Requirement Analysis and Specification

Hui Chen [a]

[a]CUNY Brooklyn College, Brooklyn, NY, USA

March 3, 2022

Outline

Outline

Preparation for Requirement Engineering

1. Plan for requirement activities
2. Agree on resources, methology, schedule for requirement activities
3. Obtain and organize the agreed upon resources and methodology

Major Requirements Engineering Activities

- Elicitation
- Analysis
  - Categorizing the requirements (by some criteria)
  - Prioritizing the requirements
- Documentation and definitions
- Prototyping
- Specification(s)
- Review and validations
- Gain agreement and acceptance

Requirements Elicitation

Requirements describe the user's needs and desires.

- "What" vs. "How"

Need to seek out the business and management perceptions and goals for the software project. Business opportunity and business needs

- Justification for the project
- Scope
- Major constraints
- Major functionality
- Success factor
- User characteristics
- ...

Requirements Categories

Most high-level: Functional vs. Non-functional

- Individual functionality (features)
- Business flow (usage "scenarios")
- Data and information needs
- User interfaces
- Other interfaces to external systems/platforms
- Various constraints (non-functional)
  - Performance
  - Security
  - Reliability
  - ...

Requirement Prioritization

Most of the time we have some limitations in developing software:

- Time
- Resources
- Technical capabilities (existing)

We need to prioritize the requirements to satisfy these limitations.

A Simple Requirements Prioritization List Example

Requirement Number	Brief Requirement Description	Requirement Source	Requirement Priority	Requirement Status
1	One-page query must respond in less than 1 second	A major account marking representative	Priority 1	Accepted for this release (release 3.1.2)
2	Help text must be field sensitive	Large account users	Priority 2	Postponed for next release release 3.1.3)

Requirements Definition/Prototyping/Review

Once the requirements are solicited, analyzed, and prioritized, more details may/must be spelled out. Three major activities that may be intertwined must be performed:

- Requirements definition/documentation
- Requirements prototyping
- Requirements reviewing

Requirements Definitions/Documentation

Requirements definitions may be written in different forms:

- Simple input/process/output (I-P-U) descriptions in English
- Dataflow diagrams (DFD)
- Entity relations diagram (ERD)
- Use case diagram from Unified Modeling Language (UML)
- User stories
- ...

Once the requirements are defined in detail using one of the above forms, they still need to be reviewed by the users/customers and other stakeholders.

Requirement Tracebility

- Capability to trace a requirement is needed to ensure that the product has fully implemented the requirements (while not part of requirements process activities – it needs to be started early).
- We need to trace requirements:
  - Requirements from source (people and documents)
  - Requirements to later steps (design, implementation, test, and release)
- We also need to link requirements to other prerequisite requirements.

Requirements Prototyping

- Requirements prototyping mostly addresses the user interface (UI) part of the requirement in terms of:
  - Visual looks (size, shape, position, color)
  - Flow (control and logical flow; depth of flow)
- The prototyping may be performed in one of the two modes:
  - Low fidelity: using paper/cardboard to represent screens and human to move the boards
  - High fidelity: using automated tools such as Visual Basic to code the screens and direct the logical flow of these screens

Requirement Specification

- Once the requirements are defined, prototyped, and reviewed, they must be placed into a Requirements Specification document.
- IEEE /EIA Standard 12207.1-1997 outlines the guideline for three major sections of a requirements specification document.
  - Introduction
  - High-level description
  - Detailed descriptions
  - Details of each functionality (input-output-process)
  - Interfaces, including user interfaces and network interfaces
  - Performance requirements (response time, throughput, etc.)
  - Design constraints, standards, etc.
  - Additional attributes such as security, reliability, etc.
  - Any additional unique requirements

Outline

Why Do Software Projects Fail?

- Don't deliver what customers want, or
- projects are late, or
- over budget, or
- hard to maintain and evolve, or
- all of the above

How does Agile try to avoid failure?

Review of Agile Model

- Work closely, continuously with stakeholders to develop requirements, tests
  - Stakeholders. Users, customers, developers, maintenance programmers, operators, project managers, ...
- Maintain working prototype while deploying new features every iteration
  - Typically every 1 or 2 weeks
  - Instead of 5 major phases, each months long
- Check with stakeholders on what's next, to validate building right thing (vs. verify)

Outline

Behavior-Driven Development – an Agile Approach

- Miscommunication is a common problem in requirements engineering:
  - How to communicate requirements clearly to all stakeholders (customers, developers, management, etc.)
- Behavior-Driven Development (BDD) is a specific method for tying requirements to testing
- BDD asks questions about behavior of app before and during development to reduce miscommunication
  - Validation (build the right thing) vs. Verification (build the thing right)

Requirement Sign-Off

Having a requirements specification agreed to and signed off on is important.

- Serves as a milestone marker and formally exits a phase of software engineering.
- Serves as baseline from which any future changes can be monitored and controlled.

How about “agile”?

Outline

User Stories For Requirements

Expressing Software Requirements as User Stories

- Requirements written down as user stories
- Lightweight descriptions of how app used

User stories concentrate on behavior of app vs. implementation of app

- No mention of implementation decisions (sort of)
- User stories are refined into implementable tasks later

User Stories

- 1 – 3 sentences in everyday language
  - Fits on small index cards
  - Written by/with customer
- "Connextra" format:
  - Title = feature name
  - As a [kind of stakeholder], I want to [do some task], so that [I can achieve some goal],
  - 3 phrases must be there, preferably in that order

Reference: https://www.agilealliance.org/glossary/user-story-template/

Example User Stories

```
Add a book

As a book club organizer,
I want to add books to the Book Club book database,
so that I am sharing the books to read
```

Outline

"SMART" User Stories

- be Specific
- be Measurable
- be Achievable, ideally, implement in 1 iteration
- be Relevant
- be Timeboxed, i.e, to know when to give up

Specific and Measurable

- Implies known *valid input* and *expected results* exist for the user story
- Each *scenario* is testable – define scenario later

Good vs Bad Examples

`Use can search for a movie`

vs.

`Use can search for a movie by title`

Good vs Bad Examples

```
BCBookClub app should have good response time
```

vs.

```
When adding a book on BCBookClub, 99% of "Add Book" screens sh
appear within 2 seconds
```

Achievable

- Complete in 1 iteration (2 weeks)
- If can't deliver feature in 1 iteration, deliver subset of stories
- Always aim for working code at end of iteration

Relevent

Why have the user story? – Discover value, or kill the story:

- Protect revenue
- Increase revenue
- Manage cost
- Increase brand value
- Making the product remarkable
- ...

Can you include stories that don't have obvious value?

The "5-why" method (https://en.wikipedia.org/wiki/Five_whys)

Timeboxed

Set a time budget to develop each user story

- Stop story development when exceed time budget
- Give up or divide into smaller stories or reschedule what is left undone
- Reassign priority (talk to the customer)
- To avoid underestimating length of project

Outline

Project Backlog?

- Real systems have 100s of user stories
- Backlog: User Stories not yet completed
- Prioritize so most valuable items highest
- Organize to form SW releases over time

Organizing User Stories

Devide based on categories, priorities, and assignments

- To Do.
- Doing with assignment
- Done

Outline

Acceptance Tests

- There are different types of tests we may want to carry out for our application
- One type is acceptance tests
  - Acceptance are very high-level (away from the details of the implementation)
  - Acceptance tests define what constitutes a finished product, i.e. the customer accepts your application and gives you a check

Mapping User stories to Acceptance Tests

- Wouldn't it be great to automatically map small index card user stories into tests for user to decide if accept the app?
- Tests from customer-friendly user stories
  - Acceptance tests: ensure satisfied customer
- How could you run the tests without a human in the loop to perform the actions?
  - Decompose user stories into smaller testable units
  - Rely on testing automation libraries/frameworks

Scenarios

Map user stories to multiple testable scenarios
Format: Given/When/Then.

- Given: some specific starting condition(s),
- When: I take specific action X,
- Then: one or more specific thing(s) should happen

Create Scenarios for User Stories

- A user story typically maps to one feature
- A feature maps to ≥ 1 scenarios that show different ways a feature is used
  - The keywords *feature* and *scenario* identify respective components
  - Both happy path & sad path scenarios
- Scenarios: typically 3 – 8 steps

Example Scenario

```
Feature: User can maually add books

Scenario: Add a book
 Givem I am on the BCBookClub app
 When I follow "Add new book"
 Then I should be on the "Create New Book" screen
 When I fill in "Title" with "Charlotte's Web", "Author" with
 And I select "Children's Book" from "Category"
 And I presss "Save Changes"
 Then I should be on the BCBookClub app's start screen
 And I should see "Charlotte's Web"
```

Outline

Summary and Questions

- Overview of requirements engineering
- BDD (an agile model)
  - User stories are the means by which we express application requirements in agile processes
  - Behavior Driven Design advocates a specific type of testable user story
  - User stories should be SMART
  - Map user stories into acceptance tests – via several scenarios

Outline

"Engineering Software as a Service" by Armando Fox and David Patterson (2nd Edition)
"Essentials of Software Engineering" by Frank Tsui, Orlando Karam, and Barbara Bernal(4th Edition)