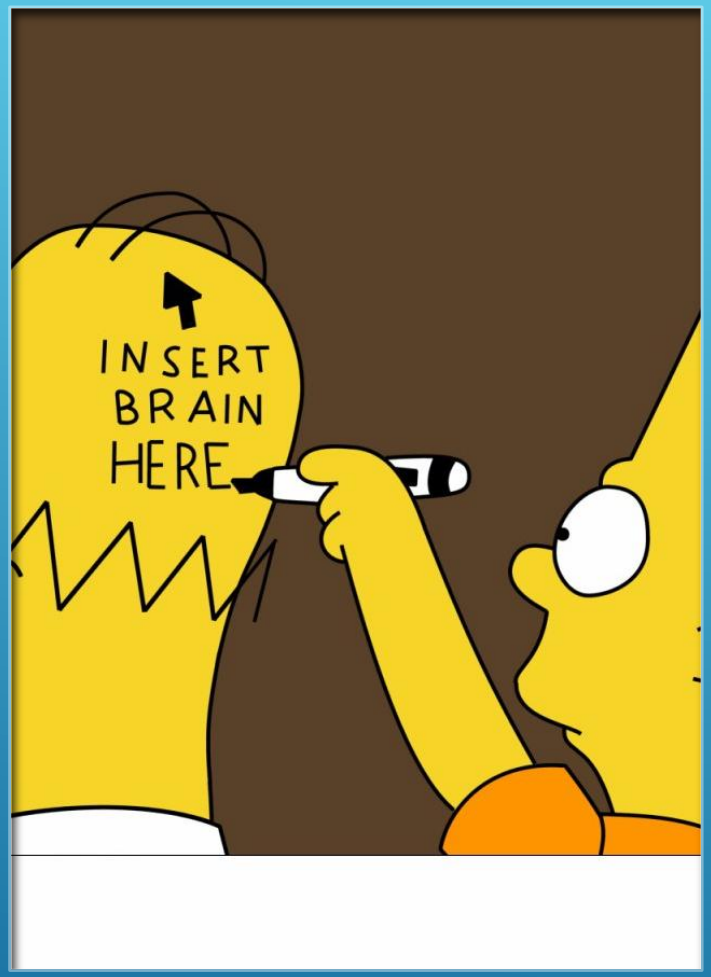
TEST-DRIVEN DEVELOPMENT

Matt Vaughn

Principal Software Engineer

www.BuildMotion.com

Matt.Vaughn@BuildMotion.com



SMART PEOPLE

"The thing about smart people is that they seem like crazy people to dumb people."

SURGEONS

"Washing hands and organizing surgical tools wastes a lot of time. I could help more patients if I just dove in."

BUILDERS

"We know how skyscrapers work. Just give us some bricks and we'll get started."

THE PROBLEM

Is it a people problem? Is it a process problem?

PROFESSIONALS

Design, plan, and prepare.

Then do the work.

PEOPLE

- Software teams must agree on benefit
  - Each individual must be on board
  - Accountability and professionalism
- Testing is a mind-set
  - Know and understand the benefits
- Long-term approach
  - It is practiced daily
  - It is a chosen developer lifestyle
  - "No more riding solo…"

PROCESS

- Produces better results faster
  - Use a plan.
- Process is the difference between:
  - Surgery and cutting people open
  - Software Engineering and “programming”
- Doesn’t matter if you are self-taught or not
  - We all need to have a proper process.
- Why should we test? Any benefits?

- Effectiveness is an outcome of good process.
- Stability is an outcome of good process.
- Successful people use good processes.
- Chaos is an outcome of bad process, not people.

PEOPLE & PROCESS TOGETHER

EXCUSES

Why we do not test or use TDD?

EXCUSES FOR NOT TESTING

- Not enough time to do it right the first time
- Fix later not now mentality…we'll come back when we have time
- The architecture/design doesn't promote unit testing.
- What are some other reasons?

NO PROCESS OUTCOME

What is the result or outcome when a process is not followed?

TECHNICAL DEBT…

- What Is Debt?
  - Accumulates over time
  - Compounds
  - Bites you, when you least expect it
- Bugs and Defects
- Orthogonal, Fragile software
- Architectural: High degree of Coupling and Dependencies
- Lack of Accountability, Professionalism, or Craftsmanship

TDD TO THE RESCUE

How can a TDD process help?

TDD PROVIDES

- Higher Quality software now, not later
- Faster development, Flawless Deployments
- Lower cost
- Lower maintenance
  - Less or no bugs
  - Easier to add new features
- Pay-as-you-go Plan
  - Regular Payments
  - Pay down existing debt
- Measurable, quantifiable results

TDD…PRACTICAL RESULTS

- Design and plan before you code
- Documents your design – How it works.
- Proof that code implements the design
- Encourages the design of testable code!

TDD :: WHERE TO START

DESIGN

- What is design?
- How much design?
- Who, what, when, where, and how?
  - Understanding the domain…the what – decompose.
  - Relationships
  - Separation of concerns
  - Single responsibility
- Take time to design or think
  - Defines how to implement
  - Continuous design…or design-as-you-go

POP MACHINE DESIGN

- You have been asked to create software for a new pop machine that will have a more intuitive user interface and interact with social media – in addition to the typical pop machine behaviors.
- The new design should make use of mobile technology to manage product selections, payment, and tracking soda-points for users – so a user can have a more interactive experience. Your team is responsible for the software.
- The web site and mobile application will be developed by a different company – and will integrate with your services.

TEST

- Test initial design
- Start small
- Tracer-bullet (integration-like)
  - aim for the target early and adjust
  - Full stack test early
- Identify goals of the system/solution
  - MVP (minimal viable product)
  - Focus on the core features first – edge cases last
- Initial tests fail…failure is expected?

IMPLEMENT

- Initial tests fail – without implementation
- Define end-points of your solution
  - entry points (layers, tiers, and domain)
  - Service APIs, Business Logic, Data Access ➔Tests
- Implement interfaces or abstractions early
  - May refactor later – when domain is more understandable
- How will I test this?
  - Principles: Separation of Concerns & Single Responsibility
  - Test should target something specific and quantifiable

- Happy path testing
  - Expected result(s) :: right results are quantifiable
- Alternative testing
  - Boundaries :: use {correct}
  - Inverse relationships :: {insert-retrieve}
  - Cross-check results :: {different algorithm}
  - error conditions :: force them
  - Performance considerations

TEST…MORE

BOUNDARY TESTING

- Conformance
- Ordering
- Range
- Reference
- Existence
- Cardinality
- Time

ITERATIVE DEVELOPMENT

- Definition: repetitious or frequent, repeating
- How?
  - modify and improve on each iteration
  - Add more tests
  - Or modify existing tests
  - Test each iteration until desired outcome

TEST AUTOMATION

Why and how?

WHY AUTOMATE?

- Write once, run frequently
- Eliminates human input (less error prone)
  - Doesn't forget steps – just runs test
- Quantifiable results
  - Reportable to development team
  - Result notifications
  - Track and measure over time
- Safety net: during development, QA, and deployments

TOOLS

- Testing framework (i.e., NUnit)
  - Common tool for whole team
  - Projects and organization
- Test repository
  - Location to store tests
  - Tests are a valuable resource to protect
- Continuous integration (cruisecontrol.net)

TESTING SCENARIOS

New development, defects/refactoring, and APIs

NEW DEVELOPMENT

Where to start?

TEST BEFORE CODE

- Start with test project or test fixture
- Use tests to drive development process
- User stories provide guidance.

DEFECTS AND LEGACY CODE

WHAT IS USED MOST - LEGACY?

- Target the core of the application
- Use code profilers to learn what is used most
  - JetBrains, RedGate, Telerik, Eqatek (until 2014)
- Use code profilers to learn what is used least
- Prioritize your testing efforts

DEFECTS

- Learn what is causing the defect.
- Attempt to reproduce defect with unit test(s)
- Refactor code and modify tests
- Validate results with unit tests

DEFECTS

- Locate
- Reproduce defect
  - Understand desired behavior
  - Test for current condition
- Design, test, implement fix, test more
  - May require refactoring to test
  - Just enough
  - document

LEGACY CODE

- Understand
- Document (just enough/comments)
- Clean-up (refactor): design, test, implement, test, repeat…
- Other:
  - Add tracing and/or logging
  - Add exception handling/logging
- Improve and standardize

TEST COVERAGE

How much should it cover?

COVERAGE

- Use a test coverage tool to determine current baseline.
  - NCover: run and report
- Automate with Continuous Integration
  - Monitor during critical development cycles
  - Monitor before deployments
- Goal: Continuous Improvement over time

TEST TYPES

UNIT TESTS

WHAT IS A UNIT

- Definition: 1. an individual thing regarded as single and complete but which can also form an individual component of a larger or more complex whole.
- The intent of the test should be evident (name, structure, focus)
- Isolatable
- Can be targetable (public, private)
  - Business logic
  - Business rule
  - Data validation
  - Result specific
  - quantifiable

UNIT TEST :: INPUTS AND DEPENDENCIES

- Inputs
  - Data and parameters
- Output
  - Validate via Assertions
- Dependencies
  - Abstractions and interfaces
  - Injected?
- Controlled dependencies
    - Stubs or mocks
- Modular and decoupled

INTEGRATION TESTS

DEFINITION OF INTEGRATION TEST

- **Integration testing** (sometimes called **integration and testing**, abbreviated **I&T**) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing.
- Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

INTEGRATION EXAMPLES

- Service API Testing
  - Includes the business logic and data access modules
- Data Access
  - Includes manipulation of the application's database.

FUNCTIONAL TESTS

FUNCTIONAL TEST DEFINITION

- **Functional testing** is a quality assurance (QA) process [1] [2] and a type of black box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered (not like in white-box testing).[3] Functional Testing usually describes *what* the system does.
- Functional testing differs from system testing in that functional testing "*verifies* a program by checking it against ... design document(s) or specification(s)", while system testing "*validate[s]* a program by checking it against the published user or system requirements" (Kaner, Falk, Nguyen 1999, p. 52).

PATTERNS

Design Patterns and Principles that support TDD.

SERVICE FACADE PATTERN

FAÇADE PATTERN

- Definition: A single class that represents an entire subsystem.
- Useful in API design
- Useful in Service Oriented Architectures (SOA)
- Typically implements an *Interface*
- Unit tests will be more *integration* in nature – exercise the full implementation (full stack).

PROVIDER PATTERN

Ability to design loosely coupled components. The design patterns at work are: Strategy and Abstract Factory.

BUSINESS ACTIONS

Implement business logic using classes instead of methods to take advantage of Object-Oriented behaviors and techniques. Allows for implementation of base classes for common behavior and property data. Enables the use of abstractions.

TEMPLATE METHOD PATTERN

Allows you to create or define an algorithm (process, steps, flow) that can be redefined in implementation and at runtime.

REPOSITORY PATTERN

Use a repository to separate the logic that retrieves the data and maps it to the entity model from the business logic that acts on the model. The business logic should be agnostic to the type of data that comprises the data source layer. For example, the data source layer can be a database, a SharePoint list, or a Web service.

ABSTRACT FACTORY

AGILE METHODOLOGY

BRIEF OVERVIEW

- Product Roadmap & Goals
  - What is the business value?
- Product Owners
- Backlog
- Releases
- Sprints & Sprint Planning
- User Stories
- Tasks
- Acceptance
- Retrospective

ITERATIONS

- How long should an iteration or sprint be?
- Team Velocity
- Sprint Burndown or Team Burnout?
- Operational Support during Sprint

USER STORIES

- As a <actor>, I <would like, need, require> the ability to <what> so that I can <benefit>.
- Size

TASKS

- Smallest unit within agile.
- Creating tasks for user stories.
- Estimating.
  - Small enough to quantify and do within short time period.
- Tracking Progress.

TASK: DEFINITION OF DONE

- Analysis and Design (thinking)
- Initial tests (TDD)
  - Alternate Flow or negative tests?
- Implementation and Iteration
- Unit Testing
- Documentation
- Code Review
- Other…

CODING PRACTICES

What should we be doing?

DEVELOPER NOTEBOOK

- Notebook Content
  - What you are working on for that day. Daily Standup Information.
  - Links and references
  - Deployment/Configuration Notes
- Just a place to put information…things not to forget
  - Useful for weekly status updates.
  - Useful for keeping your resume current.

THINK & COLLABORATE

- Take time to think
  - Learn the “what”.
  - Understand the “relationships”.
  - Sketches and whiteboards
- Work with others
  - Collaborate
  - Review
  - Confirm

MASTERY OF TOOLS

- Use, understand, and use tools
- Productivity
  - Scaffolding, Templates, designers
- Refactoring
  - Names, Methods, Interfaces
- Debuggers, Watch, Autos
- Keyboard Shortcuts
- What is in your toolbox?
- Code Repository

CODING STYLES

- Using Abstractions
- Classes
  - Single Responsibility
- Methods
  - Length
  - Parameters
- Naming Conventions
- Consistent Coding Styles
  - Where are the {}?
  - Line Breaks?

CLASS DIAGRAMS

- Use diagrams to document
  - Structure
  - Relationships
- Use as a design tool
  - Domain, Class, Entity modeling

CONFIGURATION

- Identify Configuration Items
  - Document and Implement
- Do not hard-code configuration
  - Values or Credentials
  - Things that change based on environment

RESOURCES & LINKS

- Design Patterns: http://www.dofactory.com
- Provider Pattern: http://www.codeproject.com/Articles/550495/Provider-Pattern-for-Beginners
- Repository Pattern: http://msdn.microsoft.com/en-us/library/ff649690.aspx
- Adapter Pattern: http://www.dofactory.com/Patterns/PatternAdapter.aspx#_self1
- TestDriven.NET: Tool for running unit tests; includes Ncover
- GhostDoc: http://submain.com/download/ghostdoc/
- Marting Fowler on Technical Debt: http://martinfowler.com/bliki/TechnicalDebt.html