ENGINEERING

The Principles of Software Engineering

How did software engineering actually begin?

It was triggered by so called “Software Crisis” in 60’s which was a result of lack of best practices in implementing software - projects consistently failed to deliver reliably, on time and on budget. In 1968, the term “SW engineering” found its roots.

What processes were early software engineering techniques modeled after?

After traditional engineering disciplines.

What macro-scale trends helped define the arc of software engineering over the past half-century?

SW engineers learned to break giant projects into modular (and much more manageable) pieces that communicated via interfaces.

What specific advances have led to rapid change in the discipline since the 1990's?

Internet was born -> web browser was created -> open-source movement was born, the major reasons for the latest explosion in software engineering productivity.

What characteristics of the web make modern day web software development more flexible than traditional software?

The rise of commodity computers has led to the development of the "cloud", so now applications can be updated and accessed in real time as opposed to downloaded onto a user's computer. Tools accessible in the cloud means it does not matter, what device or which operating system I’m running as long as I’m connected to the Internet. The rise of cloud computing and more demanding consumers have also led to the rise of new project management techniques like Agile Development.

Why are software engineers more like architects than builders?

The compiler (which turns source code into something the computer can execute) is what actually does all the building. The engineer, on the other hand, must figure out what the compiler is actually supposed to build. That requires creative problem solving.

Why are software engineers actually very similar to visual or UX designers in some ways?

They live firmly in the design phase of the problem solving process and are frequently required to solve loosely defined or unusual problems. Their techniques for doing so therefore are less concerned with building quickly than they are with finding and designing the right solution amidst trying out new things.

What are the four steps of the high level engineering problem solving process?

1. Understand the problem
2. Plan a solution
3. Carry out that plan
4. Examine your results for accuracy

What are the specific things you need to do to implement each of these four steps, particularly the first two?

Solve your problems fully BEFORE diving into the implementation of your solution.

Why is thinking about the user so important for engineers?

Because if I don't think about the user, I'm basically designing the thing for myself and there is a high probability, that my solution will fail.

What does KISS stand for and why do engineers love it?

Keep it simple stupid. It helps to prevent tendencies to build overly complex systems.

What does DRY stand for and why is it relevant to you?

Do not repeat yourself. If I do one thing several times, there is probably a way how to automate this process, thus saving a lot of time and more focus on the solution.

What is "You ain't gonna need it?" (YAGNI)

Just stick to the plan and don't waste time by adding more features.

What do you think are the top 3 themes of good software engineering?

Think first. Break problems into pieces. Keep things simple.

Why do we say that the engineering approach covers both Product and Process?

I don't know why you say it. Product will be used by a user so in every step we take in the process, we have to think about our user.

The **Product** is what we're actually building - coming up with a software solution and being able to code it up properly.

The hidden side of engineering is the **Process**, which means how we're actually building our product. We need to make sure we're following a process that lets us create that Product in the most efficient and effective way possible.

Introduction to Agile Development

Why does Waterfall seem to follow the core engineering problem solving steps?

Because it copies it + installation and maintenance steps.

What is Waterfall good for?

The process is designed to make really sure that you're building a functional product. It's a legacy of hardware construction, where it's very costly to change things after the fact. It's ideal for projects with high barriers to future change or serious safety or reliability needs (e.g. space programs or critical health software).

What is Waterfall not good for?

It assumes that I know what I'm going to build from the beginning and can spend long periods of time building it. It is not good for companies developing consumer-facing software, SW has to be more responsive to change.

What is Agile Development?

It specifically advocates working with customers constantly during the development process to keep up with changing requirements. It promotes collaboration and adaptiveness, including co-location and pair programming.

What is Agile good for?

It is good for companies developing consumer-facing software, makes the process of making the software more responsive to changes.

What is Agile not good for?

Not suited for projects with strictly defined requirements, projects that won't change much over long periods of time.

What is eXtreme Programming?

XP provides a disciplined framework for teams to focus on producing code. It worries less about the overall project management and more about how to make developers more productive.

What development techniques does XP advocate?

1. Start with a release planning phase to figure things out
2. Have the clients define their requirements - “user stories"
3. Embed a user or their representative within the development team - feedback.
4. Mark features completed when the users have tested and accepted them - shorter "iterations" .
5. Empower the user / client to decide that enough features have been built - terminate the project.
6. Let developers integrate their branch of the code with the main code base as often as possible.
7. Measure the current development "velocity" - how much longer it's going to take.
8. Pair program (have two developers work on the same workstation).

What is SCRUM?

It is a very popular agile framework that provides both project management and an actual development process, it’s all about organizing the development of the User Stories we mentioned above in the section on XP.

What are the common principles of XP and SCRUM?

Building the software with the help of user stories and with the client/customer involved in every step of development. SCRUM teams often also use practices associated with XP like pair programming and continuous integration.

What are User Stories?

They describe the need from the perspective of the user.

What is the format of a typical story?

As a [specific type of user], I want to [accomplish some specific goal] so I can [achieve some benefit].

What are Acceptance Criteria?

They usually go to the other side of the “User Story” card. The specific tasks that need to be achieved in order for that story to be finished.

What is a Burndown Chart and how is it useful?

Each sprint has a certain number of total points. Each day I complete some stories and reduce the number of points left. Based on how many points I do every day, I can measure the velocity of the project

Sprint = feature I'm/we are working on; each sprint has number of points, how many points I add to each sprint depends on how complex I expect this feature to be/how much time it will take;. My average daily number of completed points is called velocity. The chart of my remaining points over time is called the burndown chart. Thanks to it, I can see how fast am I working and when the job should be done.

What is Pivotal Tracker?

It's a SCRUM based project management tool, great for working with user stories. It helps me to track my progress - implemented Burndown Chart.

What are the 6 steps of implementing a typical new Agile project?

1. **Gathering Requirements** from your client and/or users, "mocking up" with them; done by the product owner.
2. **Stories Breakdown** by turning the requirements into a series of bite-sized stories, thinking about the overall system architecture.
3. **Story Estimation** - estimating how many points it should be. Stories too large get broken down again.
4. The product owner will **Prioritize the Stories** based on client needs.
5. **Build the Software**. Complete the story. Clarify.
6. **Acceptance phase** - the Product Owner will accept or reject with a reason.

What is Software Testing?

It is a critically important part of the production process.

Why do we need automated testing?

To not to repeat ourselves when testing a large chunk of code behind the web application. Automated testing does the testing for me.

What does a QA department do?

Quality Assurance department - people manually executing checklists to check if everything works as it should.

What are the general characteristics of automated testing software?

Set of instructions that I give to the computer to run.

1. How to prepare for the test
2. Some action to take - clicking a button or calling a method with a specific input
3. Your expectation for what should happen afterwards

What does Test Coverage mean?

It describes how much of the code is actually tested. My duty is to find the balance between an acceptable level of coverage and the complexity (more time, higher costs).

What is Unit Testing?

Isolating a very specific unit of code (by faking any inputs/outputs/methods inside it) and testing its characteristics.

What is Integration Testing?

Verify the interfaces between units. This can occur on many levels, from small modules to high level architectural components.

What is Acceptance Testing?

The system is delivered to the user (or Product Owner in Agile) for acceptance or rejection.

What general types of behavior should you test?

1. Anything you would otherwise manually test (e.g. by checking the page to verify it loaded properly)
2. All critical paths the users will take (like signing in and using the app's core functions)
3. Anything that might reasonably break later (e.g. if it relies on code that may get changed later or produce unexpected outputs)
4. All critical methods that are run during the critical user paths.
5. Any bugs you discovered along the way (write a test for the bug that fails, then fix the bug, then your test will pass)

What is TDD?

Test-Driven Development. Start by writing the test and then write the code to make the test pass. I cannot write a piece of code without there being a failing test waiting to pass because of it.

1. Add a Test for the first piece of work in your new story.
2. Run all the Tests to make sure everything is good in your test suite.
3. Write the Code to Pass the Test in the simplest possible way.
4. Run all the Tests to make sure your new test is now "Green", or passing.
5. Refactor your Code - make the code better organized and more efficient.
6. Repeat

What are the advantages of TDD?

What does "Red/Green/Refactor" mean?

Test failure -> test passing -> refactoring

What is BDD?

Behaviour Driven Development. The business stakeholders (usually non-technical people) work with the developers to write high level tests called Scenarios based on their needs (or the "desired behavior" of the code).

How does BDD unite the ideas of User-Centered Design with TDD?

They are both follow the standard Agile Development story template - their desired behavior is set up like the acceptance criteria of those stories.

Pseudocoding and Good Software Design

What is Pseudocoding?

Coding translated to english.

What is Whiteboarding?

When engies are trying to figure out a problem by writeing a pseudocode on a piece of paper or boar.and pseudocode (aka Whiteboard)

Why is pseudocoding useful for a beginner?

It’s grear for learning basic logical structures of coding.

Why is pseudocoding useful for an experienced engineer?

Because it focuses on the building block concepts of programming languages without me having to worry about whether I’m using the "right" words. Once I have figured out the logic of coding, the rest is just filling in the pseudocode with the syntax of a particular programming language.

Why do we say that code follows "Sequence"?

Program runs from top to bottom in a continuous sequence, like a recipe for a dish.

What is Flow Control?

Diverting the flow by IF statements.

What is Iteration?

Performing the same set of instructions multiple times - While statement, for each statement, times statement.

What is Modularity?

Breaking apart a giant complicated system of tasks into individual specialized sub-tasks - modules.

How does modularity help us think about systems?

1. What parts of this problem can be broken into independent tasks (ie modules) and only called on when necessary?
2. What information does each task require to get started?
3. What do you need to know about how each task works in order to properly use it?
4. What is the minimum level of communication between tasks so you don't need to get involved more than necessary?

What is an Interface?

Interface is what allows the exchange of information between it and any other modules.

How can the real world be modeled as a modular system?

1. Lets me use abstractions to avoid re-inventing the wheel.
2. Specialization of tasks - each component gets very good at its job.
3. Allows me to identify trouble areas in my system much easier - system is not just one bloated module.
4. Allows me to add, remove, or modify specific parts of the system without changing other parts.

How specific should your modules be?

I got to start with a few high-level steps before diving into the details of the specific sub-tasks. Until a point, when it would be wiser to give a particular subtask to someone else.

How can you apply modules to your pseudocode "programs"?

Program is a system of separate and mostly-independent sub-tasks working together towards a common goal. Pseudocode helps me to solve more complex problems.

What is SOA?

Service Oriented Architecture. It embodies the idea of modularity on a large scale. In it, each separate sub-section of the overall architecture is designed to be completely independent from every other and they communicate using specified interfaces (called Application Programming Interfaces (APIs)).

How was Amazon's transition to using SOA an example of high level modularity at work?

The code base also becomes so large and interconnected that it is nearly impossible to fix or understand at once. Amazon decided to break theirs apart to make it more manageable. Because the services are only allowed to talk to each other using specified interfaces, you know that you won't break other things by mucking around in the implementation details of a particular module

Why was it important for Amazon to make this transition?

When a company gets a huge amount of traffic, it's just not feasible to keep everything under "one roof" because some parts of the code are being used constantly while the others are just causing bloat. The code base also becomes so large and interconnected that it is nearly impossible to fix or understand at once.

What are the three characteristics of good modules?

1. Low Coupling -- they should be minimally dependent on each other and communicate using specified interfaces.
2. High Cohesion -- they should be focused completely on achieving the overall goal.
3. High Encapsulation -- they shouldn't reveal their implementation details to anyone else.

What is Coupling?

Coupling is just the degree to which the components of a system rely on each other. It should make intuitive sense that you don't want elements of a system relying too heavily on each other because that sort of defeats the purpose of having different components in the first place. Modules should be minimally dependent on each other and communicate using specified interfaces

How much coupling should good modules have?

When thinking about modules, we encourage low coupling by forcing them to communicate with each other only by using specified interfaces. That prevents one module from getting too deeply involved in the workings of another.

What is Cohesion?

Cohesion is how closely the components of a system are working towards a common goal. A system with high cohesion has many highly specialized modules instead of a few big bloated ones that try to do too much.

How much cohesion should good systems of modules have?

In the real world, we embrace specialization because we know it provides efficiency gains. When thinking about modules in general, each one should stick with doing one thing well. Modules should be focused completely on achieving the overall goal

What is Encapsulation?

Encapsulation is the extent to which the implementation details are hidden. Usually, when you want something done in the real world, you don't care how it gets done.

How much encapsulation should good modules have?

I should be able to fully use a module purely by knowing how its interface works and nothing else.

What are the SOLID principles good for?

5 key engineering principles. They are guidlines for making a good modular software.

(optional) What are each of the SOLID principles?

1. Single Responsibility Principle (SRP) - modules should only exist to serve one purpose and may only change if that purpose is modified
2. Open/Closed Principle (OCP) - modules should be open for extension but closed to modification
3. Liskov Substitution Principle (LSP) - modules that inherit from a parent should not alter any of that parent's functionality
4. Interface Segregation Principle (ISP) - each different user of a module should get to access it via a specialized interface that only requires them to supply the minimal amount of information.
5. Dependency Inversion Principle (DIP) - Higher level modules should dictate the implementation details of lower level modules, not the other way around.