike” is a special story for gathering information

- E.g., to develop a proof of concept, compare two approaches, do research, etc.
- As a story, brings the need for the activity to the attention of the customer (visibility)
- Is short in duration (like a normal user story)
Some Distinctions ....

The **product backlog**

_Prioritized_ list of the work to do (“backlog items”)

- Stories (features), spikes, bugs, non-functional requirements, …, meaningful to the customer

A **living document** throughout the project

- High priority items (generally) done first, and are smaller and more detailed than low priority items

Reviewed and updated regularly (every sprint) …

sometimes called “grooming”

---

User Story “Tools”

**Tools for Managing User Stories & Product Backlog**

- 3x5 cards (e.g., taped to wall in lab)
- Shared document (e.g., text or spreadsheet)
- Trello (see next slide)
- Others:
  - GitHub issues (more later)
  - Various commercial products (e.g., confluence, sprintly)

_How to Use User Stories_:

- _User Stories_ are the primary source of requirements.
- _User Stories_ are _written_ not _told_.
- _User Stories_ are _created_ by _everyone_.
- _User Stories_ are _organized_ by _product backlog_.
- _User Stories_ are _reviewed_ and _updated_ regularly.

_Note:_ You don’t have to follow the user story “template”, but your stories should be consistently worded!