1. Introduction to TDD

Java Test Driven Development with JUnit

Module Topics

1 Defining Test Driven Development

2 The TDD Community

3 TDD and Software Development Process Efficiency

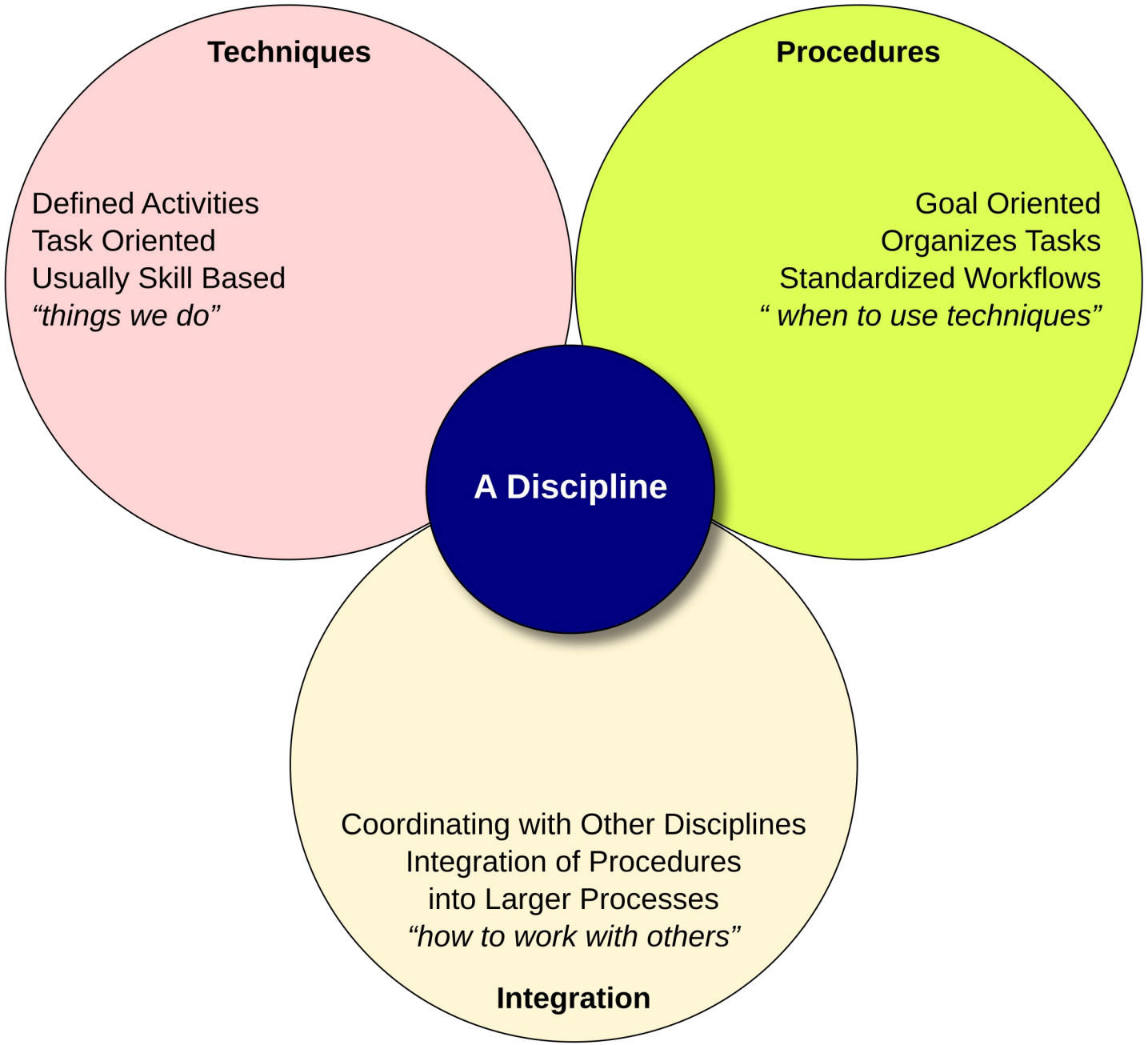
4 Writing Better Code Using TDD

Defining Test Driven Development

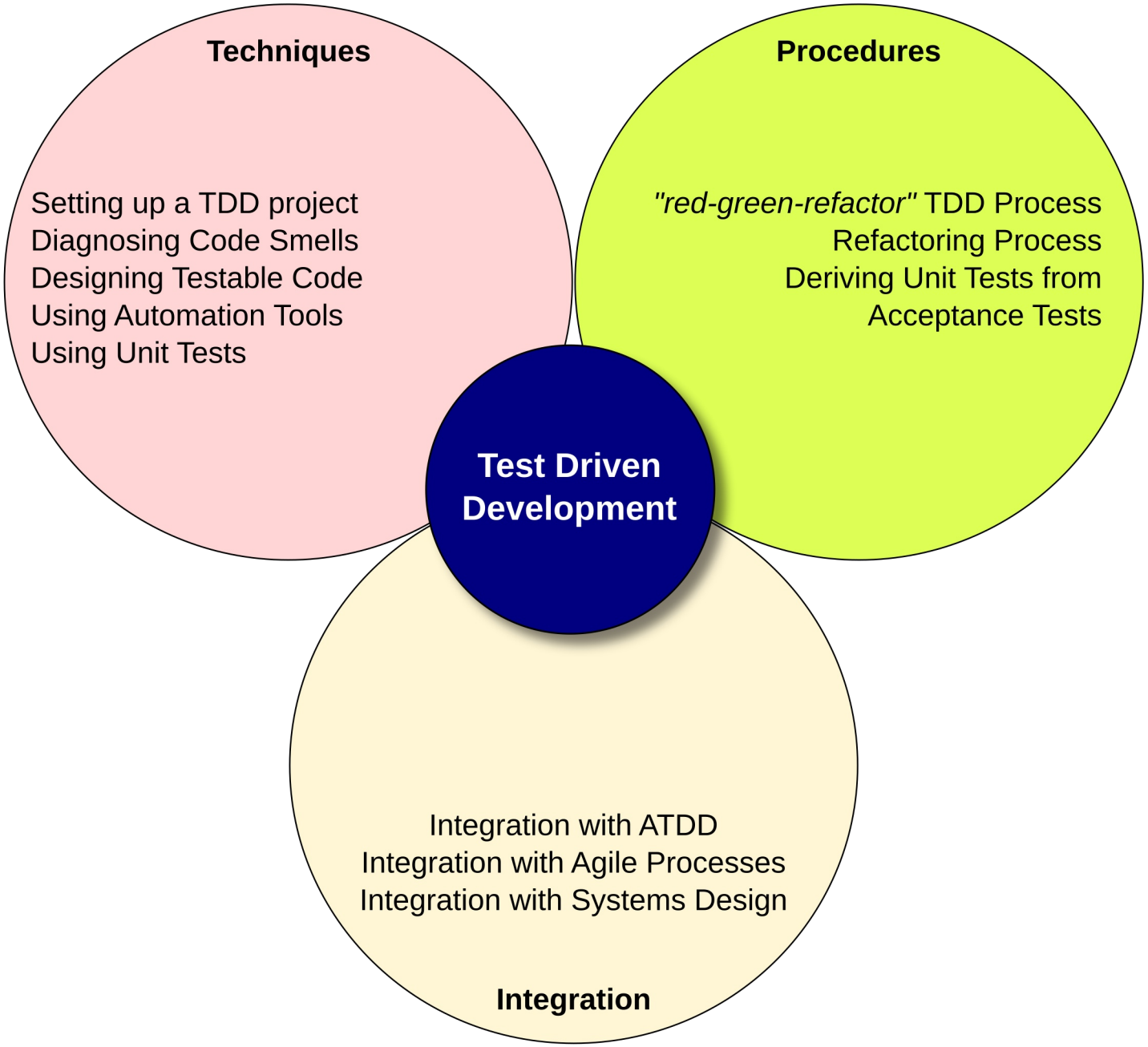
Features of Test Driven Development

- TDD is a programming discipline
- TDD is a popular best practice for programming
- TDD allows programmers to write better code faster
  – *but only if it is followed correctly!*
- Requires programmers follow best object oriented design and coding practices
- Characterized by the integrated use of automated tools
- Integrates well with Agile and other practices

Definition of "Discipline"



TDD as a Discipline



The TDD Community

I don’t like user interface-based tests.

In my experience, tests based on user interface scripts are too brittle to be useful. When I was on a project where we used user interface testing, it was common to arrive in the morning to a test report with twenty or thirty failed tests.

A quick examination would show that most or all of the failures were actually the program running as expected. Some cosmetic change in the interface had caused the actual output to no longer match the expected output.

Our testers spent more time keeping the tests up to date and tracking down false failures and false successes than they did writing new tests.

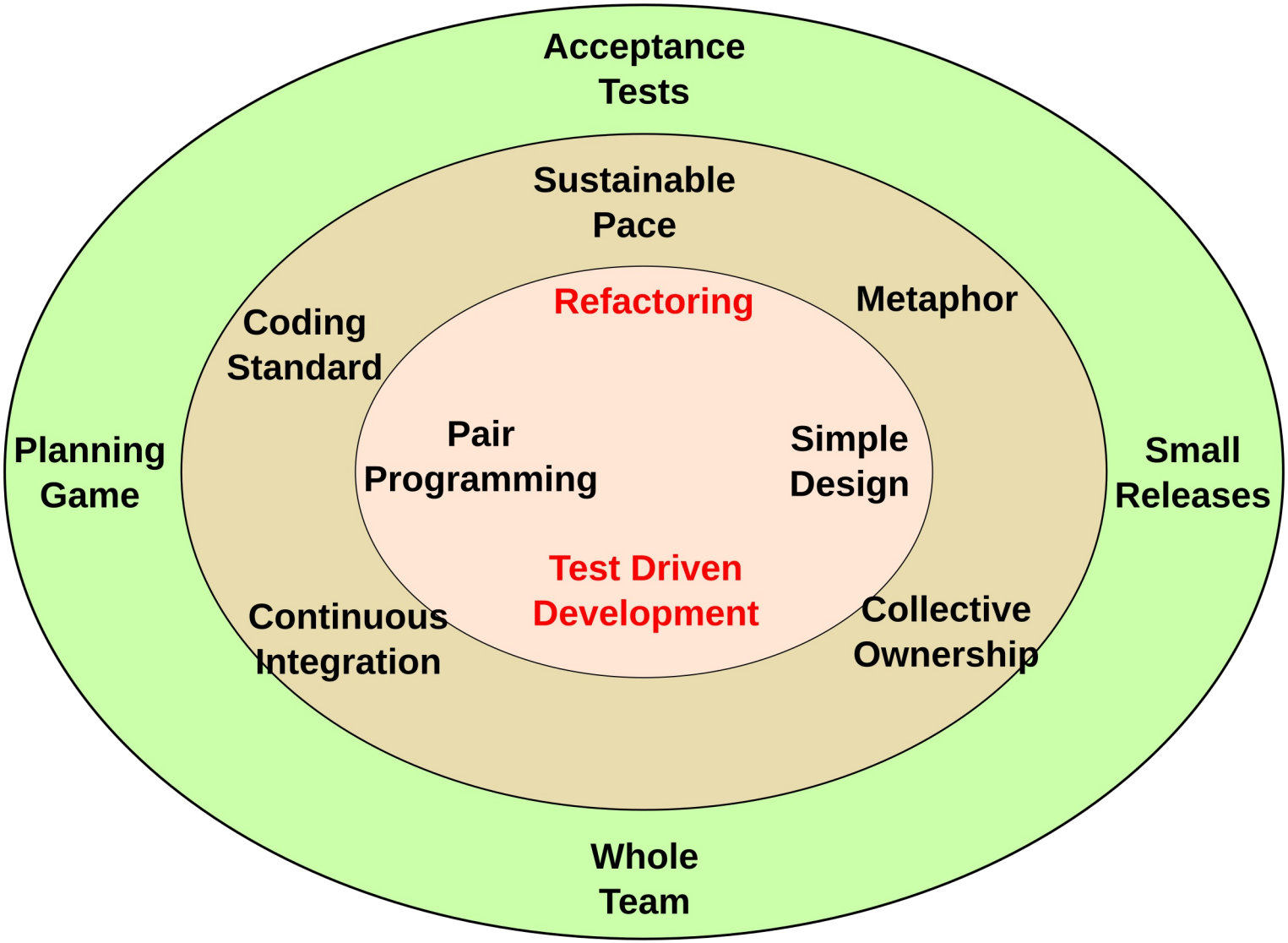
Kent Beck describing his motivation for creating TDD

Kent Beck

Kent Beck and XP

- Kent Beck developed TDD as a way to write better code faster
  - *“I'm not a great programmer; I'm just a good programmer with great habits”*
- He was one of the founders of the Agile XP methodology
  - *Two of the core XP practices were TDD and Refactoring*
- TDD and Refactoring were adopted by the larger Agile community
- TDD has now become recognized as a coding best practice
- A primary reason is how well TDD integrates with other Agile practices

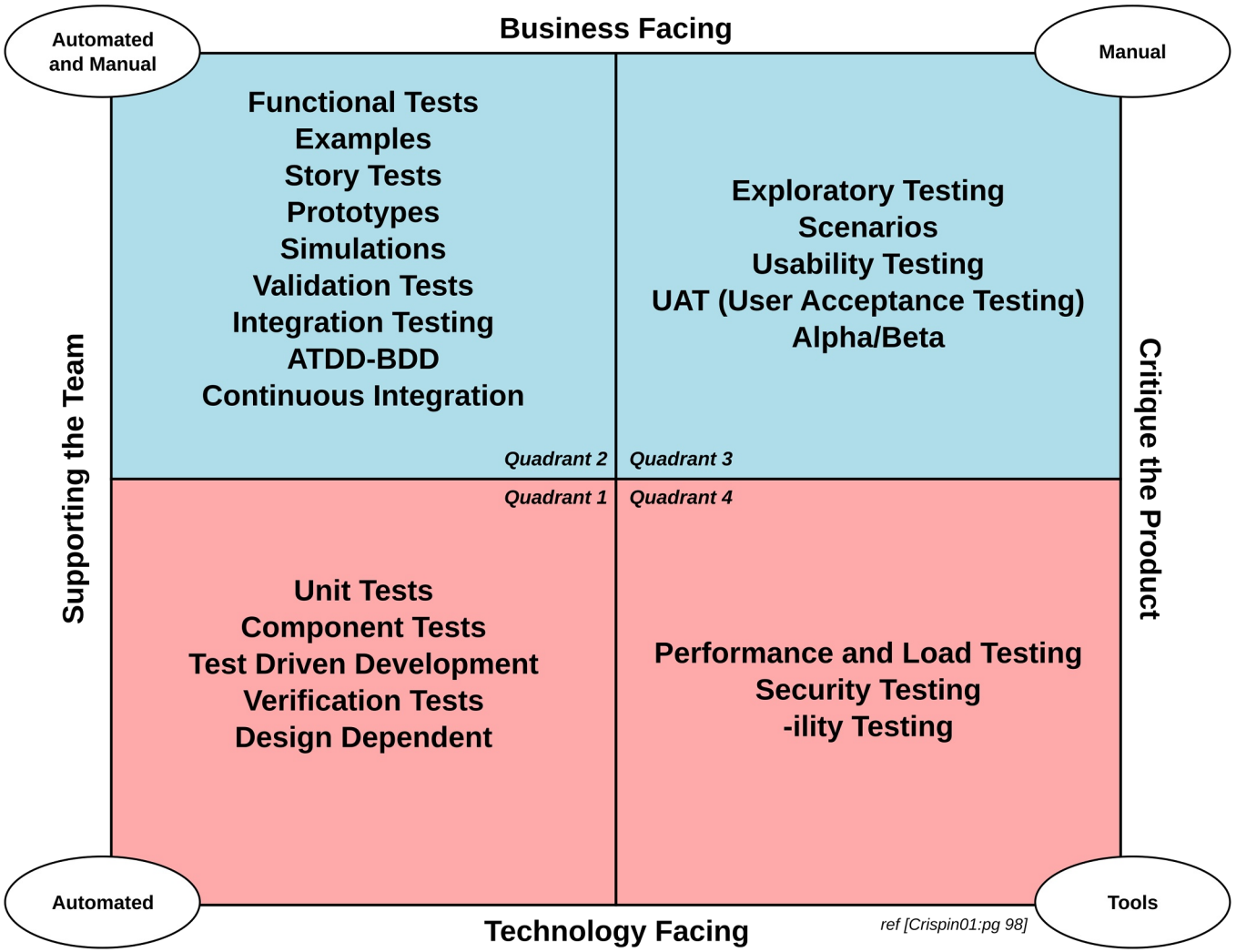
The XP Onion


Acceptance Tests
Sustainable Pace
Refactoring
Metaphor
Coding Standard
Planning Game
Pair Programming
Simple Design
Small Releases
Test Driven Development
Continuous Integration
Collective Ownership
Whole Team

Moving Beyond XP

- The XP Onion describes the basic practices of XP
- Many XP practices are now mainstream, for example:
  - *Continuous integration is a central tenet of DevOps*
  - *Developing to acceptance tests is now practiced as ATDD and BDD*
- TDD and refactoring were designed to integrate with these processes
  - *Made transitioning out of the XP world simple*

TDD and Agile Testing



The Agile Testing Quadrants

TDD is now an integral part in the emerging standard model of Agile testing and quality assurance.

The State of TDD

- There is no TDD standard or canonical TDD process
  - *Like a language, TDD has different "dialects"*
  - *All of the dialects tend to share a more or less common core of ideas*
  - *TDD is often adapted to fit into specific environments or different methodologies*
- Many developers use "lazy" TDD
  - *TDD practices are used in a very casual fashion or inconsistently applied*
  - *Lazy TDD runs into problems when the projects scale up*
- The "right" way to do TDD is the way that produces results for you

TDD and Software Development Process Efficiency

I’ve worked on three different development teams using TDD and on several more teams that didn’t. I can tell you from first-hand experience that TDD produces code that has orders of magnitude fewer unit-level bugs, far fewer functional bugs, and an exponentially higher probability of meeting stakeholder expectations when compared to code produced by conventional programming techniques

Lisa Crispin

Empirical Evidence

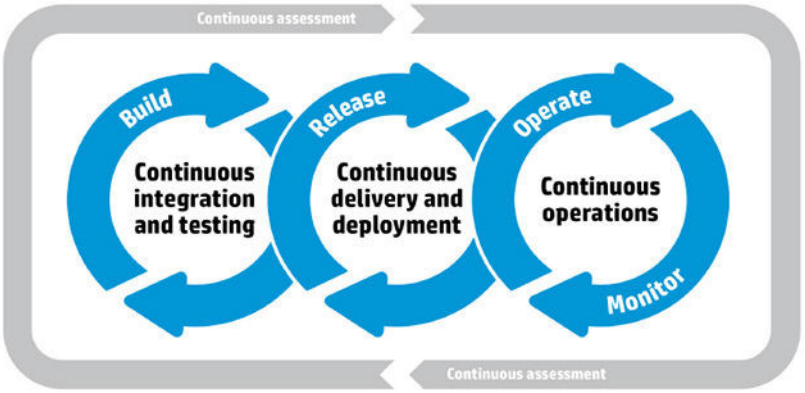
- There have been a number of research studies to see if TDD produces measurable benefits

- On average code produced with TDD:
  1. *Has a lower defect density*
  2. *Has better coverage*
  3. *Shows mixed results as to increases in programmer productivity and decreased development time*
  4. *Often shows initial increase in development time but off-set by "down the road" savings in development and application support*
  5. *Is often more loosely coupled*
  6. *Some studies show dramatic improvements in programmer productivity, some show no change, and a few show a decrease*
  7. *Has higher client satisfaction measures on average*

The Critical Success Factor

- One factor that explains the mixed results is how rigorously TDD is followed
- Best results come from engineering and industrial environments
  - *TDD is applied rigorously and consistently*
  - *Culture is focused on quality of code, risk and productivity*
- Neutral or negatives results tend to come from academic or non-industrial context
  - *Culture is not usually highly quality oriented*
  - *TDD tends to be only partly applied or inconsistently applied*

DevOps Example



DevOps: Depending on TDD

One of tshe reasons TDD has moved into a central position in software development is that it integrates well (because it is a disciple) with current innovations in software development and delivery. DevOps, for example, is built around the goal of continuous delivery of software. In order to meet the requirement for continuous testing as part of continuous delivery, software has to be tested as it is under development. DevOps relies on TDD and other related discipline to be able to meet this requirement.

This is just one example, but it is representative of the current state of software development, especially large scale software development.

Writing Better Code Using TDD

Programmer Comments on TDD

- Collected quotes on TDD collected by the author:

  “I’m writing code faster with less rework and less profanity”

  “It’s like the code is writing itself, I’m not doodling in code any more trying to figure out why I wrote that piece of code a month ago”

  “My code is simpler, cleaner and easier to understand”

  “Amount of rework is reduced since bugs are caught early, which means I go home on time”

  “Continuous regression testing means that adding new code doesn’t break my existing code”

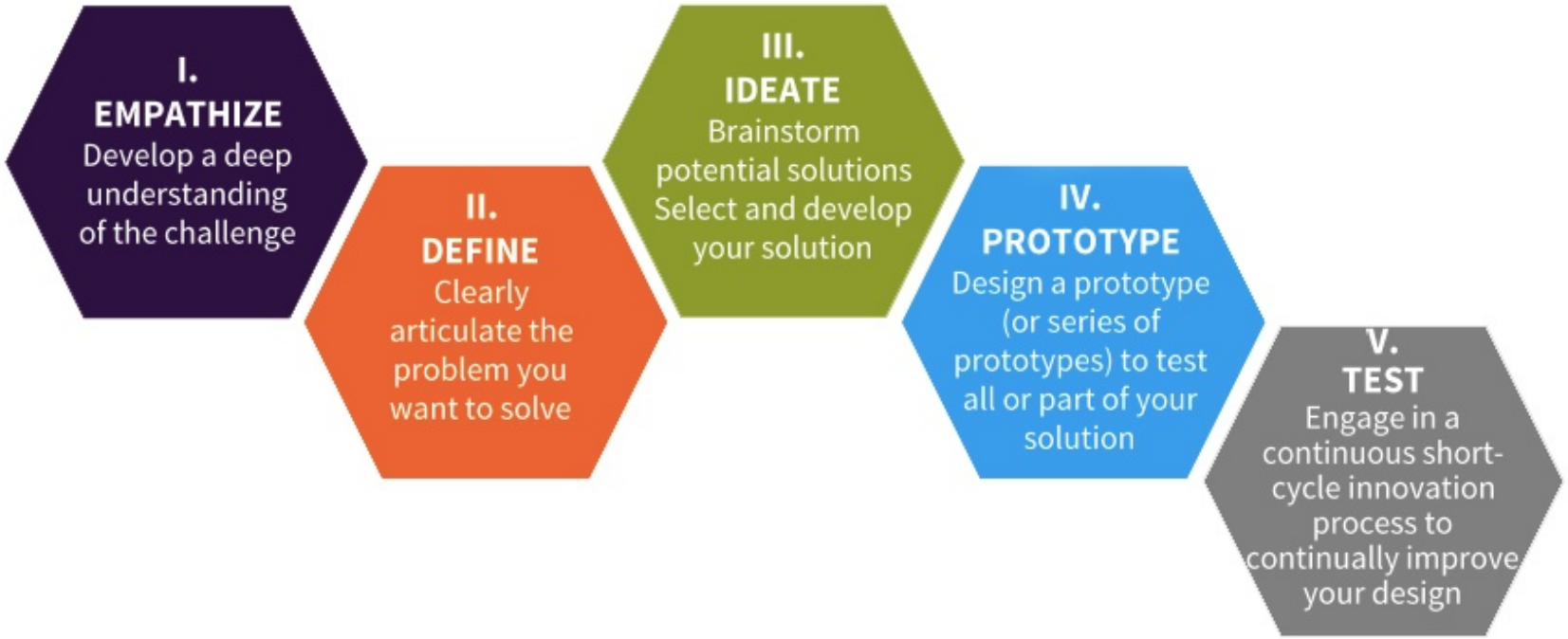
  “Programming is fun again. I don’t spend all my time on bug hunts and I get to think instead about optimizing my design and implementation”

Why Does TDD Improve Programming?

- TDD emphasizes working through the tests first, then writing the code
  - *This forces programmers to immerse themselves in the problem before writing any code*
  - *Also forces a critical analysis as to whether the developers know what the software is supposed to do*
  - *"If you can't figure out what the right result of a test should be, then you don't know what your code should do in that situation"*
  - *Requires a global view of the problem to be solved before crafting a solution*

- More importantly, TDD is consistent with the cognitive mechanisms used to solve problems
  - *For those interested, the topic is discussed in more detail in the student manual*

The Stanford Design Process



The Stanford Design Process

The structure of the TDD process is remarkably like the Stanford Design Process, which has been widely adopted as a methodology that consistently develops creative and innovative design thinking.

From Wikipedia “Design thinking is a method for practical, creative resolution of problems and creation of solutions. It is a form of solution-based, or solution-focused thinking with the intent of producing a constructive future result. By considering both present and future conditions and parameters of the problem, several alternative solutions may be explored.”

There are references to Stanford's publicly available design thinking materials in the student manual.

More than the act of testing, the act of designing tests is one of the best bug preventers known. The thinking that must be done to create a useful test can discover and eliminate bugs at every stage in the creation of software, from conception to specification, to design, coding and the rest.

If you can't test it, don't build it.
If you don't test it, rip it out.

Boris Beizer

End of Module 1