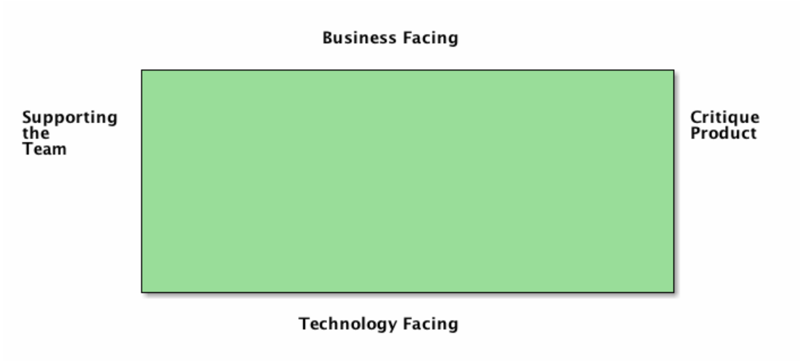
Software Testing

Mikael Svahnberg[1]

2017-03-30

[1] Mikael.Svahnberg@bth.se

Testing Goals



(L. Crispin and J. Gregory, “Agile Testing – A Practical Guide for Testers and Agile Teams”, Pearson Education, 2009.)

Discuss: What types of testing can you expect to see in each corner of this square?

Lower Left Quadrant: Technology Facing / Supporting the Team

- Testability of
  - API's, Ports, Adapters
- Test database access, updates
- Business Logic and Presentation separated
- Isolated tests
  - to isolate problems
- Internal Quality
- Infrastructure

Safety Net!

- build confidence
- go faster, do more
- support refactoring

Examples of Techniques

- Unit Tests
- Component Tests

What, Who, When?

- Unit Tests
  - Developer Intent and program design
  - Does the code-in-the-small do what it is expected to do
- Component Tests
  - Architect’s intents – system design
  - Do components work together as expected
- The Programmer writes the test and test them
- Run tests in Continuous Integration tool

Toolkit

- Source Code Management
  - Version Control
  - Who changed what?
  - Be able to restore to older version
- IDE
  - Compile, Debug, Build GUI, Unit Test, Refactor
- Unit Tests
  - e.g. xUnit
- CI tools
  - e.g. Jenkins, Travis.ci, Drone.io, ...

Upper Left Quadrant: Business Facing / Support the Team

- Drive Development with business-facing tests
- Ask the right questions
- Help customers clarify
- Captuer examples, express as executable tests
- External Quaity
- Know when we're done.

Examples of Techniques

- Functional Tests
- Examples
- Story Tests
- Prototypes
- Simulations

What, Who, When?

- Testers, Developers
- Collaboration with customers
- Team responsibilty
- Start of Iteration
    - Business facing tests drive development
- Throughout Iteration
    - No story done until tested

Toolkit

- Checklists
- Mind Maps
  - Brainstorming
  - Words, ideas, tasks
- Mockups / Paper Prototypes
  - User-centered design
- Flow Diagrams
- Whiteboards
- Behaviour-Driven-Development
  - Cucumber, easyB, nbehave, rspec
- GUI Test tools/libraries/frameworks
  - e.g. Selenium, Cucumber, Canoo WebTest, Robot Framework . . .

Upper Right Quadrant: Business Facing / Critique Product

- Recreate actual user experiences
- Realistic use
- Learn as you test
- Context
    - What works for your situation
- Constructive

Examples of Techniques

- Customer Demos
- Exploratory Testing
- Scenarios
- Usability Testing
- User Acceptance Testing
- Alpha/Beta Testing

Discuss How does this relate to UML and RUP? Discuss Are these tests automated or manual?

Also behind the GUI

- Test API
    - Input/Output
    - Sequence of API calls
    - Checking log files
    - States and Transitions

What Who, When?

- Require good skills, experience, intuition, critical thinking
- Invole the customers
- As early as possible

Toolkit

- Time
- Experience
- Some of upper left quadrant tools may apply
    - e.g. Selenium, Cucumber, Canoo WebTest, Robot Framework . . .

Lower Right Quadrant: Technology Facing / Critique Product

Quality Attributes, e.g.:

- Performance
- Stability
- Reliability
- Scalability
- Maintainability
- . . .

Also

- Memory Management
- Data Migration
- Recovery

Test Environment

- Independent, production-like environment

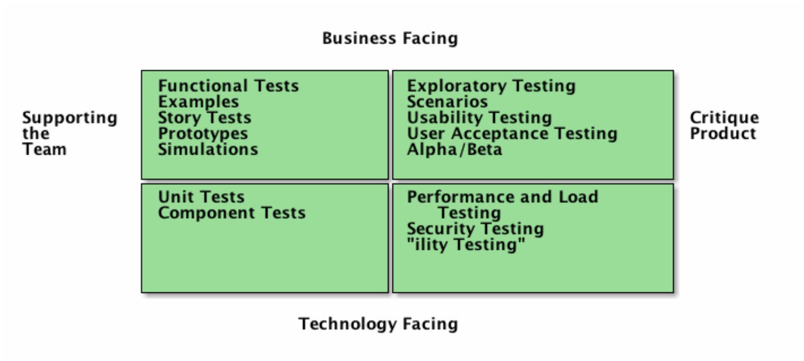
What, Who, When?

- Depends on priorities
- May be needed already from the get-go
- At least get an early baseline

Toolkit

- Data Migration, Recovery:
  - Native Database Tools
  - Shell Scripts
- Monitoring tools
  - jConsole : Application bottlenecks, memory leaks
  - jProfiler: Database usage
- Load Tests
  - Loadrunner, SilkPerformer
- Other tools
  - jMeter, jUnitPerf, . . .

Test Quadrants, Summary



Plan your Test Strategy

- Scope
- Priorities, Risks
- Tools
- Customers
- What to Document
- Consider all four quadrants
- Use lessons learned to improve

TDD: Test Driven Development

nano-cycle (second by second) Three laws of TDD:

1. You must write a failing test before you write any production code.
2. You must not write more of a test than is sufficient to fail, or fail to compile.
3. You must not write more production code than is sufficient to make the currently failing test pass.

micro-cycle (minute by minute) Red-Green-Refactor cycle

1. Create a unit tests that fails
2. Write production code that makes that test pass.
3. Clean up the mess you just made.

TDD: Test Driven Development

milli-cycle (10 minute intervals)

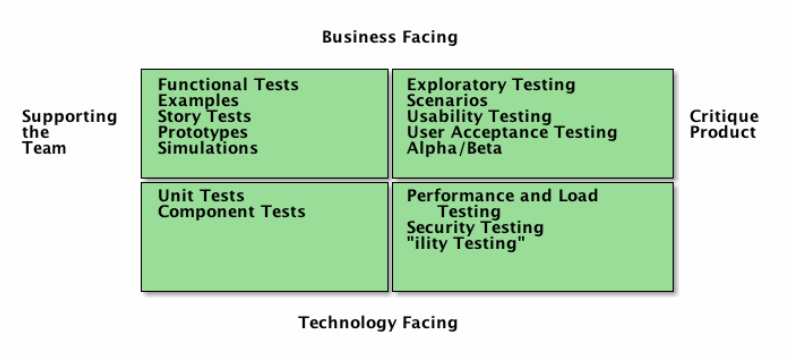
- More specific test cases $\rightarrow$ more generic code
- Code is no longer a series of special cases
- “Big Picture”
- Backtrack from too specific test cases or not general enough code

primary cycle (hour by hour)

- ensure architectural boundaries

Discuss: Testing and RUP/UML

- How does RUP/UML deal with Testing?
- What areas do RUP/UML focus on?


Business Facing
Supporting the Team
Functional Tests
Examples
Story Tests
Prototypes
Simulations
Exploratory Testing
Scenarios
Usability Testing
User Acceptance Testing
Alpha/Beta
Critique Product
Unit Tests
Component Tests
Performance and Load Testing
Security Testing
"ility Testing"
Technology Facing