Social Coding

- → Open source practice
- → All repositories are public
- → Everyone is encouraged to contribute
- → Branch is controlled via Pull Requests.

Code reuse dilemma

- → code has 80% of what you need but 20% is missing.
- → How do you add 20% missing features!
- → Make a feature request and depend on another team?
- → Rebuild 100% of what you need (no dependencies).

Social coding solution:

- → Discuss with repo owner
  - → Agree to develop
  - → open issue & assign it to yourself
  - → fork & make changes
  - → Issue pull request to review & merge back.

Pair Programming:

- → Two programmers on one workstation
- → driver is typing
- → navigator reviews
- → every 20 min they switch.

benifits

- → higher code quality
- → defects found earlier
- → lower maintainence costs
- → skills transfer
- → Two set of eyes on every line of code.

Minimum Viable Product (MVP)

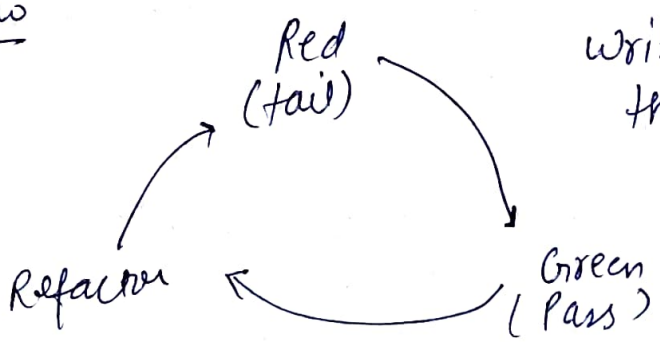
→ MVP is not phase (1) of a product.
→ MVP is an experiment to test your value hypothesis and learn
→ MVP is focused on learning, not delivery.
→ At the end of each MVP, you decide whether to pivot or presevere.

- → MVP is a tool for learning
- → The experiment may fail & that's okay
- → Failure leads to understanding
- → what did you learn from it?
- → what will you do differently?

Test Driven Development:-

→ Test cases drive the design
→ You write tests first → write code to make test pass
→ Keeps you focused on the purpose of code.

TDD workflow



→ Saves time
→ code faster & with confidence
→ future changes won't break code

Behaviour Driven Development (BDD)

→ Describes the behaviour of system from outside.
→ Great for integration testing
→ Uses a syntax both devs & stakeholders can easily understand.

BDD ensures that you are building the "right thing".
TDD ensures that you are building the "thing right".

Gherkin

→ An easy to read structural language syntax.
→ Given .... when .... then
→ understandable by everyone.

Gherkin Syntax

→ Given (some context)
→ When (some event happens)
→ Then (some testable outcome)
→ And (more context, events or outcomes).

Benefits of BDD

→ Improves communication
→ More precise guidance
→ Provides a common syntax
→ Self-documenting
→ Higher code quality
→ Acceptance criteria for user stories.

Think Cloud Native

→ The 12 factor app
  → A collection of stateless microservices
  → Each service maintains its own database
  → Resilience through horizontal scaling.
  → failing instances are killed & spawned
→ CD of services.

Failure Happens

→ Embrace failures - they will happen
→ MTTF → MTTR (changing evaluation metrices)
→ How to avoid → how to identify & what to do with it
→ operational concern → developer concern.
→ Plan to be throttled

Patterns that make applications resilient:

→ Retry Pattern
→ Circuit breaker pattern
→ Bulkhead Pattern
→ Chaos Engineering