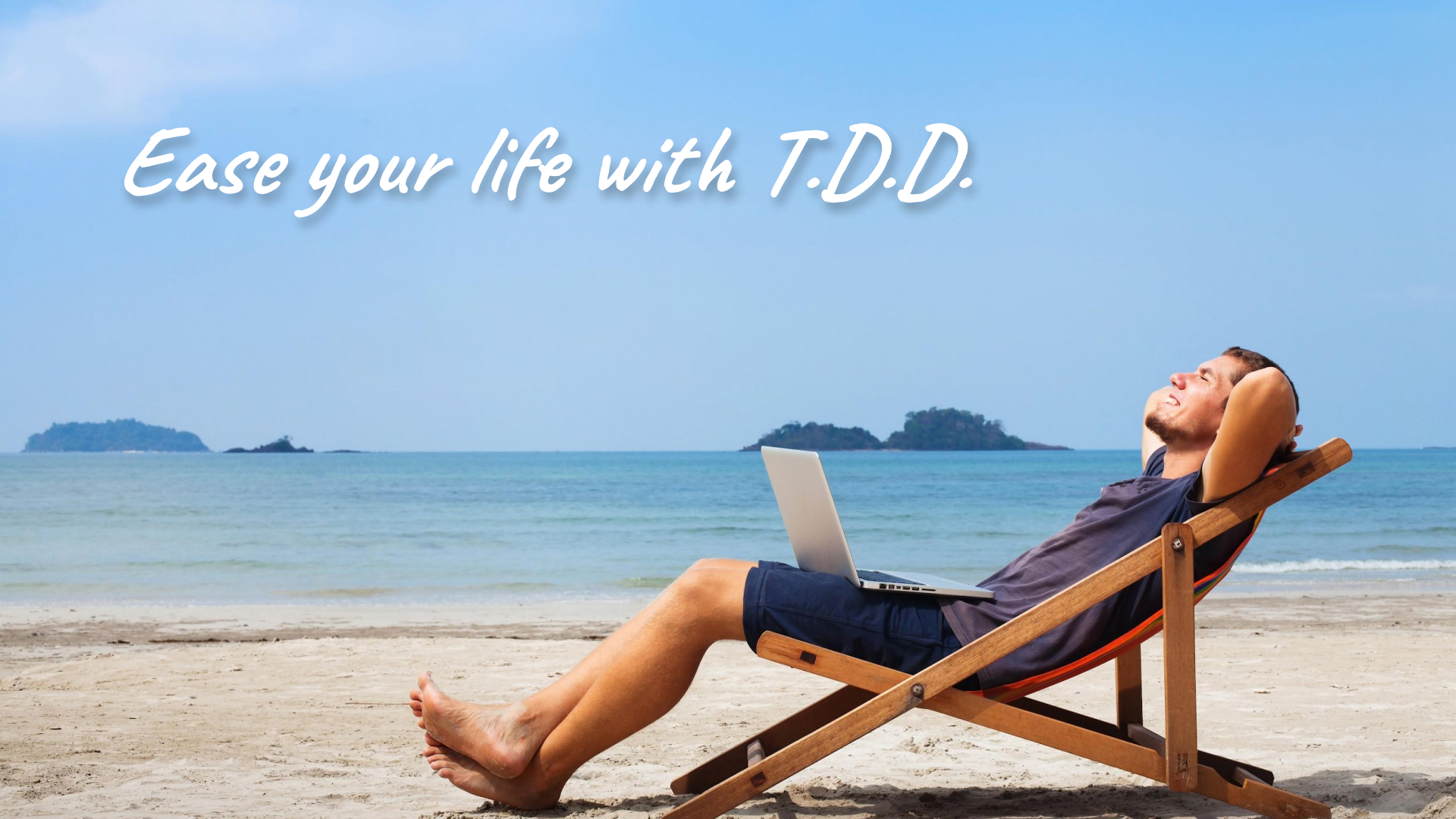
Ease your life with T.D.D.



Contents

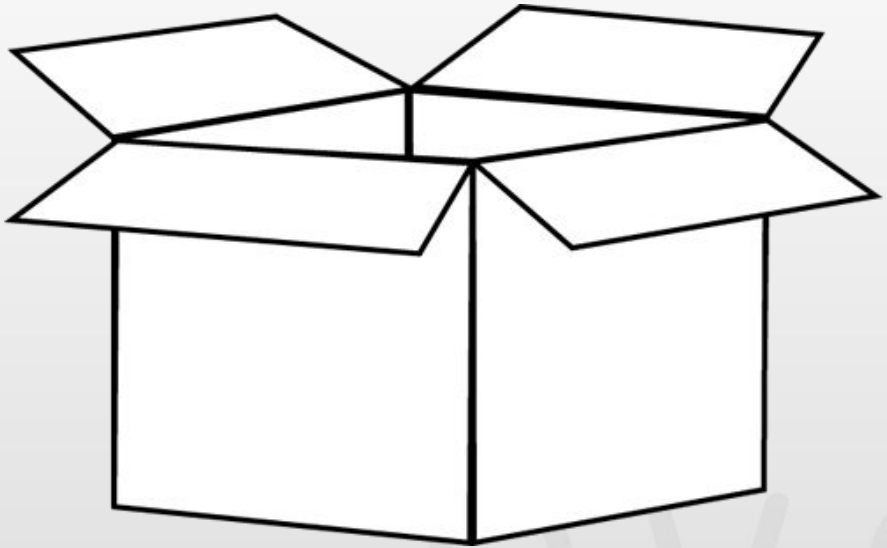
- Introduction to software testing
- What is wrong with test-last development?
- What is Test-Driven Development?
- Why TDD is important?
- TDD misconceptions
- Acceptance TDD
- A short note about XP

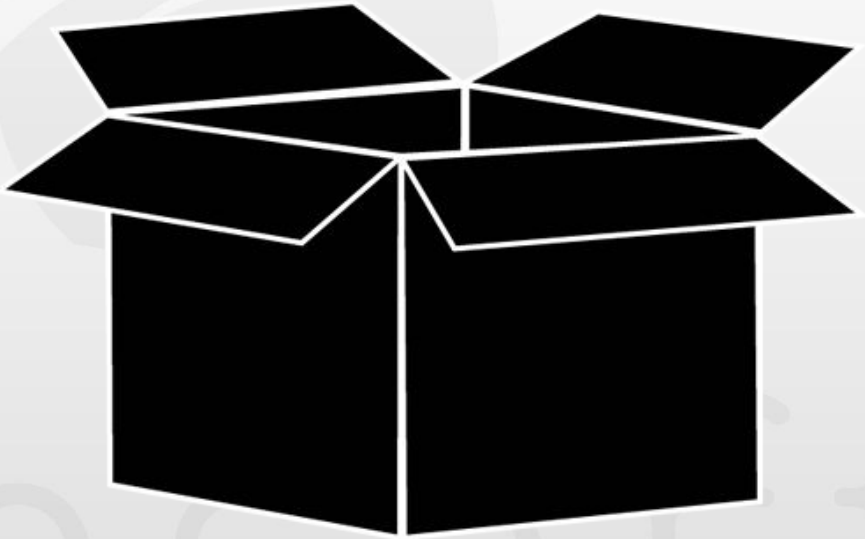
Introduction to software testing

- Test: a procedure intended to establish the quality, performance, or reliability of something, especially before it is taken into widespread use.

- Software testing involves the execution of a software component or system component to evaluate one or more properties of interest:
  - meets the requirements that guided its design and development
  - responds correctly to all kinds of inputs
  - performs its functions within an acceptable time
  - is sufficiently usable
  - can be installed and run in it's intended environments
  - achieves the general result it's stakeholders desire

- Software testing is a verification and validation process
  - Verification: are we building the product right?
  - Validation: are we building the right product? *(see ATDD)*






ROMER

Test doubles

- removes external dependencies
- creates known state
- focus test on a specific code path
- types:
  - **Stubs**: provides canned or no answer to calls during the test
  - **Fakes**: working implementation with some shortcuts (not suitable for prod.)
  - **Mocks**: objects pre-programmed with expectations what can be verified

Types of testing

- Unit tests
- Integration tests
- Functional/Acceptance tests
- Performance tests
- Regression tests
- Manual tests

Types of testing

- Unit tests
  - shows that individual parts works correctly in isolation
  - testing smallest unit of functionality
  - fast and reliable
  - everything should be done in memory (network, file system, database)
  - any dependency that is hard to understand, initialise or manipulate should be stubbed or mocked
  - cover most of your codebase
  - use for development and refactoring
  - living documentation

Types of testing

- Integration tests
  - individual modules are tested as a group
  - verify the communication paths and interactions between components
  - much slower than unit test
  - more complicated to set-up and debug

Types of testing

- Functional tests
  - check a particular feature for correctness
  - intermediate results are not relevant (it works or doesn't)
  - internal program structure is rarely considered
  - required applications should be up and running (nothing is mocked)
  - verifies a program by checking against specifications
  - very hard to create and maintain
  - very slow, should run once a day
  - logs are important to track problems

Types of testing

- Acceptance tests
  - particular type of functional tests
  - if all passes, it means software should meet all user requirements and it is feature complete

Types of testing

- Performance tests
  - used to determine the system's behavior under normal and peak load conditions
  - helps to identify maximum operating capacity and bottlenecks
  - requires a specific production-like environment

Types of testing

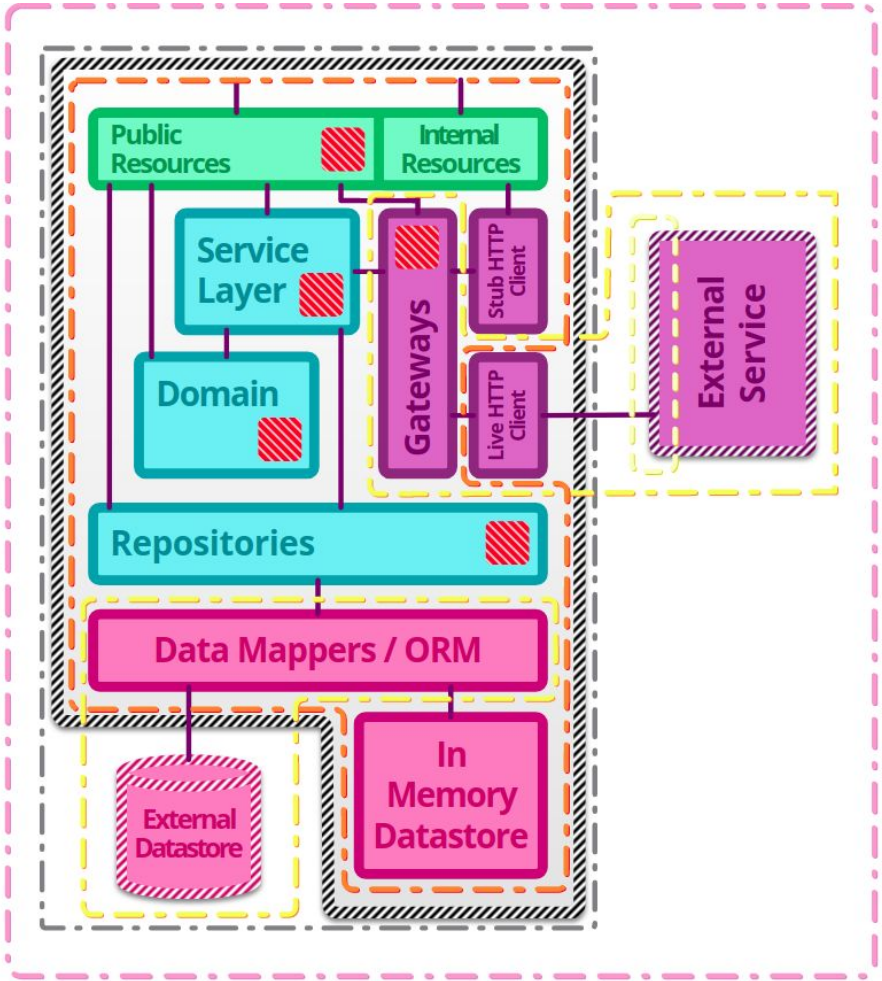
- Regression tests
  - verifies that changes does not introduced any faults
  - new bugs or regressions may be uncovered
  - requires a specific production-like environment

Types of testing

- Manual tests
  - human testers checks predefined scenarios
  - main flows are tested usually
  - requires a specific production-like environment
  - checking the correct behavior prior to release to end users

Unit tests : exercise the smallest pieces of testable software in the application to determine whether they behave as expected.

Integration tests : verify the communication paths and interactions between components to detect interface defects.

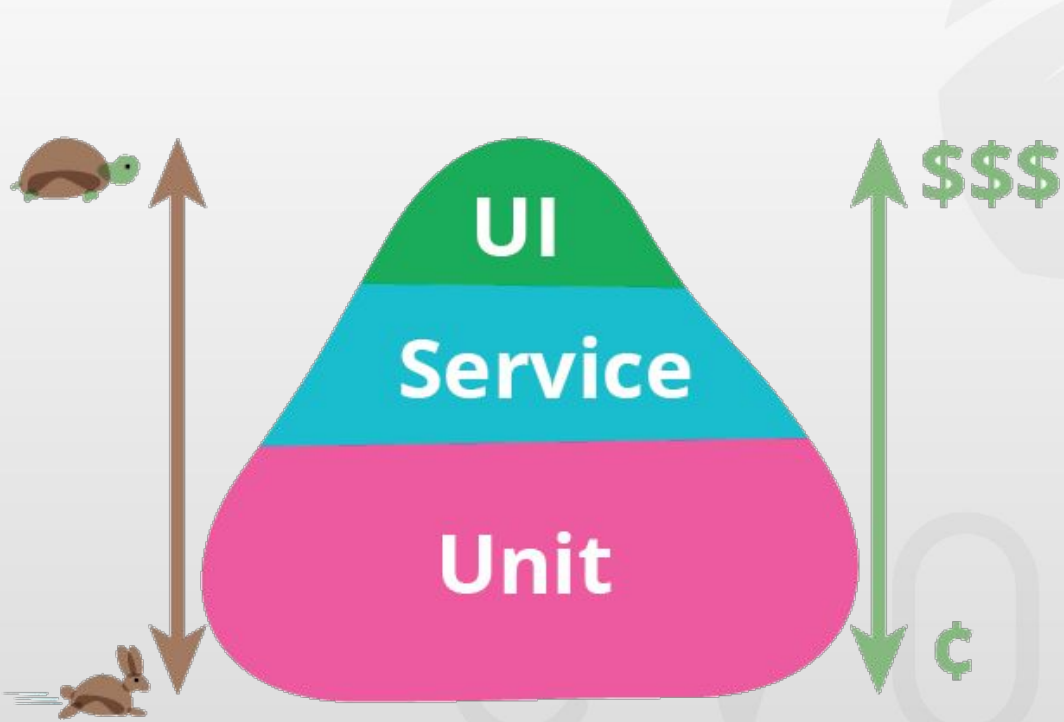


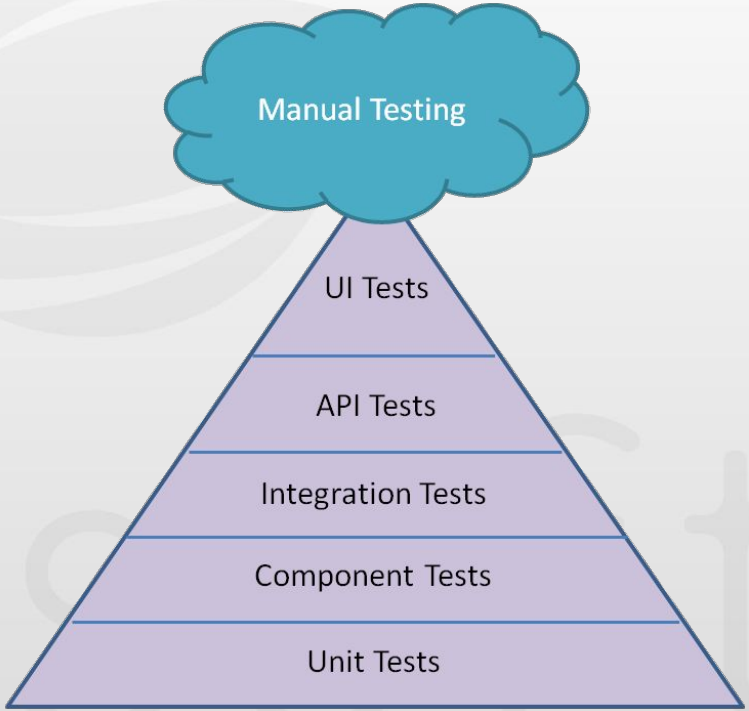
Component tests : limit the scope of the exercised software to a portion of the system under test, manipulating the system through internal code interfaces and using test doubles to isolate the code under test from other components.

Contract tests : verify interactions at the boundary of an external service asserting that it meets the contract expected by a consuming service.

End-to-end tests : verify that a system meets external requirements and achieves its goals, testing the entire system, from end to end.

Testing pyramid





Bad tests are only slightly better than NO tests

- *“Code without tests is a bad code”* - Michael C. Feathers

- Project(s) without tests often end up looking like they are stuck together with duct tape… Change one part and the other part stops working

99 little bugs in the code.
99 little bugs in the code.
Take one down, patch it around.

127 little bugs in the code...


RELEASE
ADDING NEW FEATURES
ADDING NEW FEATURES
MAINTAINING OLD FEATURES


DEBT

Smells

- constructor/method/class does too much work (SRP)
- instantiation by using `new` keyword (hard-coded dependencies)
- anything more than field assignment in constructor
- too many fields
- class has fields used only in some methods
- too many dependencies
- excessive scrolling
- high maintenance cost

Smells

- no / meaningless asserts in tests
- testing private / protected methods
- extensive setup / teardown
- brittle tests
- slow tests

Don’t let your software rot!

“If the answer is not obvious,

or it looks like the test would be ugly or hard to write,

then take that as a warning signal.

Your design probably needs to be modified;

change things around until the code is easy to test,

and your design will end up being far better for the effort”

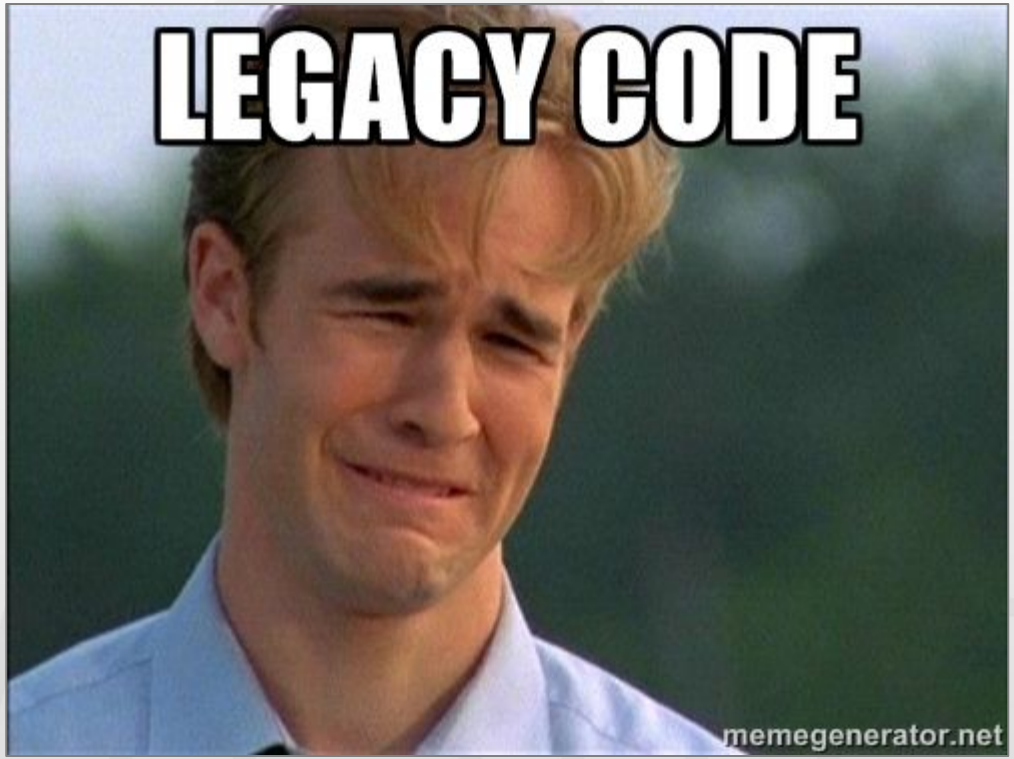
/ Thomas Hunt /

Problems with test-last approach:

- errors in production
- programmers moves onto other projects
- test and code written by different programmers
- tests based on outdated information
- infrequently testing
- fixes that create other problems

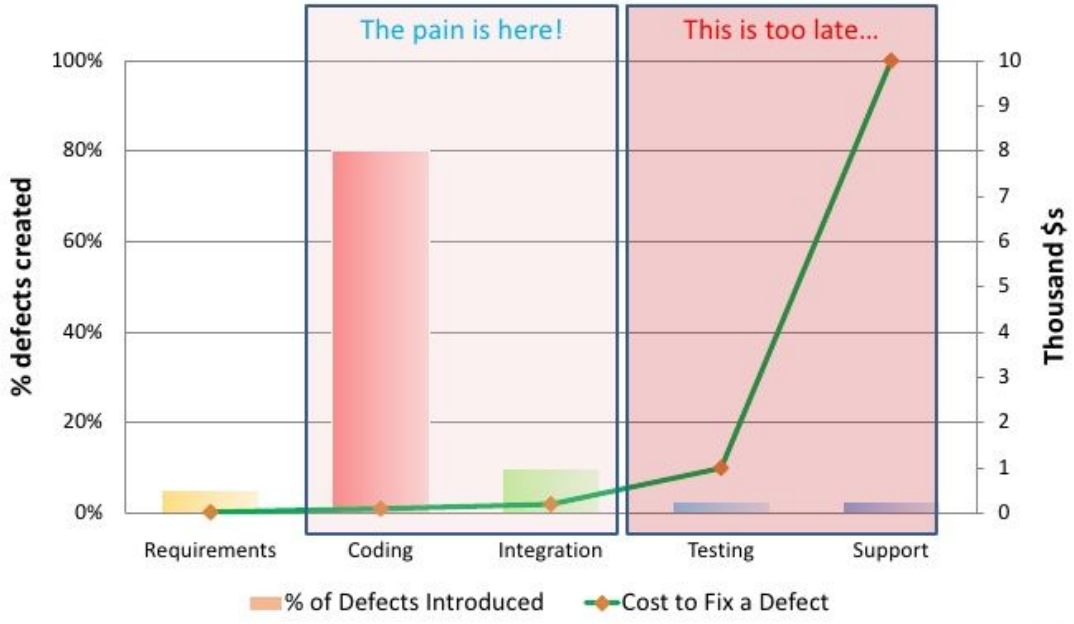
Legacy code is:

- code without tests
- code you’re afraid to change




One day, son...
All this legacy code will be yours

WHERE DOES IT HURT



creuna

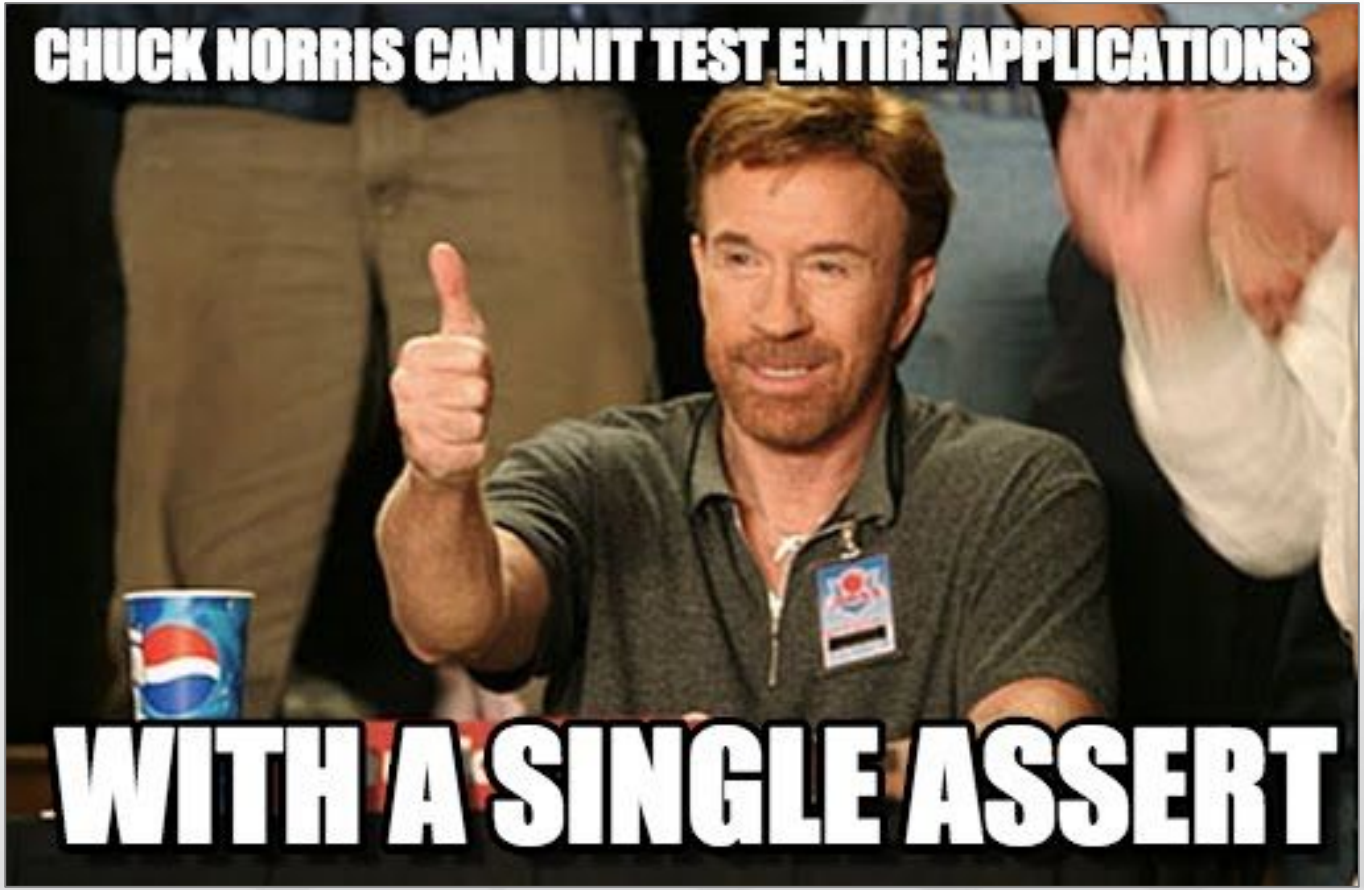

TOTALLY
COOL
PIX
TEST-LAST DEVELOPMENT
IS PAIN IN THE ASS

How to write good tests

- *“The secret of testing is writing testable code”* - Misko Hevery

How to write good tests

- the name should clearly describe the purpose of the test
- Given / When / Then + readability
- each test should be for a single concept (1 assert)
- test in isolation


CHUCK NORRIS CAN UNIT TEST ENTIRE APPLICATIONS
WITH A SINGLE ASSERT

How to write good tests

- the name should clearly describe the purpose of the test
- Given / When / Then + readability
- each test should be for a single concept (1 assert)
- test in isolation
    - **F**ast: to run them frequently
    - **I**ndependent: of each other
    - **R**epeatable: in any environment
    - **S**elf-Validating: either passing or failing
    - **T**imely: written before the production code

“Experience is a hard teacher

because she gives the test first,

the lesson afterward.”

/ Chinese proverb /

What is Test-Driven Development?

Test-driven development (TDD) is an evolutionary approach to development

which combines test-first development where you write a test

before you write just enough production code to fulfill that test and refactoring.

In other words, it’s one way to think through your requirements or design

before your write your functional code.

What is Test-Driven Development?

“TDD is risk averse programming,

investing work in the near term

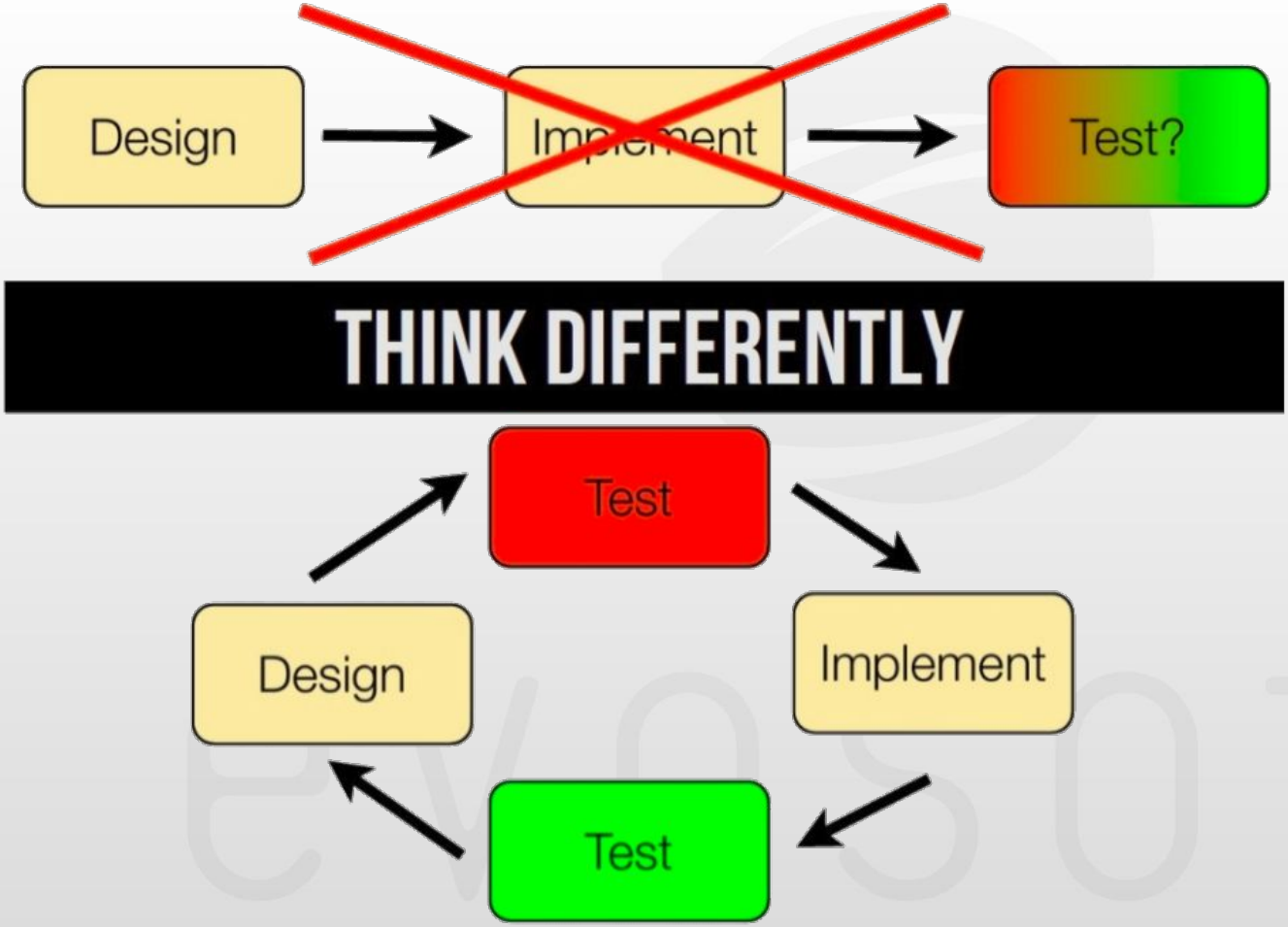
to avoid failures later on”

What is Test-Driven Development?

It’s a design technique:

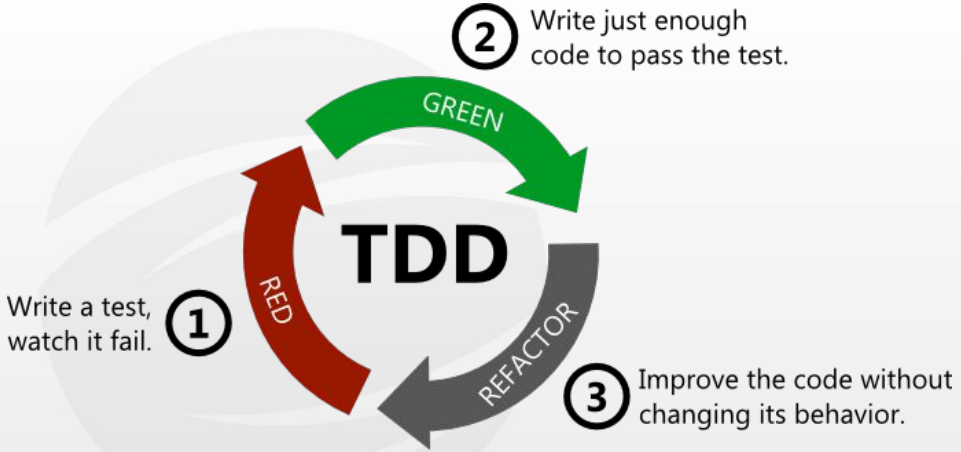
lets you think “how to use” first,

rather than “how to implement”


Design
Implement
Test?
THINK DIFFERENTLY
Test
Implement
Test
Design

The Three Laws of TDD

- You are not allowed to write any production code unless it is to make a failing unit test pass.



- You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.

- You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

Why TDD is important? Why does it work?

- The routine leads the programmers to think about details they otherwise don't.
- Specifically, test cases are thoughts through before the programmer is allowed to think about the "interesting part" of how to implement the functionality.
- Encourages "divide-and-conquer".
- Programmers are never scared to make a change that might "break" the system.
- The testing time that is often squeezed out of the end of a traditional development cycle cannot be squeezed out.

Advantages and results

- a well designed application
- enables change
- gives confidence / removes fear
- user requirements are more easily understood
- validates your design
- shortens the programming feedback loop (really fast feedback)
- reduced software defect rate and maintenance significantly
- less debug time (hard stuff and surprises are tackled early on)
- brings professionalism to software development / requires discipline

Advantages and results (regarding to code quality)

- flexibility
- readability
- maintainability
- predictability
- simplicity (KISS)
- loosely coupled
- YAGNI (no dead code)
- SOLID (especially SRP)

“Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away”

“The most powerful designs are always the result of a continuous process of simplification and refinement.”

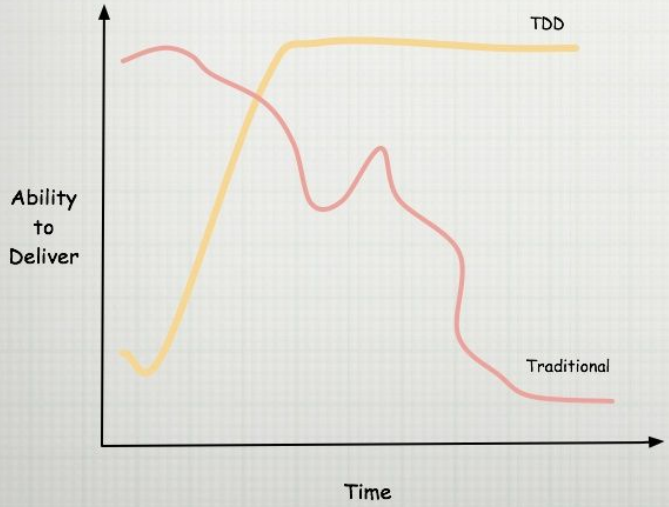
Top 5 excuses for not unit testing / TDD

- I don’t have time to unit test
- The client pays me to develop code, not write unit test
- I am supporting a legacy application without unit tests
- QA and User Acceptance Testing is far more effective in finding bugs
- I don’t know how to unit test, or I don’t know how to write good unit tests


Nice try...
...but ,no
quickmeme.com

But How Much Longer Does TDD Take?

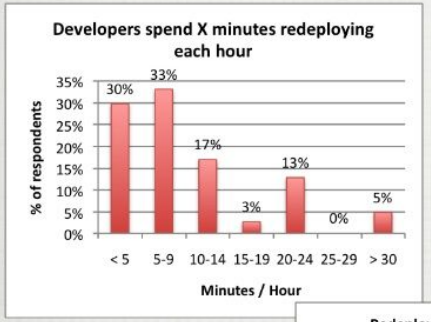
"If it doesn't have to work, I can get it done a lot faster!" - Kent Beck paraphrased

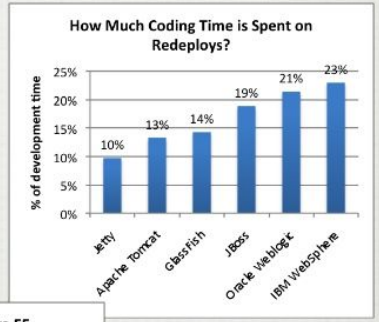


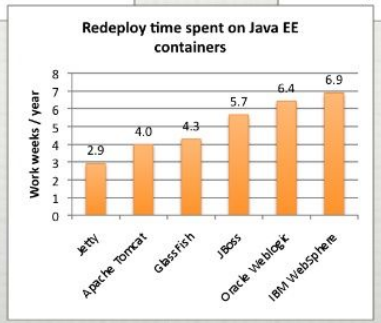
My Experience
- Initial Progress Will Be Slower
- Greater Consistency
- Less Experienced Developer Learning/Ramp-up Time
- I personally think I can develop faster using TDD than without, but it took a while to get there
- Long-term cost is drastically lower

Studies
- Takes 15-30% longer
- 45-90% fewer bugs
- Takes 10x as long to fix a bug in later phases
- One study with Microsoft and IBM: http://research.microsoft.com/en-us/projects/esm/nagappan_tdd.pdf

Speed: Traditional Development



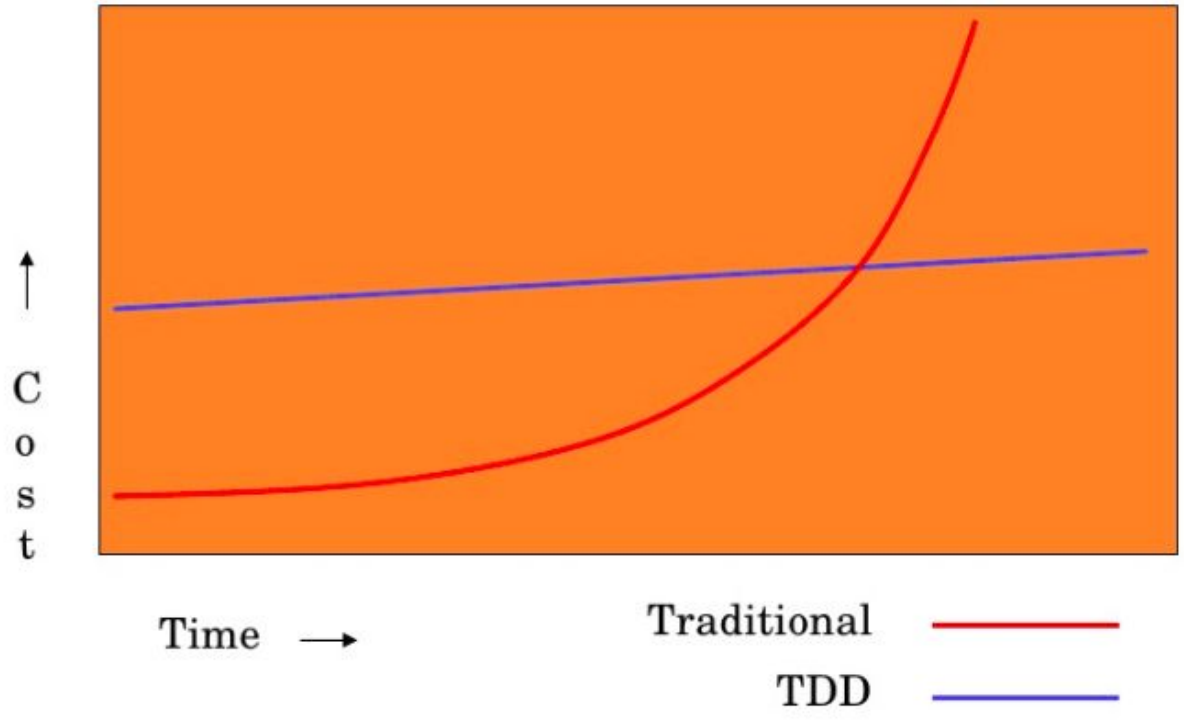




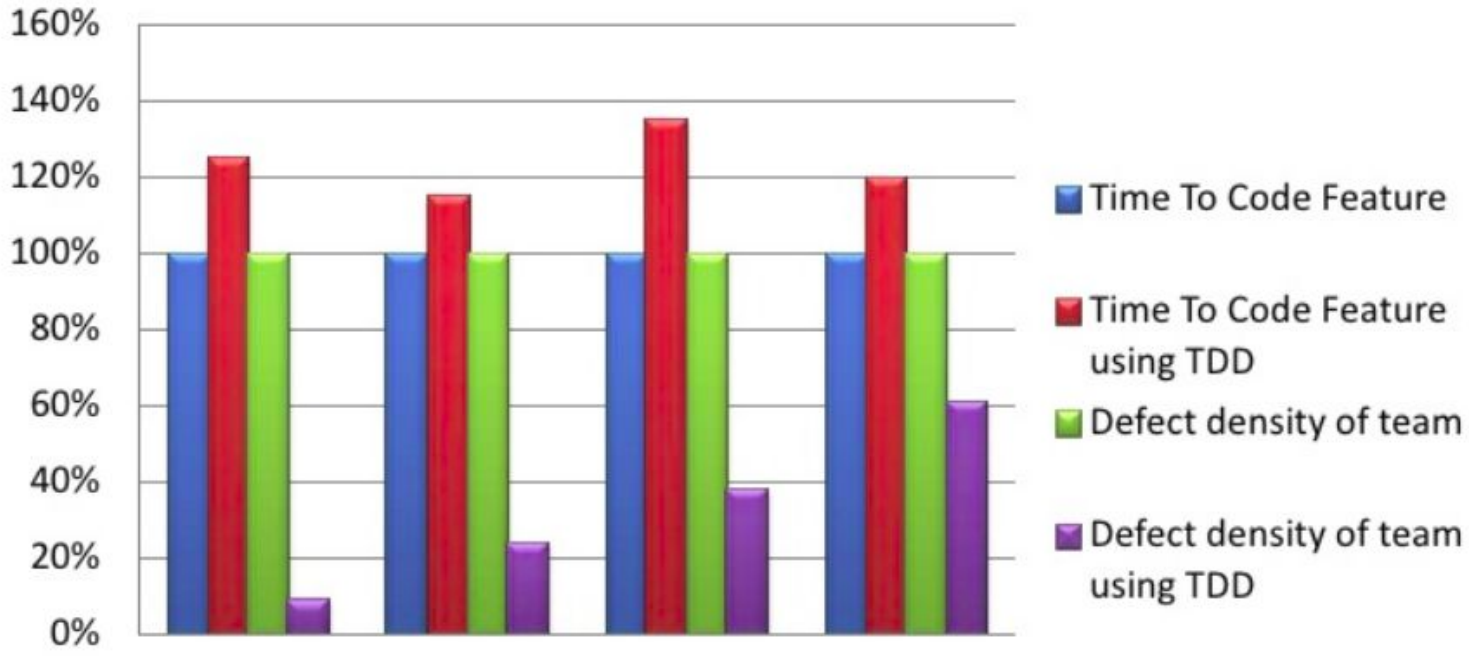
- 5 mins per coding hour becomes 6000 minutes annually (2.5, 40-hour workweeks)
- 9 mins per coding hour becomes 10800 minutes annually (4.5 weeks)
- 14 mins per coding hour becomes 16800 mins annually (7 weeks)

Survey by ZeroTurnaround
http://www.zeroturnaround.com/blog/java-ee-container-redeploy-restart-turnaround-report/

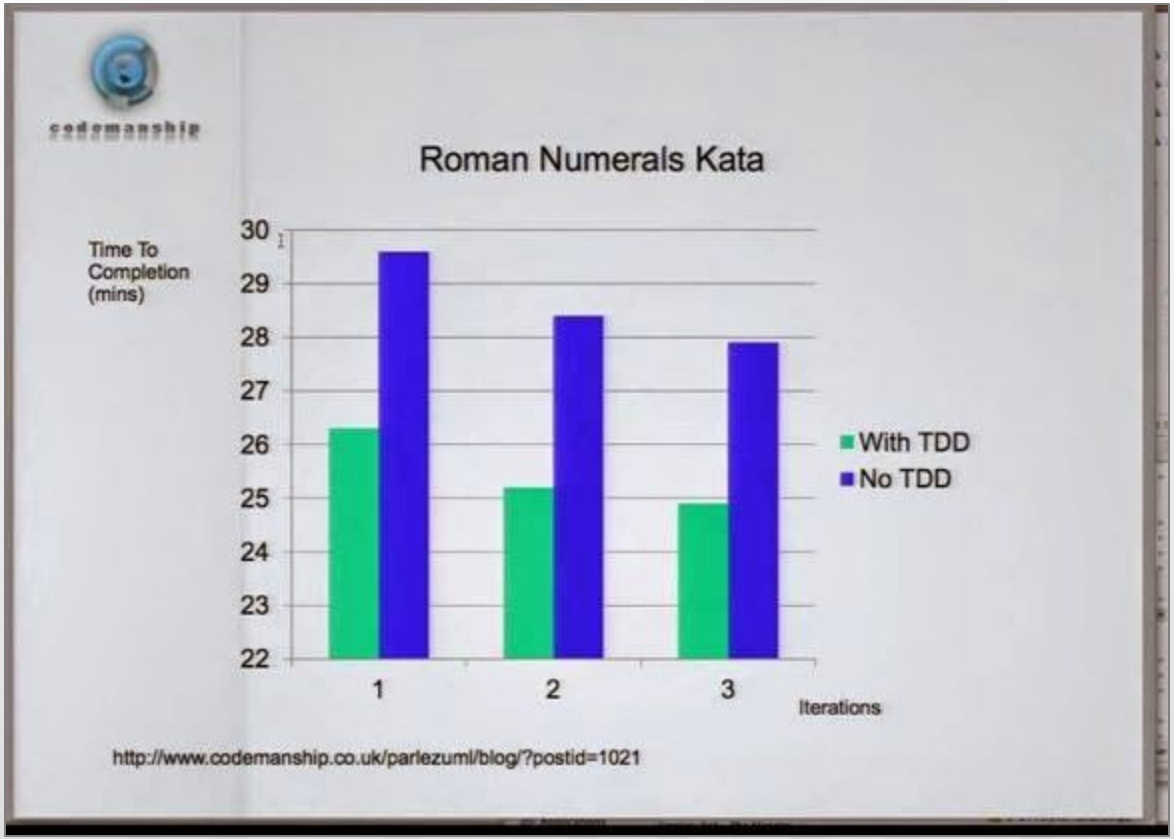
COST OF DEVELOPMENT



IS TDD A WASTE OF TIME (MICROSOFT RESEARCH)



Major quality improvement for minor time investment


codemanship
Roman Numerals Kata
Time To Completion (mins)
30
29
28
27
26
25
24
23
22
1
2
3
Iterations
With TDD
No TDD
http://www.codemanship.co.uk/parlezuml/blog/?postid=1021

Top 5 excuses for not unit testing / TDD

- I don’t have time to unit test
  You will need to spend your time to debug and fix defects
- The client pays me to develop code, not write unit test
  The client pays you to deliver working product
- I am supporting a legacy application without unit tests
  You will be afraid to improve/change it without tests
- QA and User Acceptance Testing is far more effective in finding bugs
  It is rather more costly and time consuming
- I don’t know how to unit test, or I don’t know how to write good unit tests
  Learn it!

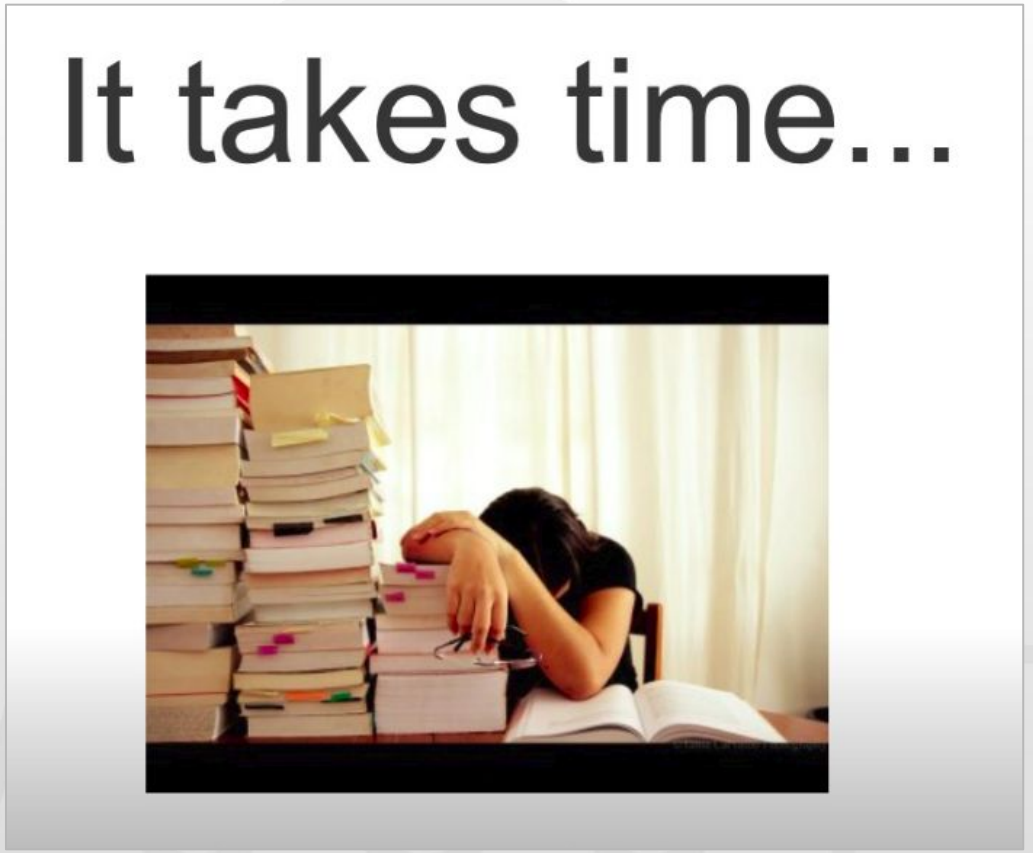
TDD challenges:

- management doesn't often understand "internal quality"
- discipline is required
- developers are often stubborn and lazy
- it is hard for developers to drop bad habits

"I'm not a great programmer; I'm just a good programmer with great habits" - Kent Beck

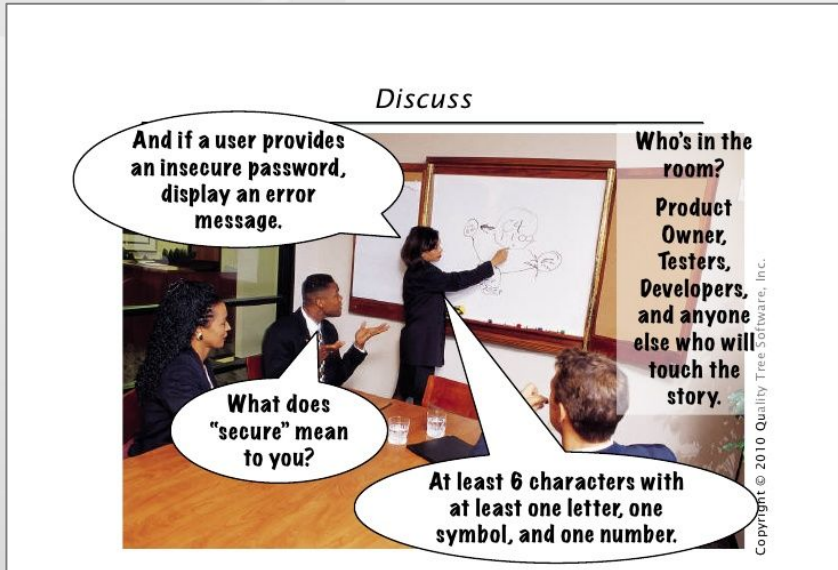
How to learn?

- coding kata
- coding dojo
- baby steps
- pair programming
- patience

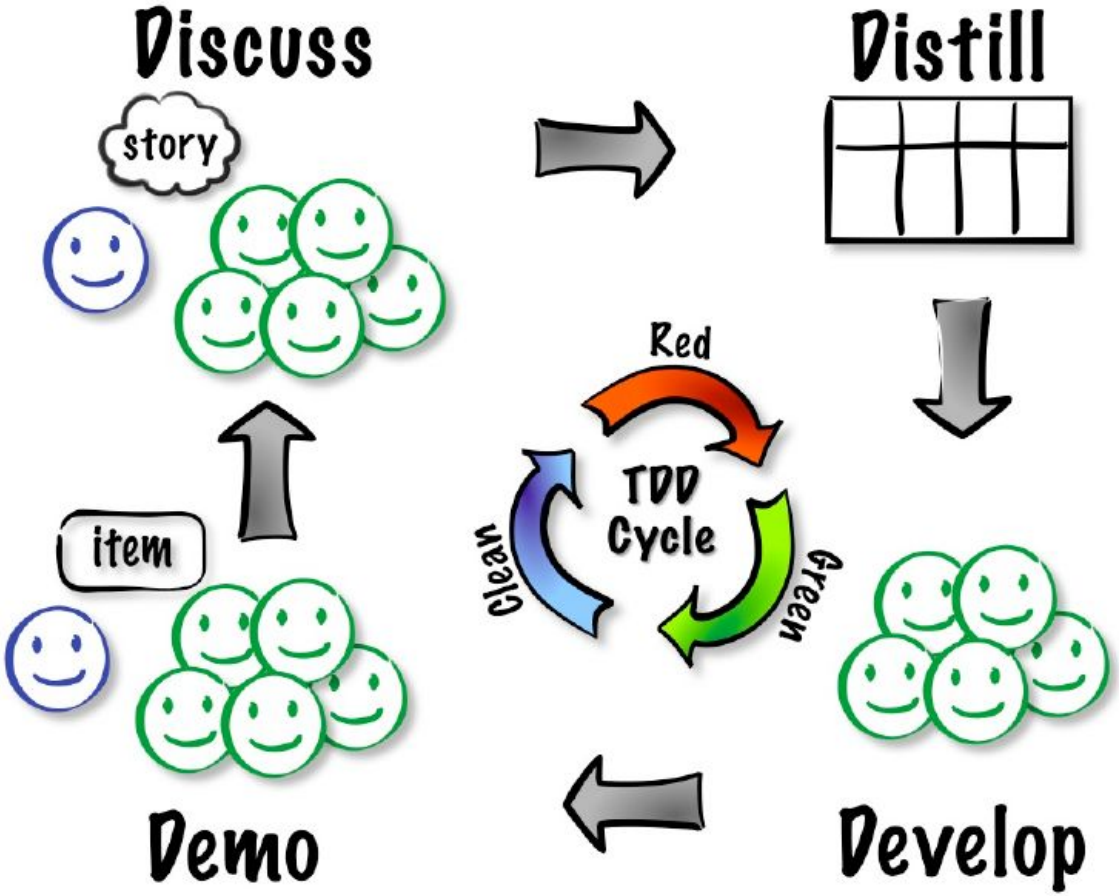


Acceptance Test-Driven Development

- ATDD is a development methodology based on communication between the business customers, the developers, and the testers.

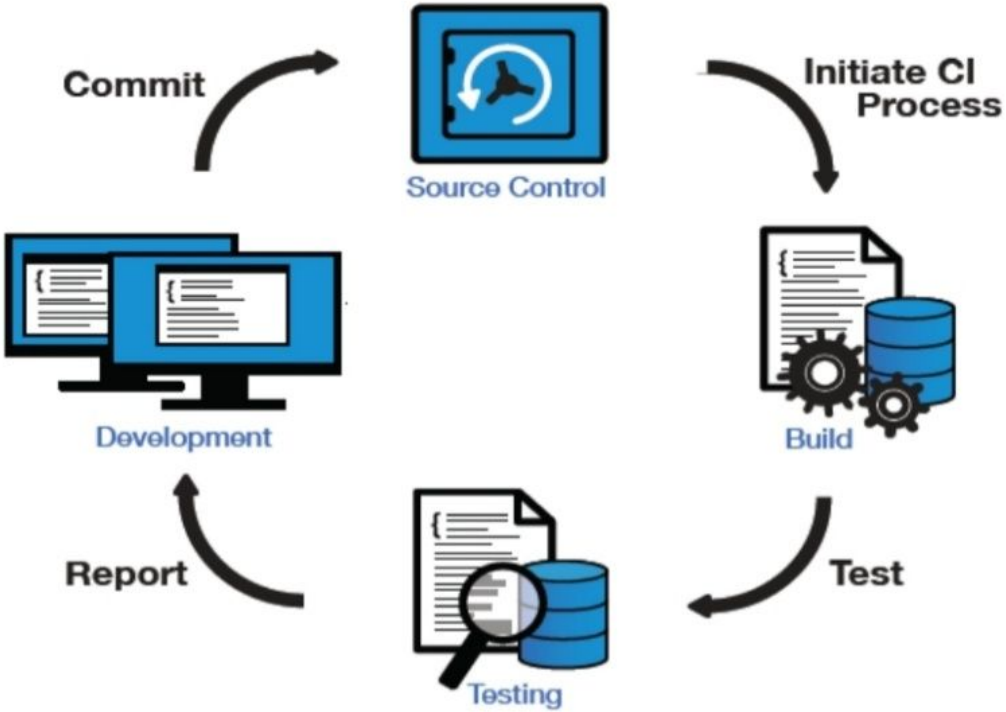


Development (ATDD) Cycle



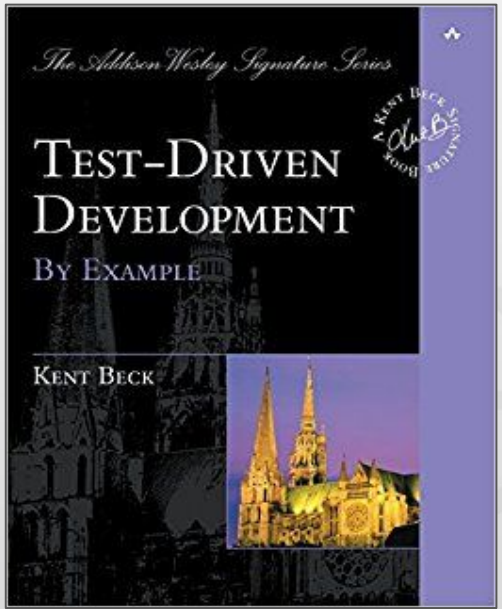
Continuous Integration

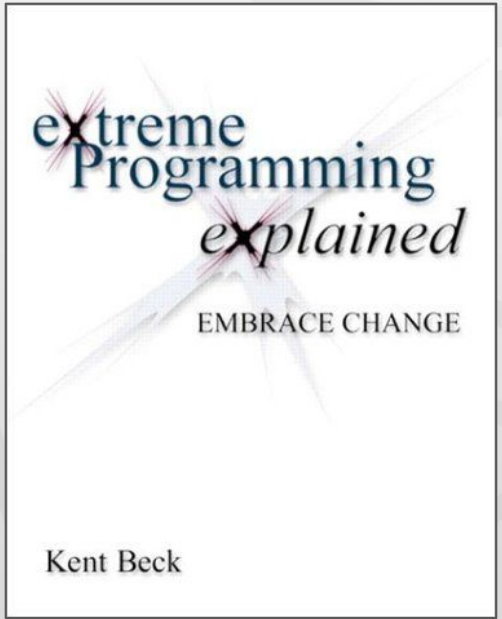
- CI is the practice of merging all developer working copies to a shared mainline several times a day.


Commit
Source Control
Initiate CI Process
Development
Build
Report
Testing
Test

eXtreme Programming

- XP is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements.





eXtreme Programming

- TDD / ATDD
- pair programming
- incremental design
- simplicity (YAGNI)
- clean code
- CI
- frequent small releases
- embracing change
- shared understanding (ATDD)

XP is about

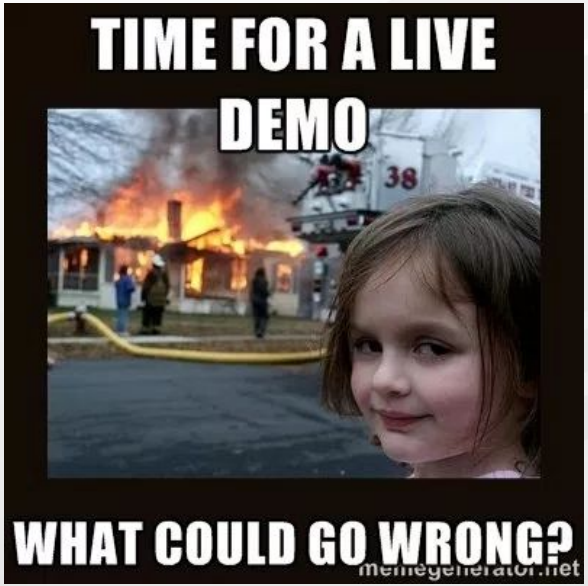
- writing great code
- improving skills and giving up bad habits
- keeping the costs of change low

"No matter what the client says the problem is,
it is always a people problem.
Technical fixes alone are not enough."

“A methodology is only as effective as the people involved”

“A significant advantage of TDD is that it enables you to take small steps when writing software. This is a practice that I have promoted for years because it is far more productive than attempting to code in large steps.”

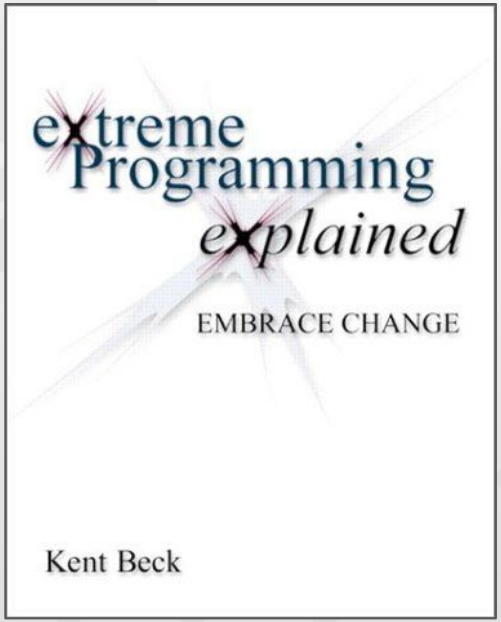
/ Kent Beck /



https://github.com/domahidizoltan/presentation-ease-your-life-with-tdd

Resources

- https://en.wikipedia.org/wiki/Software_testing
- https://martinfowler.com/articles/microservice-testing
- https://www.slideshare.net/lemiorhan/the-engines-of-software-development-testing-and-test-driven-development/
- http://agiledata.org/essays/tdd.html
- https://www.slideshare.net/guestc8093a6/test-driven-development-1000421
- https://www.slideshare.net/haochenglee/test-driven-developmentcontinuousintegration
- https://en.wikipedia.org/wiki/Test-driven_development



Ease your life with T.D.D.

Contact

Evosoft Hungary

Kaposvár utca 14-18

H-1117 Budapest

Web: www.evosoft.com

Counterpart

Zoltán Domahidi

O6.06 / Java Developer

Email: Zoltan.Domahidi@evosoft.com