

Test-Driven Development and Functional Testing

- ▶ What is test-driven development?
- ▶ How TDD is typically applied to software projects
- ▶ Benefits of extending TDD to the functional test level
- ▶ Impact on project methodology and deliverables
- ▶ Tools

- ▶ *Origin*: An Agile technique for code construction
- ▶ *Intent*: Guarantee functional requirements are met
- ▶ *Scope*: Typically limited to the unit test level
- ▶ *Value*: Clear communication of requirements
- ▶ *Value*: Executable demonstration req's are satisfied

The TDD cycle

1. Write an executable test that expresses a functional requirement. Let it fail.
2. Write code that causes the test to pass.
3. Refactor the code to clean it up.

Repeat until all requirements are satisfied.

Check in the tests along with the code.

Next project (or iteration, or maintenance...)

Check out the tests and the code together

Run the test suite (this is a regression test)

Fix any problems (someone didn't follow TDD!)

Now you can start development

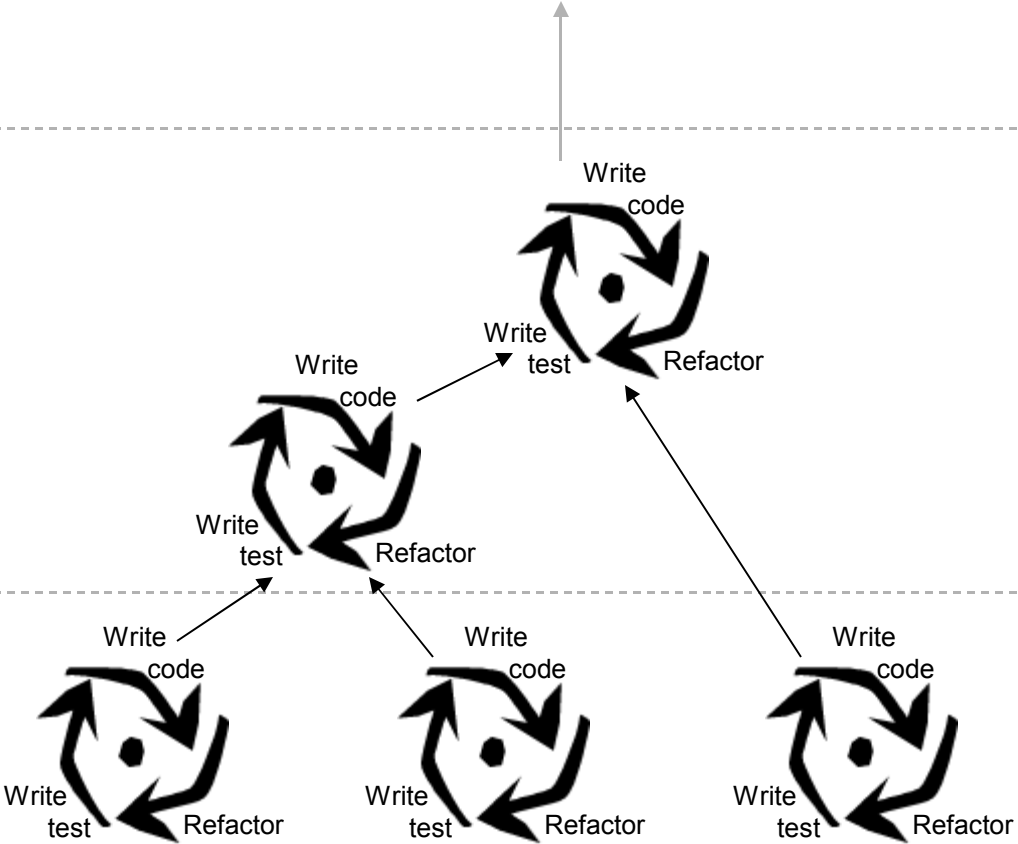
Follow the TDD cycle again

Functional

(manual)

Integration

Unit



We know...

- ▶ TDD keeps unit code tightly aligned with requirements
- ▶ TDD keeps integration code tightly aligned with requirements
- ▶ TDD automatically produces accurate regression test suites

We ask...

- ▶ What value might *functional-test* driven development offer?
- ▶ What impact would it have on our project methodology, roles, and deliverables?
- ▶ Are tools available to support it?

A problem...

- ▶ UI code quality is not at the same level as quality of other code.

Why?

- ▶ Developers work in a way that makes their own lives easier.
- ▶ With TDD, developers write code that is easy to test.
- ▶ Code that is easy to test tends to be well-structured, efficient, simple, and compliant with standards.
- ▶ Developers don't (presently) write tests for UI code.

With functional-test driven development...

- ▶ Developers will produce well-structured, efficient, simple UI code that is compliant with standards because they will write code that is easy to test – to make their own lives easier!

Another problem...

- ▶ We're producing more and more formal documentation that does not necessarily provide any value.

Why?

- ▶ Story cards are intended only to start conversations.
- ▶ We use story narratives to flesh out technical specifications.
- ▶ Story narratives are gradually evolving into requirements documents.

With functional-test driven development...

- ▶ Developers and Analysts jointly produce executable functional test cases based on the Analysts' notes and spreadsheets. The tests serve as unambiguous and accurate technical specifications.
- ▶ An executable functional test provides a demonstrable proof of functionality that a 'document' cannot provide.

An enterprise goal...

- ▶ We want to develop a repository of reliable regression tests for all our production applications.

The predictive approach...

- ▶ Functional/regression tests are developed after the fact.
- ▶ Tests can be automated only after application code is stable.
- ▶ Test cases and application code are stored/managed separately.

With functional-test driven development...

- ▶ The same test cases that guide development during a project serve as regression tests after the project.
- ▶ Test cases are stored/managed together with application code.
- ▶ No need for complex, expensive software to 'tie' things together, because everything is already together.

Organizational simplification...

- ▶ We want to focus our human resources directly on customer service to the maximum extent possible.

Issues...

- ▶ Any IT role that directly supports a customer generates value.
- ▶ An IT role that supports another IT role is overhead.
- ▶ The predictive approach requires a separate team dedicated to after-the-fact testing in support of project teams = overhead.

With functional-test driven development...

- ▶ Test cases of all kinds are built along with solutions, stored under version control with the solutions, and are accessible and usable by anyone who needs them at any time.
- ▶ There is no need for a separate team dedicated to testing, and one step removed from customers.

Today...

Analyst works with Customer to determine functional requirements. Makes notes, possibly summarizes requirements in a spreadsheet.

Analyst writes Story Card.

Analyst writes Story Narrative spelling out additional details about the functional requirements.

Developer takes Story Card.

Developer gets Story Narrative.

Analyst and Developer discuss functional requirements in detail.

Developer writes unit test cases.

Developer writes application code.

Developer writes integration test cases.

Developer writes application code.

Developer hands off to Analyst.

Analyst performs manual functional testing.

Analyst develops automated functional tests.

Analyst hands off to separate testing team.

With FTD approach...

Analyst works with Customer to determine functional requirements. Makes notes, possibly summarizes requirements in a spreadsheet.

Analyst writes Story Card.

Developer takes Story Card.

Analyst and Developer create executable functional test case(s).

Developer writes unit test cases.

Developer writes application code.

Developer writes integration test cases.

Developer writes application code.

Developer satisfies functional test cases.

Analyst performs manual functional testing.

Differences...

- ▶ Two (2) steps eliminated from the development process:
 - > Analyst creates Story Narrative
 - > Analyst develops automated functional tests
- ▶ Two (2) new steps in the development process:
 - > Analyst and Developer create executable functional tests
 - > Developer satisfies functional test cases
- ▶ Three (3) fewer handoffs:
 - > Analyst to Developer (requirements)
 - > Developer to Analyst (manual functional tests)
 - > Analyst to external testing group
- ▶ One (1) fewer external group:
 - > External testing group
- ▶ Two (2) fewer deliverables:
 - > Story Narrative
 - > Handoff materials to external testing group
- ▶ One(1) new deliverable:
 - > Functional/regression test suite
- ▶ Two (2) fewer levels of abstraction between Customer and those providing services to Customer:
 - > Between Analyst and Developer for requirements specification
 - > Between external testing group and Customer

Levels of testing...

- ▶ **Regression** – comprehensive test of previous version of the system prior to making any modifications to the code
- ▶ **Unit** – a single, discrete unit of code (unit = big enough to mean something, small enough to have few or no dependencies on other code)
- ▶ **Integration** – two or more units of code that work together to perform a definable function (not necessarily a “function” visible to the end user)
- ▶ **Functional** – a business function visible to the end user, tested at the UI level or as “report” output (same test cases as Regression test)
- ▶ **System** – comprehensive test of the solution including live interfaces to external systems, network resources, databases, etc.
- ▶ **Acceptance** – demonstration of the solution to the Customer (out of scope for this presentation)
- ▶ **Performance, recoverability, load**, etc. – environmental tests (out of scope for this presentation)

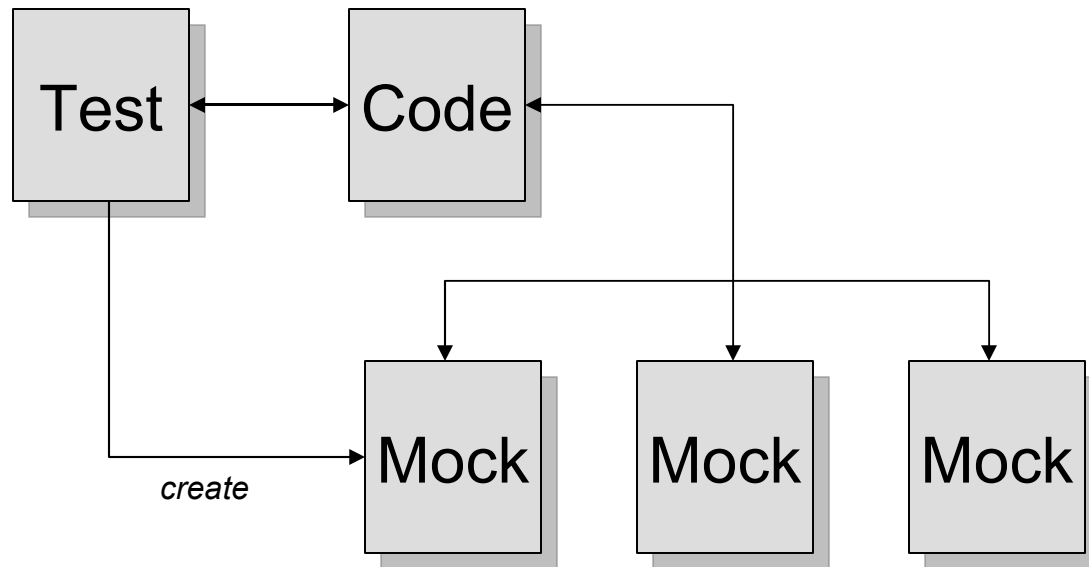
Unit tests may involve:

- “Business” logic
- External interfaces
- Presentation layer
- UI layer

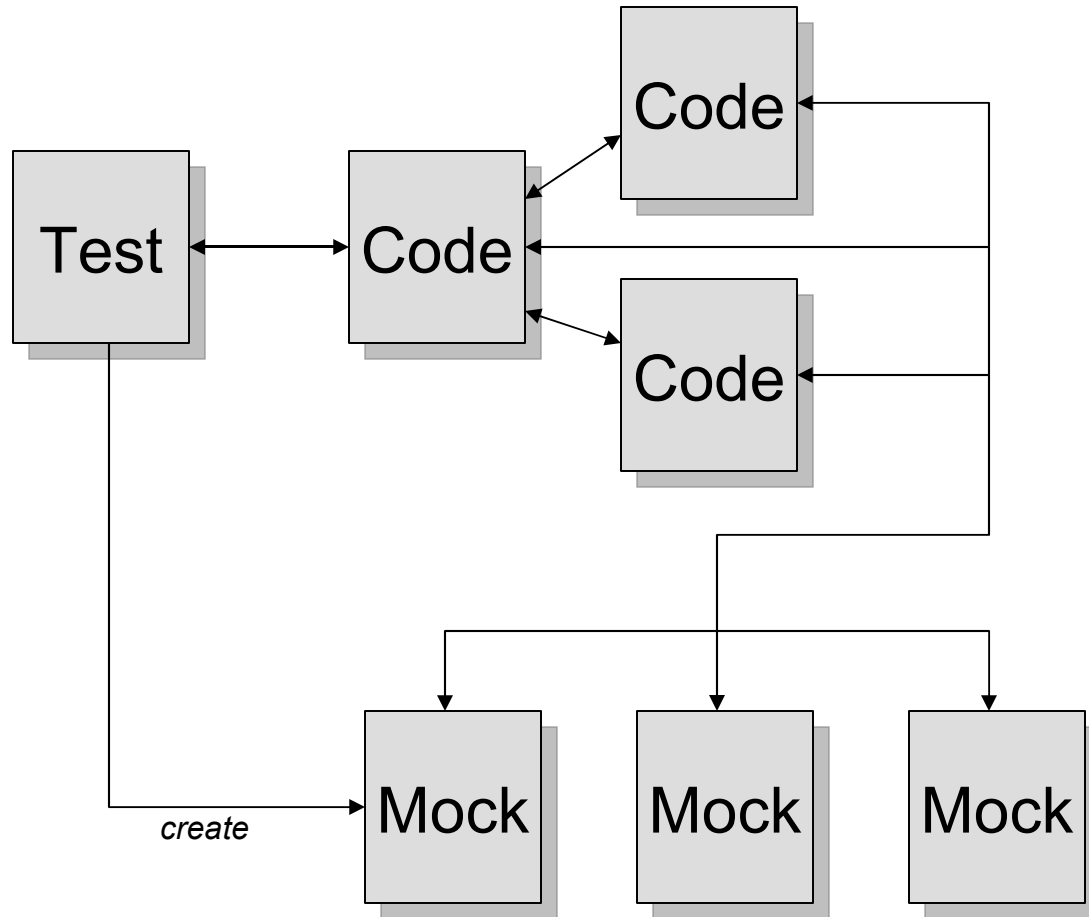
Where does functional testing fit in?

| | Who | Scripted? | Auto? | Mock | Today? |
|-------------|------------|-----------|-------|-------------------------|--------|
| Regression | Anlst, Dev | Yes | No | External interfaces | No |
| Unit | Dev | Yes | Yes | Everything outside unit | Yes |
| Integration | Dev | Yes | Yes | External to component | Yes |
| Functional | Anlst, Dev | Yes | No | External interfaces | No |
| System | Dev | Yes | No | Nothing | Yes |

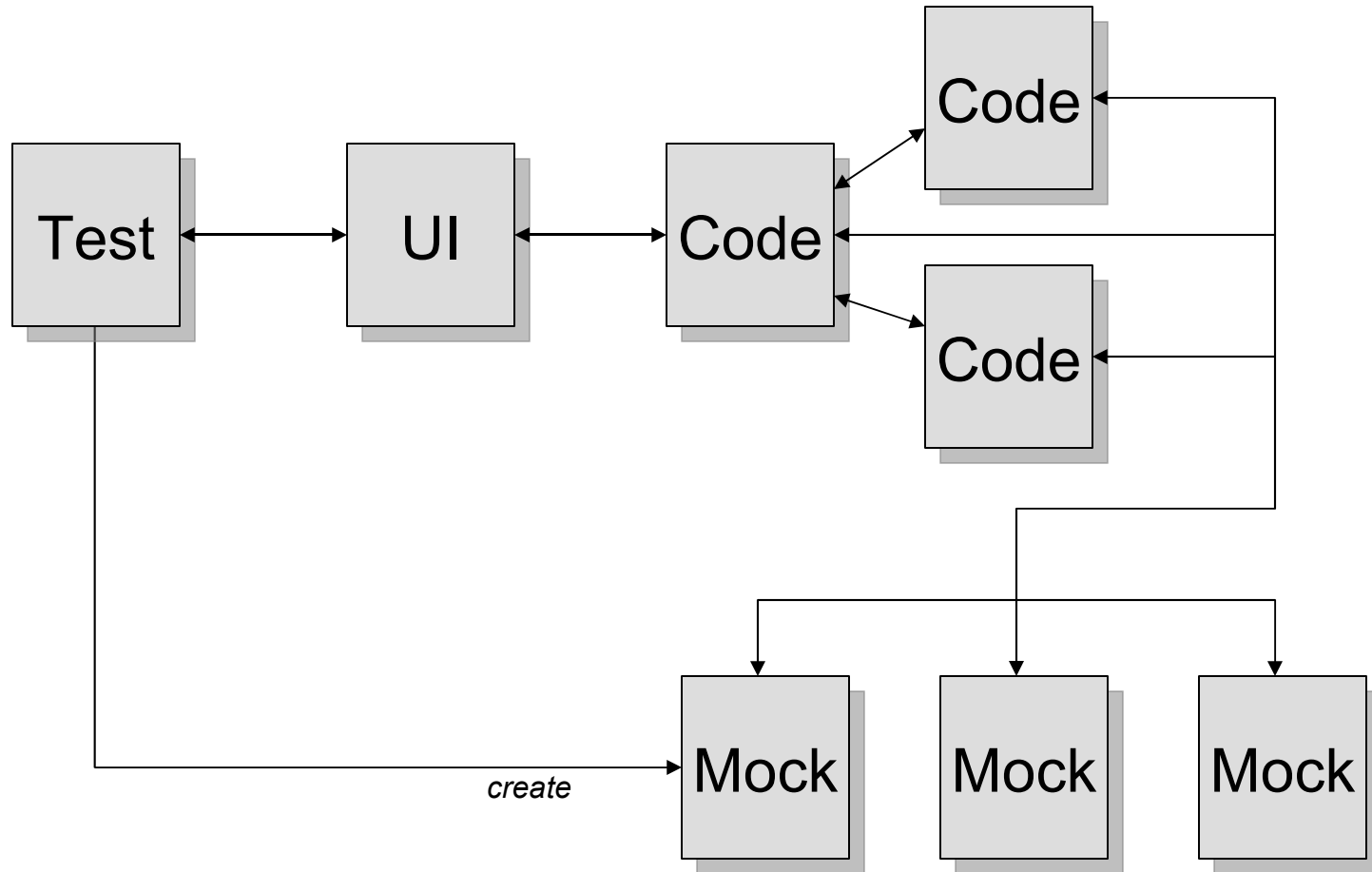
Unit test...



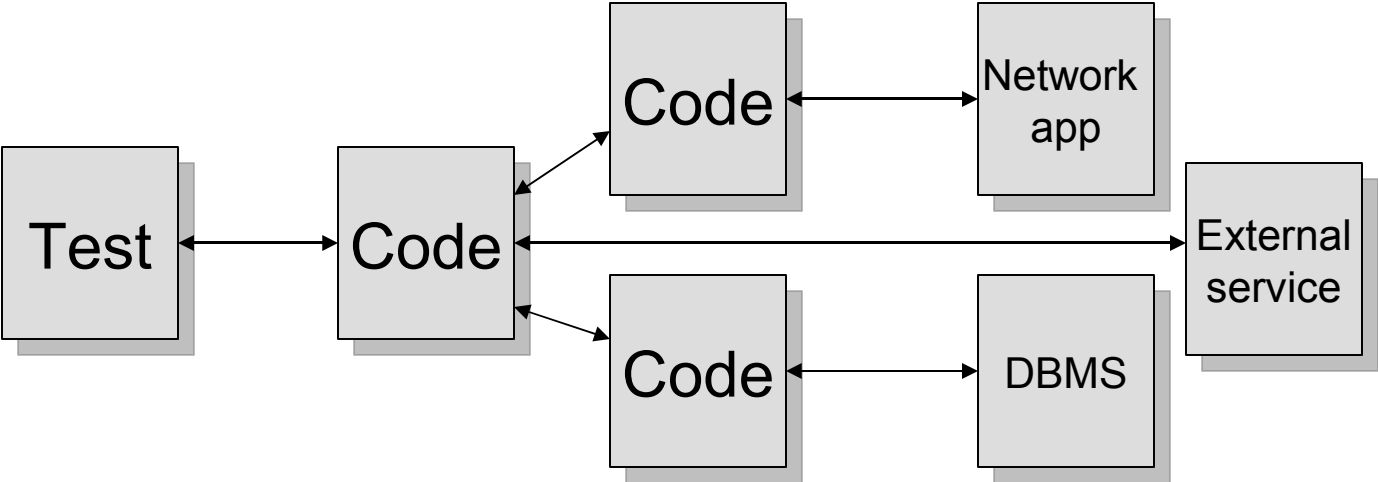
Integration test...



Functional test...



System test...



Summary of benefits...

- ▶ Closes a gap in test coverage
- ▶ Assures high-quality UI code
- ▶ Simplifies organizational structure
- ▶ Reduces the amount of formal paperwork
- ▶ Improves communication within the team
- ▶ Reduces the number of hand-offs
- ▶ Guarantees a reliable regression test suite
- ▶ Reduces distance between Customers and IT services

Testing tools must fit the methodology...

Tools that work well for a predictive approach may not work as well for an adaptive approach.

- ▶ Testing occurs at different points in the development cycle
- ▶ Testing is performed for different reasons
- ▶ Test cases are created by people in different roles, different skillsets
- ▶ Test results are evaluated by people in different roles, different skillsets

General characteristics of testing tools...

for functional testing of webapps (both predictive and adaptive)

- ▶ Supports industry standard browsers
- ▶ Supports JavaScript in HTML pages
- ▶ Supports AJAX
- ▶ Works with our desktops, servers, source code repository, etc.
(no requirements to purchase anything additional)

Characteristics that support adaptive methods...

for functional testing of webapps

- ▶ Supports TDD – test cases can be written before code exists
- ▶ Supports continuous integration – test cases can be executed automatically by continuous integration server.
- ▶ Supports refactoring – Developers can use the same tools and environment to work with test cases as they do for applications.
- ▶ Supports tech environment – test cases can be stored in the same version control repository as application code.
- ▶ Usability tailored for appropriate user community - Developers