Quality Strategy from A to Z (with a big focus on C)

Building a quality-focused approach in software development

Olivier Voyer

February 27, 2025

$whoami

Olivier Voyer, Eng., M.Eng

- Director, Software Development at Progi
- 20+ years of experience in software development
- C++, C#, Java, Python
- Emergency Solutions, Power Engineering, Health & Safety, Insurance, Blockchain
- Agilist & People-centric

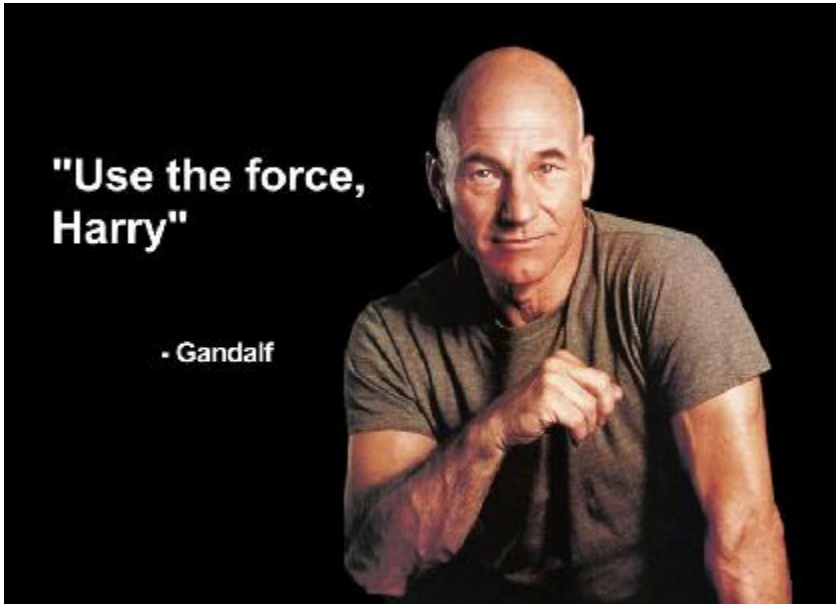
“I believe software development is more about people than technology.”

Objectives

- Explore the pillars of a solid quality strategy
- Balance theory with real-world application
- Provide actionable tools for programmers, QA, tech leads, managers, and directors

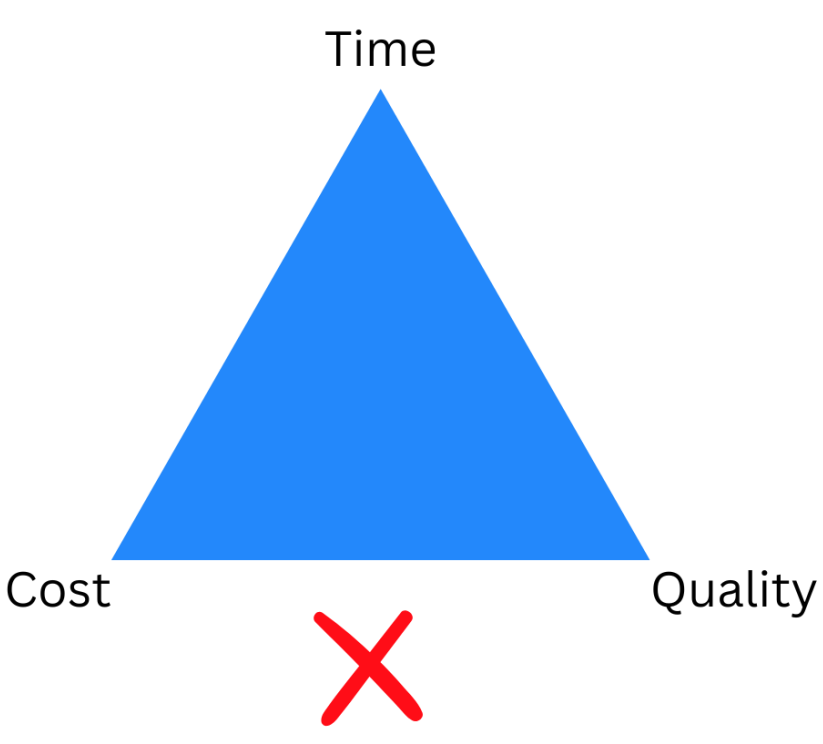
Disclaimer

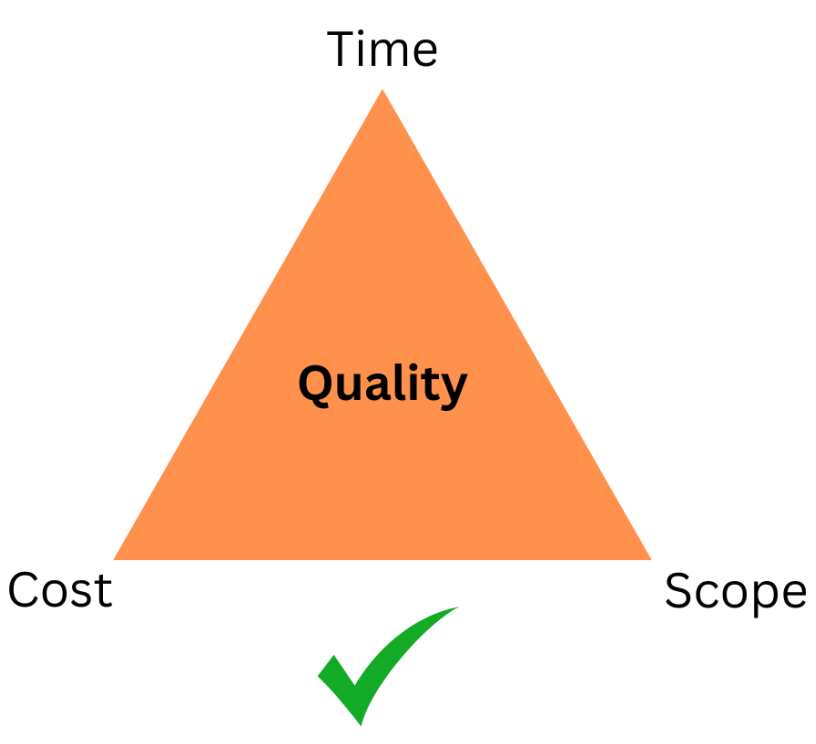
« This presentation is full of quotes. Most of them are completely made up, but they sound convincing. »



What Is Quality?

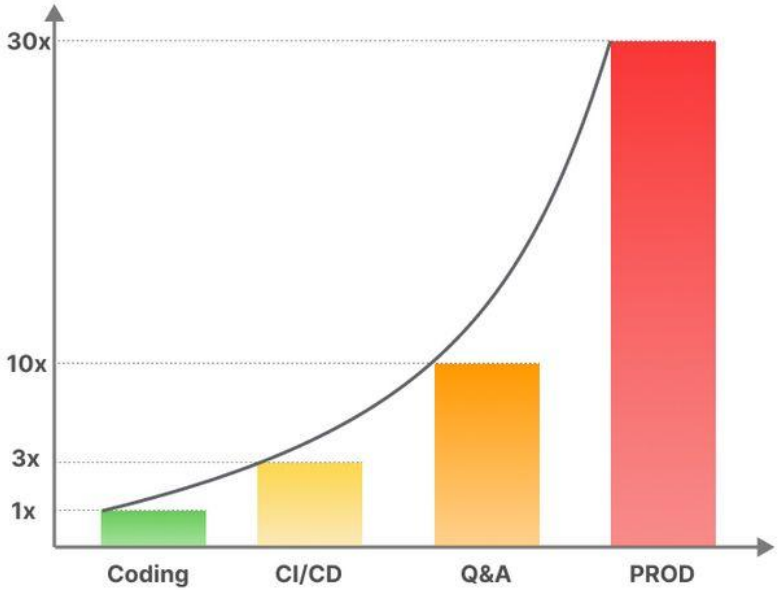
it’s about… **delivering value**


Time
Cost
Quality


Time
Quality
Cost
Scope

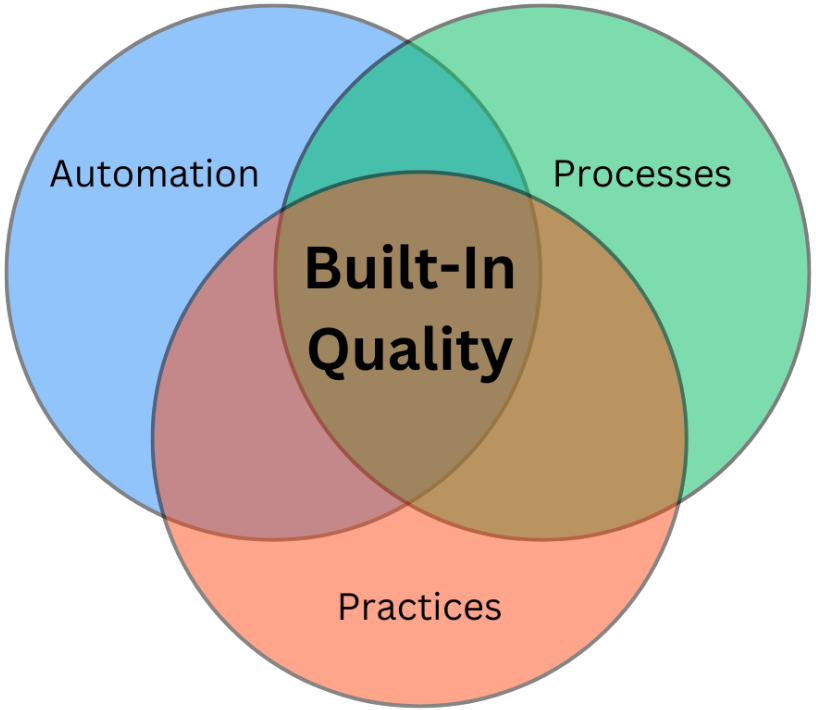
Relative Cost of Fixing Bugs

Daniel Moka



« Low performing teams don’t have time to do it “right”. But hope they have time to do it twice… » - John Crickett

Where do we start?

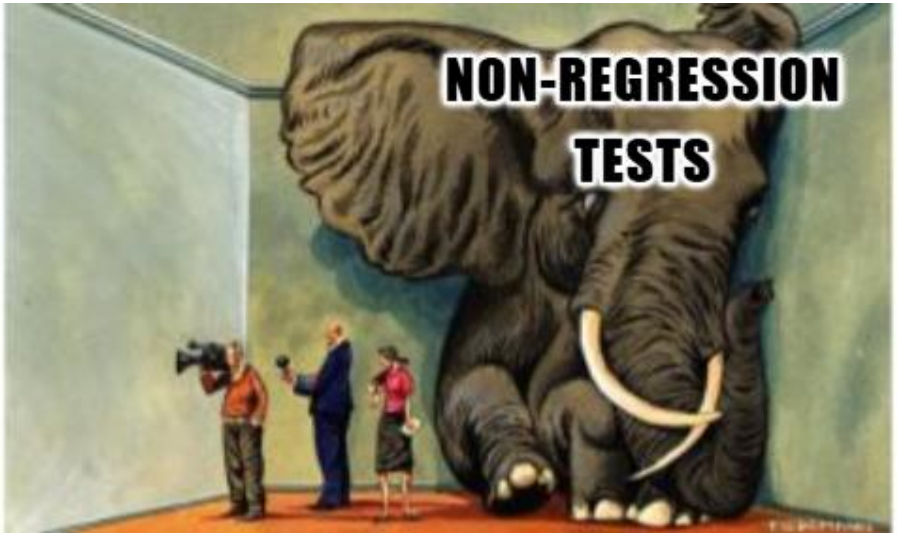

Automation
Processes
Built-In
Quality
Practices

Pillar 1: Automation

« If you rely on manual operations to assure quality you will never achieve built-in quality. »

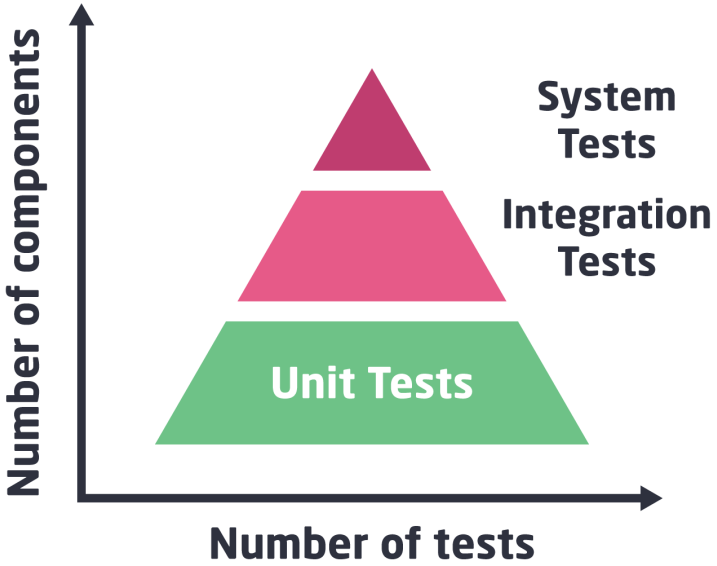
Automate Everything!

- Enforces best practices
- Reduces human error
- Speeds up feedback loops
- Ensures consistency
- Enables the scaling of operations



« The objective is to increase confidence in your codebase. »

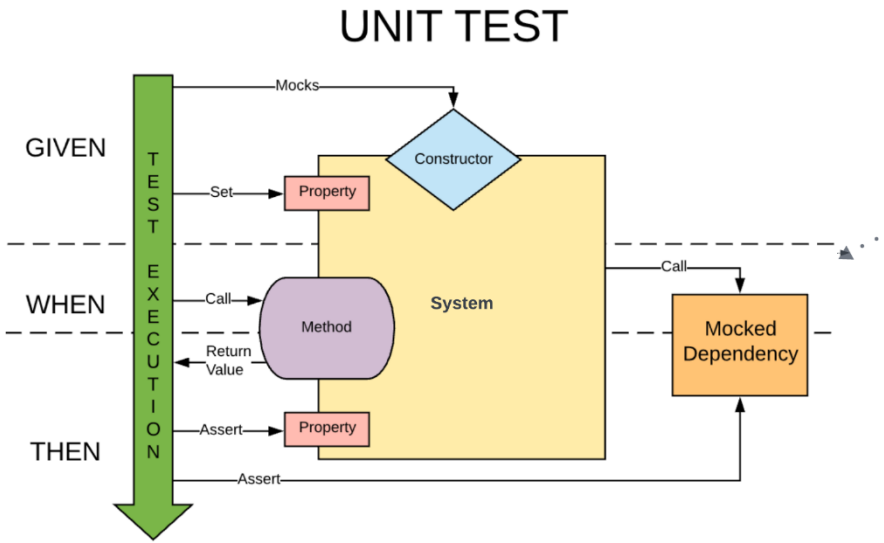
Still Relevant In 2025?



Unit Tests

Unit tests focus on testing individual components in isolation. They're fast, reliable, and help you catch issues early.

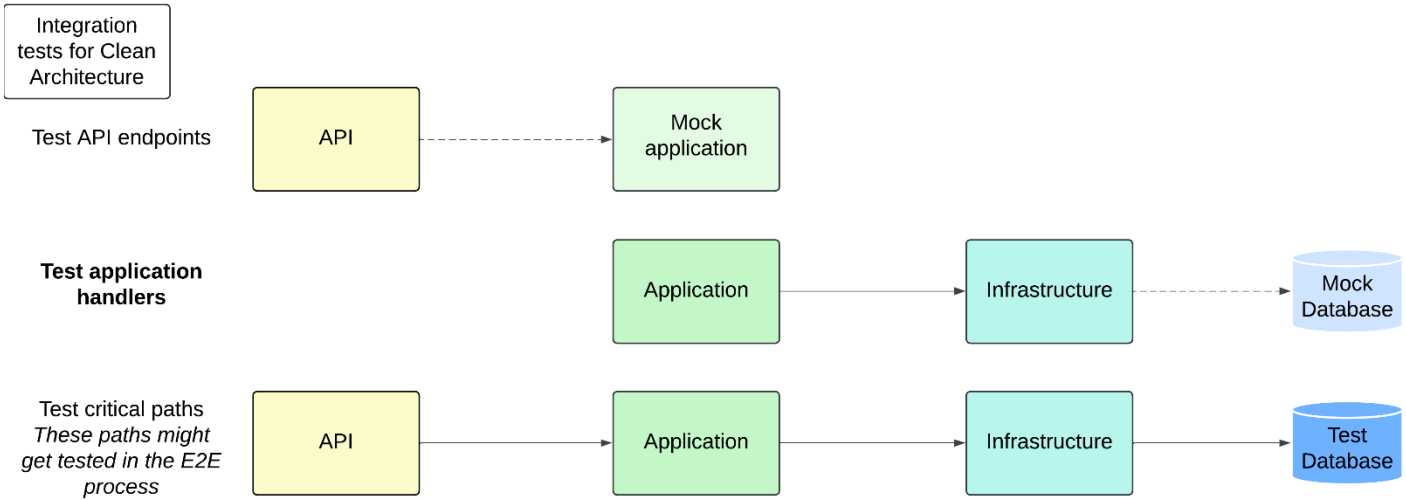
- Mock dependencies
- Test all the happy paths
- Test all the failure paths
- Verify exception handling



« Unit Tests are your first line of defense against regression. »

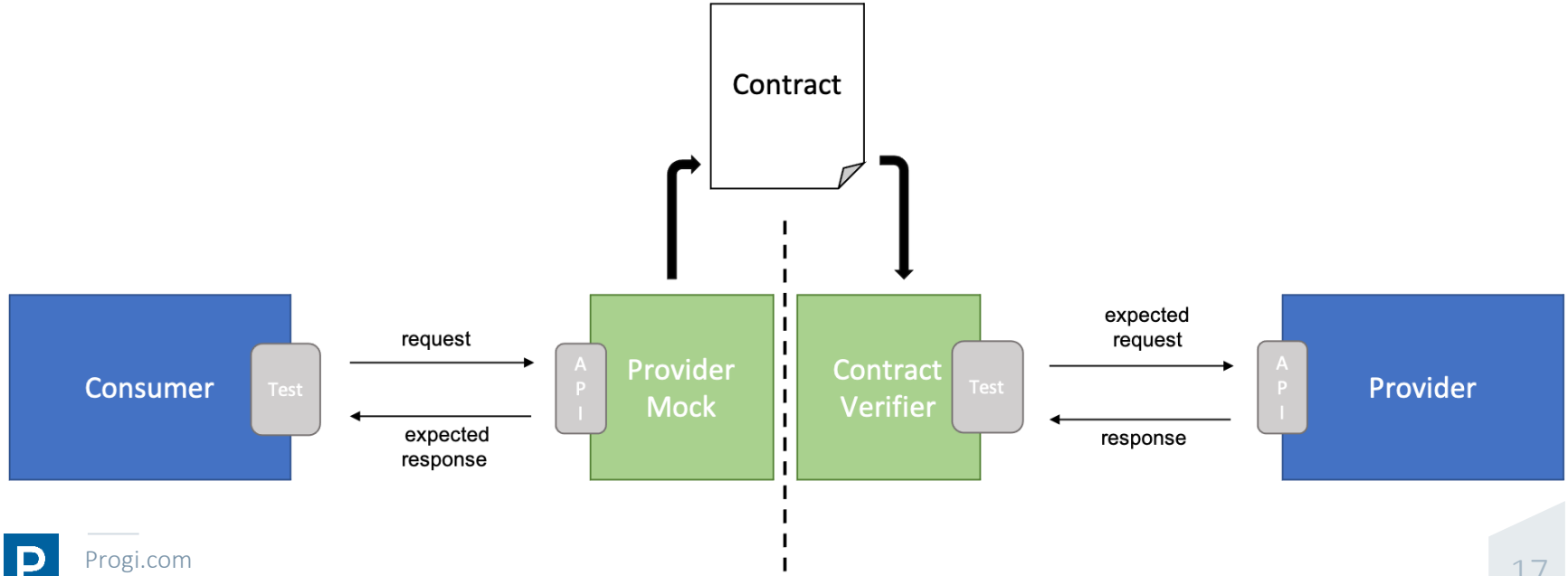
Integration Tests

Integration tests verify that different components work together correctly. They're slower but essential for testing database operations or external services.



Consumer-Driven Contract Testing (CDC)

CDC is used to test components of a system in isolation while ensuring that provider components are compatible with the expectations that consumer components have of them.

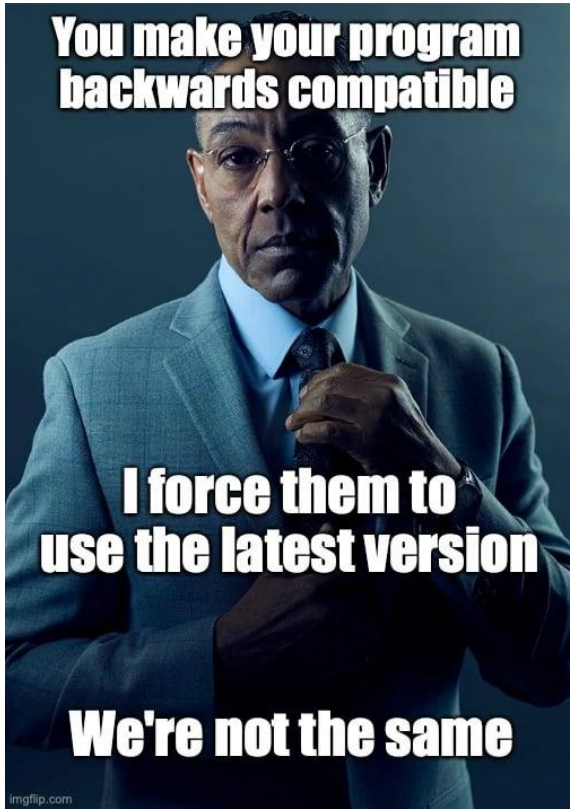


Breaking Changes

- ⭐ **Rule #1**: Don’t break contracts
- ⭐ **Rule #2**: See Rule #1

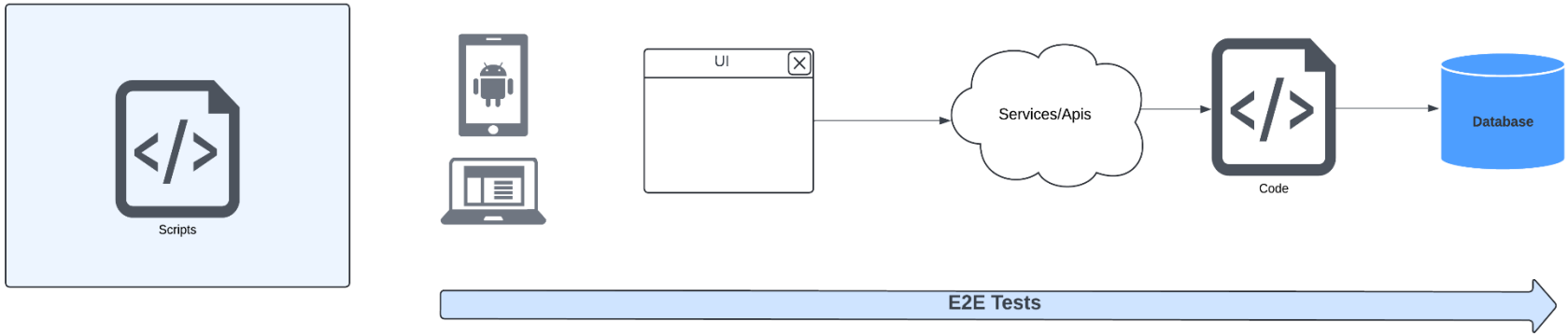
Best Practices:

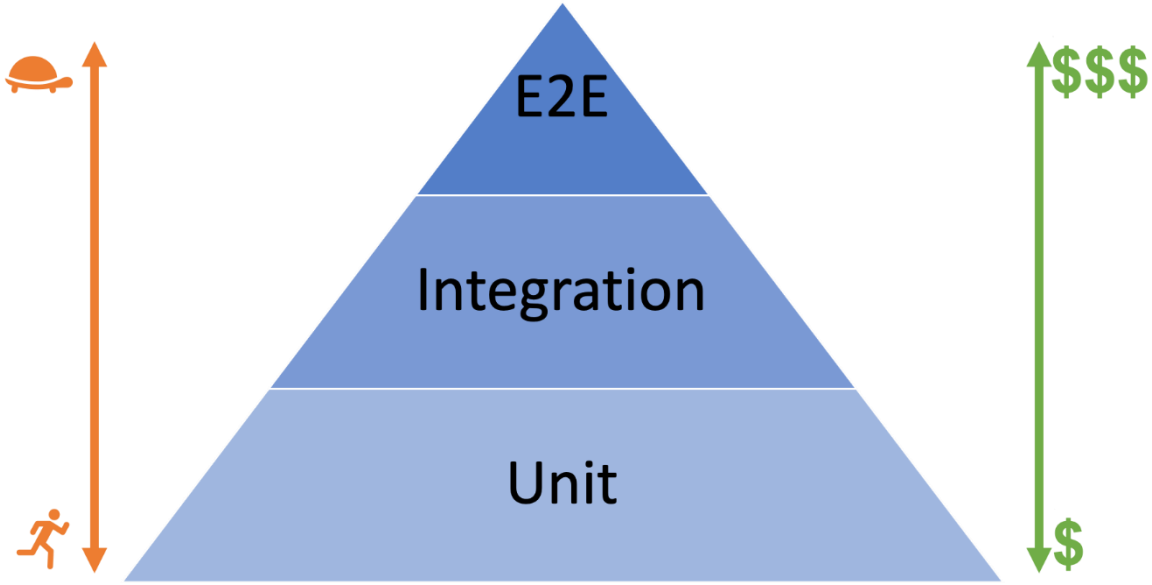
1. Establish API contracts
2. Adhere to existing contracts
3. Prevent accidental breaking changes (https://pact.io/)
4. Agree on a procedure if you must introduce a breaking change



End-to-End Tests

End-to-end tests check the entire system flow. They're the slowest but provide the most confidence that everything works together.





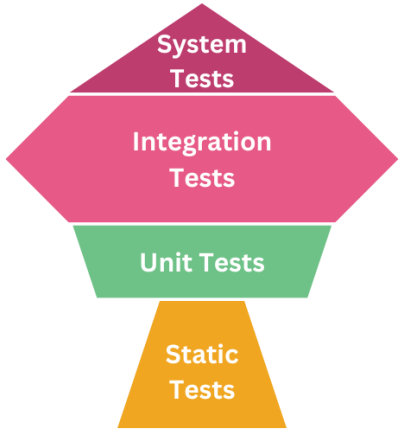
Where do we focus our time?


passing unit tests


2 UNIT TESTS
NO INTEGRATION TESTS

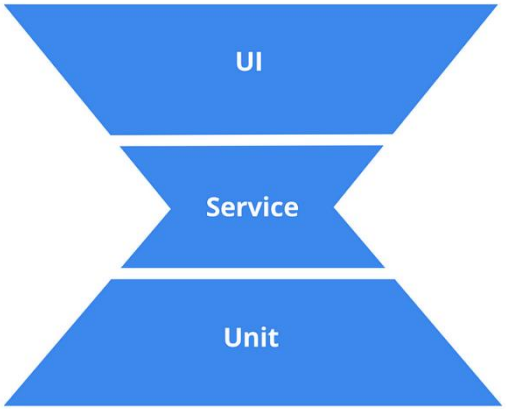

UNIT TESTS PASSING
NO INTEGRATION TESTS
imgflip.com

Testing Trophy



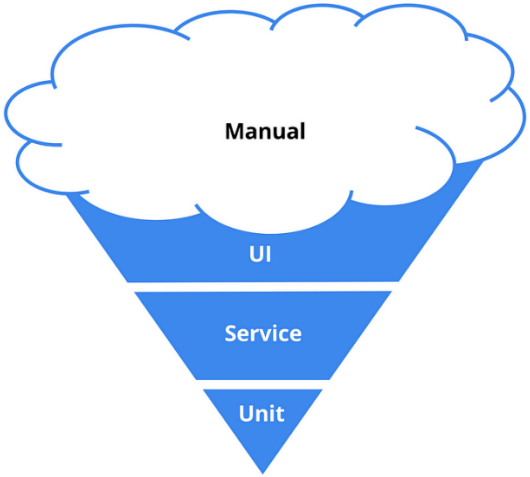
The new standard?

Testing Hourglass



Best for legacy projects

Testing Ice Cream Cone



Not scalable

Not Just Tests…

Your Deployments…

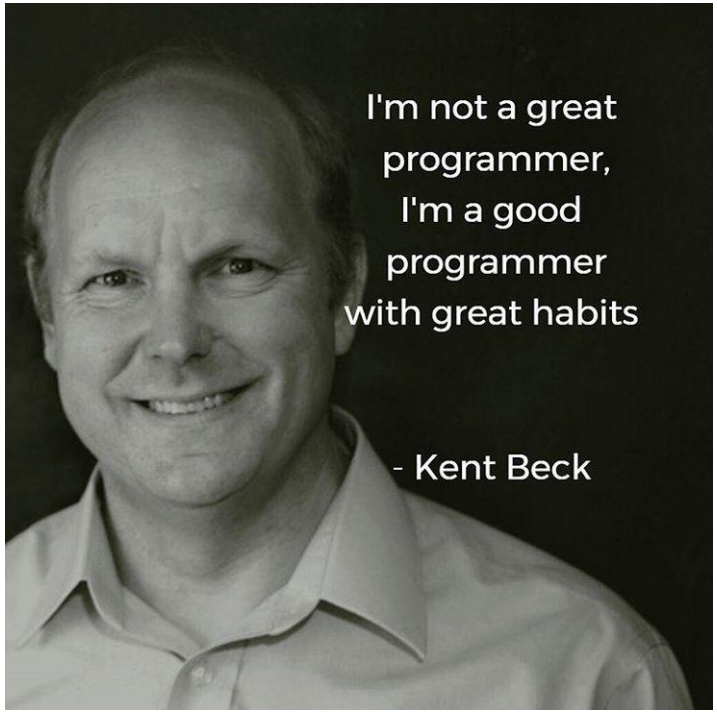
- CI/CD automates repetitive and high-risk processes, ensuring they are consistent, testable, and maintainable.
- It improves efficiency, quality, and scalability.

And More

- Monitoring and alerts
- Infrastructure provisioning

« Good tests act as documentation. They show how the code should behave and catch issues before they reach production. »
- Milan Jovanović

Pillar 2: Best Practices



```
// Complex code == Complex bugs
// Maintainable > Clever
```

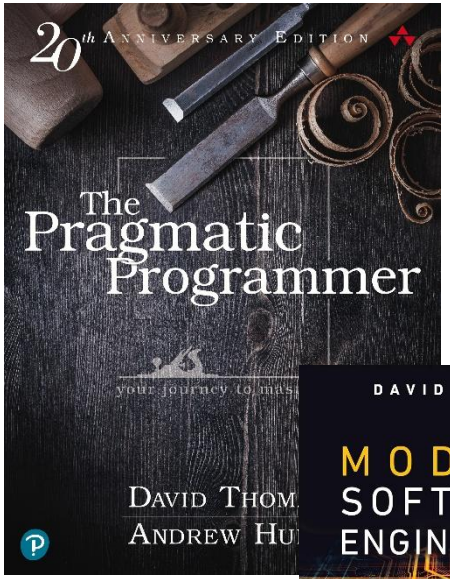
The Essentials

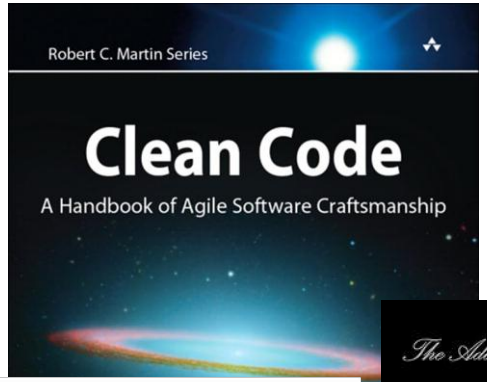
- Clean Code/Architecture. *Uncle Bob is your best friend.*
- SOLID. *The infamous 5 letters.*
- Design Patterns. *Don’t force them—use them when they add value.*
- Cool Acronyms. *DRY, YAGNI, INVEST, KISS, etc.*
- Tell, don’t ask. *Don't let services do work the objects should do themselves.*
- TDD. *We’ll get to it later…*

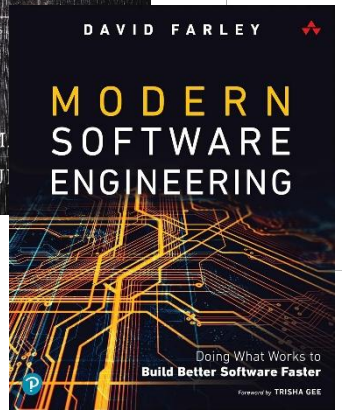
Guidelines

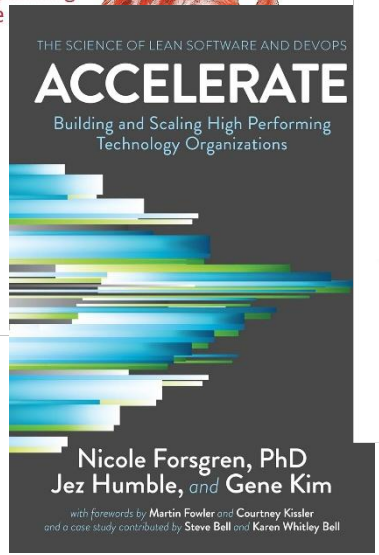
- Beware of the “one-size fits all” approach.
- You can’t know everything. Define your baseline.
- Be **pragmatic**, not dogmatic.
- Encourage collective ownership.
- Measure and iterate.

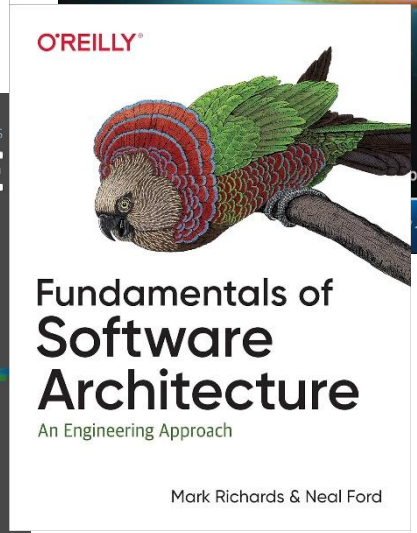
Read Much, Bruh?

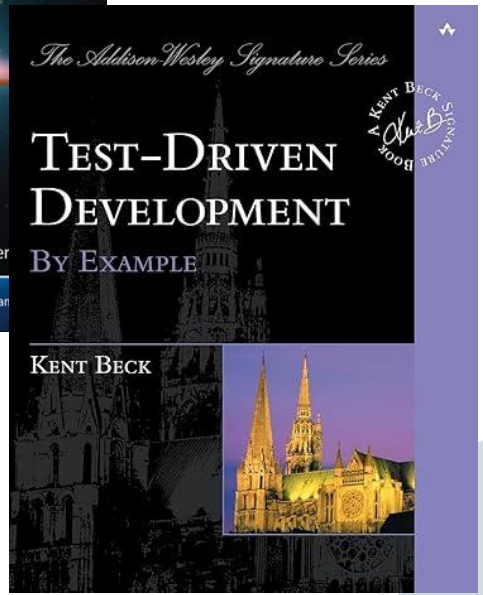












The Case Of TDD

« Once a programming team has adopted a methodology it's almost inevitable that a few members of the team, or maybe just one bully, will demand strict adherence and turn it into a religion. »

« TDD is the only safe way to create fault-free software »

My 2 cents

TDD works. That's a fact. It's proven. But…

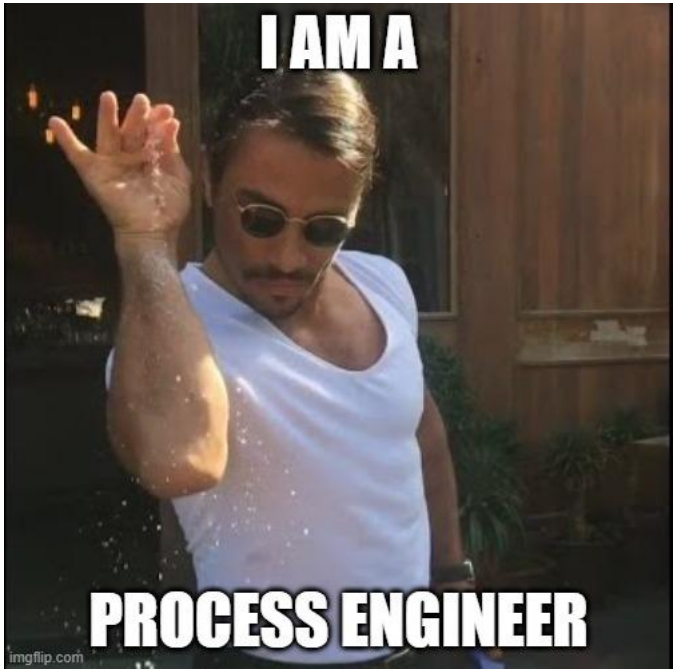
- Implementing it efficiently in a frequently changing codebase is challenging. *And yes, I'm aware unit tests should not be tied to implementation details.*
- Forcing a methodology on someone who isn't convinced *won't work*!

My Strategy:

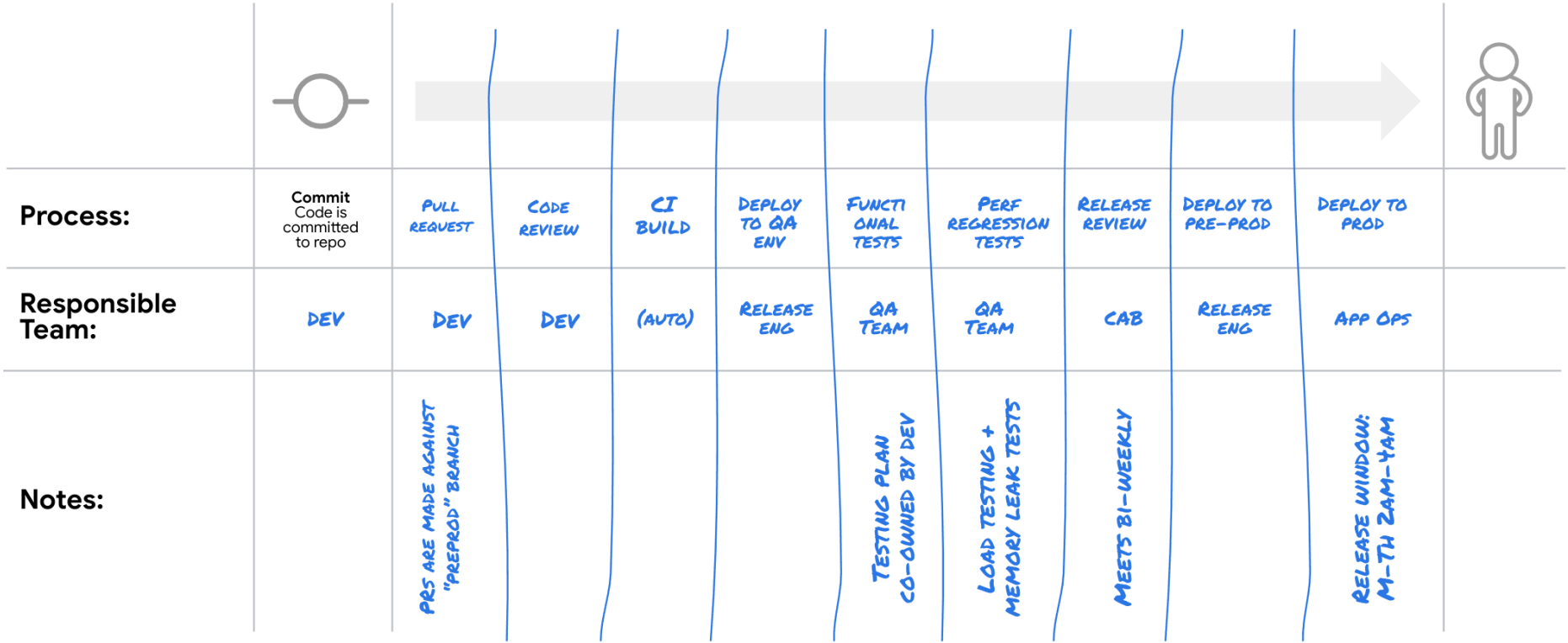
1. Demonstrate the value of testing
2. Reinforce those fundamentals
3. Iterate
4. Introduce TDD organically

Pillar 3: Processes

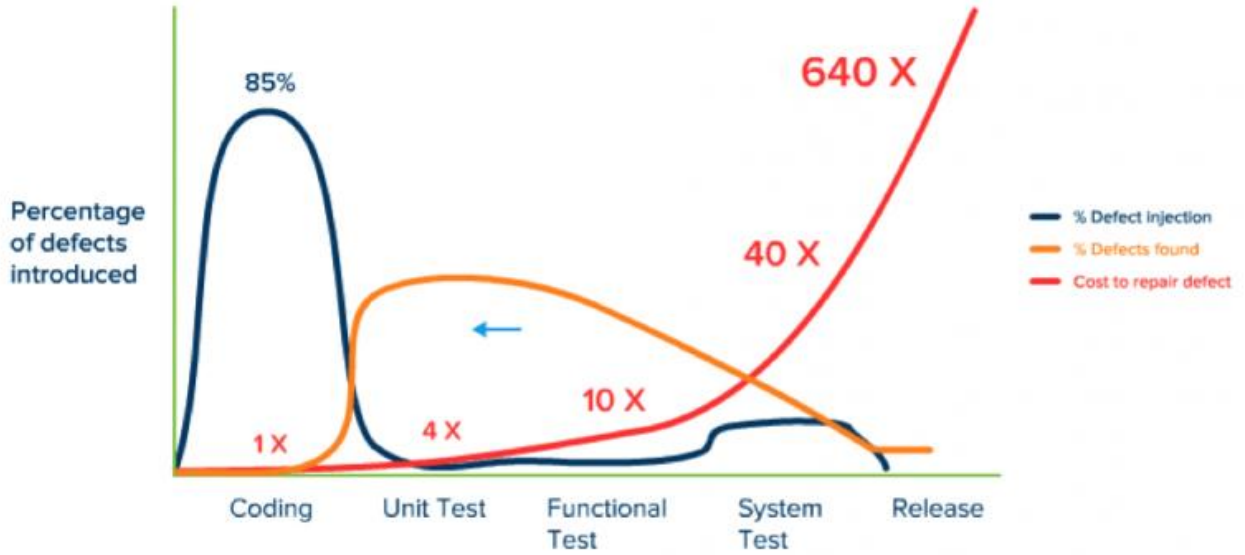
« Process is like seasoning. You need some to enhance the result. Too much, and it overwhelms. It's different for everyone. »



Value Stream Mapping (VSM)

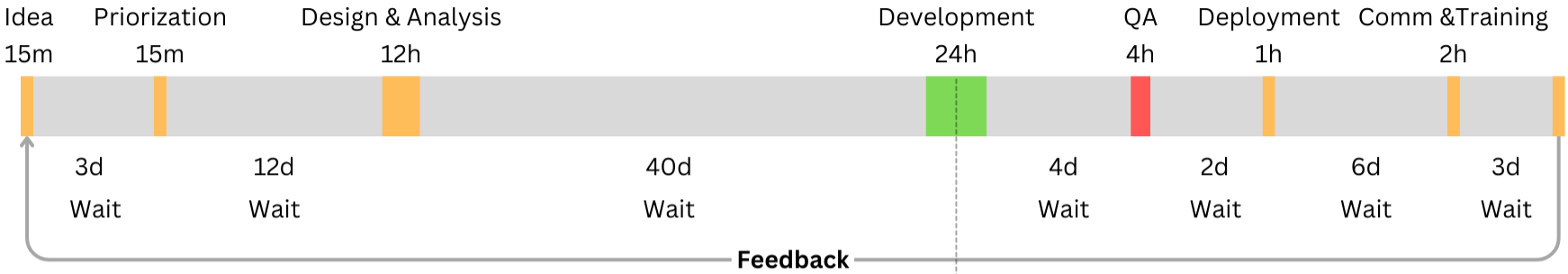


Fail Fast

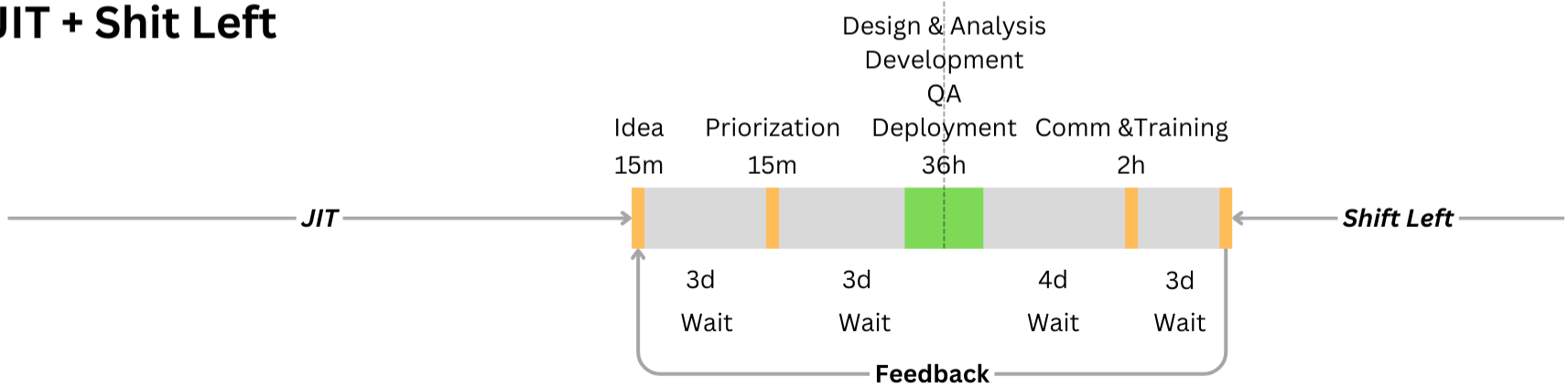


Jones, Capers. *Applied Software Measurement: Global Analysis of Productivity and Quality.*

Traditional



JIT + Shit Left

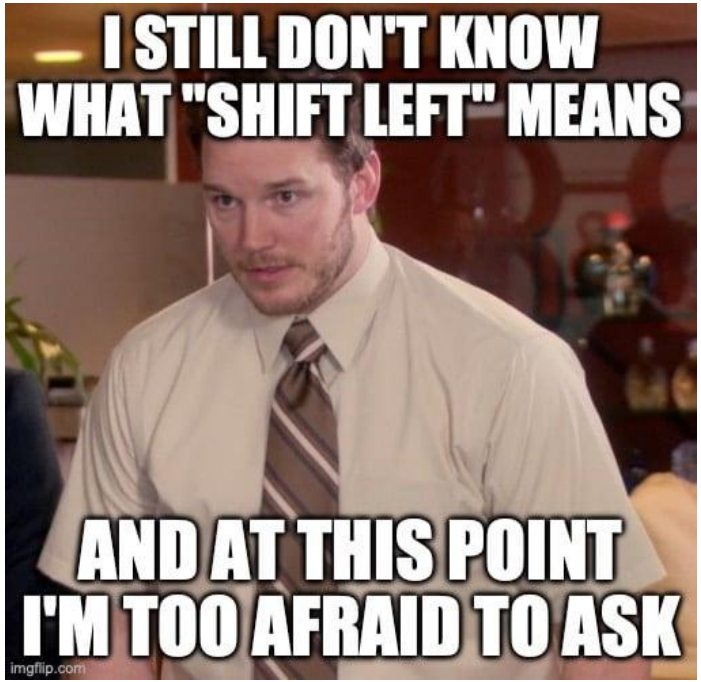


Just-In-Time (JIT)

- User Stories: Just enough to start, not all at once.
- Documentation: Keep it minimal. We want developers to ask questions ☺.
- Architecture: Only what's needed for current user stories. "Do you really need caching *right now*?"
- Planning: Focus on the iteration, not beyond. If you need some long-time planning, use *Sunset Graphs*.
- Collaboration: Encourage it to keep the team effective.

Shift Left

- Release Sooner > Release Quickly
- If something breaks, you want to know it as soon as possible
- Involve the team members at all steps
- Identify and eliminate waste - Have a **Lean** mentality
- Monitor Lead Time/Cycle Time



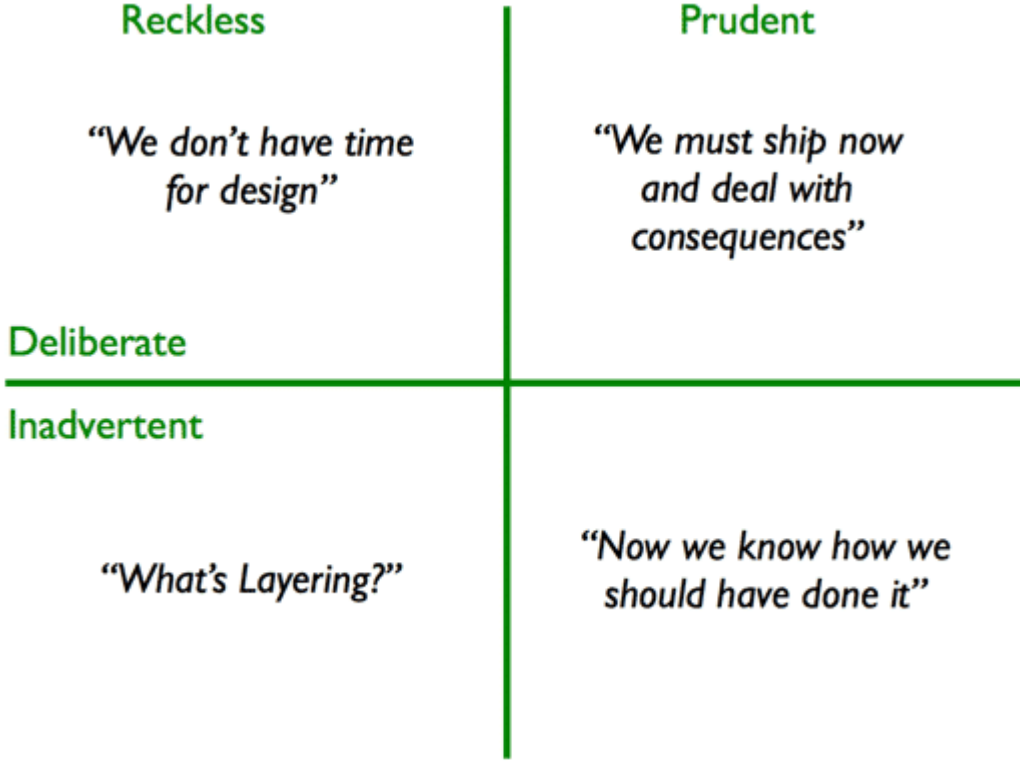
The big « C »: Culture!

« Having a QA department is a sign of incompetency in your Development department. » - Cory Foy

Culture Is Everything

- Leadership buy-in is essential. *Quality is often the first aspect to be compromised when pressure arises.*
- Prioritizing speed over quality leads to higher long-term costs. *Balance matters.*
- Continuous Improvement must be embedded within your value creation process. *It's not optional — it's a mindset.*
- Psychological safety drives innovation. *Teams need the freedom to experiment and learn. Have fun!*

Technical Debt



Deal With It!

- ⭐ Every piece of code adds *some* technical debt.
- ⭐ Without a plan, it grows and becomes harder to handle.

How to stay in control:

1. Stop feeding the beast
2. Make it visible and prioritize
3. Pay it off in small, consistent steps

“You should not need to ask permission every time you want to fix stuff in your codebase.”

How Did We Do It?

We took the time to…

1. Understand business realities. *We listened to our stakeholders.*
2. Educate, iterate, and… wait. *Change happened progressively.*
3. Define our direction and core values. *What's our team's identity and long-term vision?*
4. Negotiate a budget for Continuous Improvement (CI). *We treat it as an investment ☺.*
5. Create a CI backlog. *We made it a team-owned initiative — No PO allowed!*
6. Monitor, encourage, and support progress. *Keep momentum alive.*


I deploy
on every Fri days.
// progi.com

Let’s stay in touch!

Come visit us at our booth

Olivier Voyer

Add me on LinkedIn

progi

1-855-310-6343 Progi.com