CS3733 - Software Engineering

Kyle Corry

September 30, 2017

Contents

1 Software Development Phases

- Requirements Gathering
- Analysis
- Design
  - System design
  - Object design
- Development
- Testing
- Release
- Maintenance

2 Requirements Gathering

In the requirements gathering phase, you must work with the user to determine what the program must do. You will also be conducting research and starting on brainstorming on what your program is required to do.

There are two types of requirements: Functional and non-functional. Functional requirements specify what your program must be able to do, non-functional specifies how your program should work.

Users stories and use cases are also created during this phase.

2.1 Types of Requirements

2.1.1 Functional Requirements

Functional requirements are things that the program must do and are normally represented as users stories. These are essentially actions. They are not worried with how the program will look, how fast it will be, or how it gets the task done. These focus on the bigger picture, such as the program must be able to accept and store an image from the user; it does not worry about the details in the action, such as what the interface for the user to upload an image to will look like.

2.1.2 Non-functional Requirements

Non-functional requirements are things that the program does but they are not as important as the functional requirements. These cover how the program will look, how fast it will be, and how it gets the task done internally.

2.2 User Stories

A typical user story will represent a single action that the program will do. These actions can be broad, or very specific. Broad actions, covering a good deal of functionality are called epics.

2.2.1 Epics

An epic is a user story which in itself is composed of many user stories. A program will typically have one or more epics. The epic will represent an entire functionality of an application. An example of an epic could be the program must be able to allow users to submit an application; that task in itself is composed of many user stories such as what the application will contain and the submission details.

2.2.2 User Story Format

The typical user story look like this: As a [type of user] I want to be able to [action] so that [reason]. For example, with a Facebook application to block certain posts that user story would be as follows: As a Facebook user I want to be able to block posts so that I don't have to see them in my feed.

User stories can also have conversations attached to them. The conversations are specific point of the user story which will state the conditions of what makes the user story happen. These can be thought of as accepting criteria. For example, going back to the Facebook blocking user story, a conversation for that could be: all posts specified by the user will be blocked.

3 Analysis

In the analysis phase of the design process, you must analyze the problem and create basic object and state diagrams. The object diagrams are the representation of your system in terms of objects and classes. During this phase you will also identify the different types of objects in your program and how they interact with each other. The state diagrams will map out the flow of state in your program. This phase will allow the design phase to be easier and go smoother.

3.1 Object Diagram

Object diagrams will map out the objects that will be present in your program and determine how they will interact to create the functionality of the system. This involves identifying the entities, boundaries, and controllers in your problem space. On top of this, it may also be useful to separate the objects into subsystems, grouping objects with common functionality together.

3.2 State Diagram

State diagrams are used to represent the different states that a system will be in. Once the states of the system are identified, you must identify how each state will transition into the next state. An example of this would be an elevator: The states could be stopped [door open], stopped [door closed], traveling up, traveling down. The transitions between the states can then be identified. The elevator can only travel up if it was first stopped with the door closed or already traveling up. State diagrams can be seen as equivalents to the finite state automaton.

4 Agile SCRUM

The Agile SCRUM software methodology is a way to go through the software development phases in an efficient and robust way. The Agile process is known for fast sprints or iterations and its flexibility to change.

4.1 Sprints

Sprints are the core of Agile SCRUM and they represent about a week development duration. Each week will have a backlog of features to be implemented, and the week itself is called the iteration. At the end of the week, most of the backlog should be implemented and the cycle can start over.

4.1.1 Sprint Planning Meeting

The sprint planning meeting happens at the beginning of each iteration. During this, the team needs to determine what needs to be completed by the end of the iteration and user story points are assigned to the backlog items. These determine the difficulty of the tasks, and the iteration as a whole. Along with the project backlog being created, any requirement changes are handled. The product owner, SCRUM master, and lead software engineer should discuss this prior to the meeting so task in the backlog can be assigned to the software engineers.

4.1.2 Daily Standup Meetings

SCRUM normally has daily meetings in which the team members will all get together to stay on track and identify possible setbacks. The meeting is structured so that each member will be able to say what they are currently working on, identifying what they need from other software engineers, any issues that they are facing, how long they will need to finish their work, and what they are going to do before the next meeting. If there are remaining tasks on the backlog and a software engineer has completed their task, they are assigned a new task. If there are no tasks left, software engineers who are finished may be assigned to help out another software engineer. The meetings are typically 10 to 20 minutes long depending on the team size.

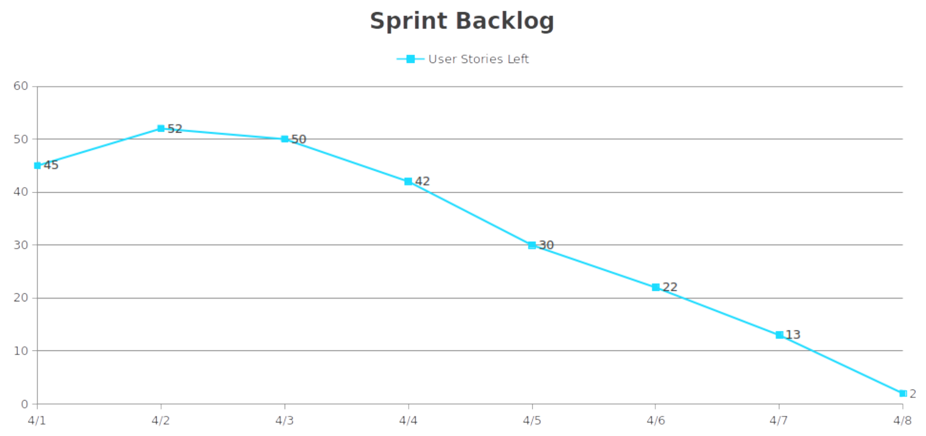
4.1.3 Reflection / Retrospective Meeting

The retrospective meeting is held at the end of each iteration. The topics discussed at this meeting are: what went well during the week, what needs to be improved in the program and/or development process, what was not finished during the sprint and why, and any changes that are needed for the following week.

4.1.4 Burndown Charts

A burndown chart is a measure of the progress made during the iteration, and should be updated every day of the sprint. Every day the amount of remaining user story points should be recorded and plotted on a line chart of user story points (Y) vs days (X). At the end of the iteration, the amount of user story points completed in that iteration becomes the sprint velocity. Note that sometimes the user story points can increase depending on if new features are required for other ones to be implemented, or if a story is worth more points than originally assigned.

Figure 1: Example of a burndown chart


Sprint Backlog
User Stories Left
4/1
4/2
4/3
4/4
4/5
4/6
4/7
4/8