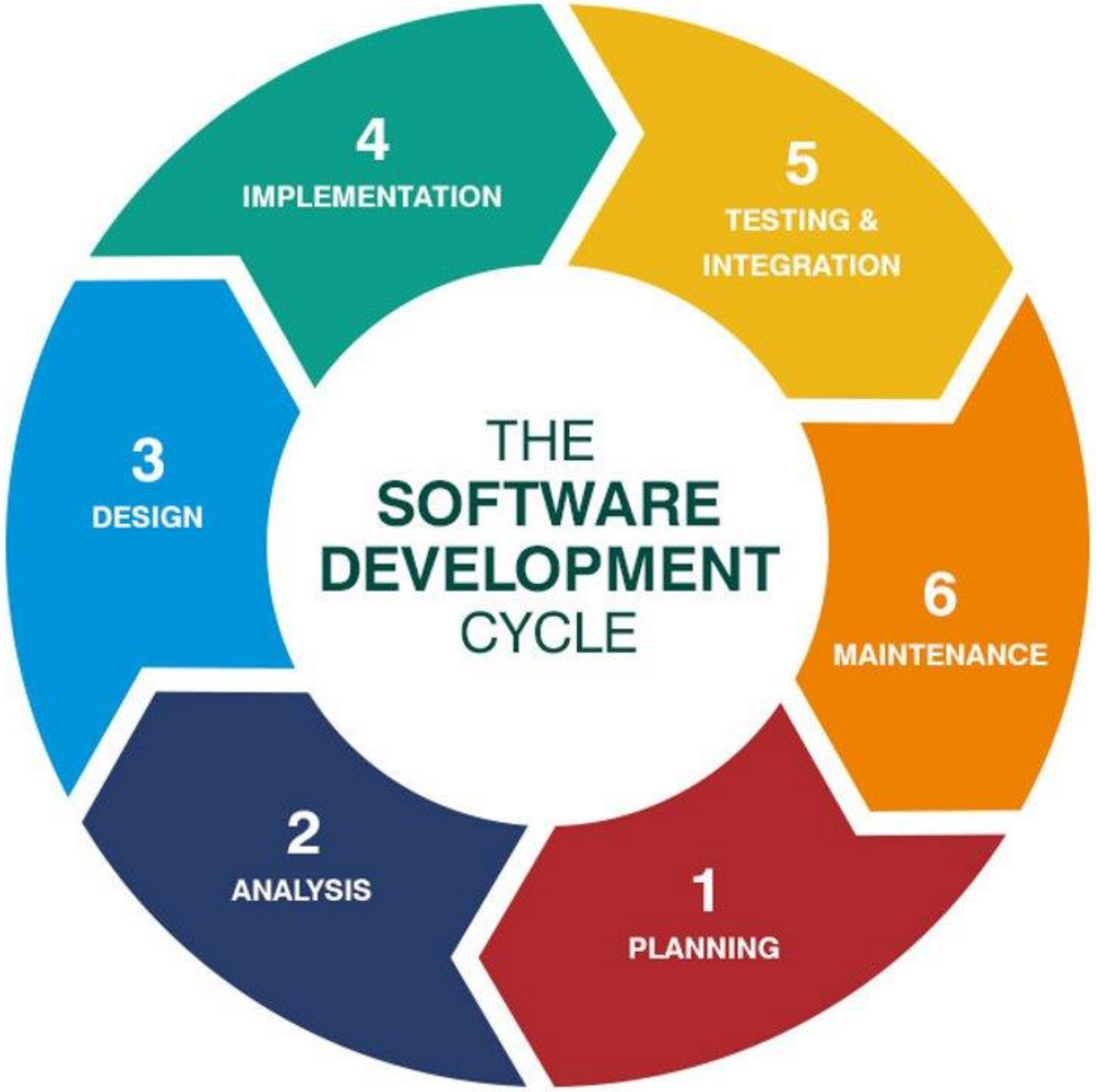
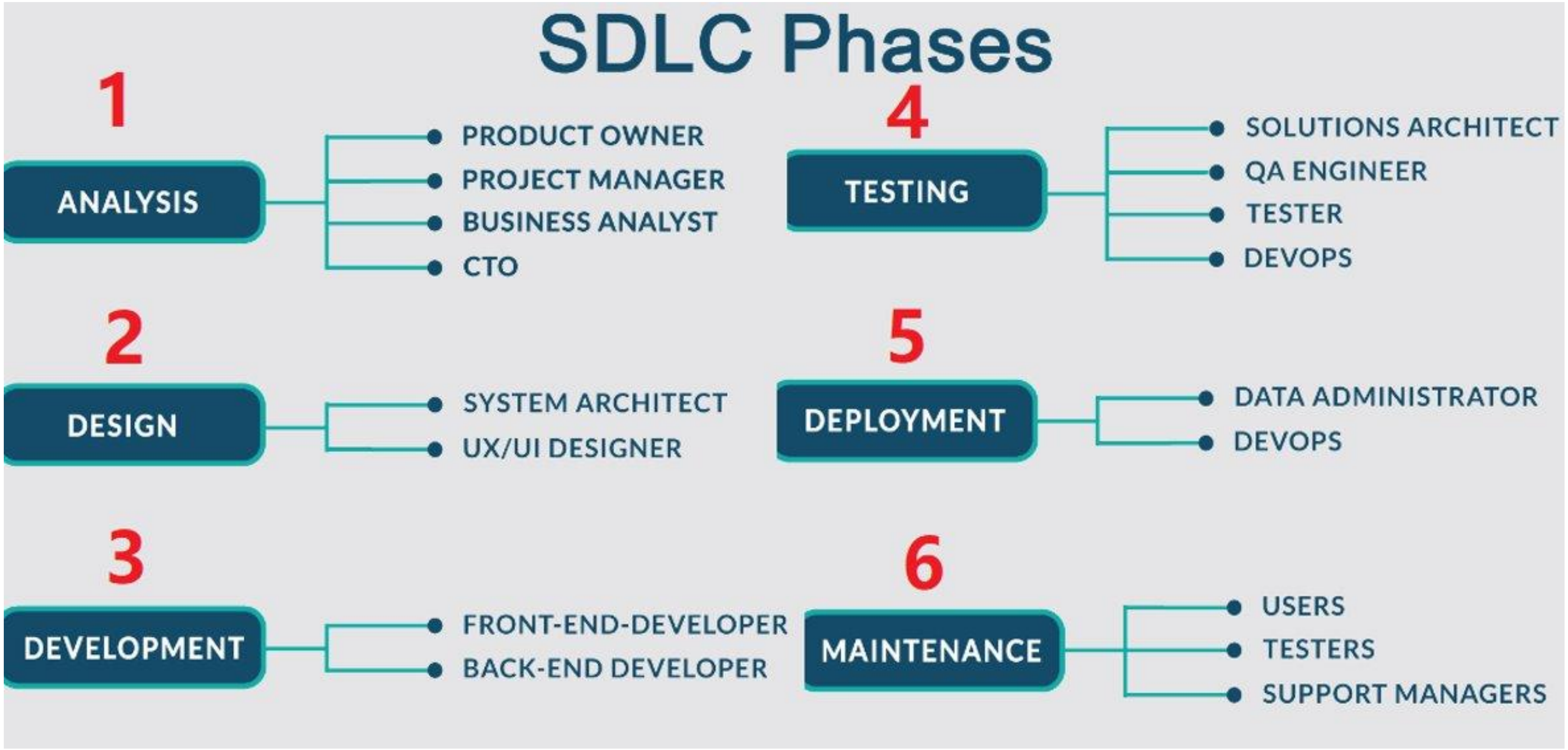
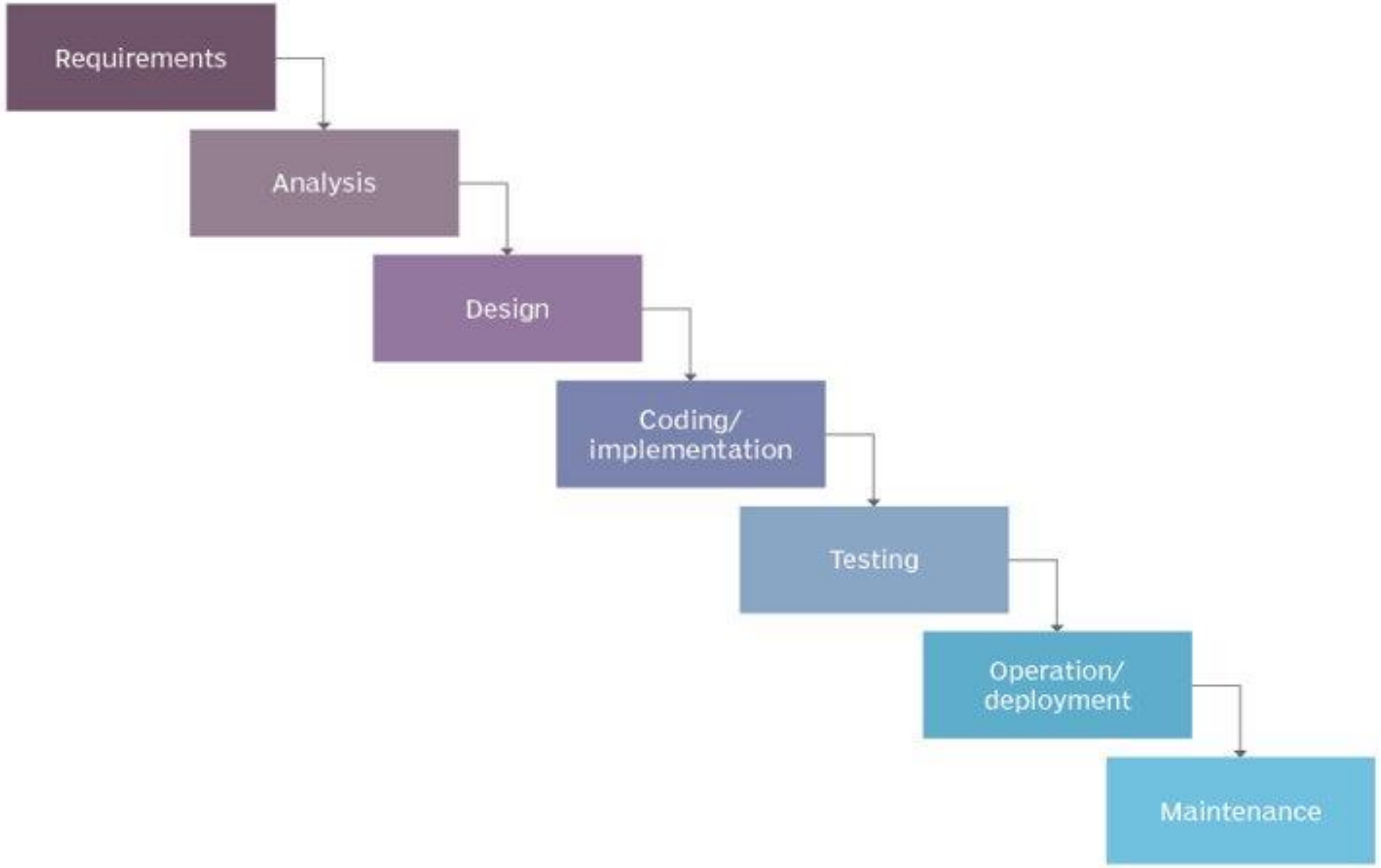
Agile

Let’s talk about SDLC and Waterfall


THE SOFTWARE DEVELOPMENT CYCLE
1 PLANNING
2 ANALYSIS
3 DESIGN
4 IMPLEMENTATION
5 TESTING & INTEGRATION
6 MAINTENANCE


SDLC Phases
1
ANALYSIS
PRODUCT OWNER
PROJECT MANAGER
BUSINESS ANALYST
CTO
2
DESIGN
SYSTEM ARCHITECT
UX/UI DESIGNER
3
DEVELOPMENT
FRONT-END-DEVELOPER
BACK-END DEVELOPER
4
TESTING
SOLUTIONS ARCHITECT
QA ENGINEER
TESTER
DEVOPS
5
DEPLOYMENT
DATA ADMINISTRATOR
DEVOPS
6
MAINTENANCE
USERS
TESTERS
SUPPORT MANAGERS

Waterfall model



Waterfall Characteristics

- **Sequential approach:** The development process is divided into sequential stages that follow each other.
- **Emphasis on documentation:** Each phase of the development process produces extensive documentation, including requirements specifications, design documents, and test plans.
- **Rigorous testing:** Testing is carried out at the end of each stage before moving to the next phase.

Waterfall Characteristics (Cont.)

- **Limited customer involvement**: The customer's involvement is limited to the requirements gathering stage, and they do not have any input into the development process until the product is completed.

- **Long development cycle**: The development cycle can be long due to the sequential approach, and any changes made later can cause delays and additional costs.

What is Agile

Agile Methodology

- "approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s). It advocates **adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages rapid and flexible response to change**."
- **An incremental development model** (software engineering activities interleaved in incremental versions)
- A philosophy of getting to work **as soon as possible** vs. big design up front

Agile Manifesto

- **Individuals and interactions** over processes and tools

  E.g. Alicia (informal communication) vs. GCS (formal / managed process)

- **Working software** over comprehensive documentation

  In most cases you can: write code / test it out / iterate with user feedback faster than authoring a design document that foresees and accounts for every scenario

- **Customer collaboration** over contract negotiation

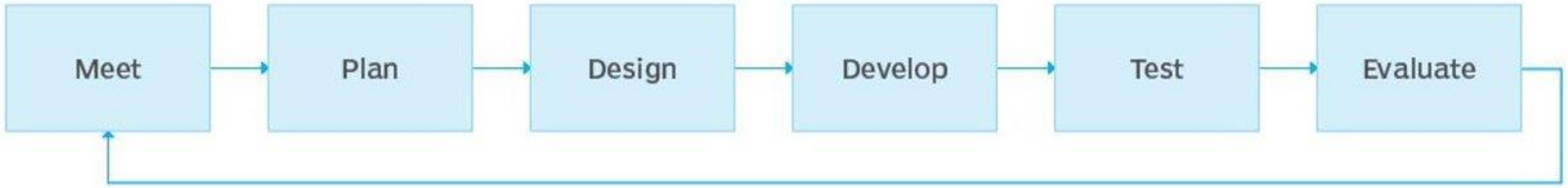
  E.g. developers from two companies interacting vs. business people defining requirements

  E.g. GCS, talking to them and finding out what they really need

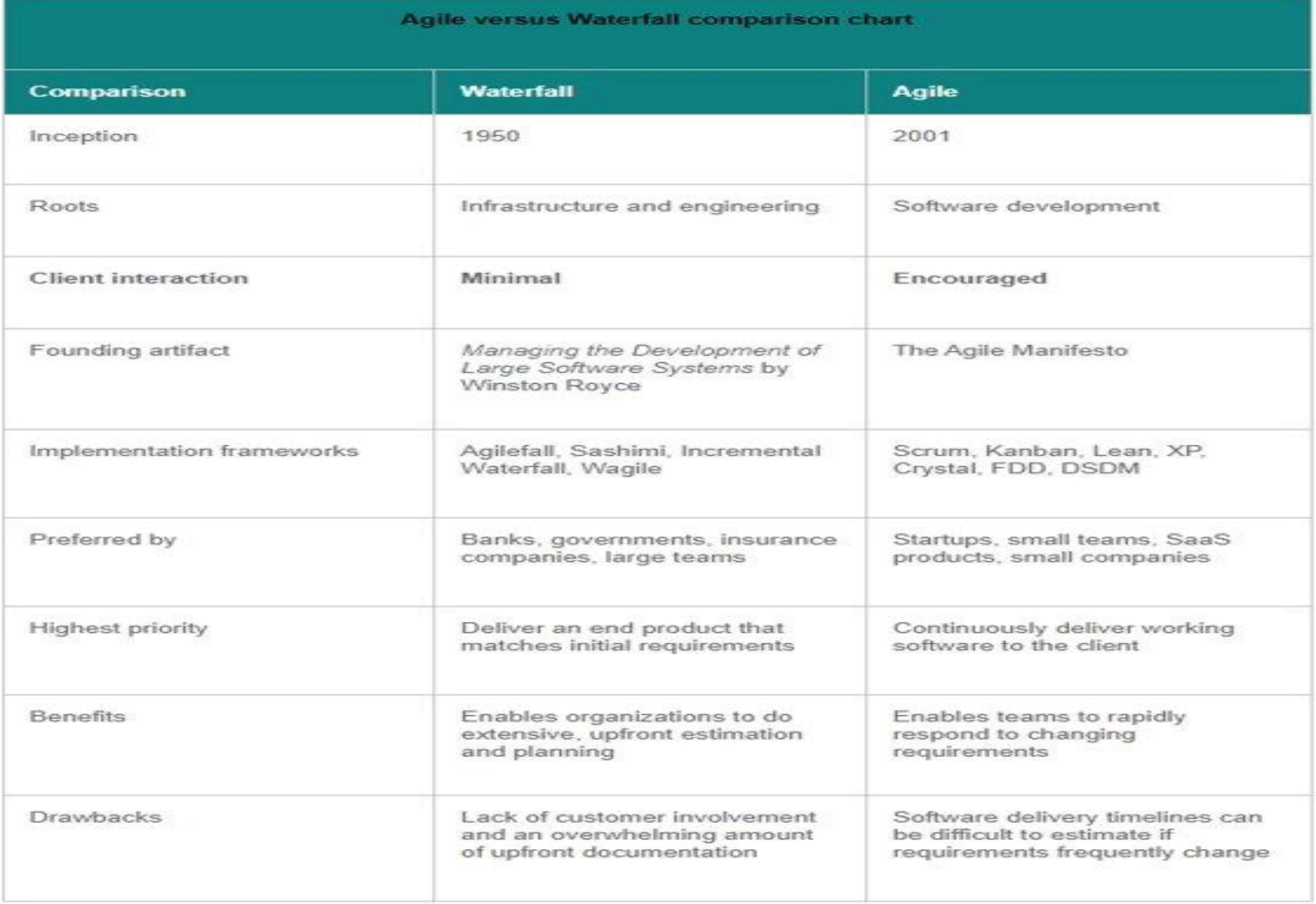
- **Responding to change** over following a plan

  Developers are BAD at planning and that’s never going to change. Own this by adapting to a rapid cycle

Agile software development cycle



Water fall vs Agile



Agile versus Waterfall comparison chart

Comparison	Waterfall	Agile
Inception	1950	2001
Roots	Infrastructure and engineering	Software development
Client interaction	**Minimal**	**Encouraged**
Founding artifact	*Managing the Development of Large Software Systems* by Winston Royce	The Agile Manifesto
Implementation frameworks	Agilefall, Sashimi, Incremental Waterfall, Wagile	Scrum, Kanban, Lean, XP, Crystal, FDD, DSDM
Preferred by	Banks, governments, insurance companies, large teams	Startups, small teams, SaaS products, small companies
Highest priority	Deliver an end product that matches initial requirements	Continuously deliver working software to the client
Benefits	Enables organizations to do extensive, upfront estimation and planning	Enables teams to rapidly respond to changing requirements
Drawbacks	Lack of customer involvement and an overwhelming amount of upfront documentation	Software delivery timelines can be difficult to estimate if requirements frequently change

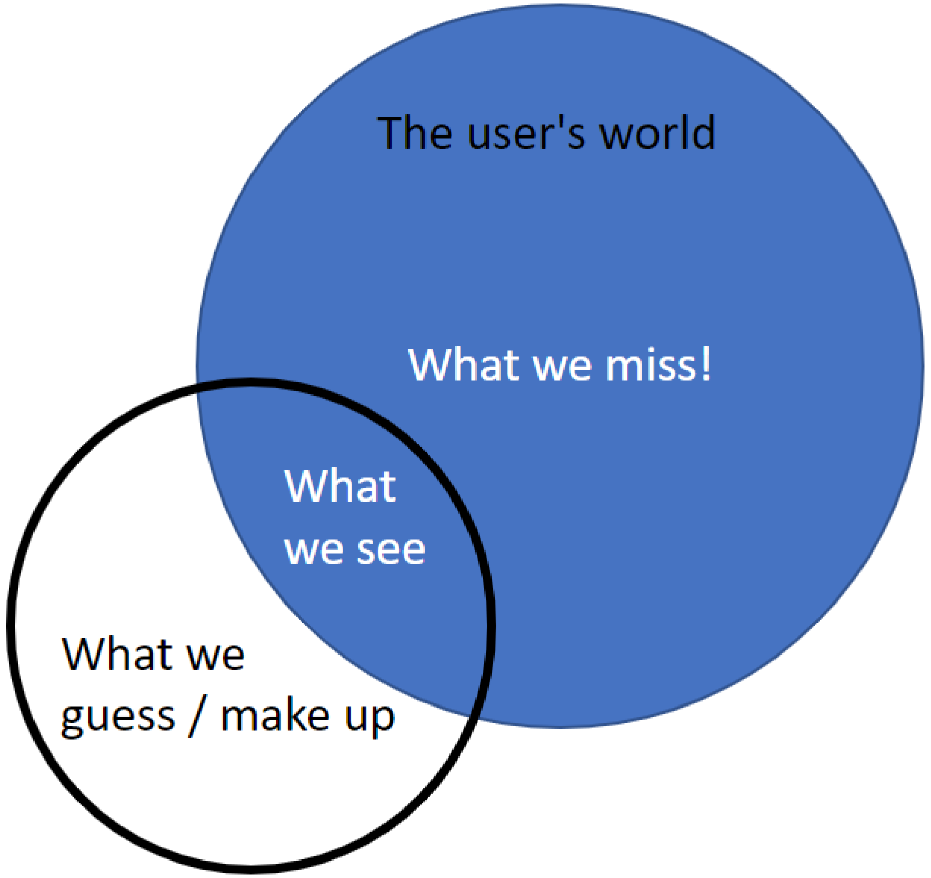
What is the reality?

Customer Collaboration (Involving Users In Development)

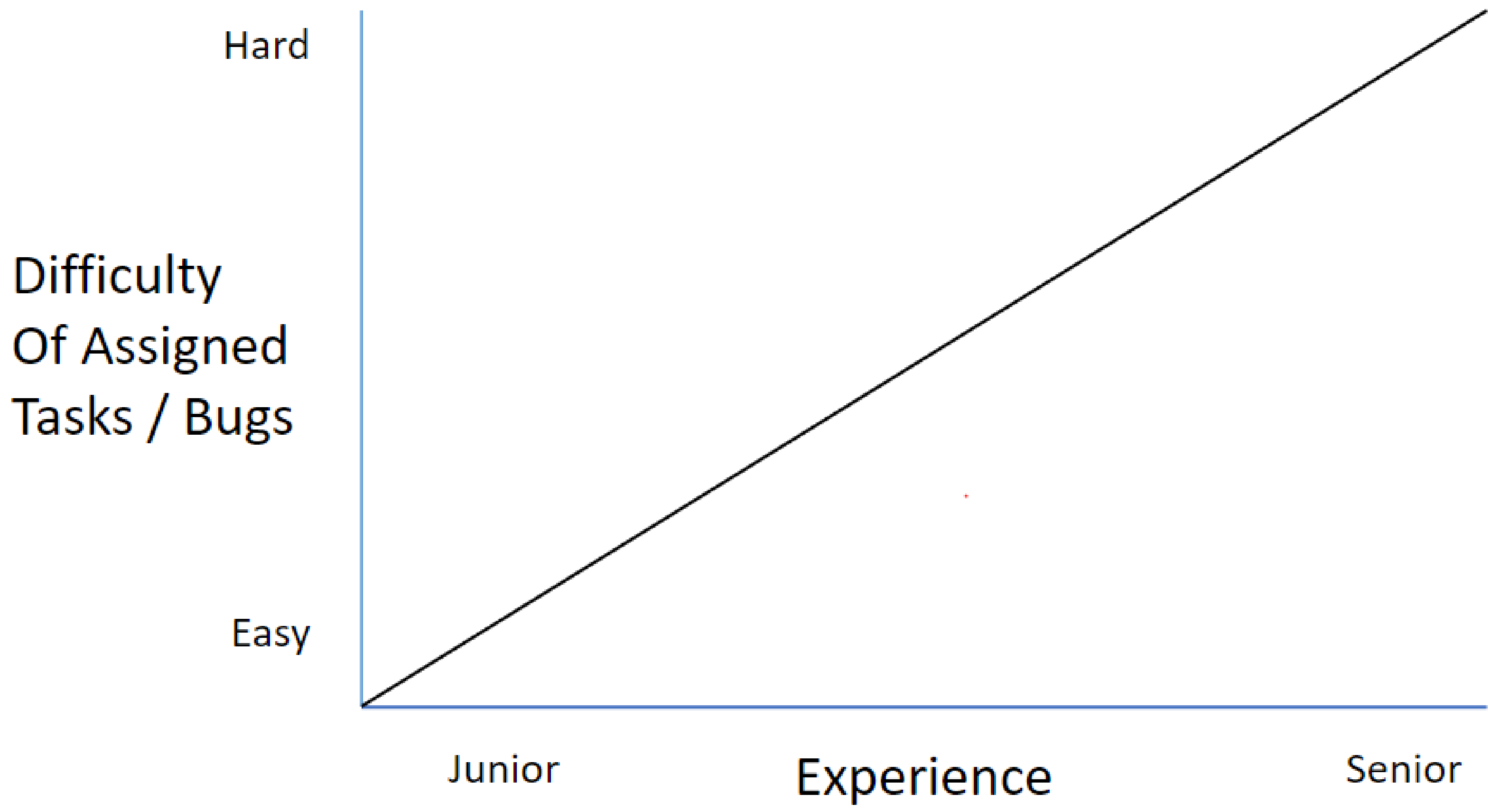
The User
a pretend person who will mould themselves to fit your system

vs.

Mary
a real person with real constraints trying to get her job done



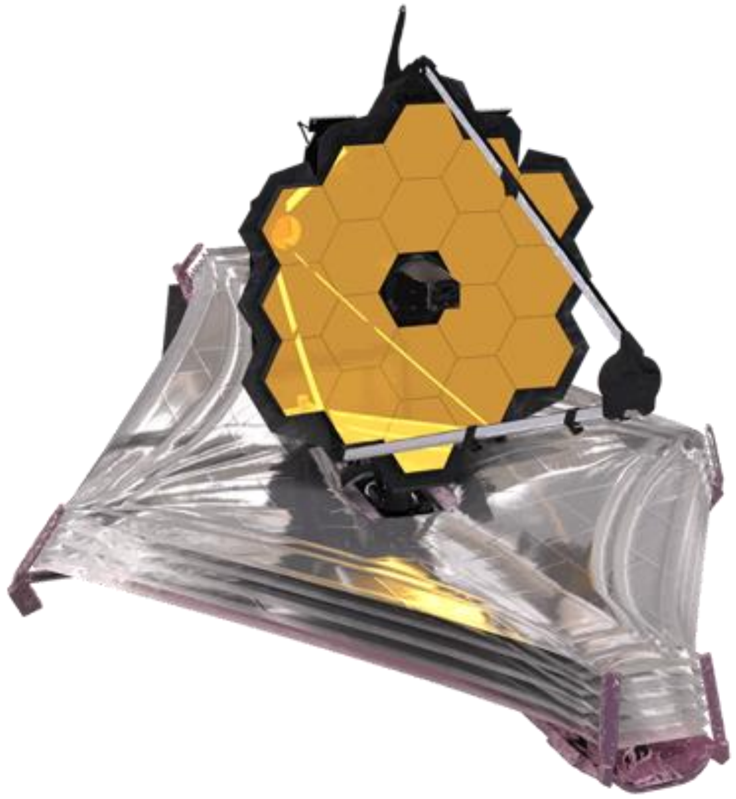
Why are programmers bad at estimating/planning?



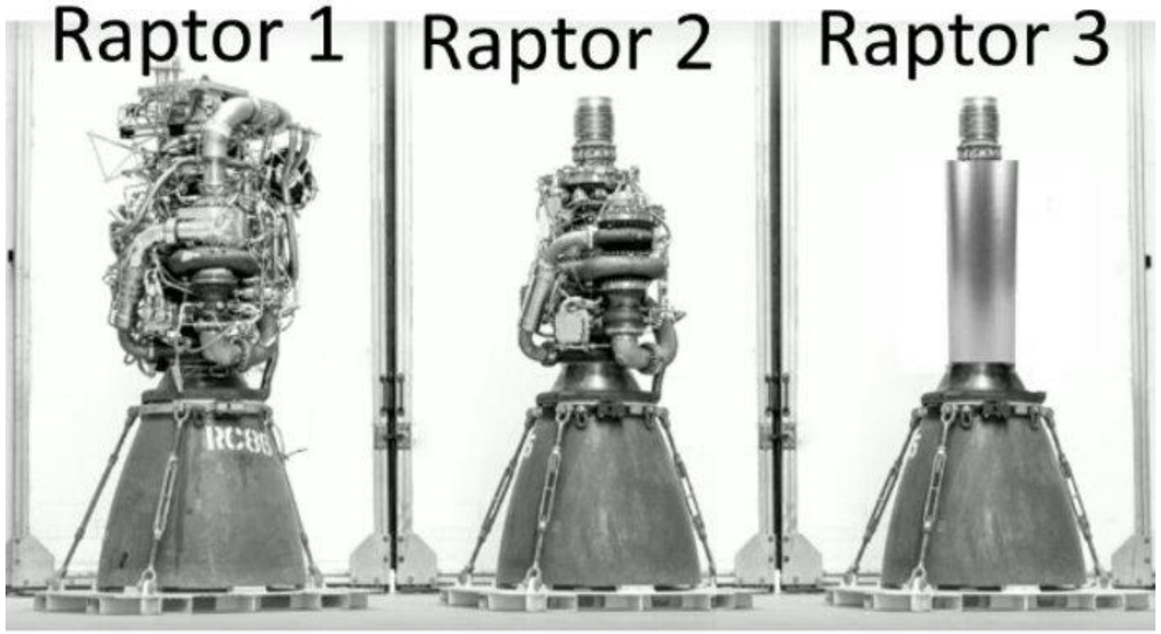
Unless you let your skills and abilities stagnate, you're usually exploring the unknown.

James Webb Space Telescope

- Waterfall: James Webb Space Telescope (so complicated and 0 error tolerance!)
- Over budget 1B -> 9B 900%!
- Over time 15 years Late



Start Ship and Raptor Engine



Twelve Principles of Agile Methodology

- **Customer satisfaction by early and continuous delivery of valuable software**
  a) Build trust with customers, bring them into the agile fold, help them understand why agile is better for their problem domain
  b) Show them what you've made and get feedback, the more you watch them use your software and involve them, the better the end product will be
  c) This must be managed, use QA to ensure you deliver high quality incremental change Manage customer expectations, predict freak outs and head them off, teach them about agile
- **Welcome changing requirements, even late in development**
  a) By adopting this philosophy and making it front of mind you prevent yourself from going down paths you can't easily come back from

Twelve Principles of Agile Methodology

- **Working software is delivered frequently (weeks rather than months)**
  a) The longer you delay delivering, the less you are receiving and integrating customer feedback
  b) Without comprehensive design documents you need to show progress before it's too late to change direction

- **Close, daily cooperation between "customers" and developers**
  a) Without a comprehensive all-encompassing design document customers and business people need to stay involved on a daily basis with developers to ensure developers are developing in the right direction, usually this involves some kind of product manager

Twelve Principles of Agile Methodology

- **Projects are built around motivated individuals, who should be trusted**
    a) Again, without the “mega design document” you must trust the implementation to the individuals on the team, trusting them to deliver what you ask for and to integrate your feedback
    b) Choose the right person for each job, assist motivation

- **Face-to-face conversation is the best form of communication (co-location)**
    a) Writing emails / design documents is a lot of work often done in isolation
    b) One person can go down a long path writing a design document with assumptions that could be cleared up very quickly by talking to the customer / business people
    c) Two developers talking to each other solve problems much quicker than written negotiations

Twelve Principles of Agile Methodology

- **Working software is the primary measure of progress**
  a) Maintain your trust with the customer by delivering working software
  b) How do you ensure quality / working? Testing and good programming practices (CI/CD)

- **Sustainable development, able to maintain a constant pace**
  a) Don’t allow sprint overloading (remember the problem with time estimates?)
  b) Don’t over commit
  c) Crazy amounts of overtime teach customers to expect more than you can deliver over the long term

Twelve Principles of Agile Methodology

- **Continuous attention to technical excellence and good design**
  a) Building things correctly the first time, learning from past mistakes.
  b) Choose the right skills, tools, and practices ,Writing clean, readable, extensible and maintainable code. This means following good coding practices.
  c) Good design reduces the risk of technical debt and rework as it is easy to understand, maintain, and modify.

- **Simplicity (the art of maximizing the amount of work not done) is essential**
  a) This contributes to the speed you can deliver code and get feedback
  b) Simplicity always means less rigid / easily changed code
  c) spaceX Raptor Engine (Picture!)

Twelve Principles of Agile Methodology

- **Best architectures, requirements, and designs emerge from self-organizing teams**

    a) It's based on the belief that the people doing the work are the best ones to decide how the work should be done.

    a) When the team is small and focused, better work gets done:

        a) less noise
        b) less miscommunication / communication overhead
        c) less interruption / contribution from unqualified people
        d) fewer meetings!

Twelve Principles of Agile Methodology

- **Regularly, the team reflects on how to become more effective, and adjusts accordingly**
  a) By regularly examining their work and making necessary adjustments, Agile teams can become more efficient, effective, and capable of delivering high-quality software that meets the needs of their customers.
  b) Impediments to agility identified and eliminated
  c) Not doing this work is usually the underlying cause of agile done wrong

Implementation of the Agile

- **Scrum:** A framework for managing and completing software development projects, where work is organized into short sprints of 1-4 weeks.

- **Continuous Integration:** A process where developers integrate their code changes into a shared repository frequently, typically several times a day, to prevent integration problems and detect issues early.

Implementation of the Agile

- **Test-Driven Development (TDD):** A development process where tests are written before the code is developed, with the aim of ensuring that the code meets the required specifications.

- **Pair Programming**: A practice where two developers work together on the same code, with one typing and the other reviewing and providing feedback.

Problems With Agile

- Informality of agile is incompatible with the legal approach to contract definition that is commonly used in large companies.
    - Have to rely on contracts that pay for time. If problems arise, then so do disputes over who pays for increased time.

- Agile methods are most appropriate for new development, not maintenance.
    - In large companies maintenance is the majority of development, kanban more appropriate

Problems With Agile

- Agile methods designed for small co-located teams, yet much software development now involves worldwide distributed teams.

- Lack of product documentation combined with development team discontinuity. Agile devs argue that the code is the documentation, however most companies do not track requirements over the long term so what the code is supposed to do is lost. Especially so if original developers move on.
    - Why were decisions made to implement the code in that way?
    - What constraints become unknown

How You Will Apply Agile To Your Projects:

- SCRUM
- Stories / Backlog
  - Jira
- Continuous Integration
- Refactoring (best way to implement / integrate change)
  - Throughout the course you will learn best practices, you will need to refactor your code to incorporate those best practices
- Test driven development
  - Your business logic must be 100% covered by unit tests
  - To achieve this you will need to mock your data layer

How You Will Apply Agile To Your Projects:

- Work is broken into “sprints” (usually 2 weeks)
- All work done is broken into:
  - a) Epics (large features of a system)
  - b) Stories (sub-features / requirements of a system)
  - c) Sub-Tasks (work that needs to be done to implement a story)
- Stories are organized in a backlog in order of priority
- Roles:
  - a) Product Owner (the person who knows how the thing should work, the priority of stories, communicates progress to the business)
  - b) Development team (the people who do the work)
  - c) Scrum master:
    1. Ensures the scrum process is followed
    2. Removes impediments to agility

Scrum (Continued…)

- Sprint planning:
  - Decide on roles
  - Load the sprint with stories:
    - Lots of debate on how to estimate how much work will fit in the sprint
    - Story points
    - Time estimates
    - Continuous analysis of story point / time estimates to actuals (burn down, burn up, velocity, blah blah blah)
  - Assign initial tasks
- Daily scrum (standup):
  - Make sure you're standing
  - 3 questions:
    - What did I do yesterday? What will I do today? Impediments?
  - DO NOT allow distraction / time wasting

Scrum (Continued…)

- Sprint Review and Retrospective:
  - Review completed work
  - Present completed work to stakeholders / get feedback
  - Reflect on the past sprint, discuss and agree on continuous improvements to the process
  - What went well?
  - What could be improved?
  - USE this information, don’t just collect it. ("The Friday Afternoon" problem)
- Backlog Grooming:
  - Reviewing stories in the backlog to make sure they’re still needed, still have valid information, and are prioritized properly
  - Usually where you create stories to address technical debt

Who Did What?

- Who’s worked in the industry?
- What software development methodology did you use?
- What were the benefits to your method?
- What were the problems with your method?
- Other thoughts?