Influences of TDD on Agility & Sustainability

What's Agile Development?

- One word: FEEDBACK.
- We develop something, have some people play around with it, get feedback, and then improve our implementation code.
- **Feedback Loop helps us to create relevant working software.**
- **If your objective is to build what your customers *wanted*, you will fail. You need to build what they *still want*.**
- It isn't about speed, but about sustainable speed.

Cost of doing TDD

There are two types of costs.

- Cost of Testing
  - Takes time to write tests. Hard to quantify the cost.
  - If we make it a practice to write the code and write test, it becomes a very natural process to write and test.
  - **Even though it takes extra time to develop code with automated tests, in the long run we benefit (takes less time to change or extend).**
  - Do you want local optimisation or global optimisation?
    - Local optimisation → Do you just want to reduce the programming cost?
    - Global optimisation → Do you wan to reduce the overall cost of developing and maintaining the software?
- Cost of Learning

    - As we (programmers) become more experienced, it becomes harder to write automated test cases.
    - To do TDD, there's an active part of unlearning.

Cost of not doing it — insanity to repeat what does not work

- The cost of not doing it is enormous.

Dealing with Entropy and Software Rot

- Software always deals with entropy. Things tend to fall apart, things change, evolve, things that once worked don't work anymore.
- Software is never actually done. We're always maintaining, improving it.

Types of Design

1. Strategic Design
2. Tactical Design

- TDD fits into the tactical design umbrella.
- When we start writing the code, we can refine the design further and start building the tactical details along the way. TDD helps to drive this tactical design.

Why Automation?

- Do perform manual testing, but sparingly.
- Do manual testing, the very first time you create a feature.
- Do manual testing, when you see a feature as a tester.
- **Separate the word testing from the word verification.**
    - When we use the phrase automated testing, we really mean automated verification.

- Automate everything that’s possible.
- “Don’t confuse your inability with impossibility.”
- With any complex software, we possible cannot keep up with the change by manually testing every sequence of the path of the system.
- Manually verifying is hard to do and error prone.

How projects end up in waterfall?

- Regression Testing → testing to make sure that what worked before is still working.
- The time taken to test keeps increasing as features are added, because the entire application needs to be tested.
- Doing way too many manual testing — will have to wait for the code to stabilise — agile becomes waterfallish.

Tenets of Testing

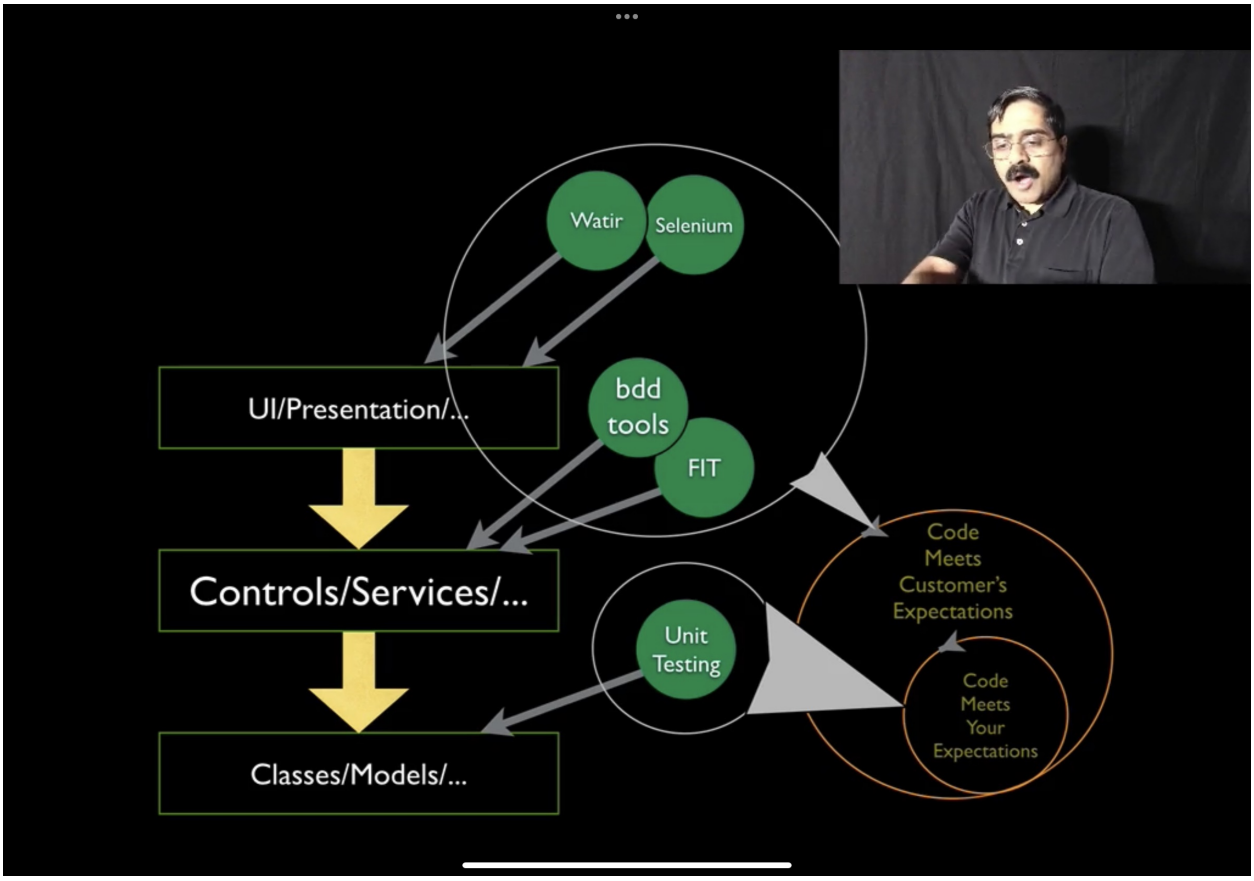
- The job of a tester is to manually verify once, and automate the verification for future executions.
- Automation is key.

Types of Test

- **White-box Testing**
  - Where the tester is keenly interested in the design and the implementation of the code.
  - Programmers usually do **unit testing**, which generally falls under this category.
- **Black-box Testing**
  - Where the tester doesn’t care about the details of the implementation or the design of the code. The tester is interested in the external behaviour, from the point of the view of the application, the problem, or the domain.
  - This is usually where business analysts and testers are involved.

- **Functional and Acceptance Testing** normally falls under this category.

Where to automate and where not to?



- Three types of companies-
  1. NO automation
     a. This is where the trouble is.
     b. A significant portion of the industry falls here.
  2. Automation
     a. Unfortunately, their programmers aren't willing to automate the verification. Companies hire 'automation engineers', who start automating at the UI level (using Selenium, UIToolKit) — 'pathway to hell testing'. These testing become brittle quickly as the UI changes quite heavily and most of the tools aren't capable of dealing with the UI.

        i. ‘Ice cream cone structure’ — more tests at the upper layers than at the lower layers.

3. Automation at the right level to the right measure
    a. You should do as much testing at the lower level as possible than at the higher levels.
        i. ‘Pyramid structure’ — more tests at the lower layers than at the upper layers.
    b. More tests at the bottom layers gives a quick feedback loop.
    c. The tests at the top level should be minimal — a smoke test, or a connectivity test, just to make sure things are wired properly.

Most important thing needed for TDD

1. Discipline.
    a. Without discipline, nothing really works.
    b. Without discipline, we don’t have the desire to do the hard work.
2. Learning and unlearning.
    a. Asking a colleague/mentor to critique your code, ask for help while writing tests.

Why is it not popular?

- Unit testing is the software equivalent of exercising. Most people know that automated testing is good for the health of the software, but very few have the discipline to do it.
- Venkat → “I don’t write automated tests because I have a lot of time, it is because I don’t have enough time.”

> Some people change when they see the light... others change when they feel the heat.
> - Caroline Schoeder