Testing Large System

From Developer Perspective Based on Real Example

bartosz.kwasniewski@gmail.com

Facts

1. Software systems are more complex than ever before
2. Writing, maintaining and changing large system is hard
3. Large systems erode quickly towards legacy one
4. Teams are more complex than ever before
5. Developers own and prove quality
6. Test Engineers are valuable to the project
7. Testing no longer means testing
8. The larger the project, the practices and disciplines are more important
9. Solutions are already present

Question

How to develop, test, integrate and deliver large and complex system?

Solution

Continuous Integration (CI) with Multi-Stage System with Automated Testing Pipeline supported by Test Driven Development (TDD)

Large and Complex System

1. Many separated binaries and libraries
2. Hundred thousand lines of code each/some
3. Different compilers for same code base
4. Different control version systems
5. Developed by hundred of developers in many separated locations
6. Used in many different products and platforms
7. Different languages
8. Part of bigger system that is a part of another bigger system, …
9. Developed over decades (some developers are dead for ages)
10. Complicated communication between all services
11. Complicated and multi-tier build system
12. No tests

Legacy Code

- Difficult-to-change code that we don’t understand
- Code without tests – by Michael Feathers

Code without tests is bad code. It doesn’t matter how well written it is; it doesn’t matter how pretty or object-oriented or well-encapsulated it is. With tests, we can change the behavior of our code quickly and verifiably. Without them, we really don’t know if our code is getting better or worse.

- Use TDD - to prevent system to become a legacy one
- Use CI - to verify quickly quality of an introduced change

Testing: Testing is no longer testing

1. Checking Implementation and Finding Bugs?
2. Testing drives the design and architecture (TDD)
3. Testing defines requirements - Acceptance Test Driven Development (ATDD)

Testing: Large Project

- Larger the project is:
  - The more roles we have
  - The more people there are
- The practices are more important
- Testing are more important
- Harder to change things
- So consider a philosophy to stick to …

Testing: Kaizen Philosophy

- Continuously Improvement
- Changing the way things are
- Things are NOT all right the way they are
- Things are a mess
- Challenge everything

If you're going to do kaizen continuously, you've got to assume that things are a mess. **Too many people assume that things are all right the way they are.** Kaizen is about changing the way things are. If you assume that things are all right the way they are, you can't do kaizen. So change something!

Testing Guides

1. Challenge assumptions about testing
2. Avoid using complex testing terminology
3. Keep testing classifications simple
4. Don’t separate development and testing
5. Testers and programmers should work together
6. Educate and coach testing
7. Create community of testing
8. Don’t have separate test automation teams
9. Feature team as test automation team
10. All tests pass - stop and fix
11. Have zero tolerance on open defects

Testing: Challenge assumptions about testing

- Testing must be independent and thus separated from development
- Testing cannot start before coding is finished
- Testing follows the sequence (a test waterfall) of: design, execution, reporting
- There must be a separate test department
- There must be a *test manager*
- Testing must be done at the end
- Testing must be “well planned”
- There must be a “testing strategy” and a “master test plan”
- 100% coverage is too expensive
- 100% test automation is too expensive
- Testing requires a sophisticated test-management tool
- Testing must be done by ‘testers’

Testing: Avoid using complex testing terminology

- unit test
- functional test
- stress test
- interoperability test
- load test
- installation test
- monkey test
- documentation test
- module test
- system test
- stability test
- compatibility test
- traffic test
- security test
- exploratory test
- acceptance test
- developer test
- integration test
- regression test
- reliability test
- performance test
- capacity test
- usability test
- user-acceptance test
- system component test
- single system component test
- multi system component test
- active module test
- smoke test
- hardware test

DO NOT READ IT !!!

Testing: Keep testing classifications simple

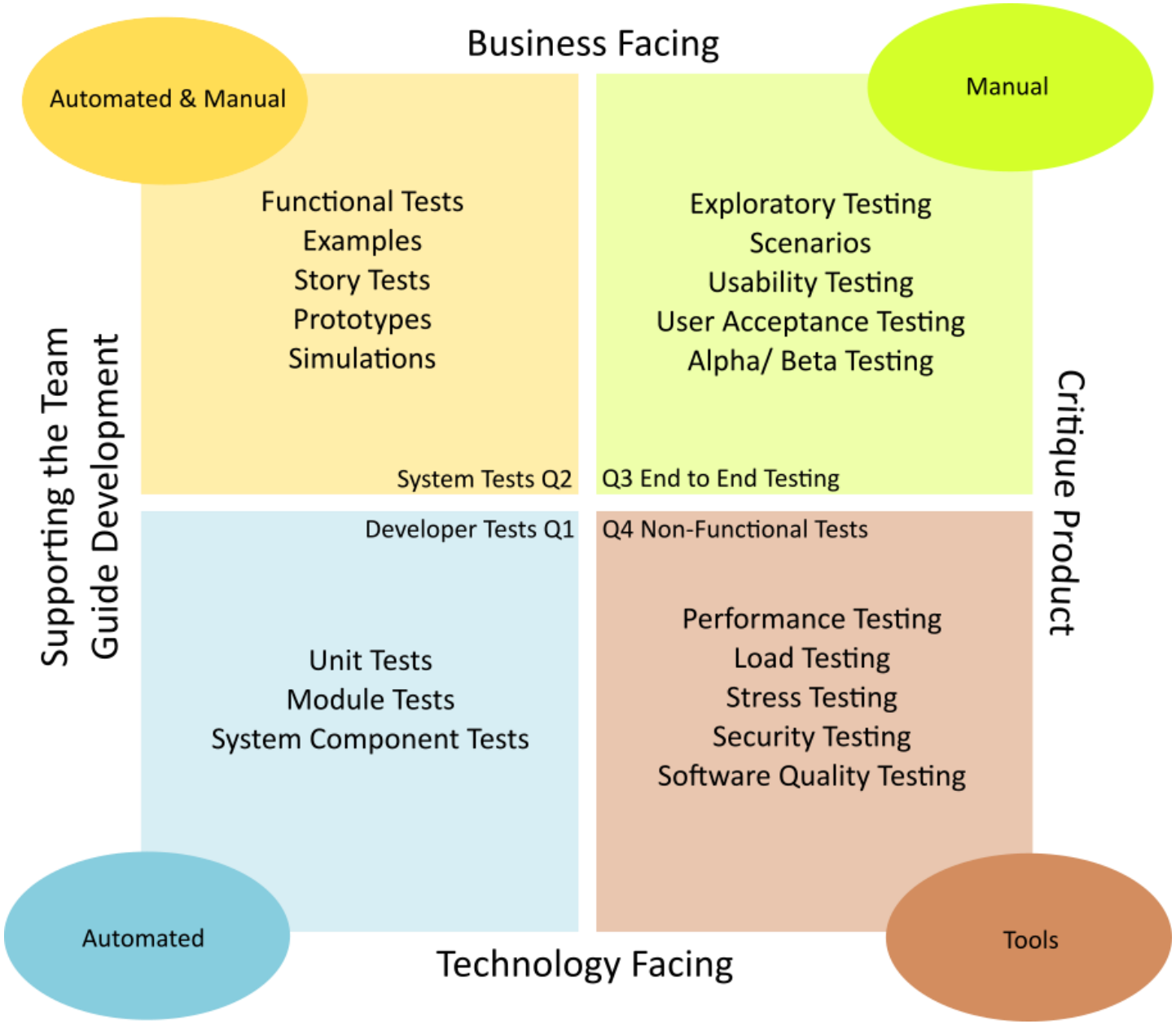
Straightforward terminology inspires intelligent behavior

Developer Tests	Customer-Facing Test
technology-facing tests	critique the product
check implementation	check the requirements
check developer intent	functional and non-functional
done by programmers	by testers
during coding	after feature is implemented

But …

Agile Testing Quadrants

- The quadrant numbering system does NOT imply any order
- Test Engineers participate in all quadrants
- Developers participate in all quadrants
- Q1 - Unit Tests and natural for Developers
- Q2 - System Tests and natural for Testers
- Q3 – End to End Tests
- Q4 – Performance and Stability Tests



Testing: Don’t separate development and testing

- Previously

> Testing should always be done by an outside party … System testing should always be done by an independent group, a separate quality-assurance department.

- In Agile Era

> Teams often confuse “independent” with “separate.” … and a division between programmers and testers is inevitable, time is wasted on duplicate meetings, programmers and testers don’t share a common goal, …

- Test independence does not mean independent testers.
- By writing tests before implementing code - TDD

Testing: Testers and programmers should work together

- Separating divisions leads to a conflict between programmers and testers
- Why?
- Programmers - with their ego in their code - defend themselves, their code, and the program.
- Testers - try to prove that part of the program is faulty

In a Scrum team, *'testers' are no longer testers* but 'simply' members of the team - with testing as their *primary specialization* . 'Programmers' are *any* members of the team who can code. Every member of the team has a shared goal and is held–as a team–accountable to that goal.

Testing: Educate and coach testing

- Especially in large organizations, testing skills are not respected.

"Everybody can do that" and people are randomly taken from the street to be testers, a temporary stage they need to go through before advancing to a "real job"

- Good testing skills come with deliberate practice and time.

Testers who don't bother to learn new skills and grow professionally contribute to the perception that testing is low-skilled work.

- Prevent this by developing the team's testing expertise.

Testing: Create community of testing

- Open discussion and experience-sharing foster learning
- Participate in conferences such as test::dive
- Test Managers can use their expertise in testing and management

Rather than keeping the testers separate, think about a community of testers. Provide a learning organization to help your testers with sharing ideas and helping each other.

- Lean thinking is a proven system that scales to large development

Testing: Happiness?

- Happiness:
  - All developers are doing TDD with big smile on their face
  - Testers and developers are working all together in a Scrum team
- But:
  - Slow Development Cycle
  - Lack of Test Automation
- Dream about:
  - Continuous Integration
  - Continuous Delivery
  - Continuous Deployment
- Who is going to be responsible for creation of test automation framework, tests creation itself and the maintenance of both

Testing: Don’t have separate test automation team

- A separate automation team
  - causes additional complexity
  - is a quick fix
  - harmful in the long run
- Why:
  - Creating testware requires deep understanding of the product.
  - Maintenance and evolution is more effort than initial creation.
  - Insights obtained during testware creation is perhaps more important than the testware itself.
  - Creating testware without using it leads to complex and unusable testware.

Testing: Feature team as test automation team

- A feature team can temporarily take on the role of test-automation team
  - They have a deep understanding of the system.
  - They can take a small feature so that the automation is concrete and realistic.
  - The learning created during test - automation will not be lost.
  - There is visibility into test automation as the items go on the Product Backlog
- Additionally
  - Create the initial test framework, produce training material, and support the teams

Testing: All tests pass - stop and fix

When automated tests fail, fix them immediately

- Regression takes precedence over any feature or bug fix
- We cannot afford to stop “production line”

Testing: Have zero tolerance on open defects

- First, focus on preventing defects
- Then, have zero tolerance on open defects
- And fix it as soon as possible
- It prevents:
  - Effort spent on tracking many defects
  - Effort spent on prioritization
  - Delaying the learning that happens when fixing a defect
  - Spending extra time on trying to remember implementation
- No creative accounting and don’t fooling yourself

Testing: Summary

- Testing is a collaborative activity
- Testers and developers should work together in Scrum team
- Developers should do testing by doing TDD
- Drives design and prevents the code becoming legacy
- Defines specification by examples

- Now its time for continuous integration and delivery to have
- There should be as short as possible development cycle

Eggs and Ham

1. It's better to catch a bug at design time than during unit testing
2. It's better to catch a bug during unit testing than during system component testing
3. It's better to catch a bug during system component testing than in beta system testing
4. It's better to catch a bug during beta system testing than have your customer catch one
5. It's better to have your customer catch a bug than to have no customers ;)

Bibliography

1. Jez Humble and David Farley. Continuous Delivery. 2010
2. Michael C. Feathers. Working Effectively with Legacy Code. 2005
3. https://less.works/less/technical-excellence/continuous-integration.html
4. https://less.works/less/technical-excellence/thinking-about-testing.html
5. https://martinfowler.com/bliki/ContinuousDelivery.html
6. https://www.atlassian.com/software-testing

Appendix: More is LeSS

Organizational Agility is constrained by Technical Agility

One of the best sources for Large Scale Development is LeSS.

Most of this presentation is based on LeSS articles.

Craig Larman & Bas Vodde