October 15, 2021

**Agile Software Methodology**

**Overview**

**Goals**

* Understand the underlying principles of Agile software development methodology
* Understand the core activities of an Agile software team
* Contrast Agile with other methodologies
* Learn about how to plan and document large software projects in an agile way

**Intro**

**Software Development Methodology**

Most simply, it’s a **way of building software**

* Helps to ensure productivity and high quality
* Fosters organized collaboration among a team of developers
* Provides a way of breaking down a large project into smaller sets of tasks

**Waterfall Methodology**

**What is Waterfall**

* Break down a project into a list of sequential steps
* Decide on a plan at the beginning
* Go through the plan until you’re done
* At the end, software is built, show it to the user

**Formal Steps**

1. Decide on system/software requirements
2. Analysis: business rules, data models-might have to use different software or programs
3. Design the software
4. Code the software-developers
5. Test the software-user acceptance testing(uat)/qa
6. Figure out maintenance, support, and installation-should be a part of initial plan. Make sure to think of the developer that comes behind you.

**Waterfall Pros**

* Easy to explain / understand-set written requirements.
* Saves time/money if you commit to design & requirements early. We don’t want to get in a situation where we’re going back and forth with business. Try to avoid a number of changes that happens within the middle of the development period.
* Allows for very strict deadlines (holds developers accountable). Own your code.

**Waterfall Cons**

* Doesn’t allow for flexibility. Set and go principle. As a developer you want to deliver your piece earlier.
* No input from users or customers until the very end
  + **A lot** can get lost in translation (uat department can circumvent all this)
* Fails to recognize that as you build software, **requirements can and need to be able to change.**
* “Overly regulated, planned, and micro-managed”

**Agile Development**

**The Agile Manifesto**

* **Individuals and interactions** over Processes and tools
* **Working software** over Comprehensive documentation
* **Customer collaboration** over Contract negotiation
* **Responding to change** over Following a plan

**Excerpt from the *Agile Manifesto***

The introduction of the manifesto reads:

*The Agile movement is not anti-methodology, in fact many of us want to restore credibility to the word methodology. We want to restore a balance. We embrace modeling, but not in order to file some diagram in a dusty corporate repository. We embrace documentation, but not hundreds of pages of never-maintained and rarely-used tomes. We plan, but recognize the limits of planning in a turbulent environment. Those who would brand proponents of XP or SCRUM or any of the other Agile Methodologies as “hackers” are ignorant of both the methodologies and the original definition of the term hacker.*

*—Jim Highsmith*

**However**

* This doesn’t mean “no planning”.
* It’s helpful to think about what Agile was a response to (Waterfall Methodology)
* It means: “build what’s needed, planning at the right time & level”

**Agile Principles**

* Daily **cooperation between client and developers**
* **Deliver frequently** (every day or week)
* **Working software** is the principal measure of progress
* **Sustainable development** at a constant pace
* **Simplicity** — the art of maximizing work not needed — is essential

**Formal Steps**

* **Step 1**: Decide on system/software requirements
* **Step 2**: Plan 2 weeks of work, or **sprint(about 2 weeks worth of work/ticketing system[Jira]) Done is when it’s ready to be released.**
* **Step 3**: Code and deliver a version of the product
* **Step 4**: Discuss how it went, reflect on requirements-(retro-how you contributed)
* **Step 5**: Repeat steps 2-4 many, many times…
* **Step ?:** Software is built!

**Pros of Agile**

* Allows for flexibility, end up with a better product (usually)
* More feedback from the user or customer
* Harder for developers to waste time building the wrong thing

**Cons of Agile**

* Ambiguity and uncertainty of deadlines is difficult for the business side
* Project Managers have to constantly be changing and re-prioritizing
* Requires some training and understanding of Agile before you dive in

**Agile Activities**

* Daily stand up or “scrum” meetings
* Pair programming
* Testing and Test-Driven Development
* Planning poker
* Velocity tracking

**Agile Project Management**

**Figure out requirements**

General term for

* **What** to build
  + e.g. “An app to automatically notify sales managers when a sales goal is achieved…””
* **Details** on how it should work
  + e.g. “Managers need to receive an email with the following information…””
* **Information** about how users will be able to use it
  + e.g. “Any employee that is logged into our company’s intranet will be able to access at [https://sales.coolcompany.com](https://sales.coolcompany.com/)”

**Make “User Stories”**

For example:

*As a user, I want to see the details of a movie, with a list of other users’ ratings and the output of my estimated rating (calculated in a separate story)*

**Use Relative Estimation**

* It’s **really hard** to know how long it takes to build software.
* Studies find we’re much better at “relative estimation”

“The movie detail page will take twice as long as the movie listing page.”

**Play Planning Poker**

* For each user story, everyone simultaneously reveals their estimate for how many “points” that story should get
* High and low outliers are asked to explain their votes
* A group decision is made (by consensus or vote)
* Record points for each story and for total of project

**Story Points**

“Story points” = “Relative units of effort”-time units(days/weeks)

**Plan Sprint**

* Once all the stories are documented and given a points estimate, figure out **2 week chunk** of work
* Project Manager usually assigns stories to developers

**Track Velocity**

As the team works on the project, revisit the estimations

* Over time, calculate *velocity*: effort-unit to developer-time
  + How much time does it take to complete one story-point’s worth of work?
* Often, the velocity is reasonably stable
* This allows you to better estimate what future phases will take

**Getting Started with Agile**

**- Don’t overbuild**

* *Keep it simple*
* *Don’t add style/design until you’ve learned what the feature needs*
* Keep track of your velocity
* Work in two-week sprints

**The End**

Retro meeting

What went well was the morning coding challenges.

What went wrong was github on the terminals on my computer.

What can be better? Afternoon labs challenges go from 0 to 100 quick. lol

Make sure retro is YOUR problems and ups.

**The brand of you:**

**Professional**