Module 1
Introduction to SaaS, Agile, Cloud Computing

CS 169A: Software Engineering

1 Overview

Welcome to the world of software engineering! As you will soon find out, becoming a capable software engineer involves much more than knowing how to write algorithms. As you'll see across the course of the semester, an apt software engineer has not only technical proficiency, but also personable, collaborative, and decisive, among many valuable characteristics.

In module 1, we introduce the concepts of Software as a Service, along with different workflows suited for different kinds of projects that allow teams to coordinate appropriately. In this worksheet, we'll ask you to recall and apply some of these concepts from the first chapter.

Do note that some of these questions don't have any one specific correct answer! Get creative! There's a lot of examples that capture and embody the same concepts. Feel free to write your answers or discuss with classmates!

2 Workflow Concepts

In this chapter, we learned about a variety of different software development processes, the main ones being Plan and Document (Waterfall), Spiral, Rational Unified Process (RUP), and Agile. Answer the following questions to refresh your understanding and think about how to apply these concepts.

What are the differences between the Waterfall Process and the Agile Process?

Agile proceeds as a series of short iterations, each one limited in the scope of work and aimed at getting working and well-tested code in front of the customer rapidly. Changes to the design, if needed during development, are accomplished by refactoring. Each Agile iteration includes all the waterfall phases "in miniature" - requirements gathering, design, development, testing, deployment/release, maintenance (in the form of refactoring). Waterfall does these phases sequentially, only moving to the next phase when the previous phase is complete, which can make changes harder to manage since the change must "propagate down" from the earlier phases to later phases.

Can you design and build hardware with the Agile Process?

Some parts of the design cycle can be adapted to Agile. Tools such as CHISEL allow specifying hardware at a much higher level and using code generation to assist with the design, thus bringing to hardware some of the productivity advantages of software engineering and thereby enabling more agility.

Describe a recent software project you've been involved with, and whether you think it would be a good fit for Agile or not. Do your classmates agree?

Use the table in Section 1.3 (Figure 1.5) in book as a guideline for determining which projects fit what kinds of workflows. These projects can be from school, research, past internships, side hobbies, anything in the realm of software engineering!

3 Verification & Validation

Testing is an extremely important skill that is integral to writing correct, efficient software. In this class, we'll explore the concepts and tools behind writing tests that verify both technical and behavioral benchmarks. for now, recall that verification is for checking "Did you build the thing right?" (Did you meet the specification?). On the other hand, validation is concerned with "Did you build the right thing?" (Is this what the customer actually wants).

You're building the shopping-cart checkout portion of an e-commerce website. What would be an example of each of the following types of tests for the shopping cart?

Unit Tests: Makes sure that a single procedure or method does what was expected. For instance, if there is an underlying function, such as a `calculatePrice` method for a shopping cart object, a unit test would be helpful for ensuring that the method returns the right price corresponding to the total of the items in the shopping cart.

Module or Functional Tests: These check functionality that may involve multiple cooperating classes. For example, if I submit the Web form for adding an item to the cart, does the cart then end up containing the correct item?

Integration Tests: These ensure that the interfaces between the units have consistent assumptions and communicate correctly. A good example would be ensuring that two classes are interacting properly (i.e. a User object can invoke the `addToCart` method of a shopping cart object.

System or Acceptance Tests: This is the highest level test. A good example would be a Cucumber test (Module 8) that simulates a mock user interacting with the website, adding several items, then making a purchase.

What are some formal methods of verification and what are their pros and cons?
This is a computationally intensive technique to verify that code abides by a certain behavior. These are mathematical proofs, either done by a person or done by a computer. The two options are automatic theorem proving or model checking. Because formal methods are so computationally intensive, they tend to be used only when the cost to repair errors is very high, the features are very hard to test, and the item being verified is not too large.

4 Programmer Productivity

With the increasing plethora of available, open source software, there's absolutely no need to reinvent the wheel when it comes to finding tools that bolster the amount of work you're able to get done. Throughout this class, you'll be introduced to a variety of platforms, websites, and libraries that will expedite the development process. All of these tools will typically fit under one or more categories of productivity.

Define and describe an example of each of the following productivity concepts: Clarity via conciseness, synthesis, reuse, automation and tools.

- Clarity via conciseness: An assumption that if programs are easier to understand, they'll have fewer bugs and to be easier to evolve. An example is higher level programming languages that raise abstraction level.
- Synthesis: Save effort by automating with code that is generated automatically rather than written manually.
- Reuse: Recycle portions from past designs rather than write everything from scratch (*NOT* Stack Overflow).
- Automation and Tools: Replace tedious manual tasks with tools to save time, improve accuracy, or both.

5 Software as a Service

SaaS applications dominate the majority of software that we use in our daily lives. The main premise of SaaS is that software is run in a web browser instead of having to be downloaded and install on a local machine. An example of SaaS would be any Google App (i.e. Maps, Drive), all of which are meant to be accessed from your browser. On the other hand, you would download a tool like Adobe Illustrator to your local computer, which is a more traditional approach.

How does SaaS differ from traditional software? Think about what features might make it more advantageous to deliver the app in a browser rather than as a local installation.

Section 1.6 has a complete list of advantages from using SaaS, along with some differences between SaaS and traditional software.

- Customers don't need to install the application. This eliminates customer concerns about having a compatible operating system or hardware.
- The service's data is kept within the service, saving customers the work of having to be responsible for local data. Services can also analyze this data to create better experiences, although the ethics behind such practices are in still debated (i.e. If a service can see your data, does that violate a user's right to privacy and security?)
- With permission, the same data can easily be shared among users.
- Only a single copy of the server software is run, voiding any concerns about users having old copies of software.
- It's much easier to roll out new features and make sure customers are on-boarded.