Test Driven Development: Benefits and Pragmatics

Venkat Subramaniam

venkats@agiledeveloper.com

http://www.agiledeveloper.com/download.aspx

Abstract

Abstract Testing is what you typically expect your QA to do. What is this TDD and why would your programmers be doing that? We all know that time is a scarce resource. How can you then justify your developers spending time writing test code when they should be writing "real" code?

There are many benefits of TDD. First, unit testing is more of an act of design than verification. A number of tools are available to help your team automate it. Imagine you being notified when a developer checks in code that breaks some functionality contract and renders the application unusable. The quicker you learn about it, the quicker you have them fix it. The quality of your code significantly improves. You actually end up saving time in the long run.

In this presentation, you will see how your developers will write test cases, when they should write those, and times when they shouldn't be writing those. You will look at tools and techniques available to effectively write and automate the test cases. You will look at facilities available to measure and monitor the quality and robustness of your code.

About the Speaker *Dr. Venkat Subramaniam*, founder of Agile Developer, Inc., has trained and mentored thousands of software developers in the US, Canada and Europe. He has significant experience in architecture, design, and development of software applications. Venkat helps his clients effectively apply and succeed with agile practices on their software projects, and speaks frequently at conferences.

He is also an adjunct faculty at the University of Houston (where he received the 2004 CS department teaching excellence award) and teaches the professional software developer series at Rice University School of continuing studies.

Venkat is the author of: *Practices of an Agile Developer* (The Pragmatic Programmers-2005), and *.NET Gotchas* (O'Reilly-2005).

Agenda

- Motivation
- Types of tests
- What's Unit Testing?
- Unit testing as design tool
- Benefits of unit testing
- When and who writes unit tests
- When not to write unit tests
- Testing core business logic
- Continuous Integration
- Conclusion

Motivation

- Project goals and concerns
  - Budget, time, quality, robustness
  - success
- Code constantly evolves
  - New piece of functionality gets added
  - Small design changes
  - Fixing bugs
- Your developer changes code, reviews it, checks it in, and then what happens
  - May be nothing for a while
  - And then a boom when code is integrated ☹
  - You don't want to let your code rot
  - You don't want it to turn into a loose cannon

Agenda

- Motivation
- Types of tests
- What's Unit Testing?
- Unit testing as design tool
- Benefits of unit testing
- When and who writes unit tests
- When not to write unit tests
- Testing core business logic
- Continuous Integration
- Conclusion

Types of Tests

- Black-box testing
  - Does not know and depend on internal structure of modules being tested
  - Acceptance testing
    - Written by customers, QA
    - Focuses on functionality of the system
- White-box testing
  - Knows and depends on internal structure of modules being tested
  - Unit Testing
    - Drives the design
    - Validates changes made
    - Insufficient as verification tool however

Acceptance Testing

- Manual testing is not the preferred way
- Need to find ways to automate this as well
- Promotes separation of business logic from UI
- May be written using scripts, XML, etc.

Agenda

- Motivation
- Types of tests
- What's Unit Testing?
- Unit testing as design tool
- Benefits of unit testing
- When and who writes unit tests
- When not to write unit tests
- Testing core business logic
- Continuous Integration
- Conclusion

Unit Testing

- Unit Testing is an act of design than an act of verification
- It helps provide instant feedback when code is changed
  - Substantially improves robustness of app
- Works as a great form of documentation
- Safety net when refactoring code

TDD vs. TFD

- Test Driven Development
  - Written to gain insight into design and implementation issues
  - Quickly alerts when things fall apart
- Test First Development
  - Test for code written before any code is written
  - You start with a failing test and write minimal code to make it work
- TFD is a subset of TDD
- TFD very helpful at times; TDD adequate at other times

Agenda

- Motivation
- Types of tests
- What's Unit Testing?
- Unit testing as design tool
- Benefits of unit testing
- When and who writes unit tests
- When not to write unit tests
- Testing core business logic
- Continuous Integration
- Conclusion

How we create classes?

- We think about what a class must do
- We focus on its implementation
- We write fields
- We write methods
- We may write a few test cases to see if it works
- We hand it off to users of our code
- We then wait for them to come back with feedback (problems)

Test First Coding

- How about starting with a test case even before we have any code for our class?
- How about first write test that fail because the code to support it does not exist?
- How about adding functionality to our system by adding tests incrementally and then adding code to make those tests succeed?

Test First Coding Benefits

- It would
  - completely revert the way we develop
  - We think about how our class will be used first
    - Helps us develop better interfaces that are easier to call and use
  - Would change the way we perceive things
  - Will have code that verifies operations
  - Will increase robustness of code
  - Will verify changes we make
  - Will give us more confidence in our code

Test First Coding Benefits…

- Forces us to make our code testable
- Tests decouple the program from its surroundings
- Serves as invaluable form of documentation
  - Shows others how to use our code

Programming by Intention

- Making your intent clear
  - Avoid Opacity – that is code that is hard to understand
- It goes a long way in writing code
  - Choose appropriate names for methods, fields, classes, etc.
  - Strive for simplicity
  - Test First, Code next

Simple Code that works

- Striving for simplicity is important
- When you write your test, you think of how the object should be used
  - And you do that before you implemented it
- This leads to finding the simplest way to communicate
  - Avoid unnecessary complication or over abstraction

Agenda

- Motivation
- Types of tests
- What's Unit Testing?
- Unit testing as design tool
- Benefits of unit testing
- When and who writes unit tests
- When not to write unit tests
- Testing core business logic
- Continuous Integration
- Conclusion

Benefits

- Safety net for refactoring
- Makes code robust
- Acts as a design tool
- Boosts confidence
- Acts as a probe to fix problems
- Serves as a documentation
- Can help to alert us of change in APIs
  - learning tests

Agenda

- Motivation
- Types of tests
- What's Unit Testing?
- Unit testing as design tool
- Benefits of unit testing
- When and who writes unit tests
- When not to write unit tests
- Testing core business logic
- Continuous Integration
- Conclusion

When and Who should write it?

- Best written
  - before writing code or
  - as it is being written
- Written by the developer
  - Can a QA person write it?
- Not effective if others write it
- Not effective if written at other times

Agenda

- Motivation
- Types of tests
- What's Unit Testing?
- Unit testing as design tool
- Benefits of unit testing
- When and who writes unit tests
- When not to write unit tests
- Testing core business logic
- Continuous Integration
- Conclusion

When not to write Unit test?

- Not effective to take existing code and start writing tests
- Why?
- You can't think of the specific issues to be tested all at once
- No point going on a marathon of writing tests for legacy code
- What's the solution then?
  - Write unit tests as you refactor or evolve code instead

Agenda

- Motivation
- Types of tests
- What's Unit Testing?
- Unit testing as design tool
- Benefits of unit testing
- When and who writes unit tests
- When not to write unit tests
- Testing core business logic
- Continuous Integration
- Conclusion

Core business logic

- What about core business logic that is critical to the system's success?
- Users/customers will test it, isn't it?
- May be too late though
- Can you get those tested sooner?
- Can you automate those tests?
- Look at FIT: Framework for Integrated Testing

Agenda

- Motivation
- Types of tests
- What's Unit Testing?
- Unit testing as design tool
- Benefits of unit testing
- When and who writes unit tests
- When not to write unit tests
- Testing core business logic
- Continuous Integration
- Conclusion

Continuous Integration

- What good are the test cases if they are not run
- How often should we run them?
- Every night at least
- How about once every hour?
- Or better still when ever code change is checked in
- When code is checked in the code is compiled automatically and all tests cases are executed
  - If a test fails the team is alerted
  - When test fails, nothing else important/high priority
    - Fix the code to make the test succeed
    - Or modify the test to fit the changes if appropriate

Tools for Continuous Integration

- Anthill
  http://www.urbancode.com/projects/anthill/default.jsp
- Cruise Control/Cruise Control.NET
  http://cruisecontrol.sourceforge.net/
- Draco.NET
  http://draconet.sourceforge.net/
- Gump
  http://jakarta.apache.org/gump/

Agenda

- Motivation
- Types of tests
- What's Unit Testing?
- Unit testing as design tool
- Benefits of unit testing
- When and who writes unit tests
- When not to write unit tests
- Testing core business logic
- Continuous Integration
- Conclusion

Conclusion

- Test Driven Development can help you
  - Create a more pragmatic design
  - Create a robust system
  - Stay alert when things start falling apart
  - Find problems before they become unmanageable
  - Serve as a good documentation
  - Isolate the system for easy problem solving
- Used along with Continuous integration can make a difference between success and failure

References

1. Agile Software Development, Principles, Patterns, and Practices, Robert Martin

2. Refactoring Improving The Design Of Existing Code, Martin Fowler

3. Test-Driven Development by Example, Kent Beck

4. Continuous Integration, Martin Fowler
   http://www.martinfowler.com/articles/continuousIntegration.html

5. Pragmatic Project Automation, Mike Clark

6. Pragmatic Unit Testing, Dave Thomas and Andy Hunt

7. FIT: http://fit.c2.com

8. Practices of an Agile Developer, Venkat Subramaniam

9. Examples, slides are for your download at

   http://www.agiledeveloper.com/download.aspx

Thanks!