CYBERINFRASTRUCTURE
INTEGRATION RESEARCH CENTER
PERVASIVE TECHNOLOGY INSTITUTE

Continuous Integration/Deployment

February 1st 2022

Suresh Marru

Software Engineering Best Practice Checklist

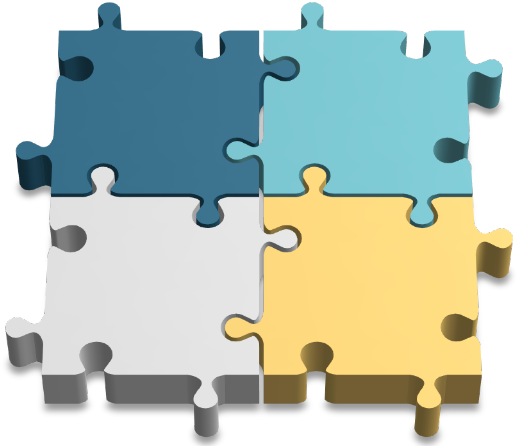
1. We have our code in a version control system that meets our needs.
2. We associate all commits with issues.
3. We have at least two main branches for all of our code: develop and release.
4. We have code reviews for all code committed to our main branches.
5. We have a build system that we regularly execute.
6. Our build system includes unit tests.
7. We fix broken builds, including test failures as well as compilation failures.
8. We use code analysis tools to help find problems such as DRY code, code that is never used, code that is too complex, inadequate unit test coverage, etc.
9. We remove obsolete code.
10. Our integration and deployment scripts are in our code base.

Continuous Integration

Continuous integration is a development practice that requires developers to integrate code into a shared repository on a daily basis.

Each check-in is validated by

- An automated build
- Automated unit, integration and acceptance tests

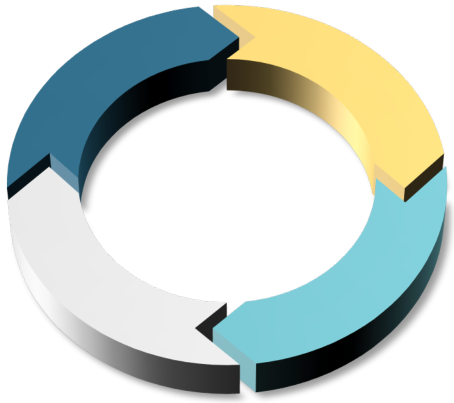


Integrating regularly in production-like environments makes it easier to quickly detect and locate conflicts and errors.

Continuous Delivery (1)

Continuous delivery is a methodology that focuses on making sure software is always in a releasable state throughout its lifecycle.

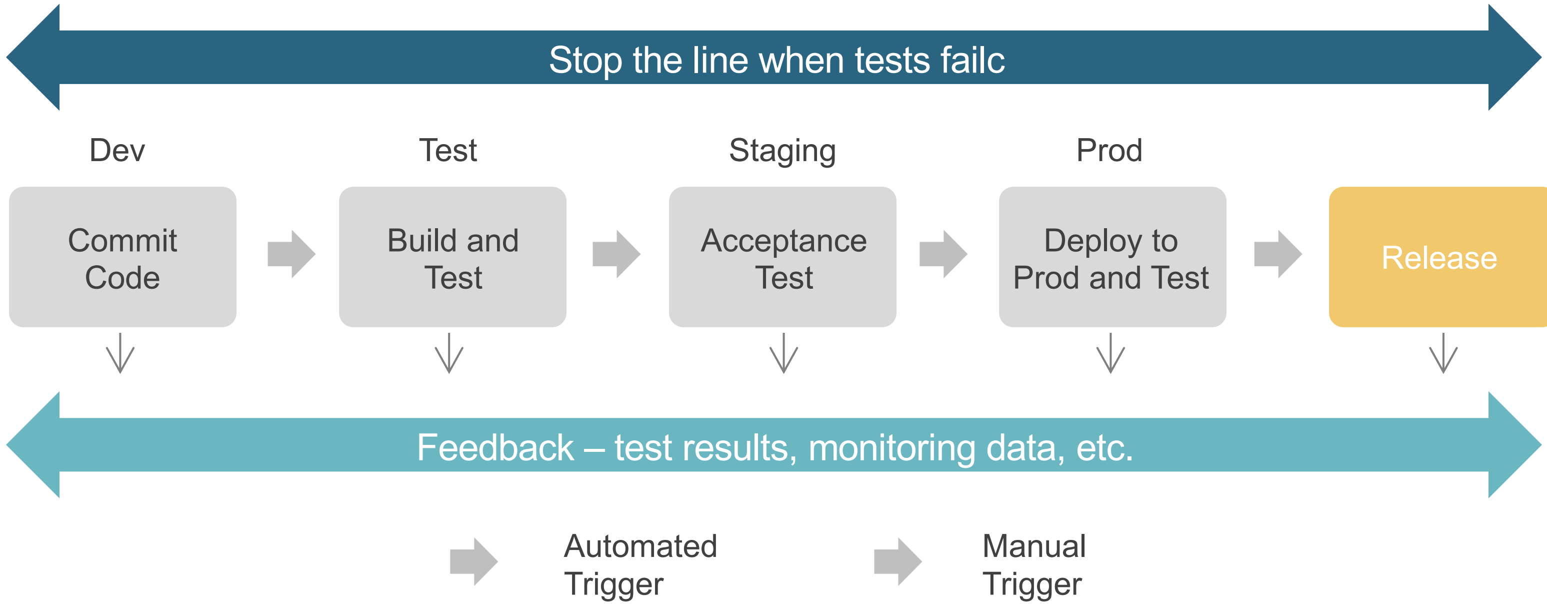
- Extends continuous integration
- Provides fact, automated feedback on the production-readiness of systems
- Prioritizes keeping software deployable over working on new features
- Enables push-button deployments on demand
- Reduces deployment risks and enables quicker user feedback



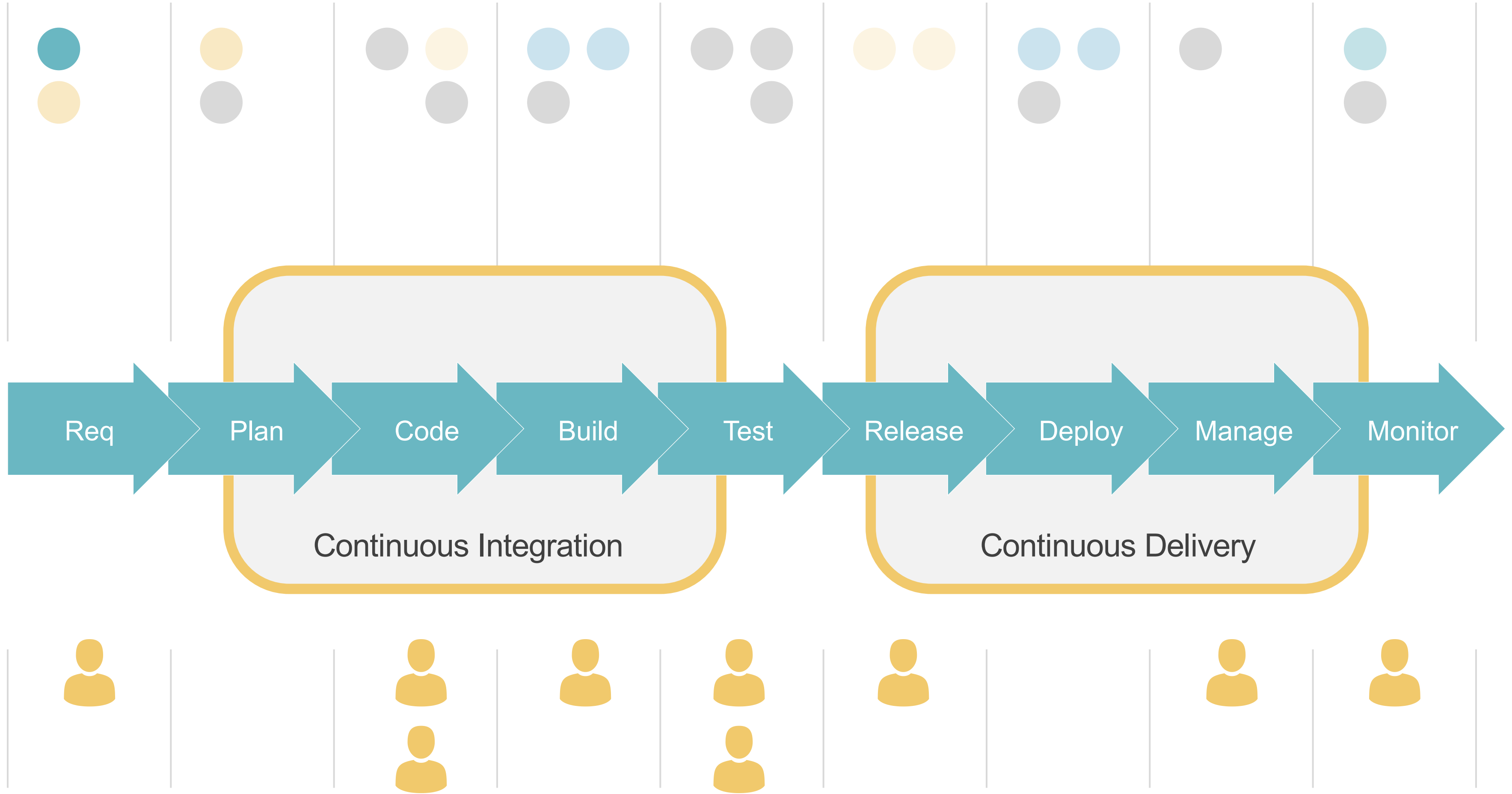
Integrating regularly in production-like environments makes it easier to quickly detect and locate conflicts and errors.

Continuous Delivery (2)

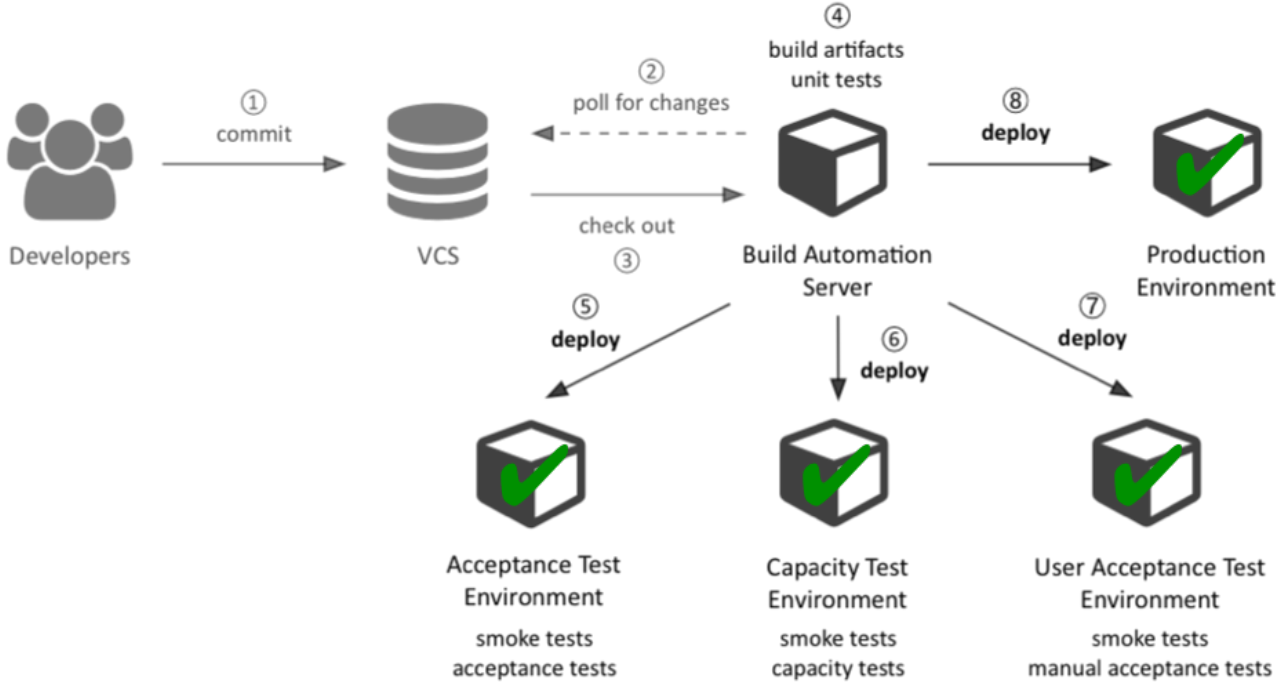
Automated tests in production-like environments assure the code and environment operation as designed and are always in a deployable state.



Deployment is the installation of a specified version of software to a given environment (e.g., promoting a new build into production).


Req
Plan
Code
Build
Test
Release
Deploy
Manage
Monitor
Continuous Integration
Continuous Delivery

Robots and Virtual Machines Will Save You



Continuous Integration Practices

- Have a defined build process
- Automate the build
- Make your build self-testing
- Every commit to the “integration” branch should build on an integration machine
- Keep the build fast
- Test in a clone of the production environment
- Make it easy for anyone to get the latest executable
- Everyone can see what’s happening
- Automate deployment

No Excuses, Plus There Are Tools!

- With containers and virtual machines, we are running out of excuses for having irreproducible deployment environments.
- There is a substantial array of tools to help you automate your integration and deployment environments.
- The best ones provide “Infrastructure as Code”

Operations Best Practices Checklist

1. We have a continuous integration system that mimics our deployment environment
2. We have automated the creation of integration and deployment environments
3. We don’t need to log into our test and production systems to update them.
4. We have integration and system tests that we use regularly, trust and rely on.
5. Our integration and systems tests run automatically (tied to commits or else run as cron jobs)
6. We apply systems tests regularly to our production system.
7. We debug most or all of our production system’s problems by inspecting logs.
8. We don’t need to log into our production system’s server hosts to inspect logs.
9. We have monitoring in place to detect failed services.
10. We have documented steps for moving our services to new deployment hosts

A Problem Solving Machine

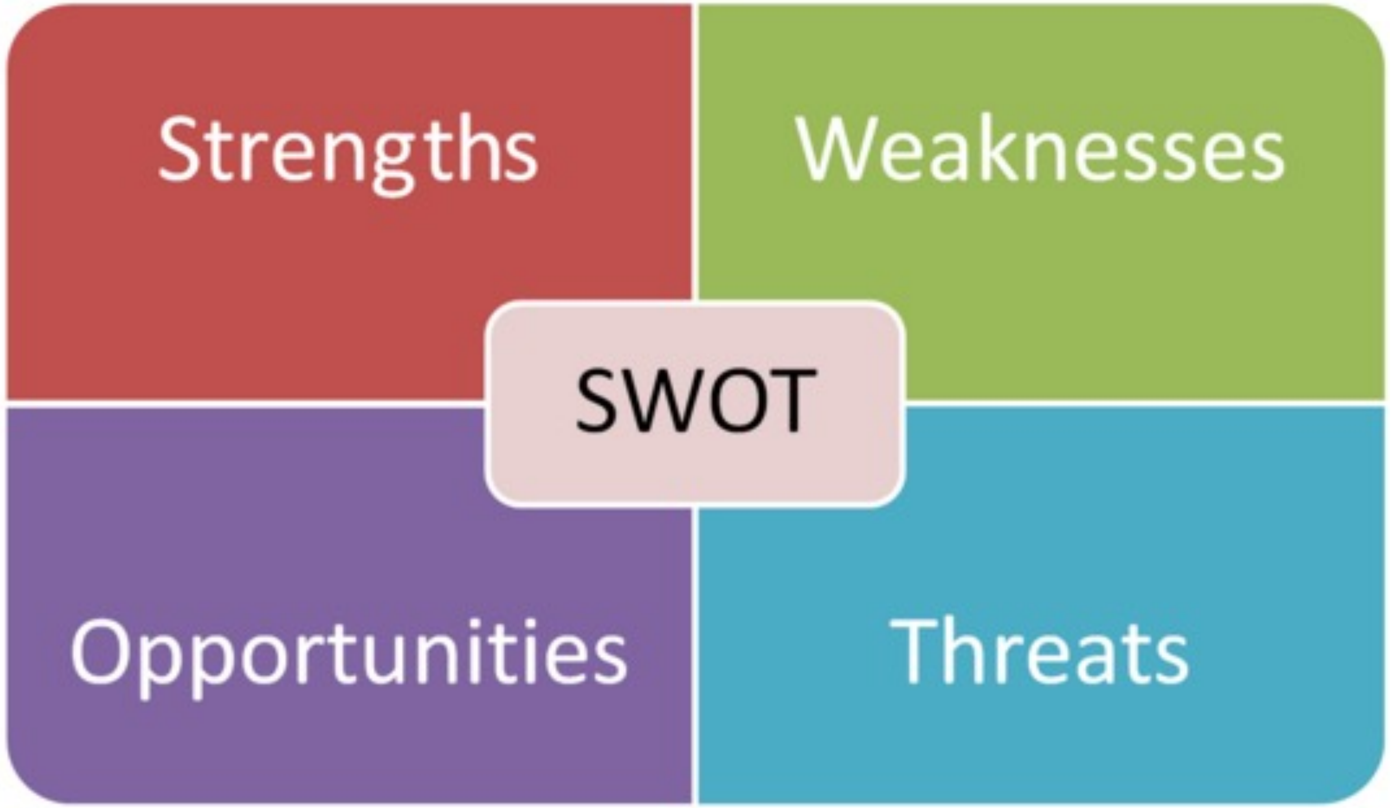
- Having a technical architecture for your system will guide you at all stages and help identify “hot spots” that need improvement
- Map your value proposition into your system.
- Choose technologies, frameworks
- Identify “hot spots” in your system: where are you spending too much effort? Where are you not spending enough effort?

Another Problem Solving Machine

- Reviewing your architecture with knowledgeable peers is a great way to identify problems and find solutions.

Expected Outcomes

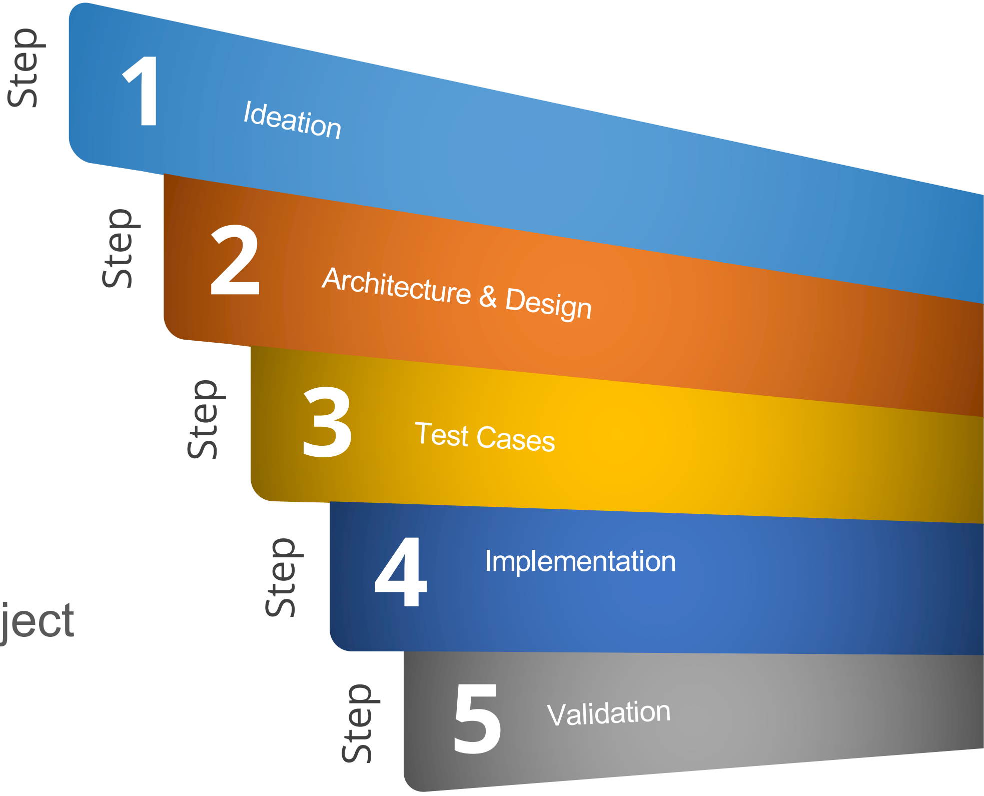
- Develop an architecture that maps you capabilities and implementations to your value proposition.
- Identify what you build and what you buy
- Review this with a peer group.
- Identify “hot spots” in your architecture
  - Introspection, peer review
  - SWOT analysis


Strengths
Weaknesses
SWOT
Opportunities
Threats

Project Life Cycle

Break every project milestone into:

5 Steps

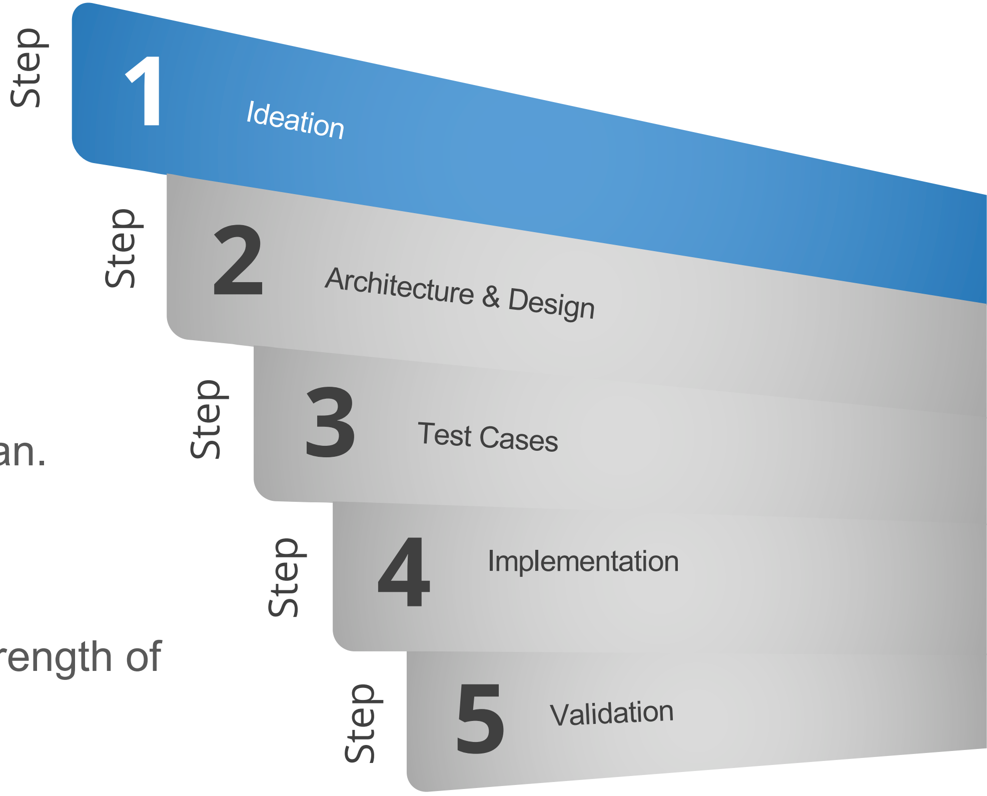


All these artifacts need to be in your project repository and will be peer-reviewed

Project Idea

01

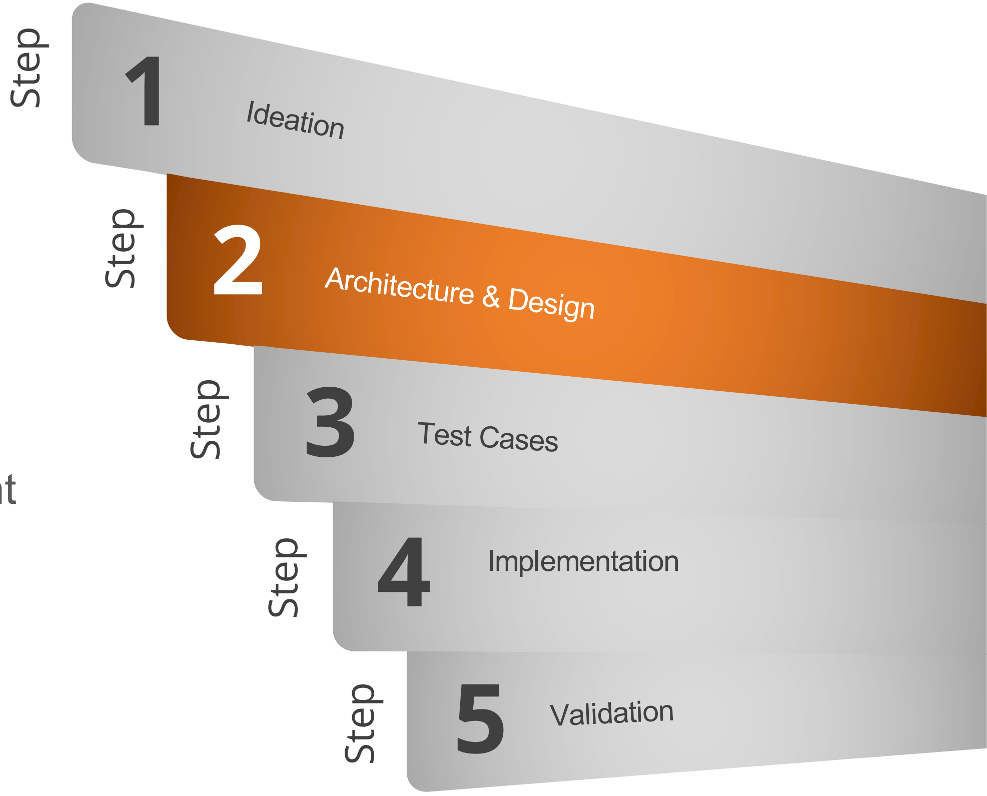
- Articulate your idea as a user story.
  - Describe as many variations as you can.
  - Do not yet describe how the system accomplishes them.
- Draw a napkin drawing of your idea.
- Articulate the value proposition (central strength of your project).



White Board Architecting

02

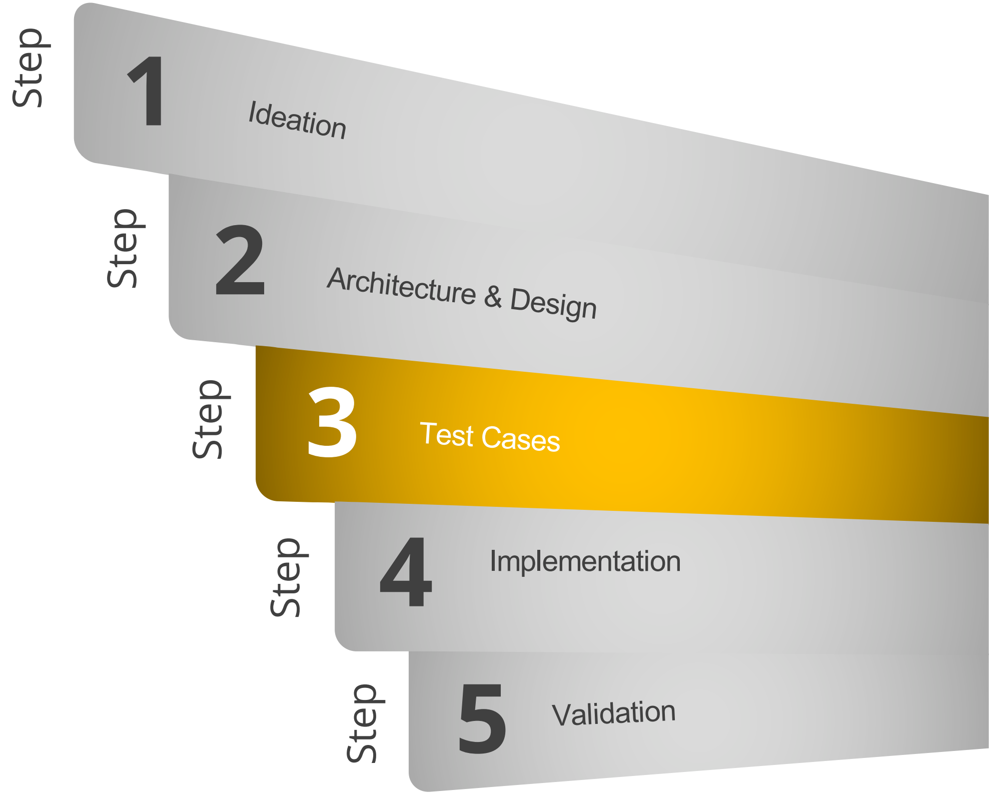
- Draw an architecture on how you will accomplish the project.
- Do not yet discuss how you will implement it.
- Identify the basic components needed.
- Leave out the implementation details.
- Identify the components which are your unique contributions (value propositions).



Test Cases – Usage Scenarios

03

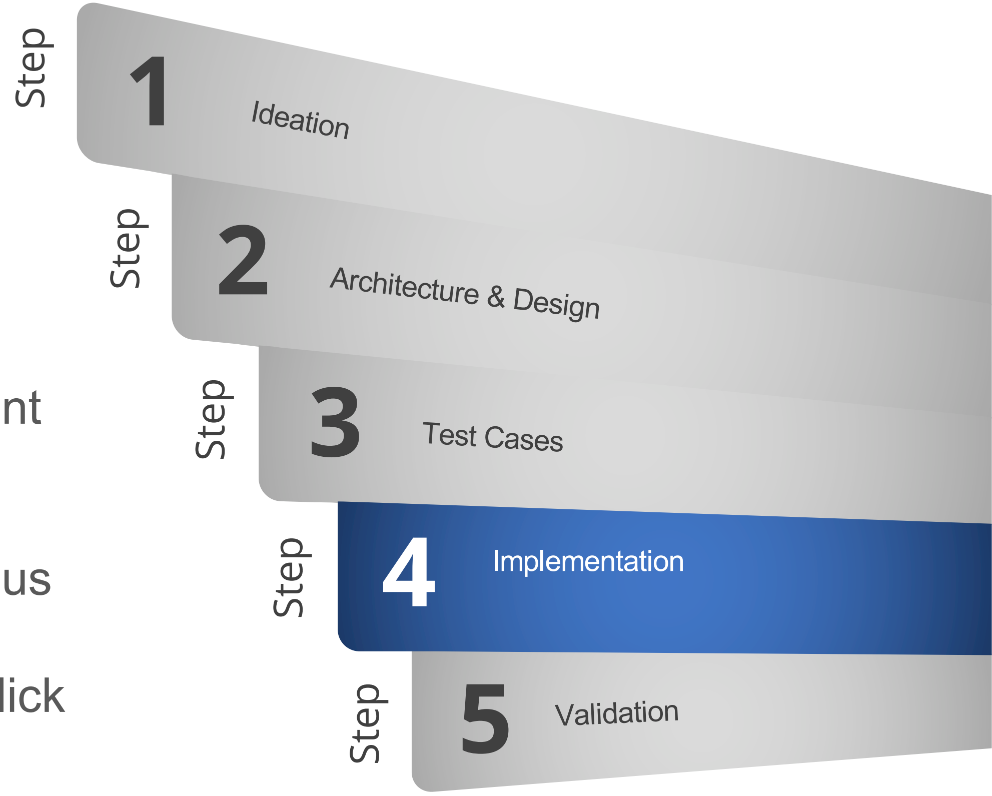
- Write usage scenarios as test cases
- Think about programmability of the test cases



Implement the Architecture

04

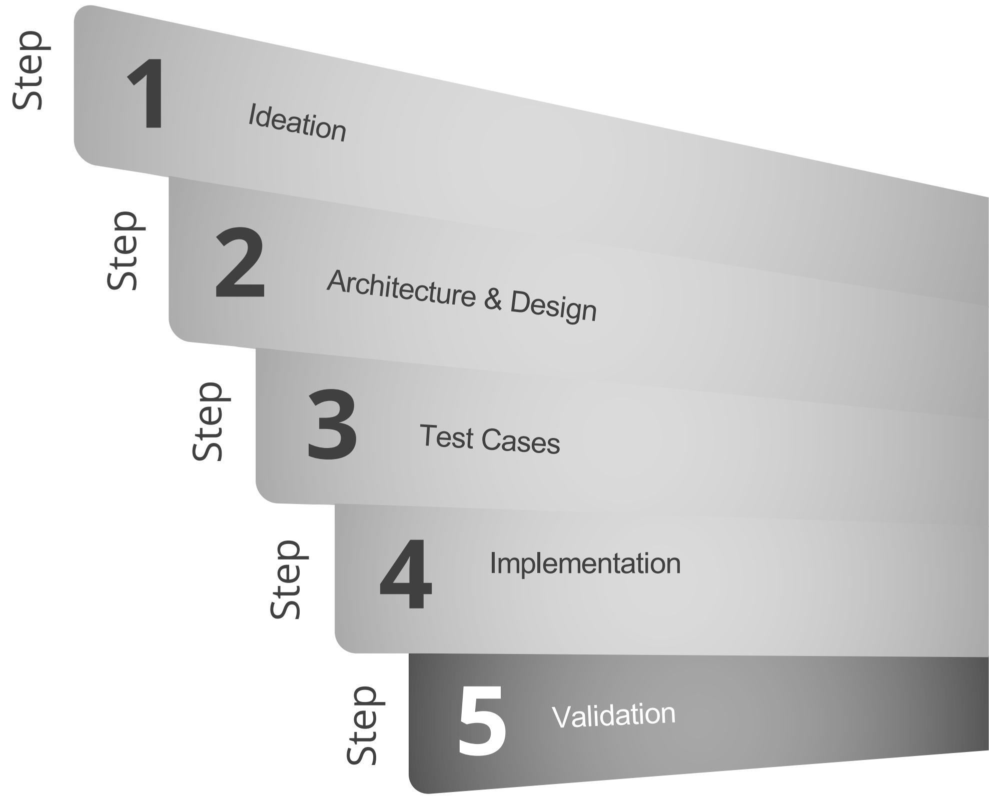
- Implement the architecture in 3 different programming languages.
- Use a build system.
- Configure Travis-CI/CircleCI Continuous Integration.
- Use Ansible like tools to have a one-click Deployment.



Validate the system

05

- Run through test cases
- Peer-reviewers will need to identify more testing scenarios.
- Rinse and Repeat


Step
1
Ideation
Step
2
Architecture & Design
Step
3
Test Cases
Step
4
Implementation
Step
5
Validation