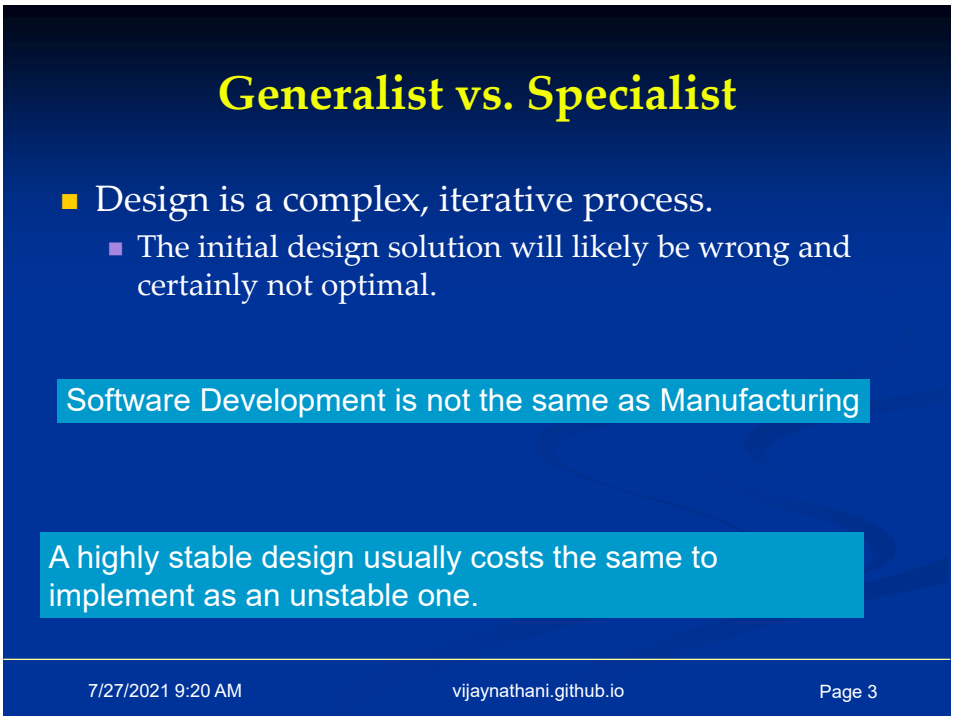
Agile Practices

A process is needed, when we would not do the same thing naturally.

Who owns the code?

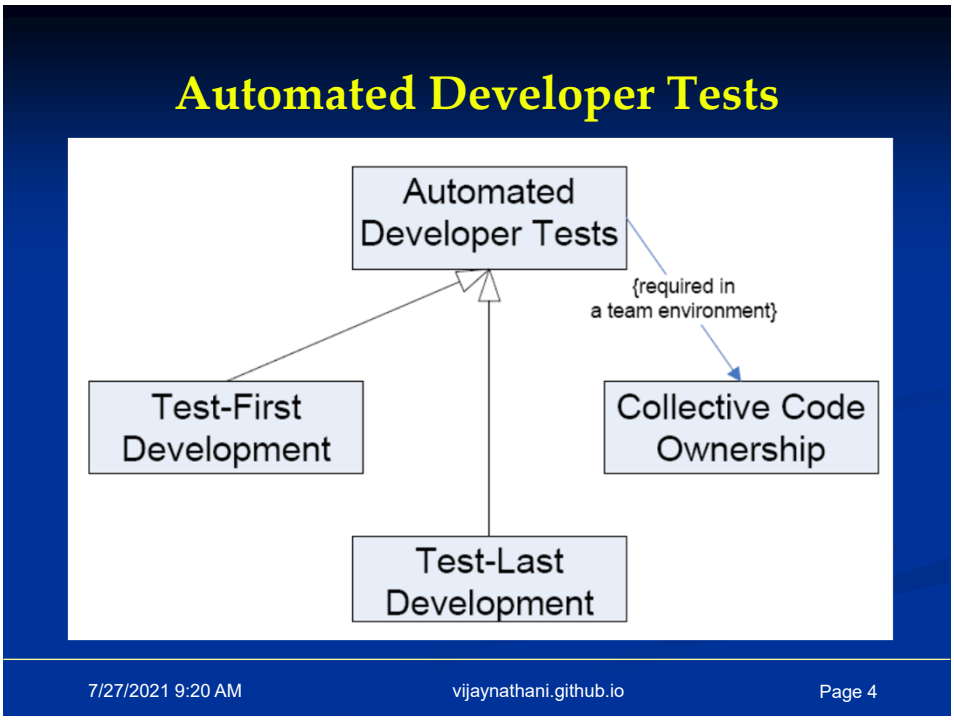
- Collective ownership is nothing more than an instantiation of the idea that products should be attributable to the team, not individuals who make up the team.
  - Strong Ownership / Loose Ownership are usually not good enough.



Programmers shift from design to coding when the problem is decomposed to a level of "primitives" that the designer has mastered. If the coder is not the same person as the designer, the designer's primitives are unlikely to match the coder's primitives, and trouble will result.

Manufacturing is a popular metaphor for software development.

> One inference from this metaphor: highly skilled engineers design; less skilled laborers assemble the products.
>
> This metaphor has messed up a lot of projects for one simple reason—software development is all design.



The way in which people run a 100m sprint is much different than running 10 Km race. In 100m, the runners are hefty and strong. In a 10 Km race, the runners are light and can change their running strategy.

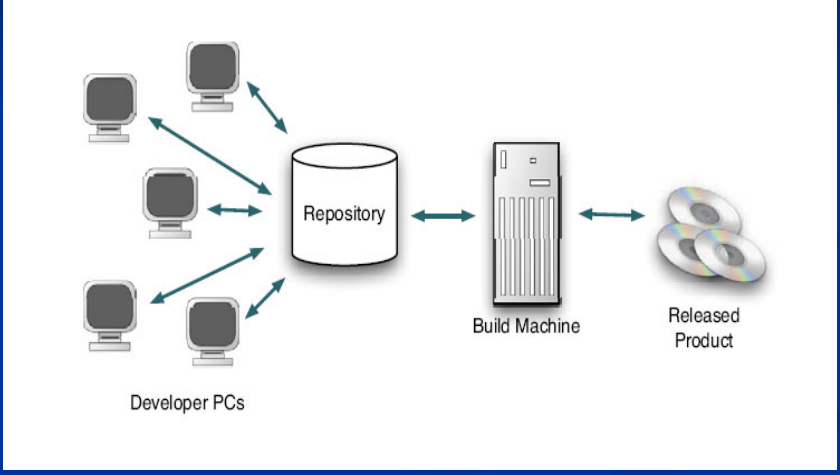
Person who is ahead a the start in 10 KM will usually not win. Runners need to pace themselves. By cutting quality, we can go fast in the short run. Software is like 10 KM race.

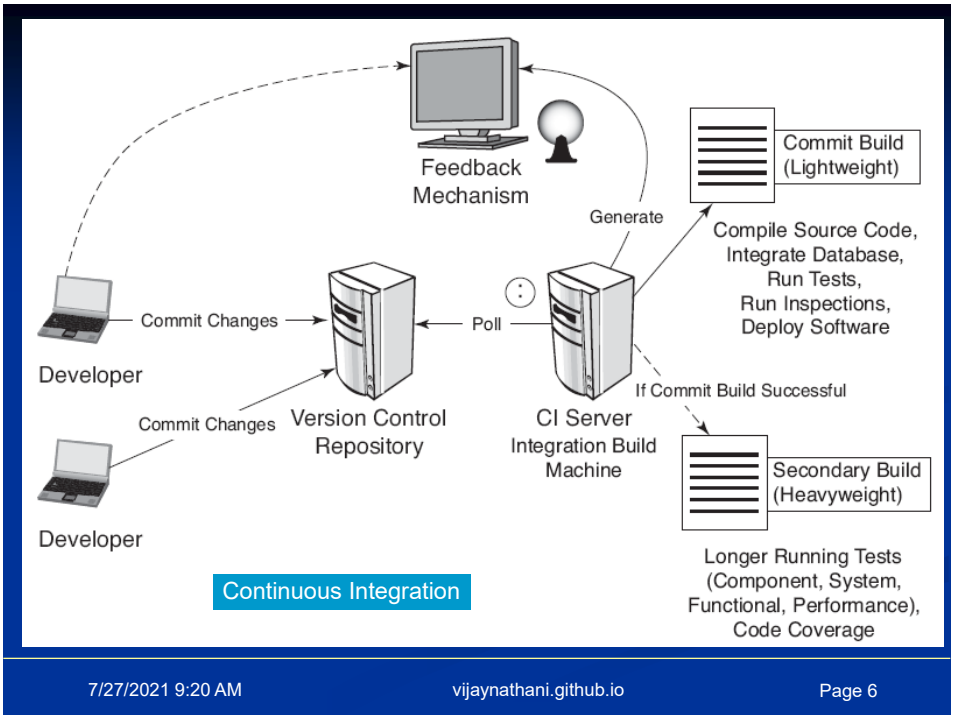
Software development is like a 10 Km race.

We need to know whether we are done. We don't want to be almost complete. Project remain almost complete for many weeks / months.

Every failing test causes the line to stop.

Sandbox Development Setup





For every check-in

Incremental Build

Fast tests (Unit)

Every Few hours

Full Build

Fast Tests

Slow Tests (Unit & Functional)

Information Radiator

==========

A build and test should take less than 10 minutes or else developers will avoid using it.

Code tested with build test suite is frequently separated from the database with mock objects to speed up execution.

These are fast tests

Acceptance tests are slow tests. They can run less frequently (but at least once a day)

CI means:

•All developers run private builds on their own workstations before committing their code to the version control repository to

ensure that their changes don’t break the integration build.

• Developers commit their code to a version control repository at least once a day.

• Integration builds occur several times a day on a separate build machine.

• 100% of tests must pass for every build.

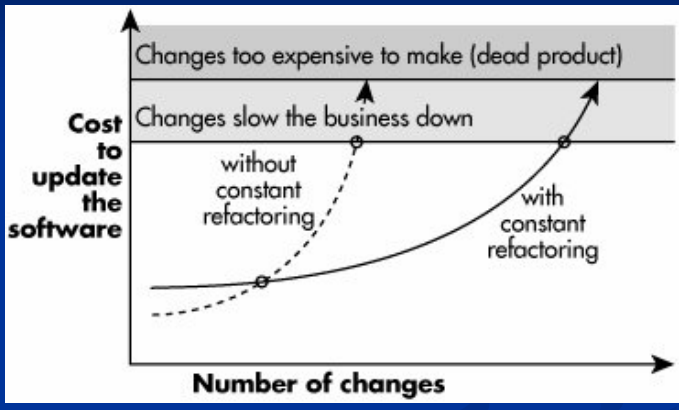
• A product is generated (e.g., WAR, assembly, executable, etc.) that can be functionally tested.

• Fixing broken builds is of the highest priority.

• Some developers review reports generated by the build, such as coding standards and dependency analysis reports, to seek areas for improvement.

Refactoring

Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code [design] yet improves its internal structure.



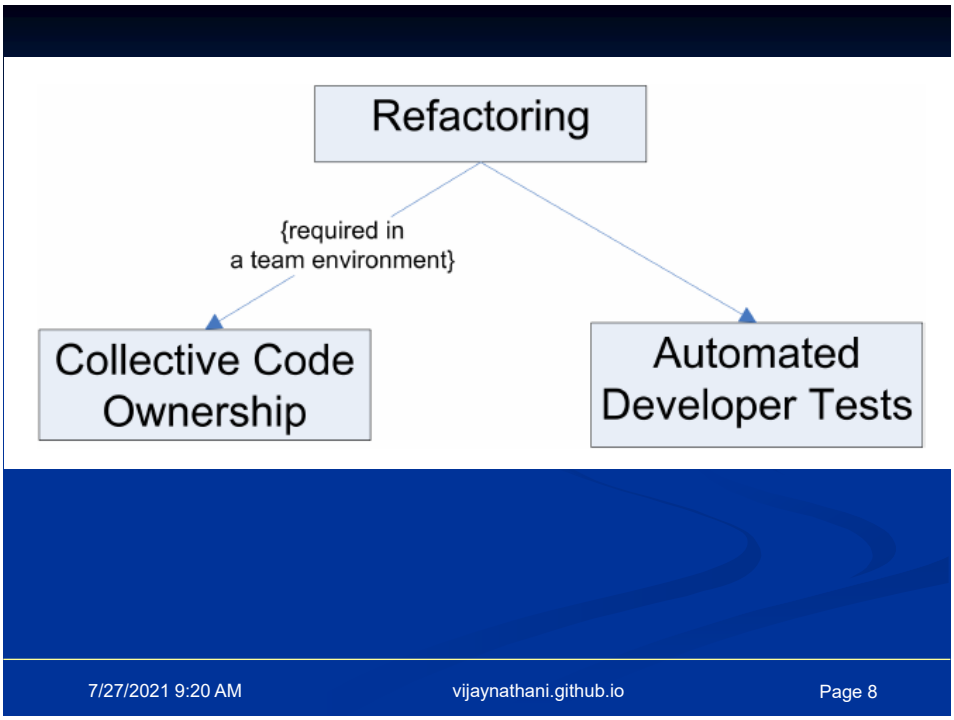
Refactoring is lot like getting our teeth cleaned. We can think of a thousand reasons to delay it, and we will get away with procrastinating for quite a while. But eventually, our delaying tactics will come back and cause pain.

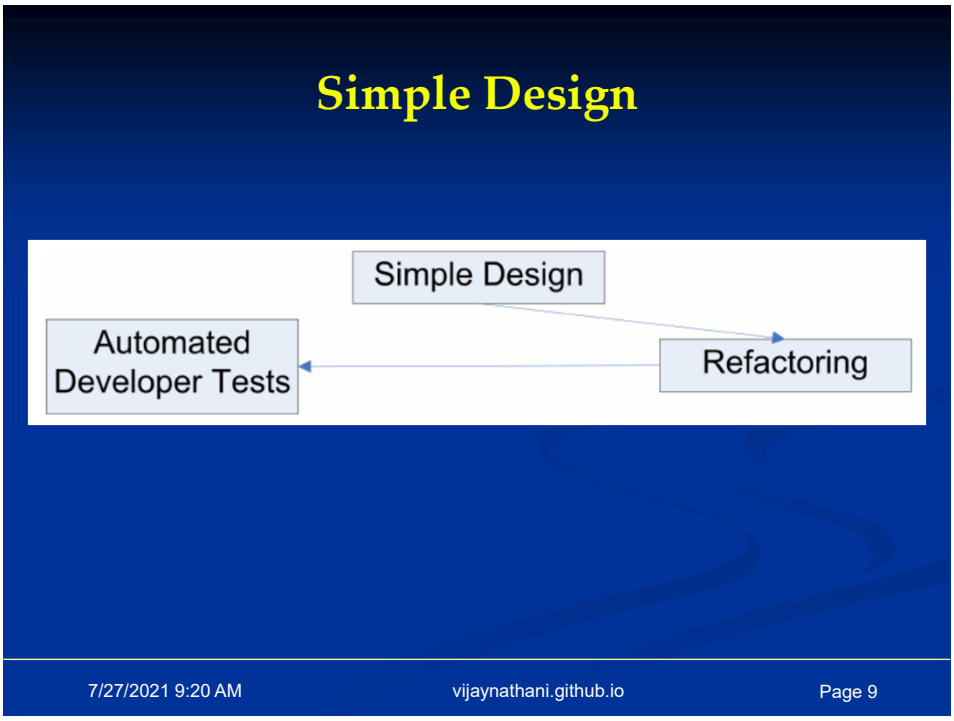
You can pay a little now, or

you can pay a lot more later.

– Sign in a garage

suggesting a tune up.


Refactoring
{required in
a team environment}
Collective Code
Ownership
Automated
Developer Tests

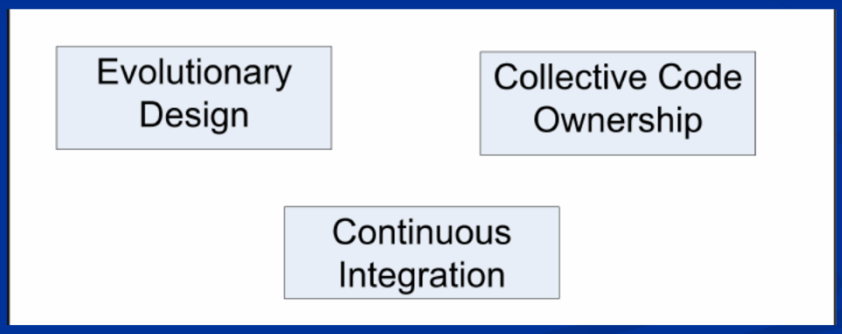


BDUF can be even worse than wasted time because of incorrect guesses. BDUF can also lead to self-fulfilled prophesies.

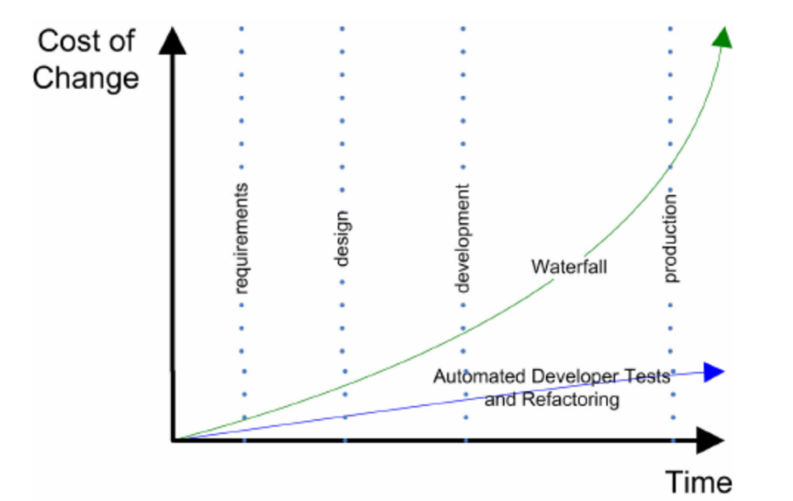
It's pretty well known that a Big Design Up-Front (BDUF) has some big problems. At the same time, most often we know some things from day one. It's a matter of balance.

Finally, a last remark regarding DDD and TDD: Domain Models are very suitable for TDD. Sure, you can also apply TDD with more database-oriented design, but I haven't been able to apply it as gracefully and productively as when I'm working with Domain Models.

Simple Design



Cost of Change in Waterfall & TDD

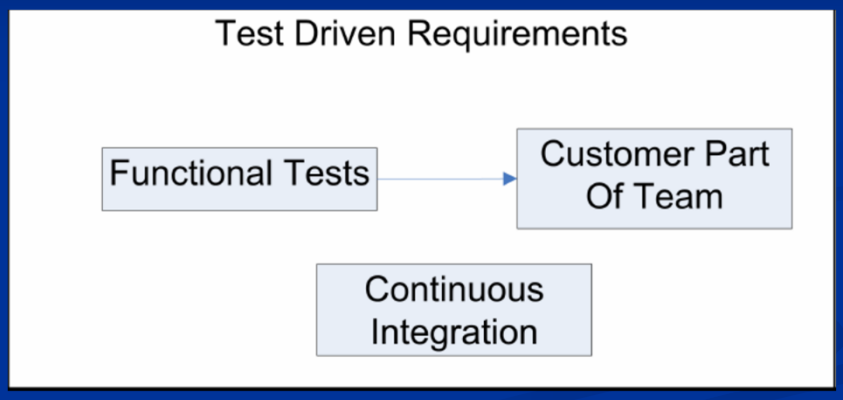


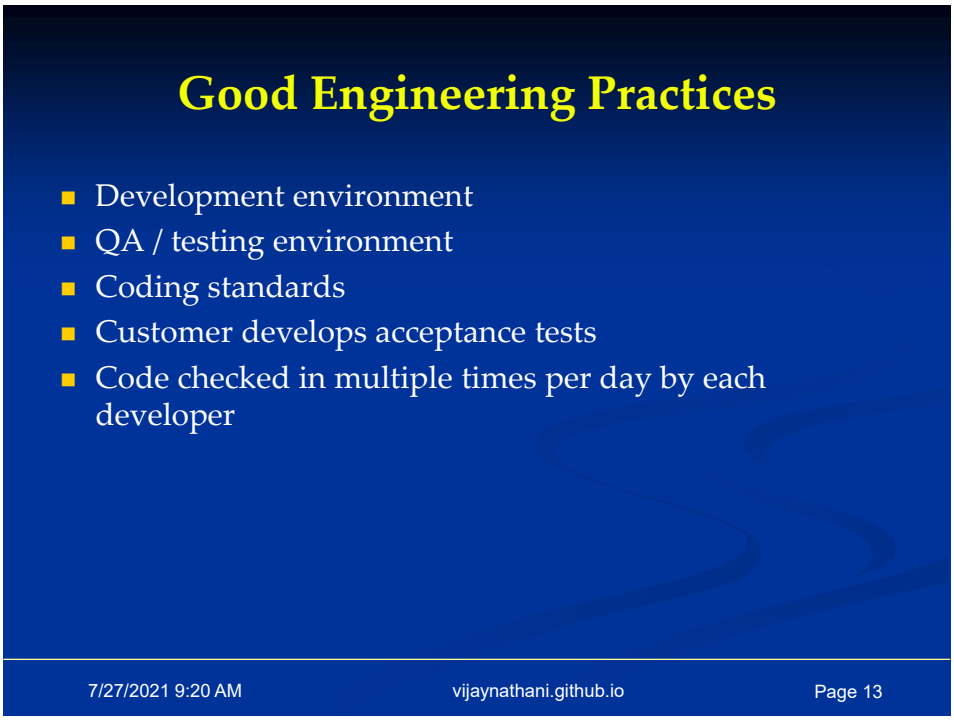
No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle. - *Bersoff, et al, 1980*

On an average, 35% of the requirements change during project execution

It is estimated that 25% to 50% of effort in waterfall goes in discussing / managing change requests.

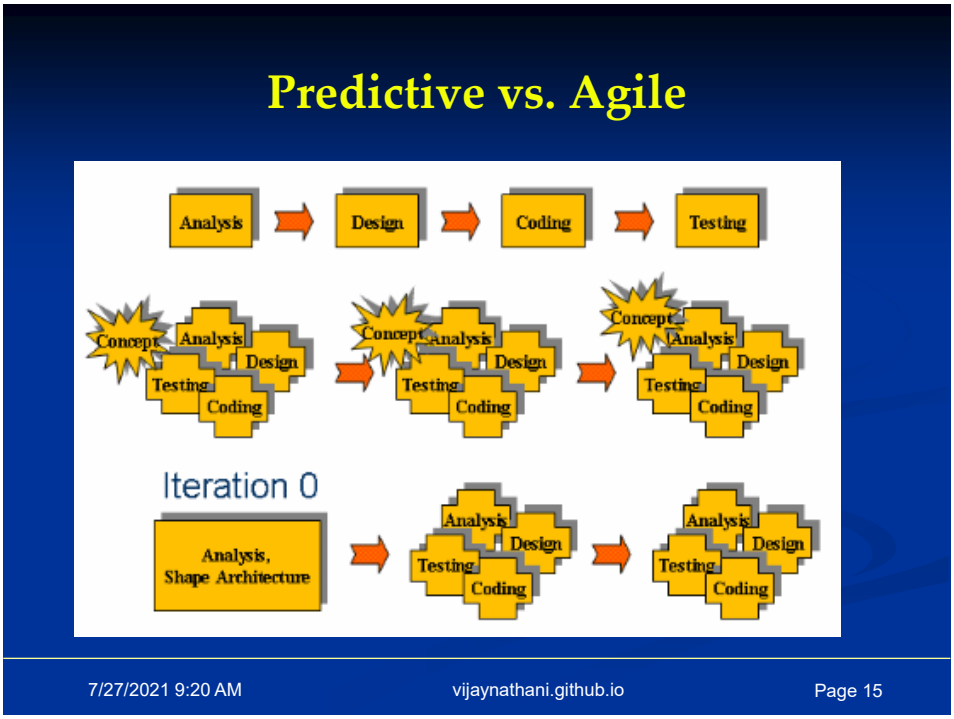
Automated Functional Tests





For distributed source code Management use Git / BitKeeper and not Subversion or CVS

Thoughtworks is primarily a XP shop.



The first row is Waterfall.

The second row is XP.

The third row is Scrum.

As I've been contemplating moving agile throughout our entire organization here at Data Transfer Solutions, I've been considering the usefulness and effectiveness of an Iteration Zero. Many agile teams use what's known as Iteration Zero to put the necessary systems in place to enable the delivery of value to the customer. It's essentially the *getting started* iteration. It takes place before any development begins. I think Peter Schuh described Iteration Zero very well in his book Integrating Agile in the Real World. Peter says:

> "An iteration zero does not deliver any functionality to the customer. Instead the project team focuses on the simple processes that will be required for the adoption and use of most agile practices. From a management point of view iteration zero may include:
>
> •Initial list of features identified and prioritized.
>
> •Project planning mechanism identified and agreed upon.
>
> •Identification of and agreement upon a team customer, essential stakeholders, and business users and the nature of iterative planning process, such as the time of planning meetings and the length of iterations."

What is XP?

- XP is a lightweight methodology for small to medium sized teams developing software in the face of vague or rapidly changing requirements. -- Kent Beck.
- The goal of XP is outstanding software development

Developer Bill of Rights

- *You have the right to know what is needed, with clear declarations of priority.*
- *You have the right to produce quality work at all times.*
- *You have the right to ask for and receive help from peers, superiors, and customers.*
- *You have the right to make, and update your own estimates.*
- *You have the right to accept your responsibilities instead of having them assigned to you.*

The Customer Bill of Rights.

- *You have the right to an overall plan, to know what can be accomplished, when, and at what cost.*
- *You have the right to get the most possible value out of every programming week.*
- *You have the right to see progress in a running system, proven to work by passing repeatable tests that you specify.*
- *You have the right to change your mind, to substitute functionality, and to change priorities without paying exorbitant costs.*
- *You have the right to be informed of schedule changes, in time to choose how to reduce scope to restore the original date. You can cancel at any time and be left with a useful working system reflecting investment to date.*

Why is it Extreme?

- Because we take good practices to extreme levels (turning the knobs up to 10!)
  - Code reviews – pair programming
  - Testing – CI
  - Design – TDD
  - Feedback in minutes

If code reviews are good, we'll review code all the time (pair programming).

If testing is good, everybody will test all the time (unit testing), even the customers (functional testing).

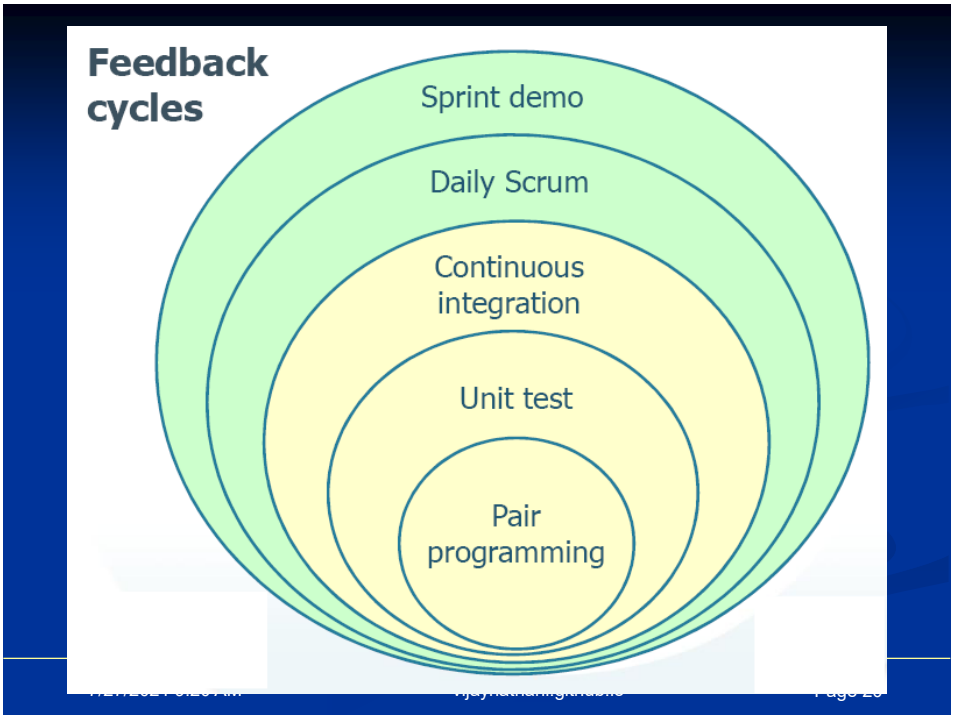
If design is good, we'll make it part of everybody's daily business (Refactoring).

If simplicity is good, we'll always leave the system with the simplest design that supports its current functionality. (The simplest thing that could possibly work.

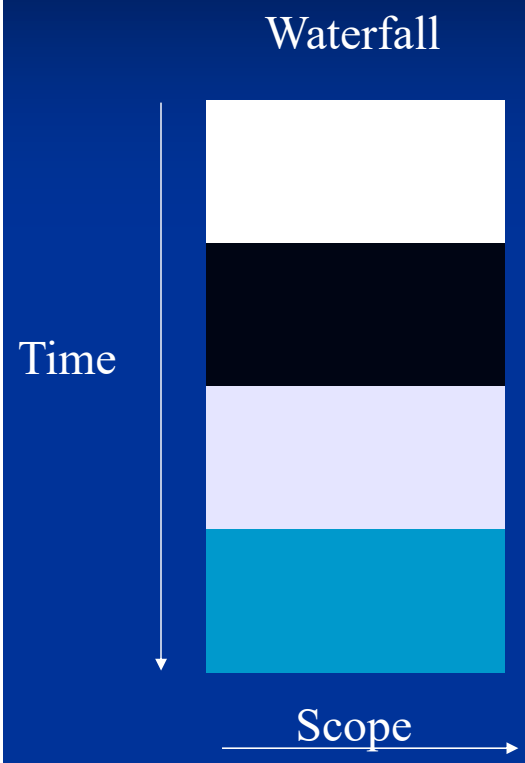
If architecture is important, everybody will work defining and refining the architecture all the time (Metaphor).

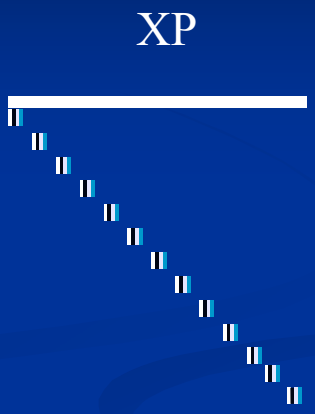
If integration testing is important, then we'll integrate and test several time a day (continuous integration).

If feedback is good, we'll get feedback quickly -- seconds and minutes and hours, not weeks and months and years (the Planning Game).

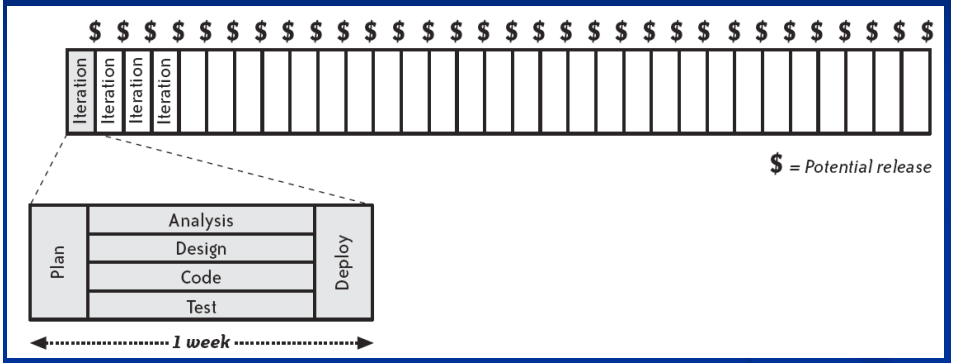

Feedback cycles
Sprint demo
Daily Scrum
Continuous integration
Unit test
Pair programming

Different Processes



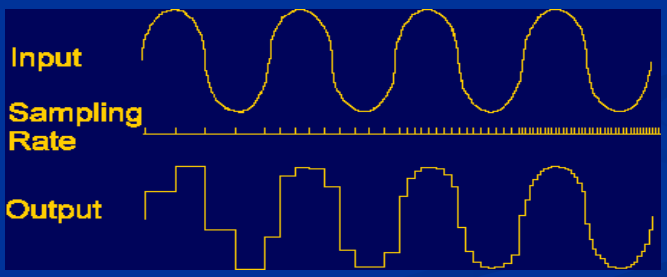


1 Week Iteration



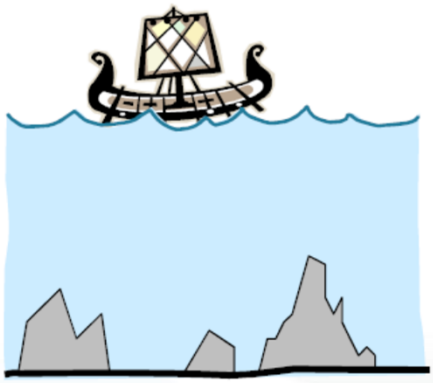
Rate of Change

- Rapid technical changes
- Global scheduling pressure
- Compounded opportunity costs



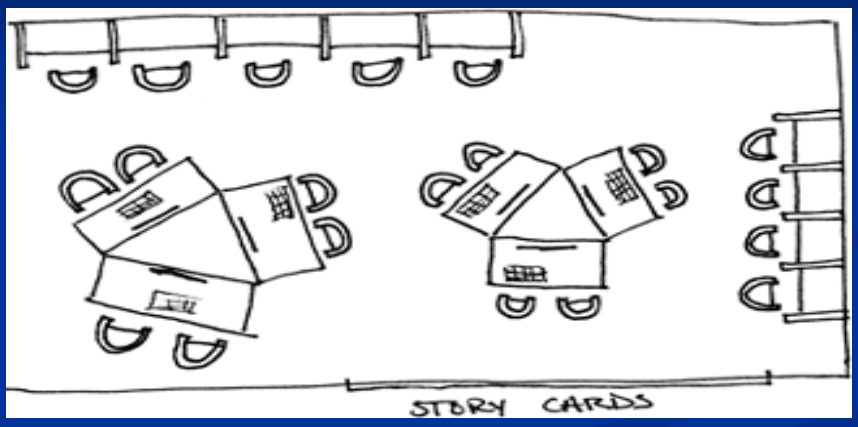
Continuous Improvement

Traditional strategy:
Avoid problems

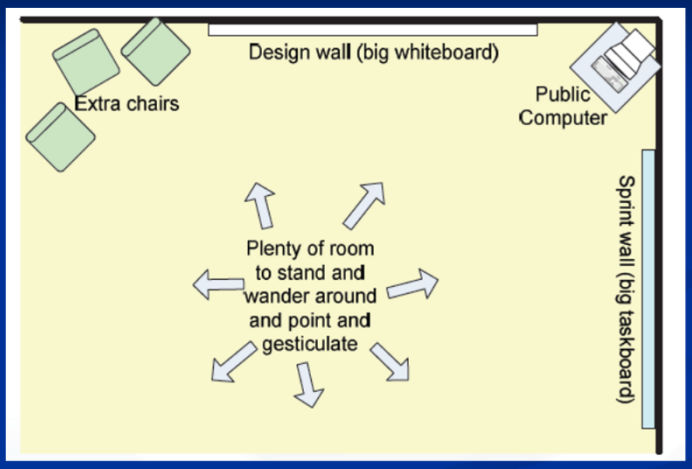


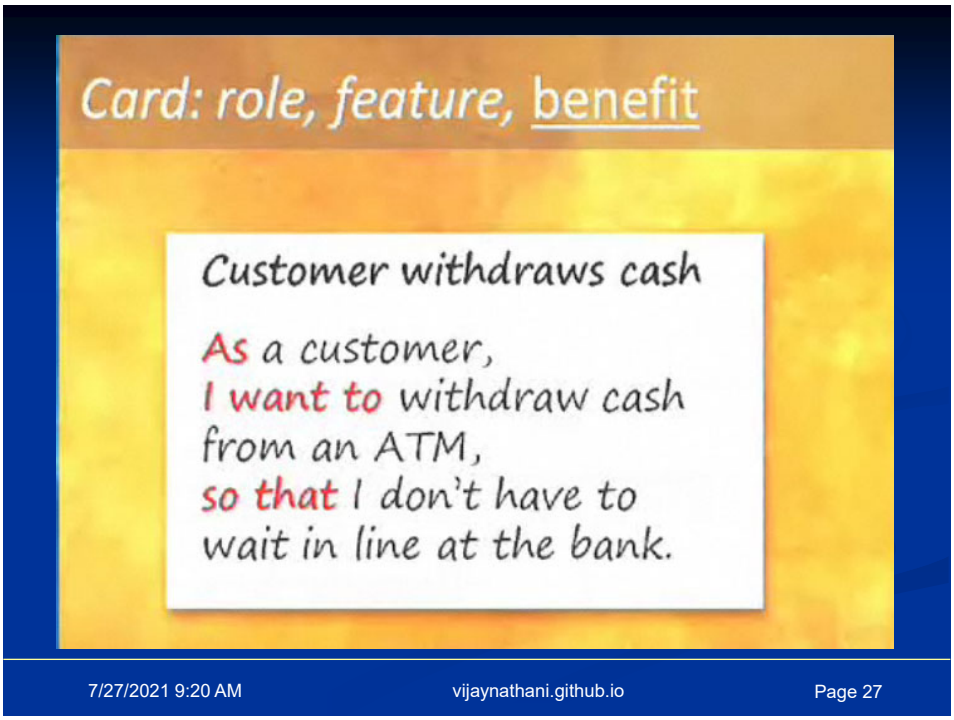
Agile strategy:
Find and remove problems

Caves & Commons



Design Corner





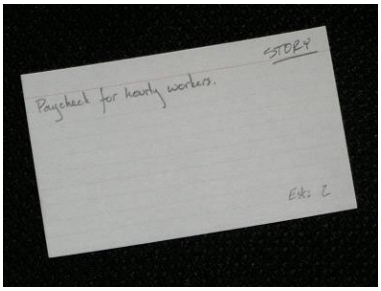
User Story Example

“I would have written a shorter letter, but I didn’t have time” – Mark Twain.

The main purpose of a story card is to act as a reminder, to discuss the feature.

In XP, the estimate is given in ideal hours. These are written at the corner of the card.

Welcome Change

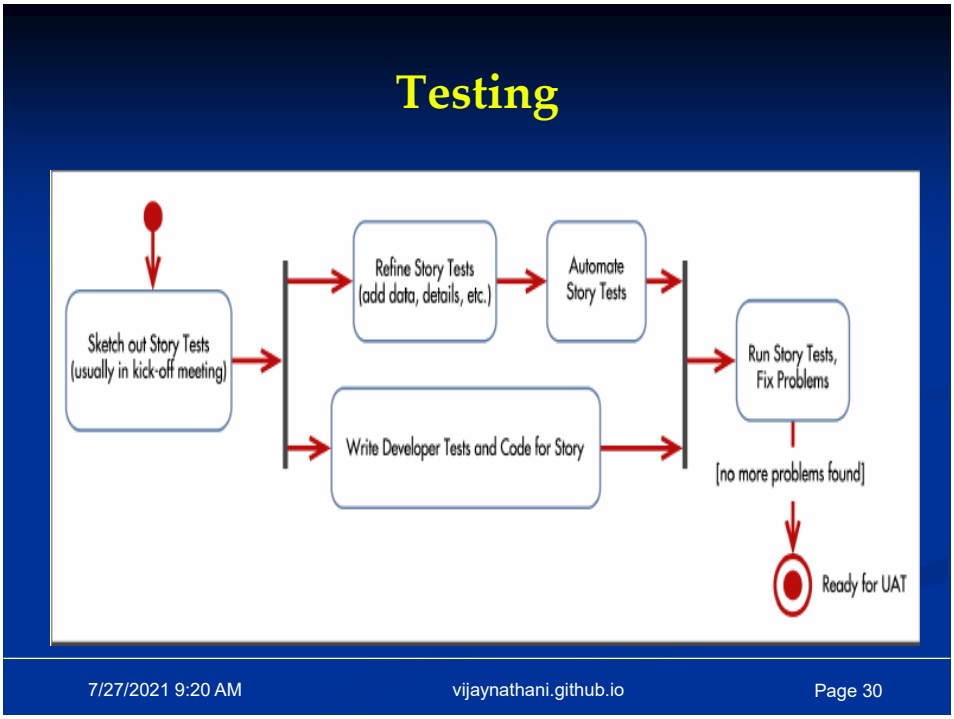


User Story Characteristics

- I – Independent
- N – Negotiable
- V – Valuable
- E – Estimatable
- S – Small
- T – Testable

A User story should be closed i.e. People must get a feeling of accomplishment, when it is done.

Every User story is 0% done or 100% done.

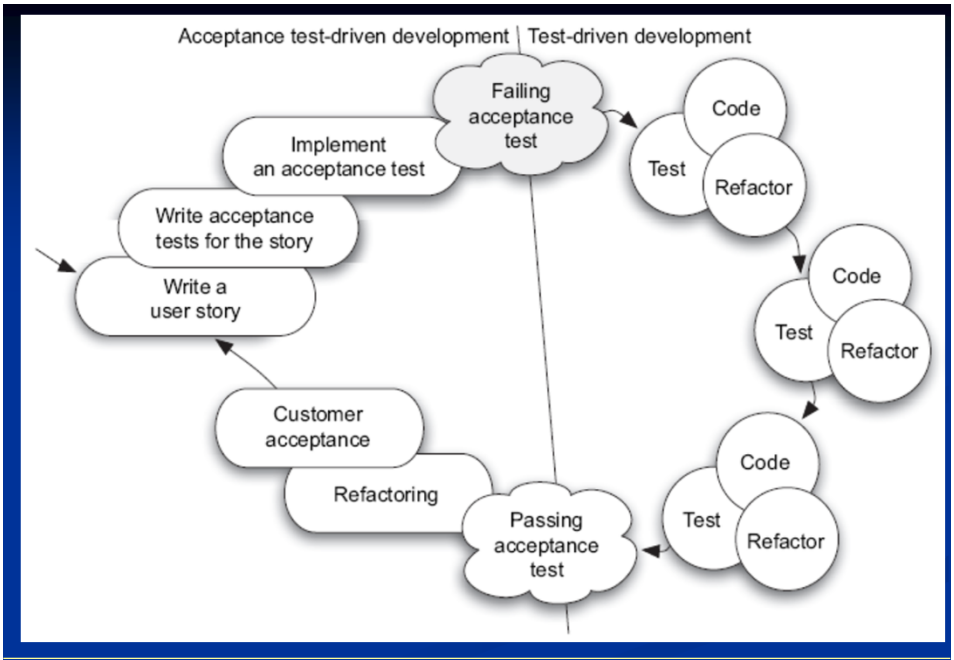


Acceptance Tests are the requirements artifacts

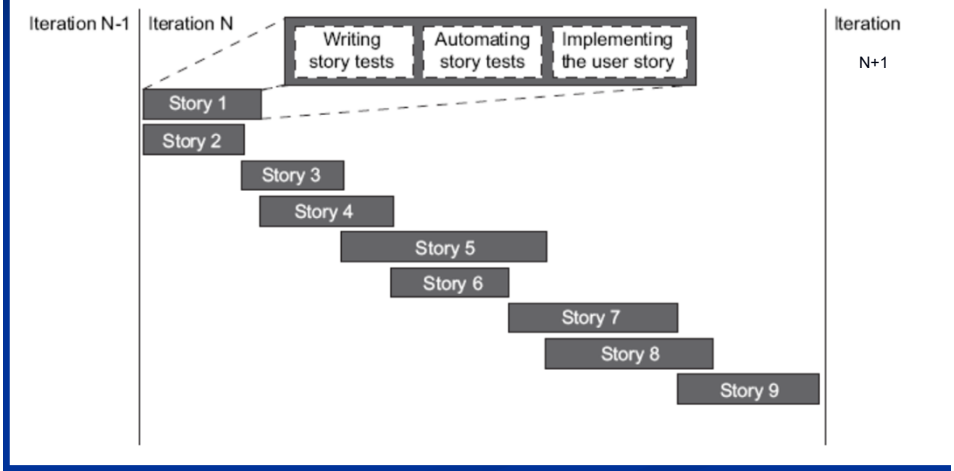
Unit Tests are the Detailed Design artifacts

Testers on an XP team help customers choose and write automated functional tests in advance of implementation

Much of the responsibility for catching trivial mistakes is accepted by the programmers.


Acceptance test-driven development
Test-driven development
Failing acceptance test
Implement an acceptance test
Write acceptance tests for the story
Write a user story
Customer acceptance
Refactoring
Passing acceptance test
Code
Test
Refactor
Code
Test
Refactor
Code
Test
Refactor

Acceptance TDD timeline



Automated TDD are

- Executable unambiguous requirements
- Inexpensive for developers to run
- Regression testing ability
- Release process confidence
- High ROI in the long run

Late Testing Vs. Early Testing

Repeated Testing Barriers

Test driven development vs. Traditional Testing

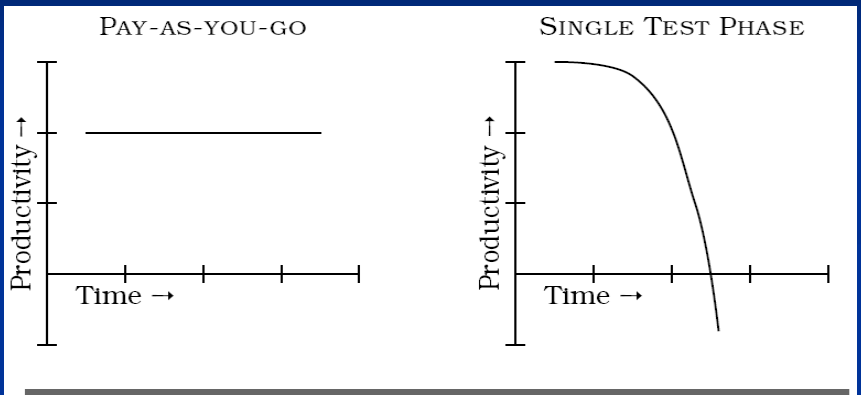


Figure – Comparison of Paying-as-you-go vs. Having a Single Testing Phase

With manual testing, the more stressed the team, the more mistakes the team members make in both coding and testing.

With automated testing, running the tests themselves is a stress-reliever.

Health of the software is more important than the quality

"The act of writing a unit test is more an act of design than of verification" - Robert Martin

"Test-driven development seeks specification, not validation, letting you think through your design before you write your functional code" - Scott Ambler

"Test-Driven Development is a powerful way to produce well designed code with fewer defects" - Martin Fowler

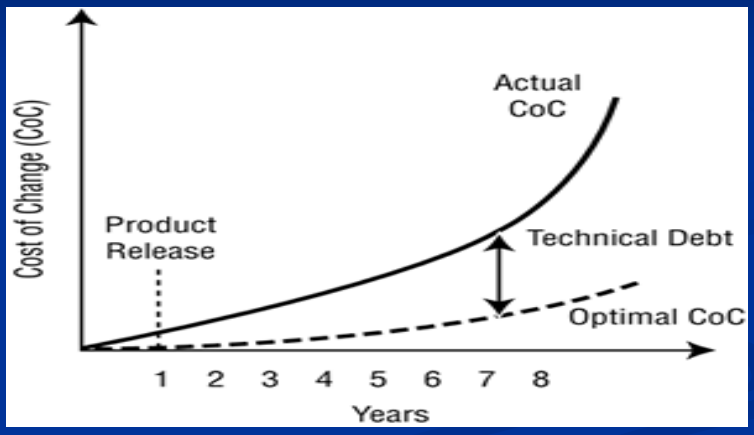
"Fewer defects, less debugging, more confidence, better design, and higher productivity in my programming practice" - Kent Beck

"The best way that I know to write code is to shape it from the beginning with tests" - Ron Jeffries

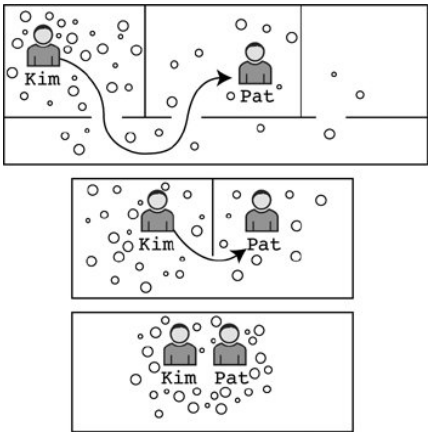
Everything is Refactorable

- We refactor code and tests
  - … Removes dead code, eases maintenance
- We refactor designs
  - … improves understandability.
- We refactor architectures
  - ..crystallizes concept and eases extensibility
- We refactor teams
  - … applies and develops skills as needed.
- We refactor plans
  - … enable better predictability and coordination
- We refactor our process methods
  - … adapts and improves ability to satisfy objectives.

Haste makes Waste

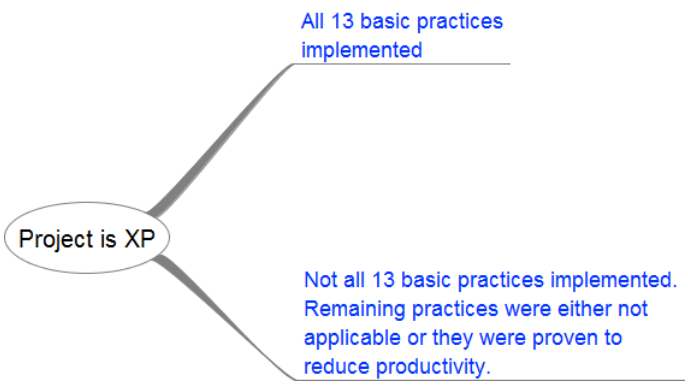


Bus Factor



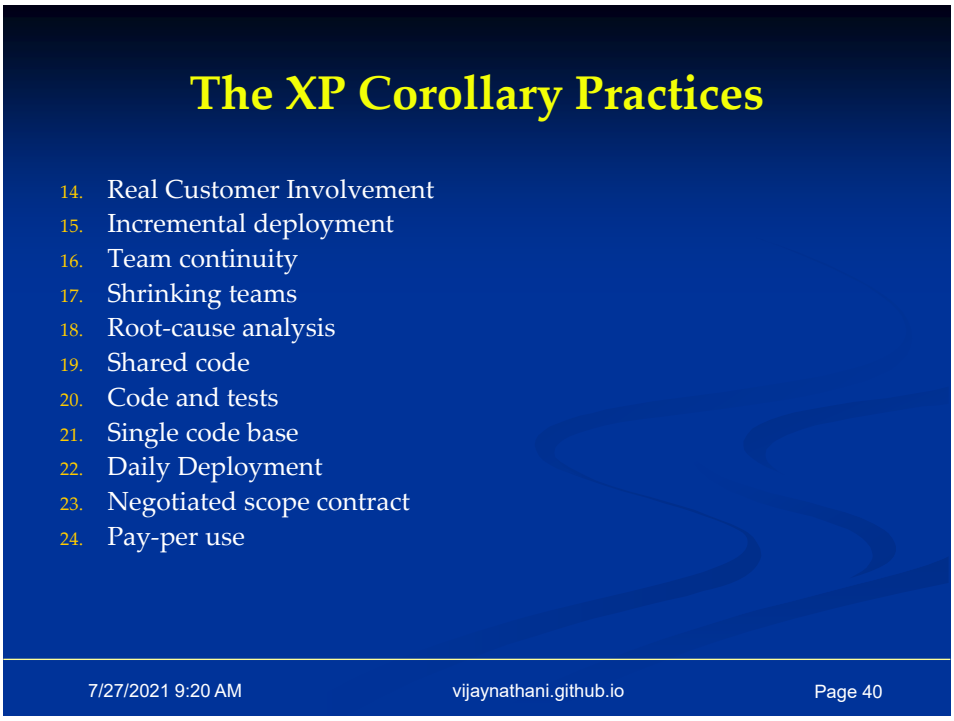
How many people have to be hit by a bus for the project to stop?

Is my Project XP?



The XP Basic Practices

1. Sit together
2. Whole Team
3. Informative Workspace
4. Energized work
5. Pair Programming
6. Stories
7. Weekly cycle
8. Quarterly cycle
9. Slack
10. Ten-minute build
11. Continuous Integration
12. Test-first programming
13. Incremental Design



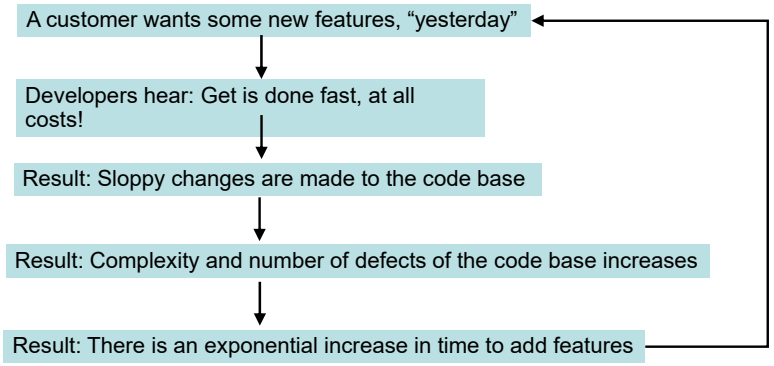
Flicr does 10+ deployments everyday. (Documented in an agile talk by James Shore)

Stackoverflow deploys everyday (http://itc.conversationsnetwork.org/audio/download/ITC.SO-Episode70-2009.10.13.mp3)

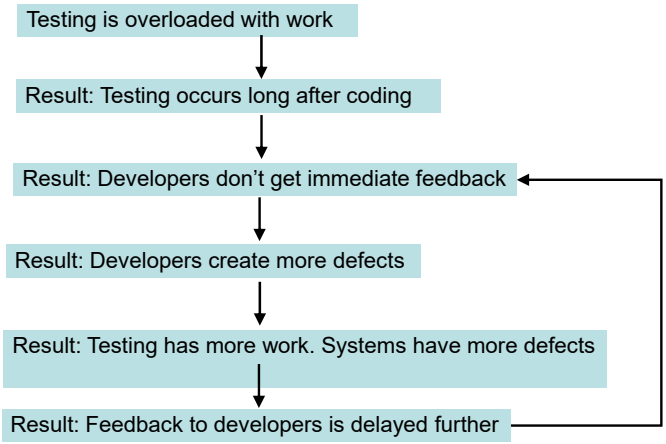
Amazon deploys everyday.

IMVU.com does continuous deployment e.g. deployment 50 times a day. For every checkin, if all tests pass, deploy.

Vicious Cycle



Vicious Cycle

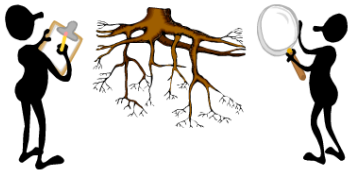


Stop The Line Mentality

- Detect abnormalities the moment they happen
- Don't fight fires!

Stop!

- Find and fix the problem
  - Determine for the root cause
    Why? Why? Why? Why? Why?
  - Investigate Countermeasures
    Try several and see what works
  - Implement the best solution
    Keep on looking for a better way



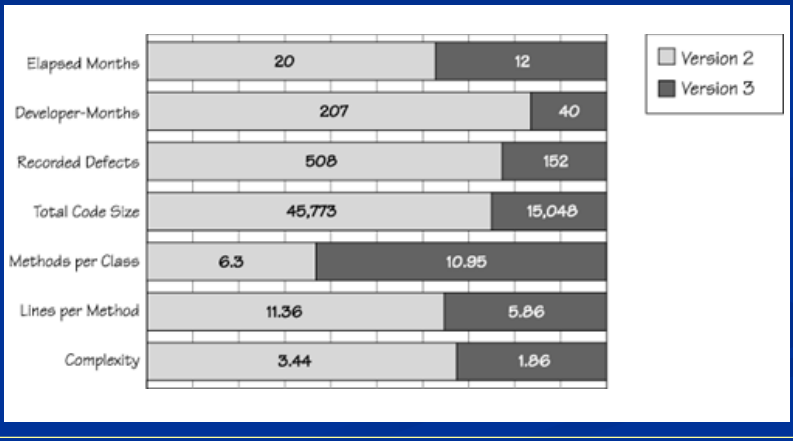
Demming. Most of the time the process is at fault and not the people.

Change the Process

- Developer didn't write the code correctly
  - Write Unit Tests
- Developer misunderstood what the customer wanted
  - Specify acceptance tests early
- Customer realized later that they mis-spoke the requirement
  - Specify acceptance tests early
- Customer spoke it properly, but realized they asked for the wrong thing once they saw what was delivered
  - Don't take a long time to show customers what you've done
- Customer spoke it properly, realized they got what they originally wanted, but they now have a better idea
  - Don't take a long time to show customers what you've done

Benefits of XP?

- Escrow.com needed a B2B e-commerce software



Version 2 was developed with predictive methodology

Version 3 was developed with XP

Example in Extreme Programming Perspectives. Both version added equal number of functionality.

Role of Project Manager

- Project manager is a facilitator and not a controller.
- Project Manager facilitates communication inside the team and coordinate communication with customers, suppliers, and the rest of the organization.

They act as team historians, reminding the team how much progress it has made.

To remain accurate, the information changes frequently; which gives project managers the challenge of communicating changes helpfully.

Role of Product Manager

- Product managers write stories, pick themes and stories in the quarterly cycle, pick stories in the weekly cycle, and answer questions as implementation uncovers under-specified areas of stories.

A plan is an example of what could happen, not a prediction of what will happen.

Stories should be sequenced for business, not technical, reasons.

The system should be "whole" at the end of every iteration.

Role of Programmers

- Programmers work in close technical collaboration with each other, pairing on production code, so they need to develop good social and relationship skills.
- Specifically the Programmers
  - Estimate stories and tasks
  - Break stories into tasks
  - Write tests
  - Write code to implement features
  - Automate tedious development process
  - Gradually improve the design of the system.