Agile Software Development

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Waterford Institute *of* Technology

INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Introduction to Test Driven Development

Test Driven Development Introduction

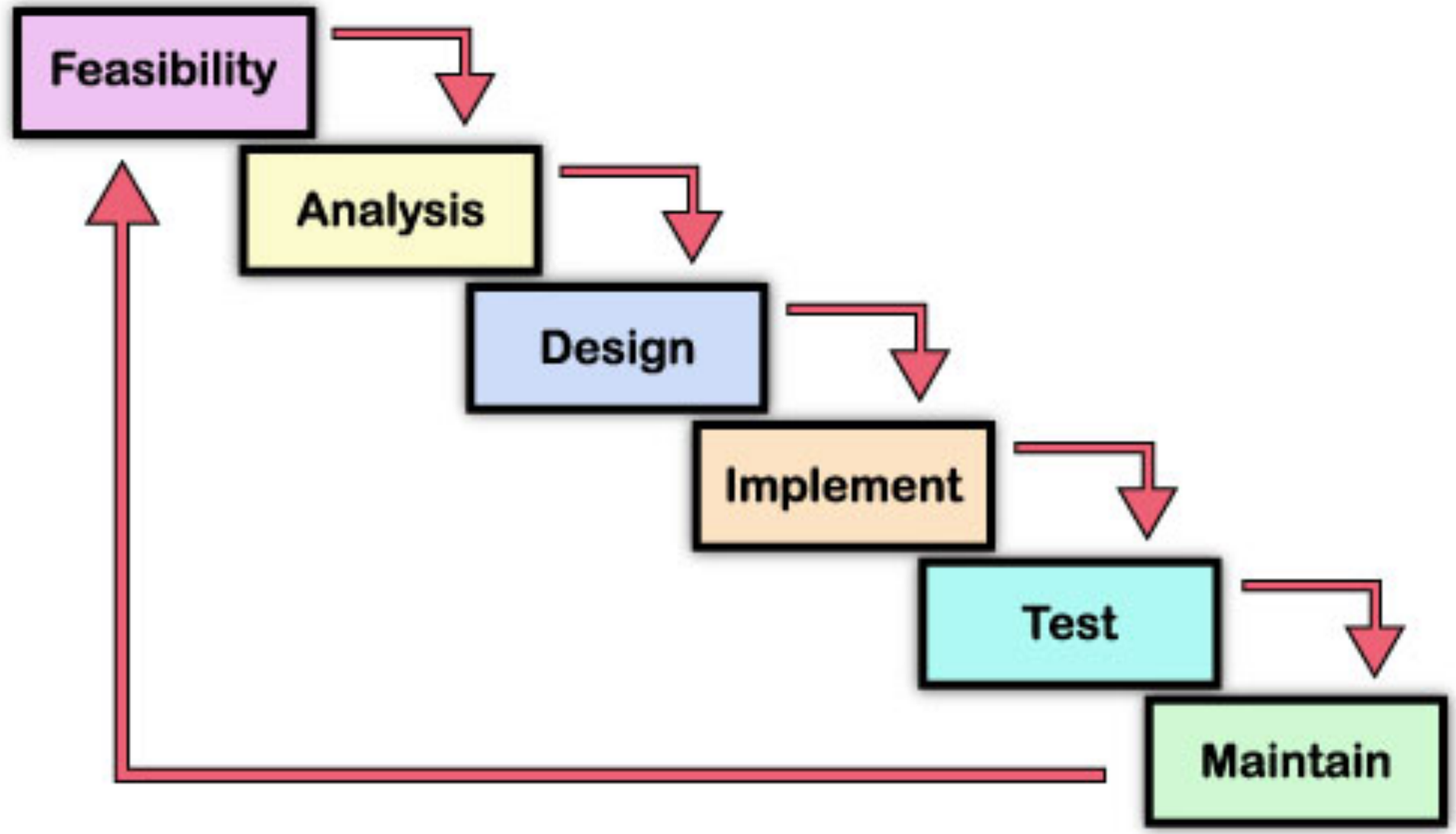
“Good programmers write code, great programmers write tests”

- Context & Motivation
- What it is Unit Testing?

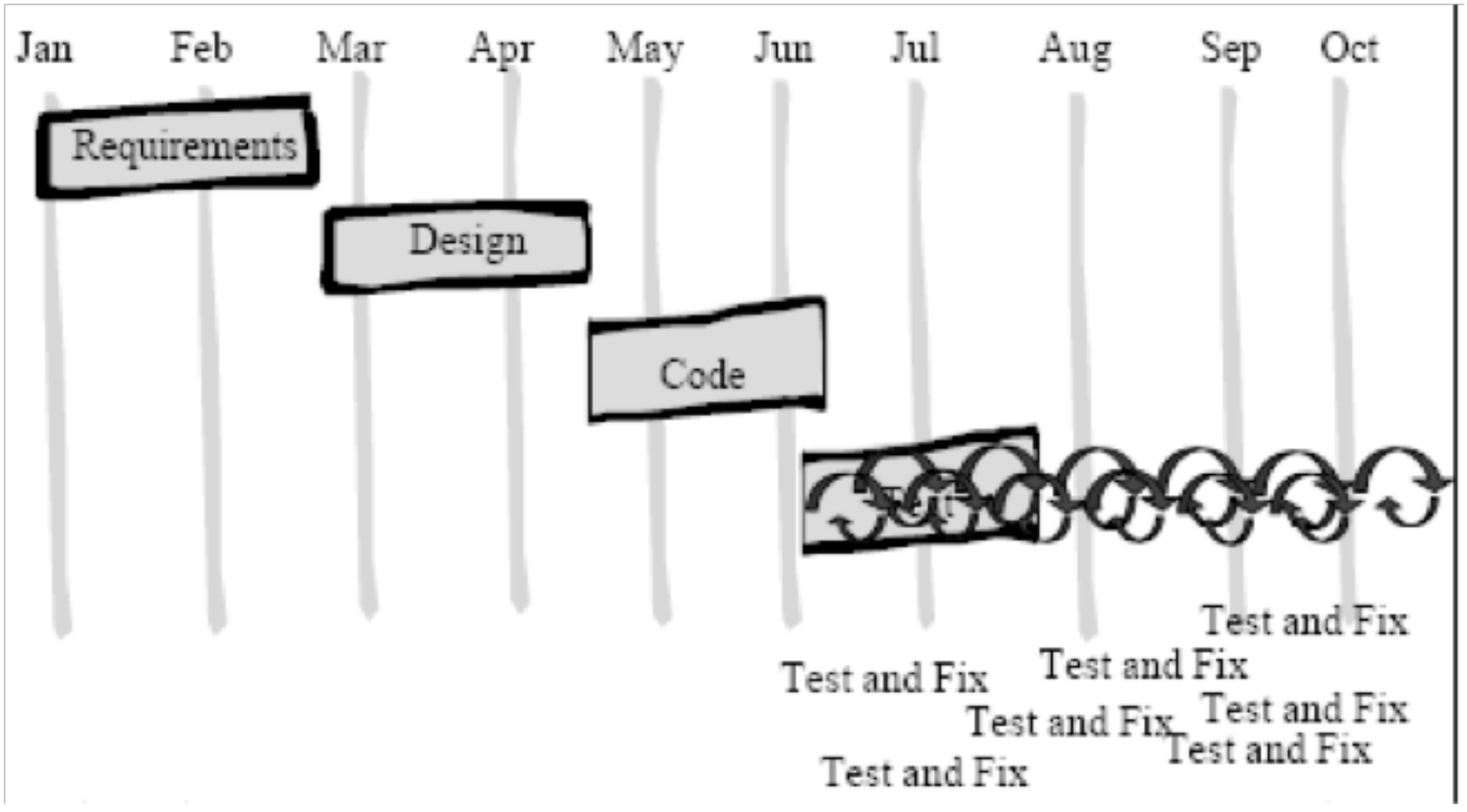
“Never, in the field of programming, have so many owed so much to so few”

- Martin Fowler on the developers behind JUint

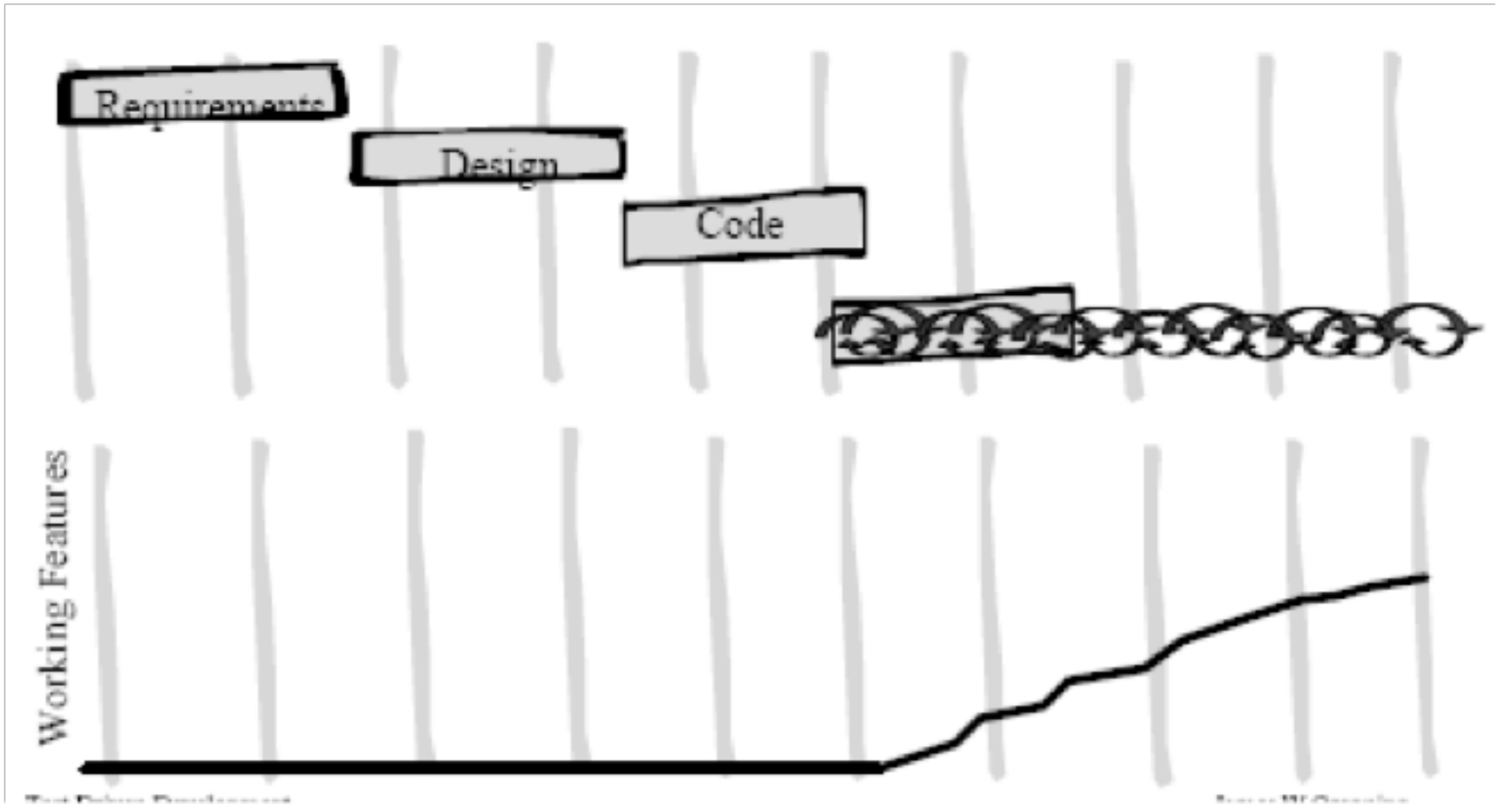
Waterfall development approach



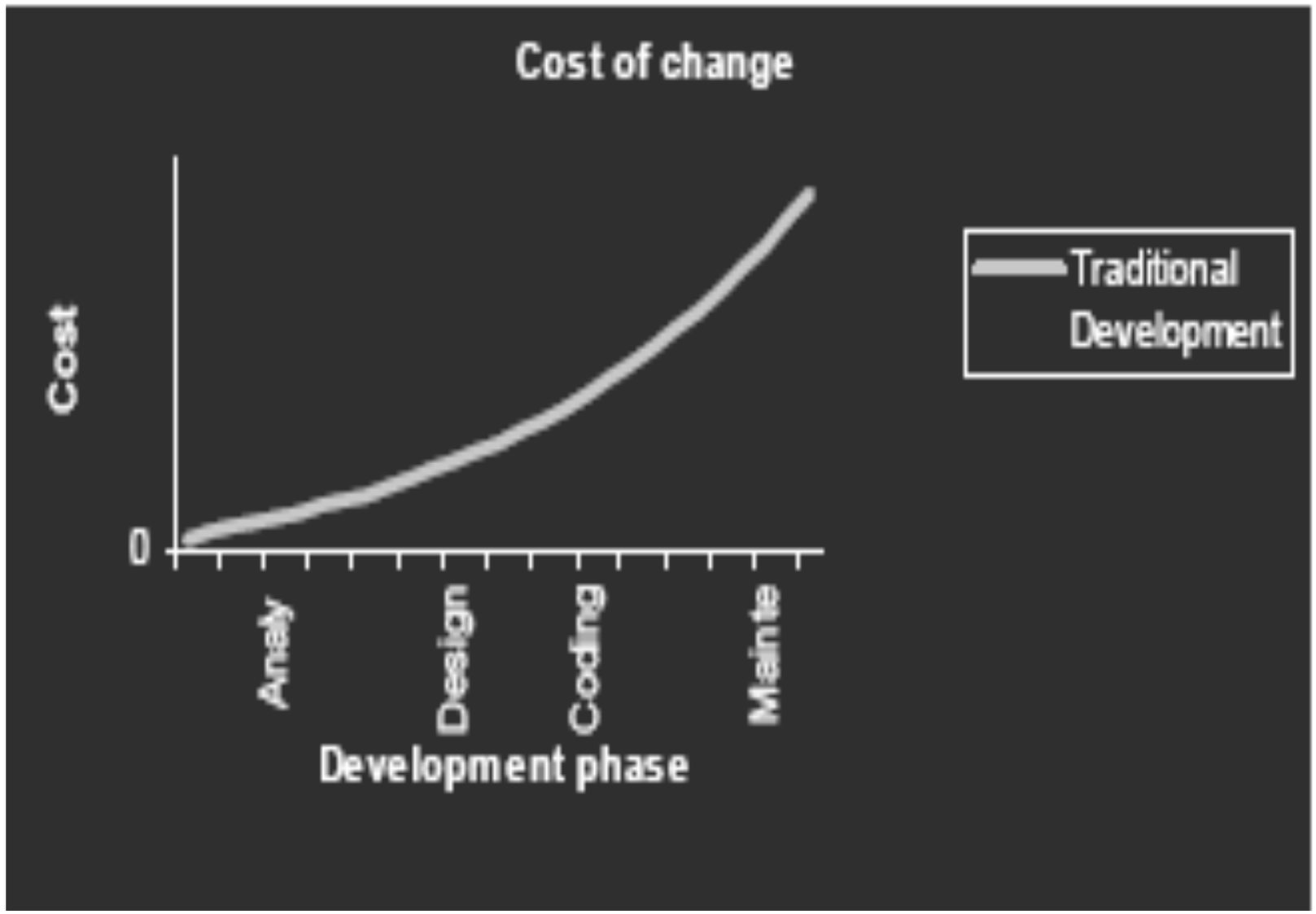
Waterfall development approach



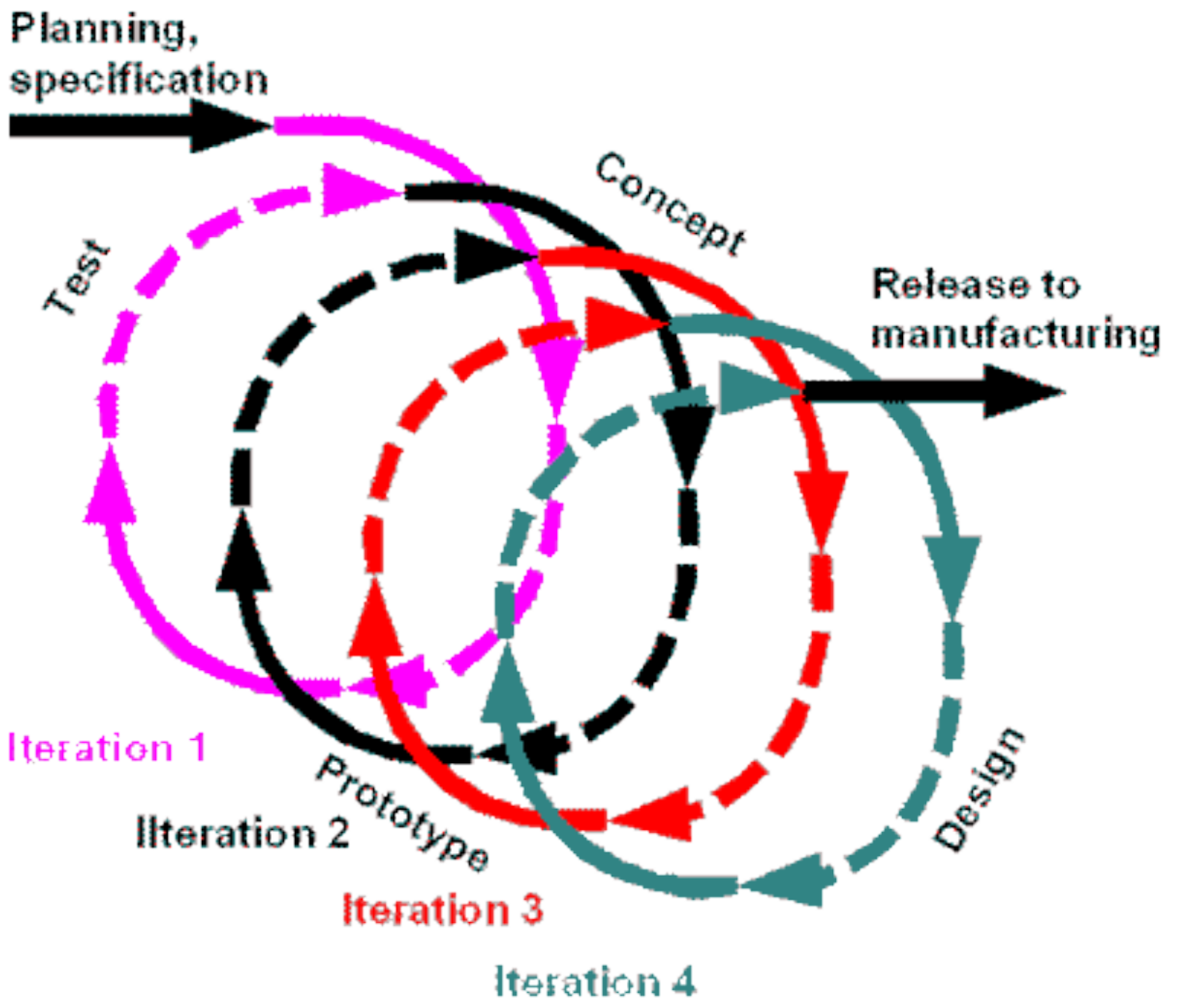
Waterfall - Working Features



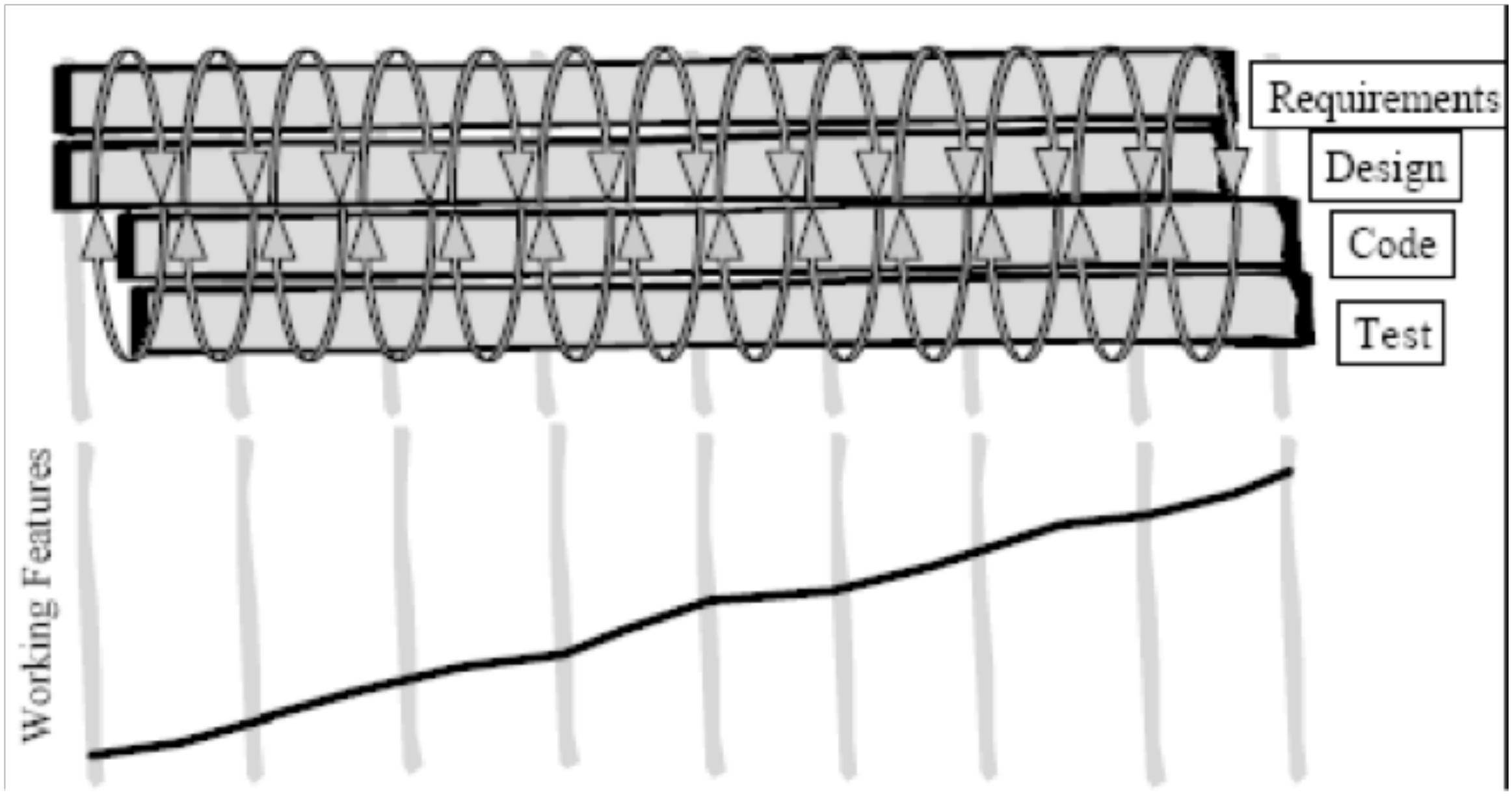
Waterfall - Cost of change



Iterative/Evolutionary approach



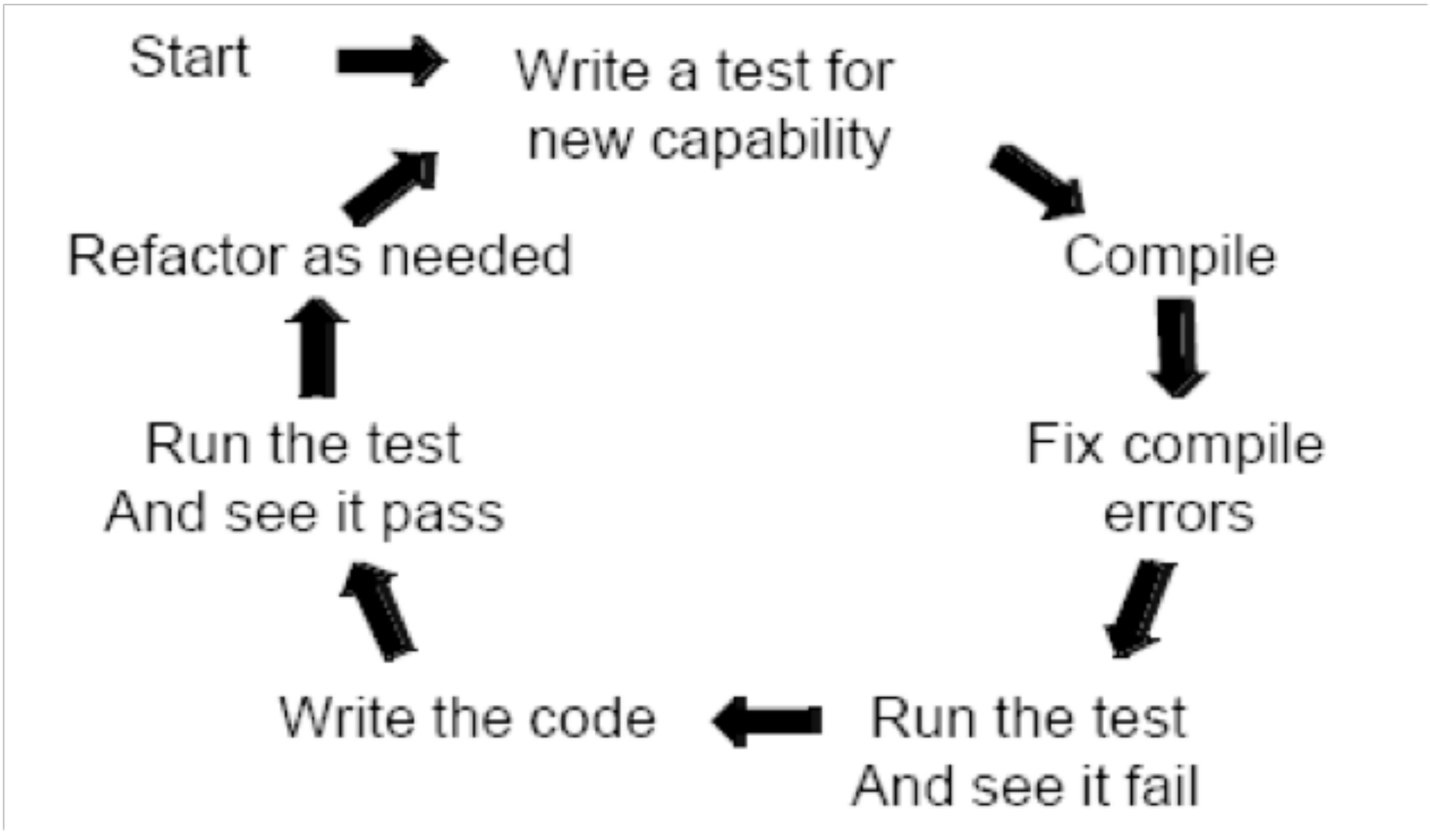
Iterative/Evolutionary - Working Features

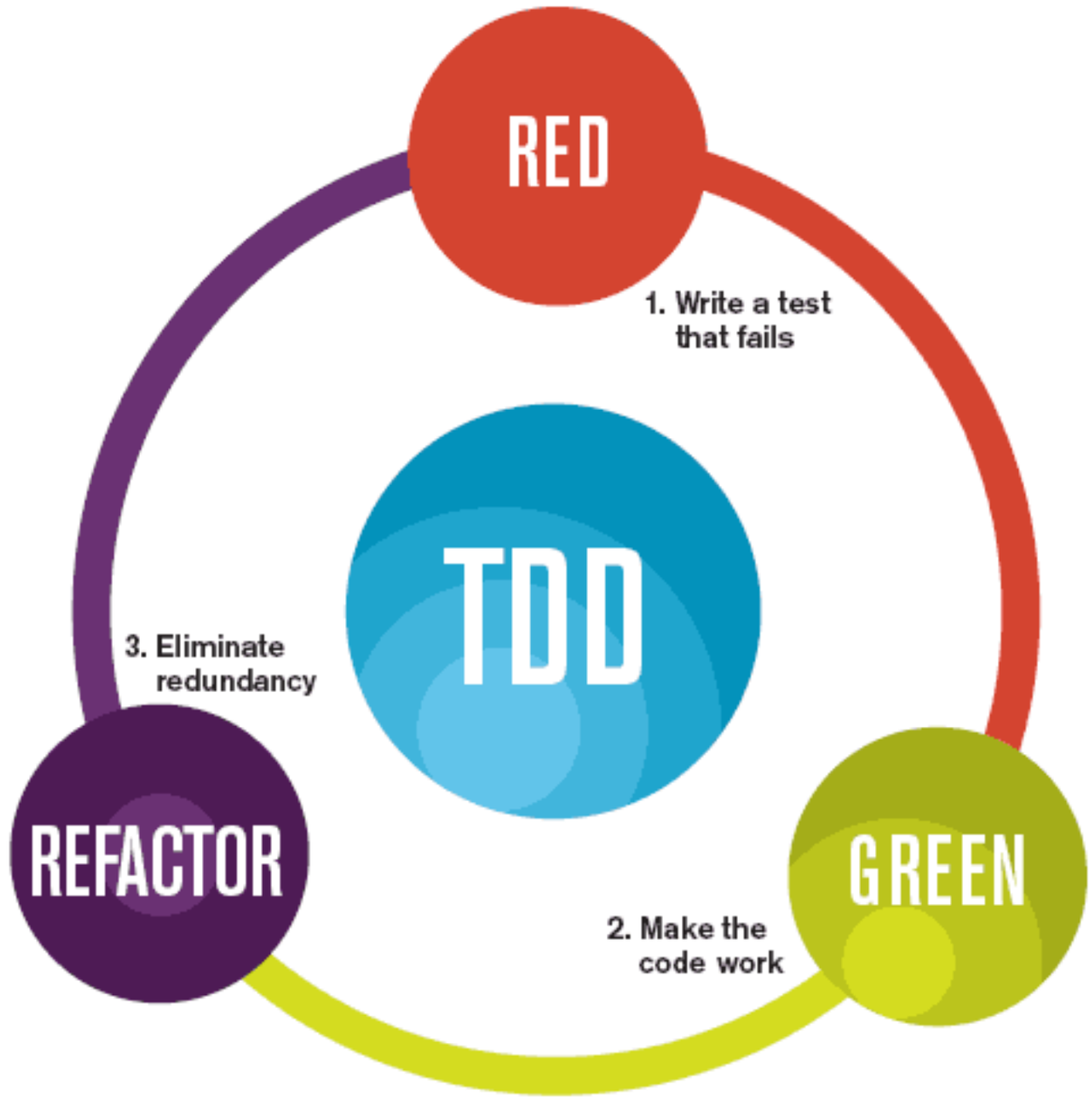


Things have changed a little…

- Computing power has increased astronomically
- New tools have dramatically eased mundane developer tasks:
  - Automated test tools.
  - System build tools.
  - Version control.
  - Continuous integration.
- Used properly, OO languages can make software much easier to change.
- The cost curve is significantly flattened, i.e. costs don’t increase dramatically with time.
- Up front modeling becomes a liability – some speculative work will certainly be wrong, especially in a business environment

Test-driven development.





The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Principles of TDD.

- Lots of small changes.
  - Use test-driven to get from A to B in very small verifiable steps
  - You often end up in a better place.
- Do the Simplest Thing
  - Assume simplicity.
    - Consider the simplest thing that could possibly work
    - Iterate to the needed solution.
  - When coding:
    - Build the simplest possible code that will pass the tests
    - Refactor the code to have the simplest design possible.
    - Eliminate duplication.

Test driven development - General

- An iterative technique to develop software.
- Tests are written before the code itself.
- As much (or more) about design as testing.
    - Encourages design from user's point of view.
    - Encourages testing classes/units in isolation – Unit testing.
- A test framework is used so that automated testing can be done after every small change to the code.
        - This may be as often as every 5 or 10 minutes.
- Axiom:
    - 'Code that isn't tested doesn't work'
    - 'Code that isn't regression tested suffers from code rot (breaks eventually)'

Test driven development – General (Contd.)

- As much (or more) about documentation as testing.
  - The tests are the documentation of what the code does.
- Must be learned and practiced.
- Consequences:
  - Fewer bugs;
  - More maintainable code - loosely-coupled, highly-cohesive systems.
  - During development, the program always works—it may not do everything required, but what it does, it does right,
  - Break the cycle of more pressure == fewer tests,

Regression testing.

- New code and changes to old code can affect the rest of the code base.
    - ‘Affect’ sometimes means ‘break’.
- We need to rerun tests on the old code, to verify it still works – this is regression testing.
- Regression testing is required for a stable, maintainable code base.
- Unit tests retain their value over time and allows others to prove the software still works (as tested).

What is Unit Testing?

- A unit test is a piece of code written by a developer that exercises a very small, specific area of functionality of the code being tested.
    - Usually a unit test exercises some particular method in a particular context
- Unit tests are performed to prove that a piece of code does what the developer thinks it should do.
- The question remains open as to whether that's the right thing to do according to the customer or end-user:
    - that is acceptance testing

What does Unit Testing Accomplish ?

- *Does the **code** do what was expected?*
  - i.e. s the code fulfilling the intent of the developer?
- *Does the **code** do what was expected all the time*?
  - exceptions get thrown, disks get full, network lines drop, buffers overflow - is the the code still perform as expected?
- *Can the **code** be depended upon?*
  - Need to know for certain both its strengths and its limitations.
- *Does the **test** document the developers Intent?*
  - An important side-effect of unit testing is that it helps communicate the code's intended use

Why Bother with Unit Testing?

- Will make designs better

- Drastically reduce the amount of time spent debugging.

How is Unit Testing Carried Out?

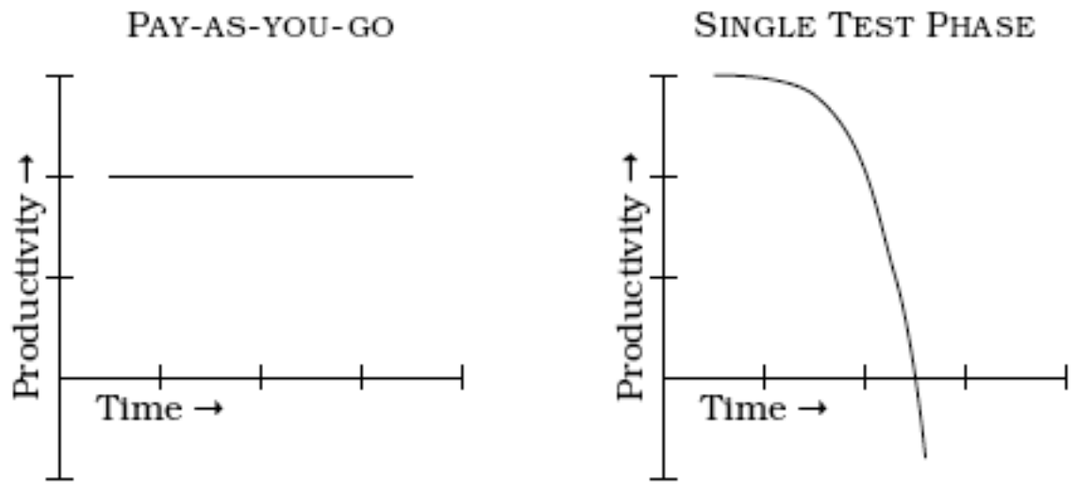
- Step 1: Decide how to test the method in question before writing the code itself

- Step 2: Write the test code itself, either before or concurrently with the implementation code.

- Step 3: Run the test itself, and probably all the other tests in that part of the system

- *Key Feature of executing tests: need to be able to determine at a glance whether all tests are succeeding/failing*

Excuses for note Testing (1)



- *It takes too much time to write the tests*
  - The trade-off is not “test now” versus “test later”
  - It's linear work now versus exponential work and complexity trying to fix and rework at the end.

Excuses for note Testing (2)

“It takes too long to run the tests”

- Separate out the longer-running tests from the short ones.
- Only run the long tests once a day, or once every few days as appropriate, and run the shorter tests constantly.

“*It's not developers job to test his/her code”*

- Integral part of developer job is to create working code

“But it compiles!”

- Compiler's blessing is a pretty shallow compliment.