Agile Testing

Towards Continuous Quality Management

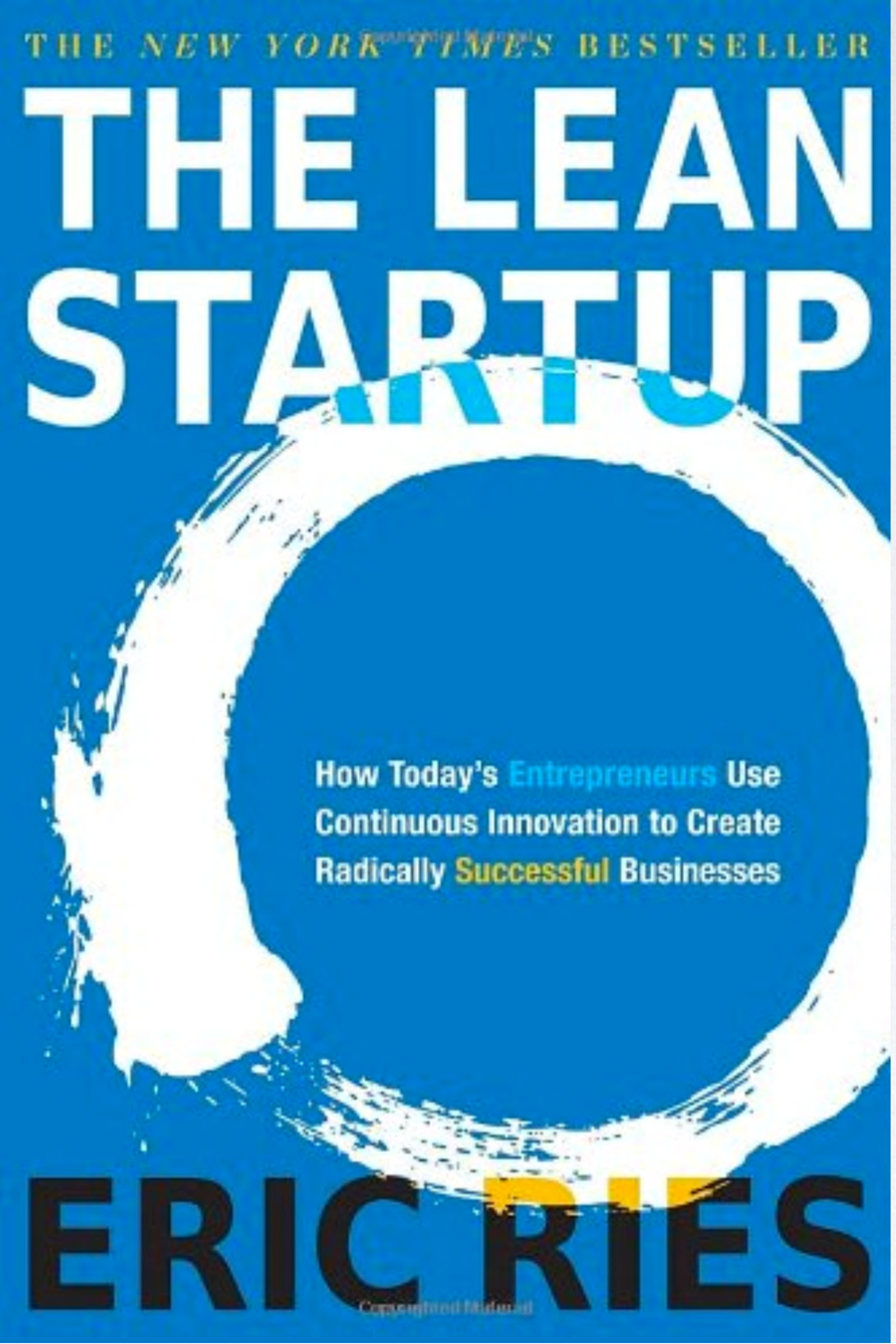
This presentation is taken from the 2016 / 17 ones of Prof. Olivier Liechti from HEIG-VD, with the demos replaced by :

- a tutorial of Docker by Pascal for continuous delivery (today);
- demos of SonarQube; EsLint and (partially Travis CI)

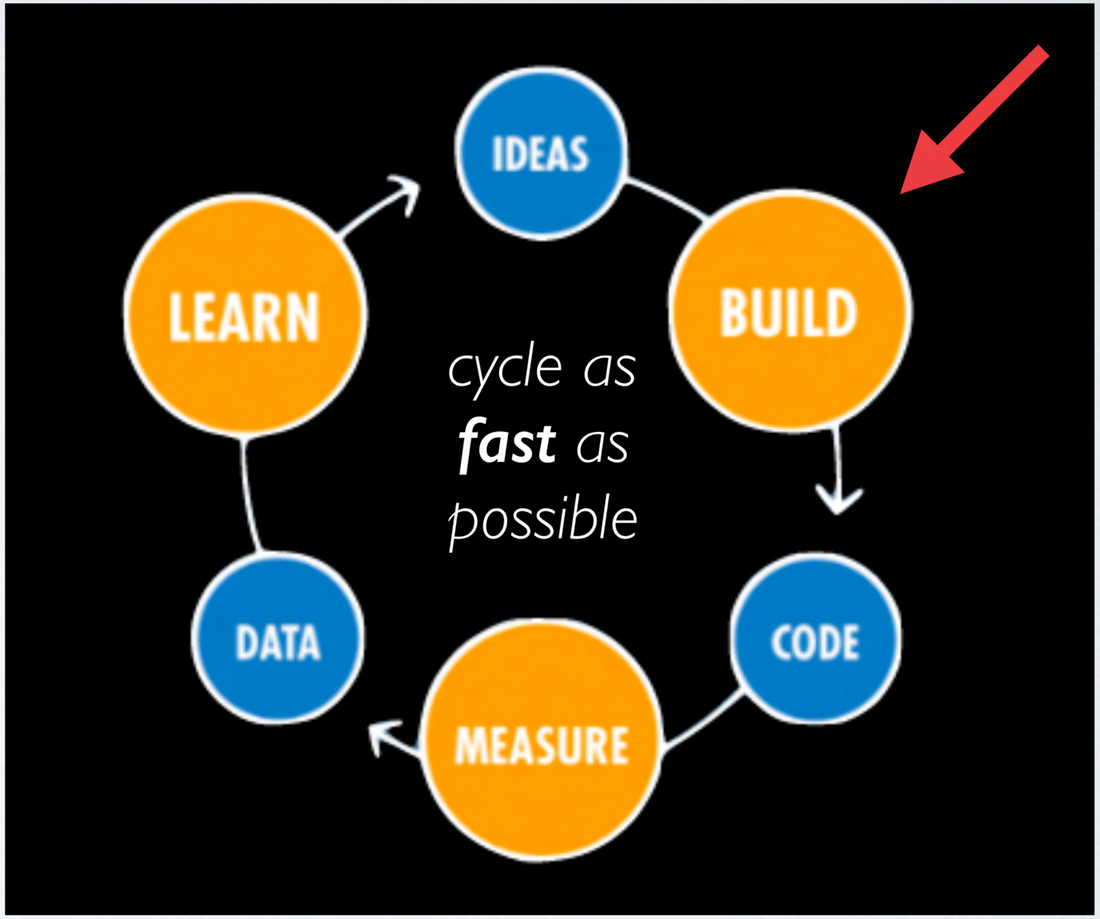
TODAY'S AGENDA

- **Context**
  - Agile software development & continuous delivery pipelines
- **Agile Testing**
  - Software quality
  - Agile testing quadrants and selected testing methods

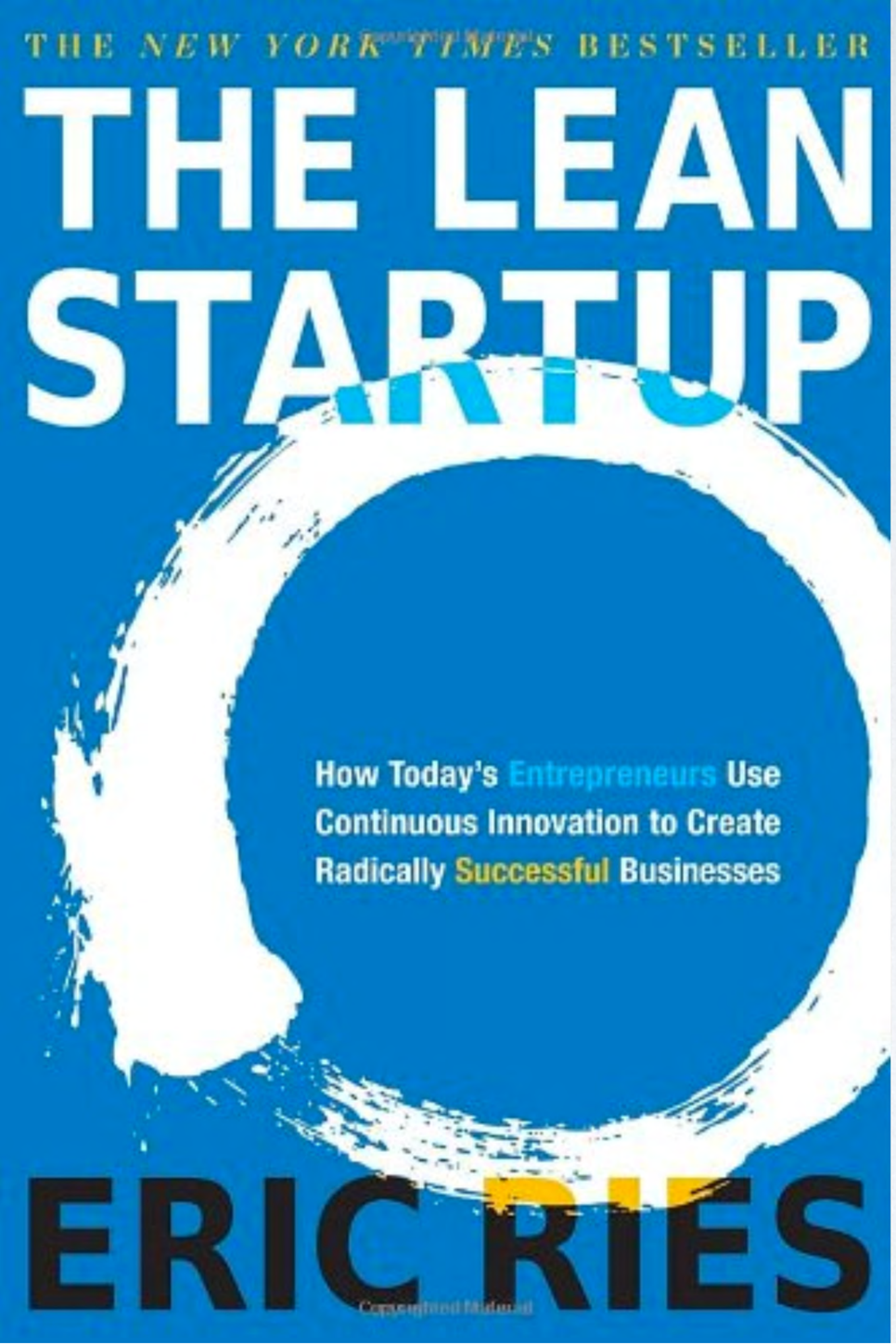
Continuous Delivery



MOTIVATION



http://www.startuplessonslearned.com/2009/06/why-continuous-deployment.html

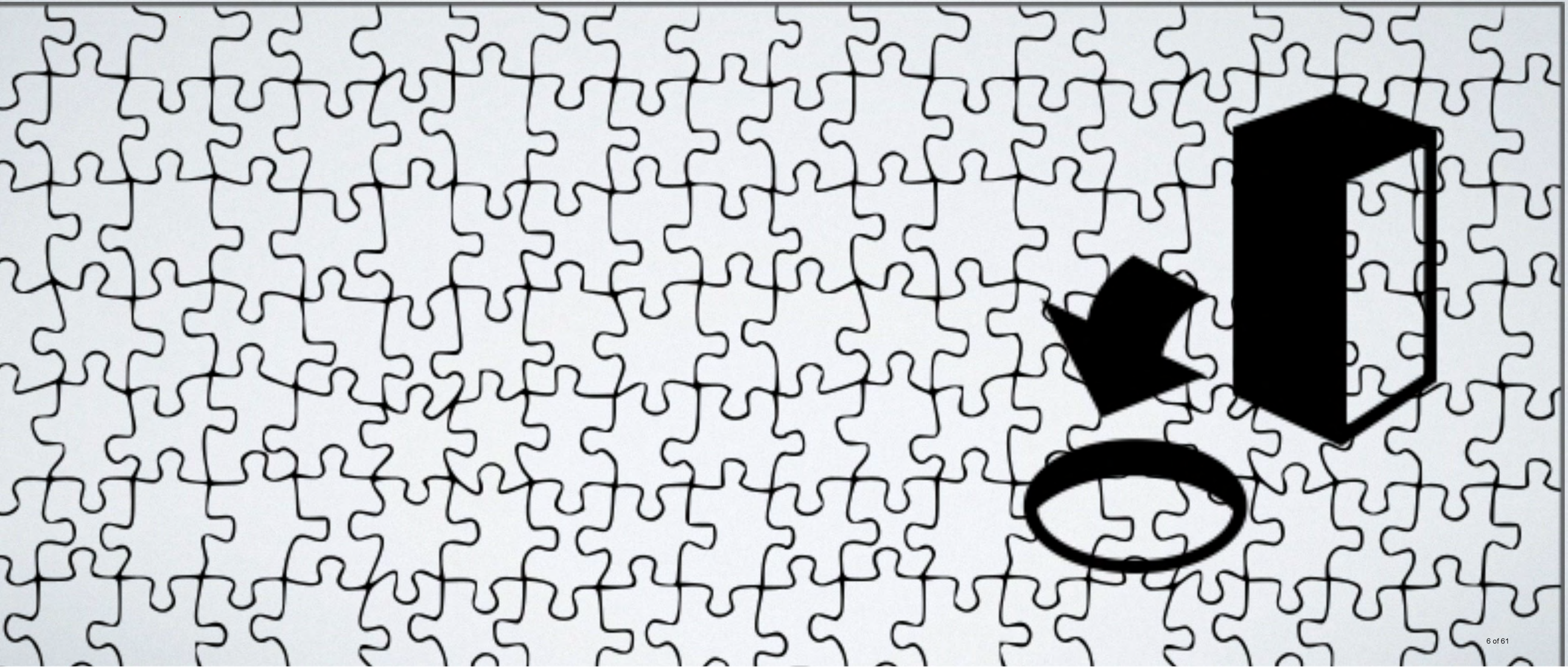


MOTIVATION

*How do we **keep a fast validated learning cycle**, even when building a complex software product, used by of people on a 24/7 basis?*

*How can **software engineering practices** and **tools** help us?*

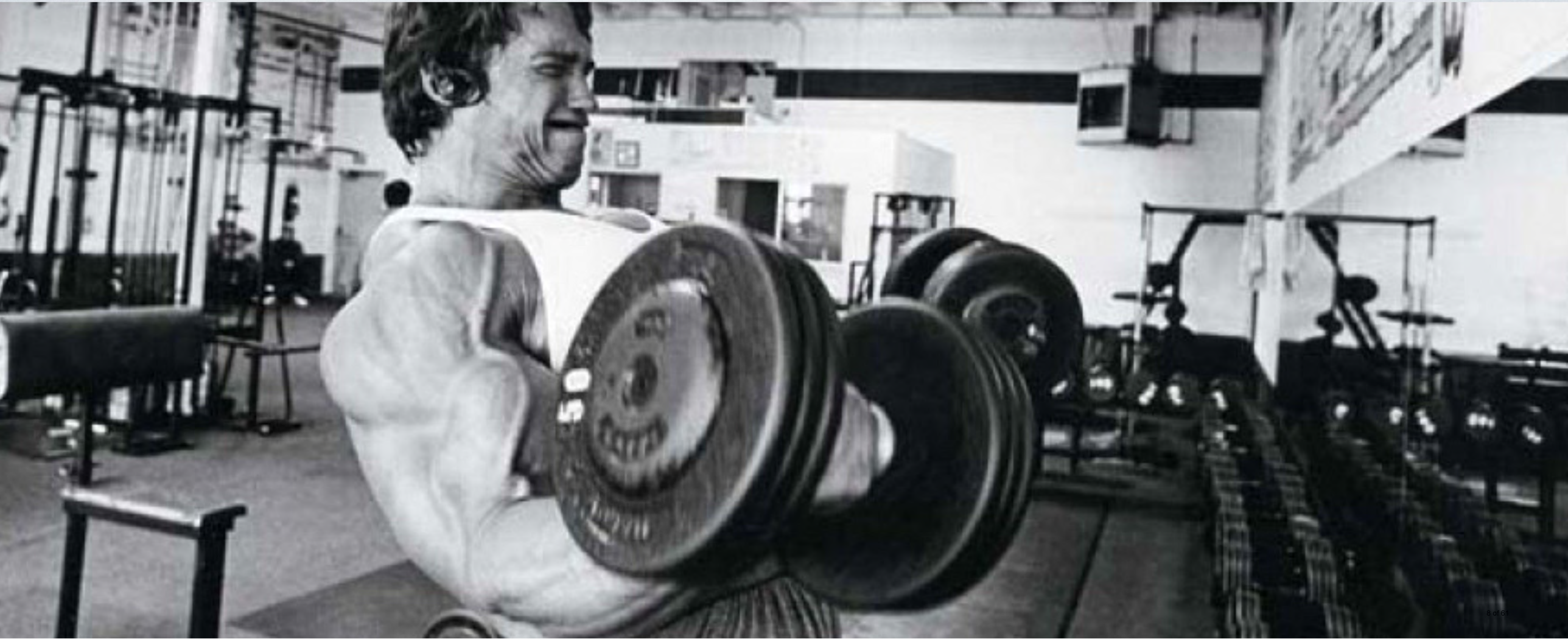
CONTINUOUS INTEGRATION



INTEGRATION IS A PITA

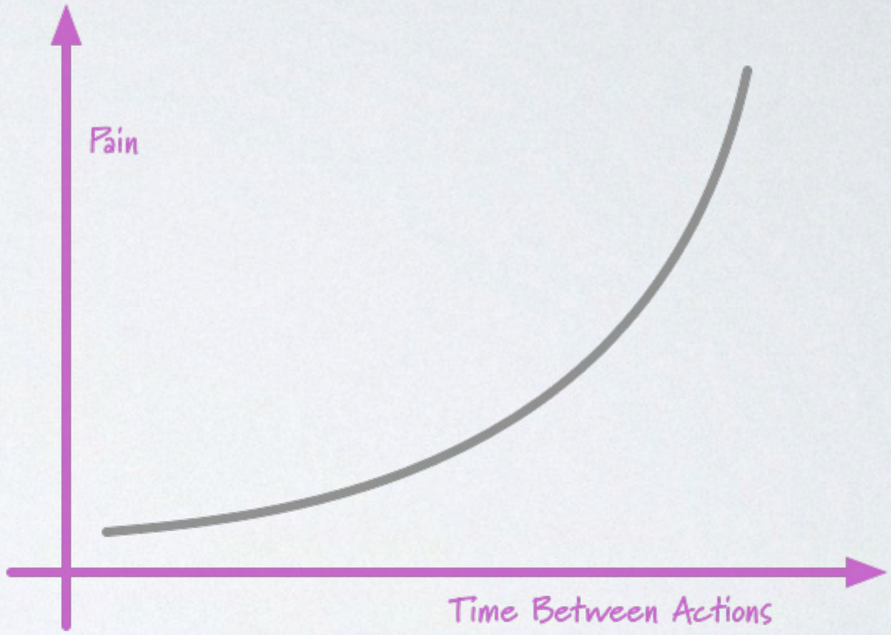
- **Team members work on several components in parallel**
- **The components need to be integrated to form a product**
- **Doing this integration is always painful**
- **But there are ways to reduce the pain**

IF IT HURTS ... DO IT MORE OFTEN!



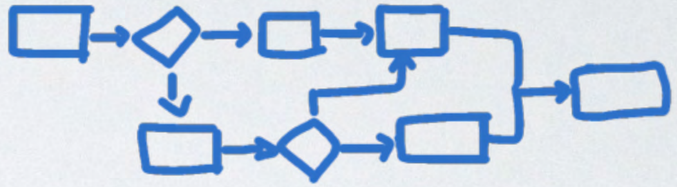
MAKE INTEGRATION VERY FREQUENT

- Doing one big task at once is way more painful than doing **many small tasks**.
- **Frequent feedback** gives us a chance to make the same task iteratively less painful.
- Doing a task repeatedly **trains** you and makes you better at it. It also encourages you to automate (some of) the process.

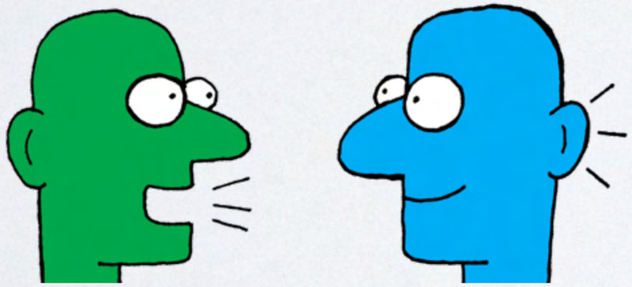


http://martinfowler.com/bliki/FrequencyReducesDifficulty.html

HOW DO WE DO IT?



Tools ***Culture*** **Processes**



Communication **Discipline**

IN PRACTICE

- Maintain a code repository
- Everyone commits to the baseline every day
- Every commit (to baseline) should be built
- Automate the build
- Make the build self-testing
- Everyone can see the results of the latest build (feedback)

SonarQube
ESLint

CONTINUOUS INTEGRATION

Your Team is happy.

Everyone is pushing commits very frequently. We know that we are always able to build a complete software system with all the components. We even have some unit tests.

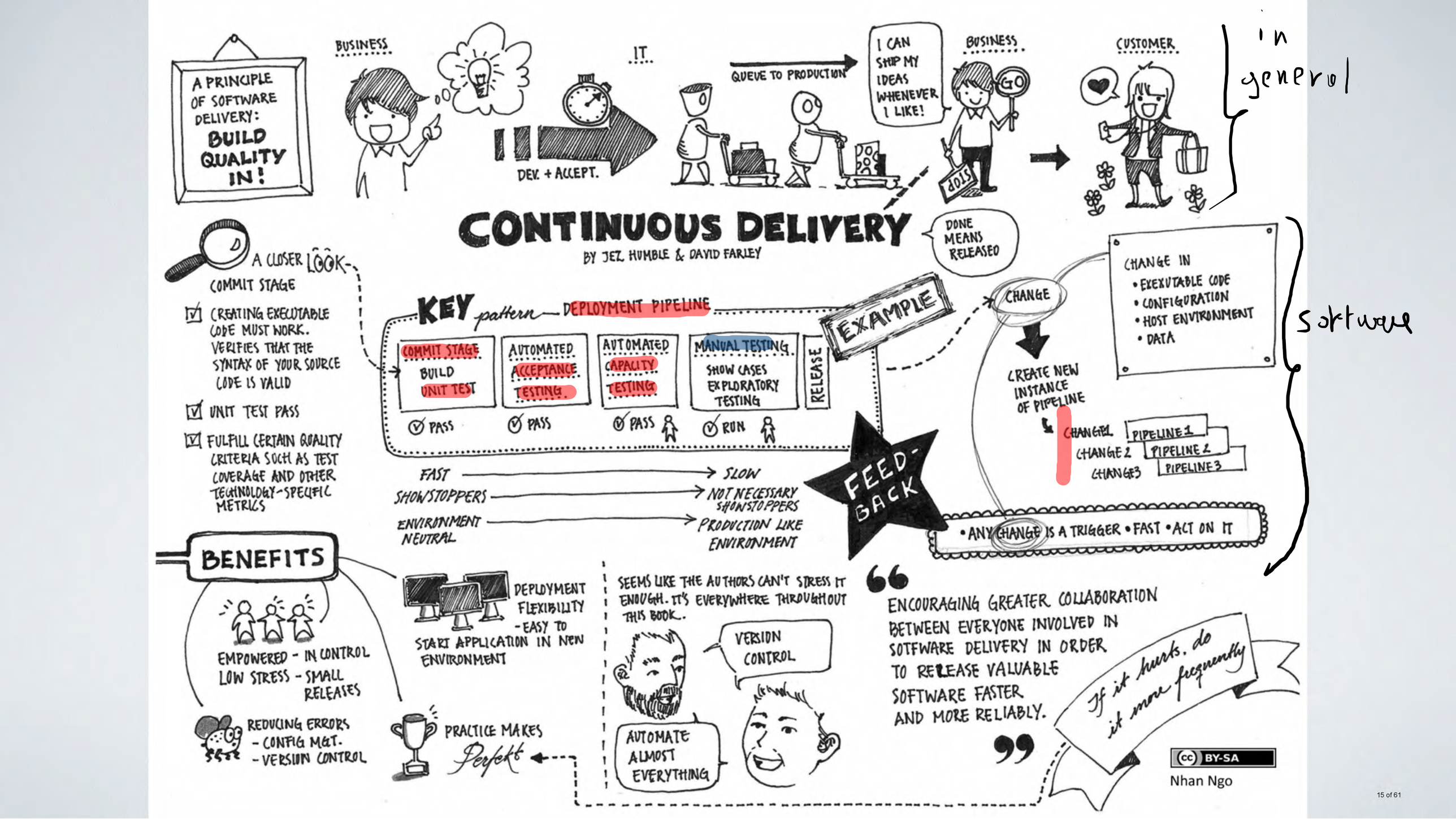
Of course, problems do happen. But we identify them quickly. Not 2 hours before an important demo or 2 days before launch.

By the way, setting up our continuous integration server was actually easy.

DON'T STOP THERE

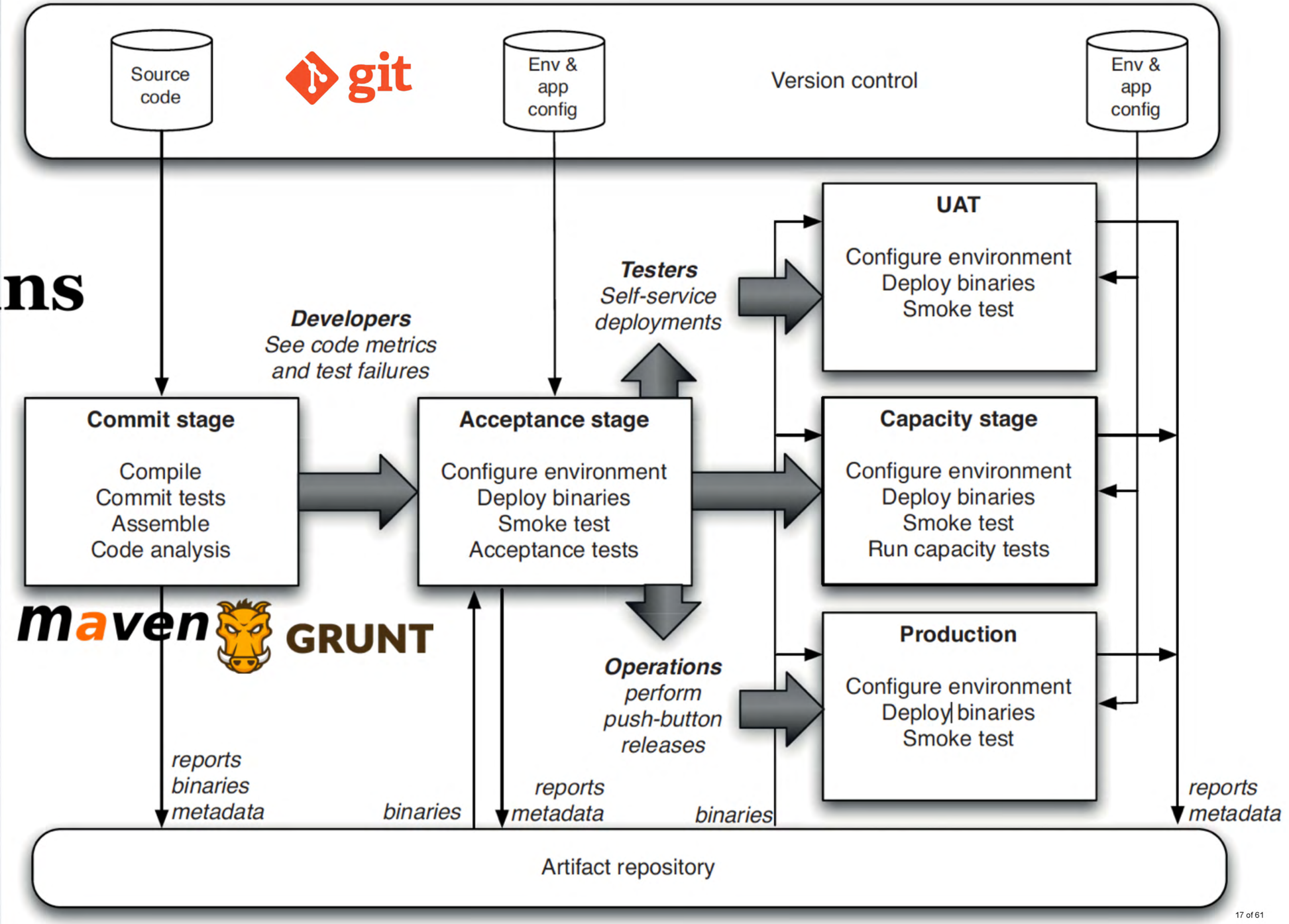
- **Continuous integration is only a first step**
- **Your team might be happy, but your customers don't care**
  - A working software binary built on a daily basis and its components work together.
  - Your customers don't have access to this binary.

	Continuous Integration	Continuous Delivery
Who	Developer centric	Whole team Business/QA/Dev/Ops
Output	**software package**	**feature available to users**
Scope of quality control	Narrow: Unit / integration tests	Broad: unit, integration, UX, perf, security, etc.
Complexity / cost	Low	High to very high
Impact on agility	Medium	High to very high


A PRINCIPLE OF SOFTWARE DELIVERY: BUILD QUALITY IN!
BUSINESS
IT
DEV. + ACCEPT.
QUEUE TO PRODUCTION
I CAN SHIP MY IDEAS WHENEVER I LIKE!
BUSINESS
GO
STOP
CUSTOMER
in general
CONTINUOUS DELIVERY
BY JEZ HUMBLE & DAVID FARLEY
DONE MEANS RELEASED
A CLOSER LOOK-
COMMIT STAGE
☑ CREATING EXECUTABLE CODE MUST WORK. VERIFIES THAT THE SYNTAX OF YOUR SOURCE CODE IS VALID
☑ UNIT TEST PASS
☑ FULFILL CERTAIN QUALITY CRITERIA SUCH AS TEST COVERAGE AND OTHER TECHNOLOGY-SPECIFIC METRICS
KEY pattern— DEPLOYMENT PIPELINE
COMMIT STAGE
BUILD
UNIT TEST
AUTOMATED ACCEPTANCE TESTING
AUTOMATED CAPACITY TESTING
MANUAL TESTING
SHOW CASES EXPLORATORY TESTING
RELEASE
✓ PASS
✓ PASS
✓ PASS
✓ RUN
EXAMPLE
CHANGE
CHANGE IN
• EXEXUTABLE CODE
• CONFIGURATION
• HOST ENVIRONMENT
• DATA
CREATE NEW INSTANCE OF PIPELINE
CHANGE1
CHANGE 2
CHANGE3
PIPELINE 1
PIPELINE 2
PIPELINE 3
FAST → SLOW
SHOWSTOPPERS → NOT NECESSARY SHOWSTOPPERS
ENVIRONMENT NEUTRAL → PRODUCTION LIKE ENVIRONMENT
FEED-BACK
• ANY CHANGE IS A TRIGGER • FAST • ACT ON IT
software
BENEFITS
EMPOWERED - IN CONTROL
LOW STRESS - SMALL RELEASES
DEPLOYMENT FLEXIBILITY - EASY TO START APPLICATION IN NEW ENVIRONMENT
REDUCING ERRORS
- CONFIG MGT.
- VERSION CONTROL
PRACTICE MAKES Perfekt
SEEMS LIKE THE AUTHORS CAN'T STRESS IT ENOUGH. IT'S EVERYWHERE THROUGHOUT THIS BOOK.
VERSION CONTROL
AUTOMATE ALMOST EVERYTHING
“ENCOURAGING GREATER COLLABORATION BETWEEN EVERYONE INVOLVED IN SOTFWARE DELIVERY IN ORDER TO RELEASE VALUABLE SOFTWARE FASTER AND MORE RELIABLY.”
If it hurts, do it more frequently
BY-SA
Nhan Ngo

QUALITY - PIPELINES - THE LAST MILE

Jenkins


Source code
git
Env & app config
Version control
Env & app config
UAT
Configure environment
Deploy binaries
Smoke test
Testers
Self-service deployments
Developers
See code metrics and test failures
Commit stage
Compile
Commit tests
Assemble
Code analysis
Acceptance stage
Configure environment
Deploy binaries
Smoke test
Acceptance tests
Capacity stage
Configure environment
Deploy binaries
Smoke test
Run capacity tests
maven
GRUNT
Production
Configure environment
Deploy binaries
Smoke test
Operations
perform push-button releases
reports
binaries
metadata
binaries
reports
metadata
binaries
reports
metadata
Artifact repository

To better understand real pipelines in action, see :

- Arnaud's demo and Pascal's tutorial,
- and / or search youtube.

For more concrete examples we:

- Pascal's tutorial about Docker
- Arnaud's demos
  - SonarQube
  - ESLint
  - ? Travis CI (partial)
- Many Jenkins tutorials on the web

Agile Testing

PRODUCT DEVELOPMENT

Analyze

Specify

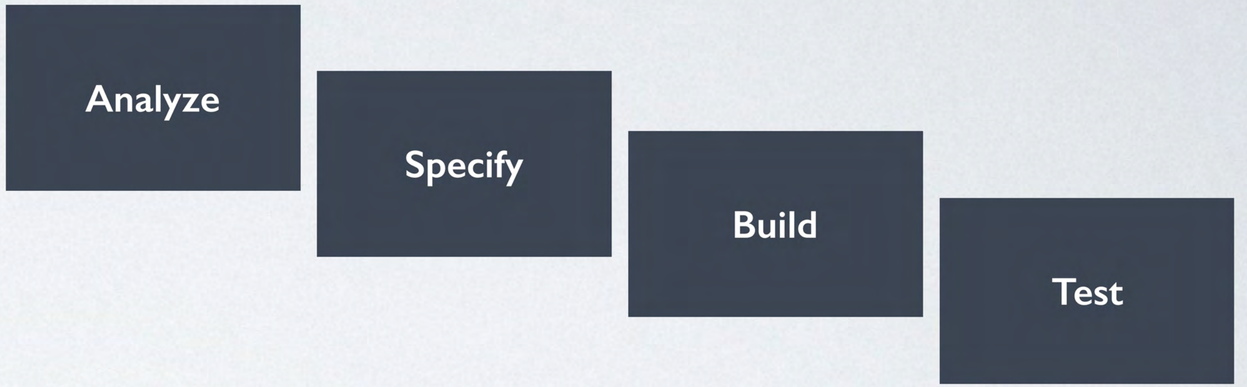
Build

Test

So... anything wrong with that?

PRODUCT DEVELOPMENT

- How much **time** do we need to go through all the phases?
- How often do we go through these phases (is it a **cycle**)?
- How much "**functionality**" is moving through these phases?



- **Who is working in each phase?**
- **How do people collaborate?**

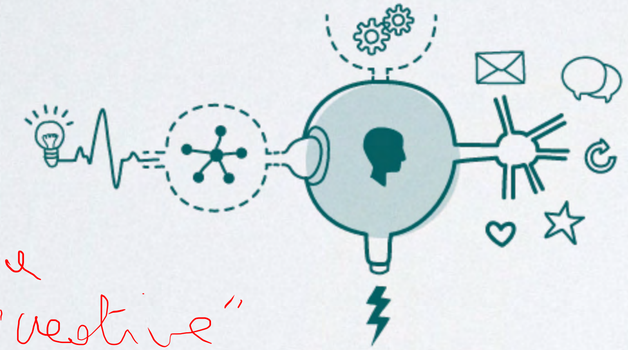
PRODUCT DEVELOPMENT

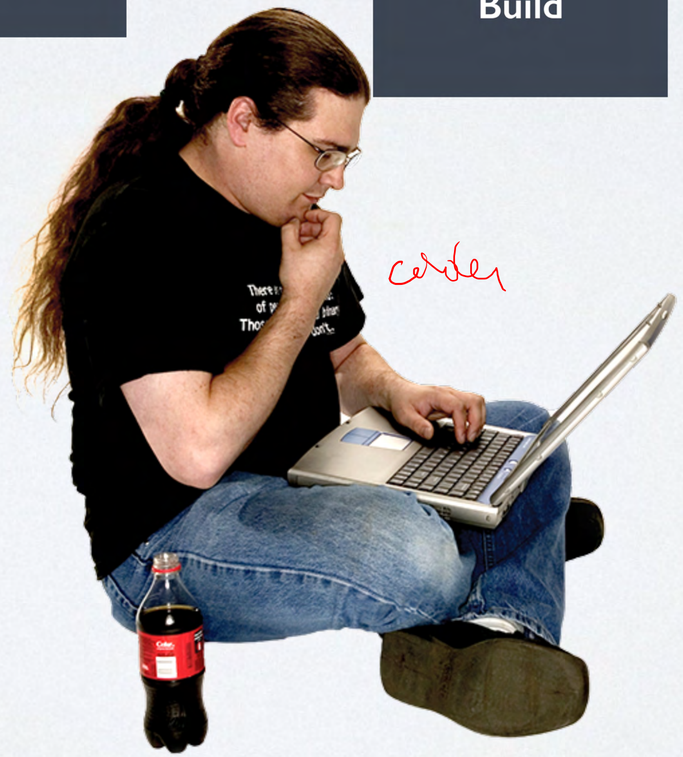
Analyze

Specify

Build

Test






AUSTIN
AUSTIN


Don’t look at me, it’s his responsibility.
Browman

New Approach

*"One of the biggest differences in agile development versus traditional development is the "**whole team" approach**.*

*With agile, it's not only the testers or a quality assurance team who feel responsible for quality. [...]. **Everyone on an agile team gets "test-infected"**. Tests, from the unit level on up, drive the coding, help the team learn how the application should work, and let us know when we're "done" with a task or story."*

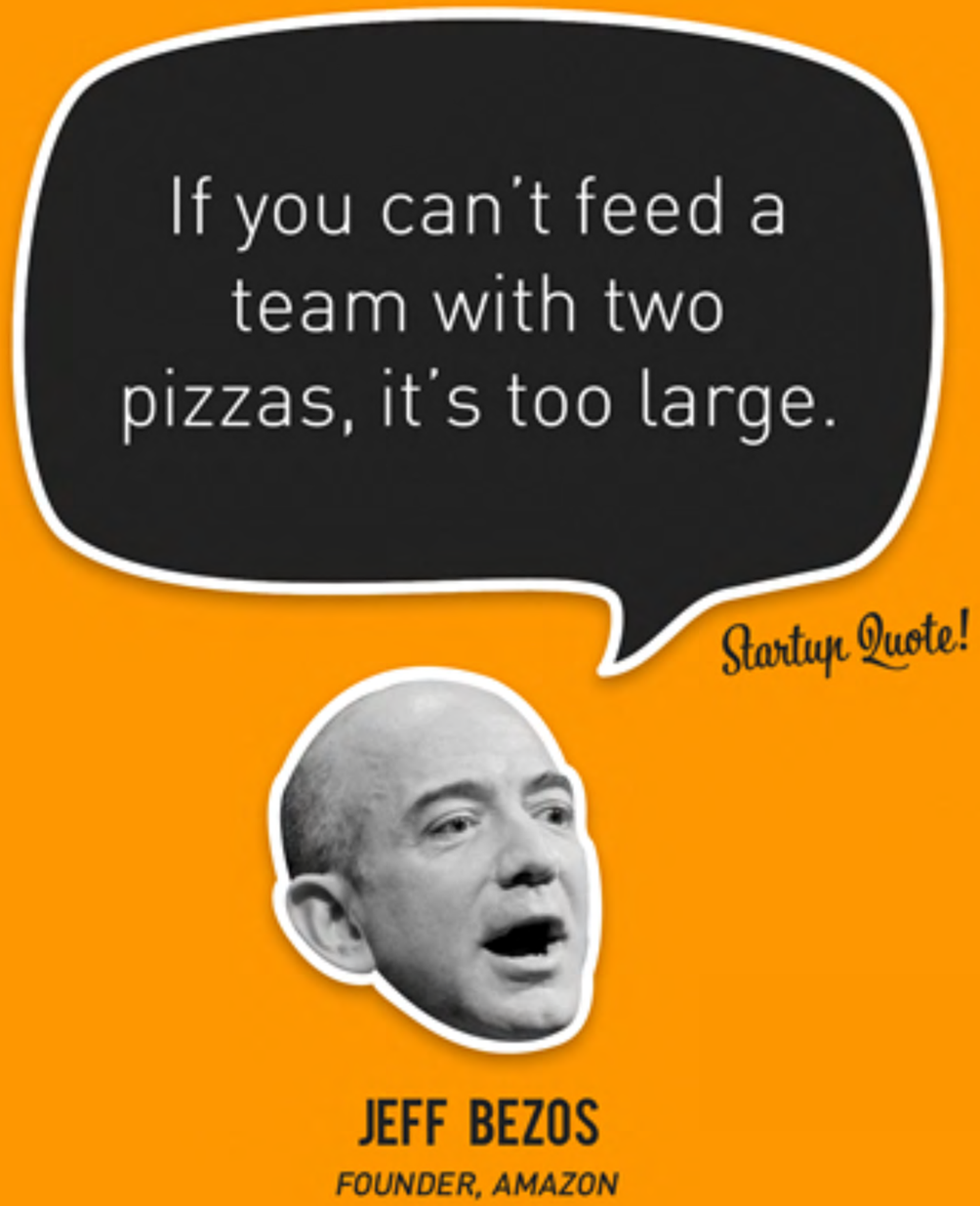
http://www.informit.com/articles/article.aspx?p=1316250&seqNum=5

Small autonomous teams, *which are given the responsibility of a complete "product" are very effective.*

Communication, collaboration and coordination are much easier than across ***organizational silos****.*

People are much more likely to feel ***empowered*** *and* ***accountable****.*

http://blog.jasoncrawford.org/two-pizza-teams



Agile Testing Quadrants

(inspired by a good (systematic) book

+ a nice video (see slide 30)

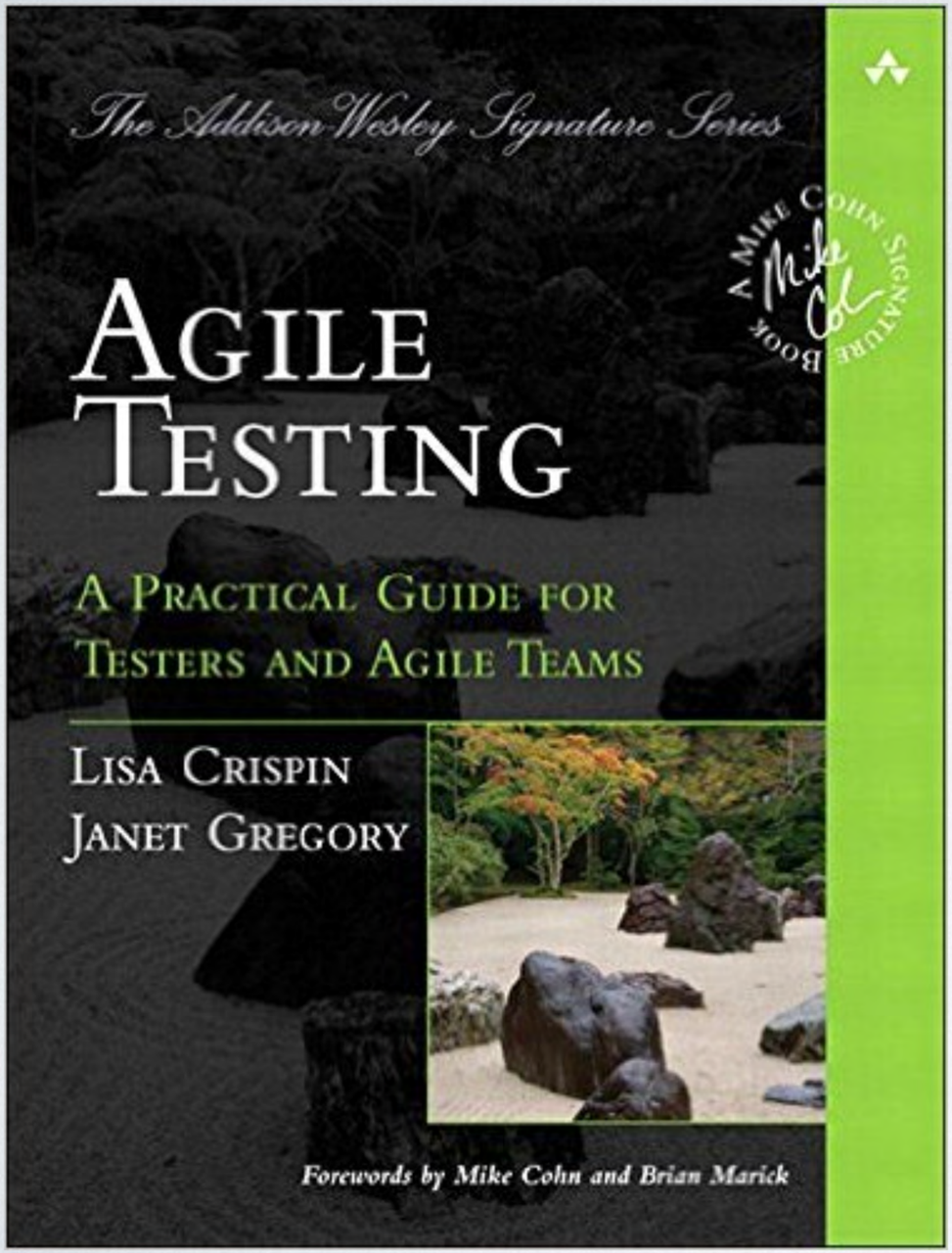
*"**Software quality**" is a **broad concept** and has many aspects (reliability, efficiency, usability, maintainability, etc.).*

*"**Software testing**" refers to methods and techniques for **assessing** certain aspects of the quality of a software system. **There are many, many of them**.*

*Some "**Software testing**" techniques do not only measure quality after the fact, but **help the team to proactively** maintain the quality of the software to an appropriate level.*

*Is there a way to **classify** all these methods, so that we can see how they relate to each other?*

*These techniques are aligned with the principles of **agile** software development.*



Elisabeth Hendrickson, Google Tech Talks

https://www.youtube.com/watch?v=bqrOnIECCSg

AGILE TESTING QUADRANTS

Are we building the right product?

Business facing

Support the team

Critique product

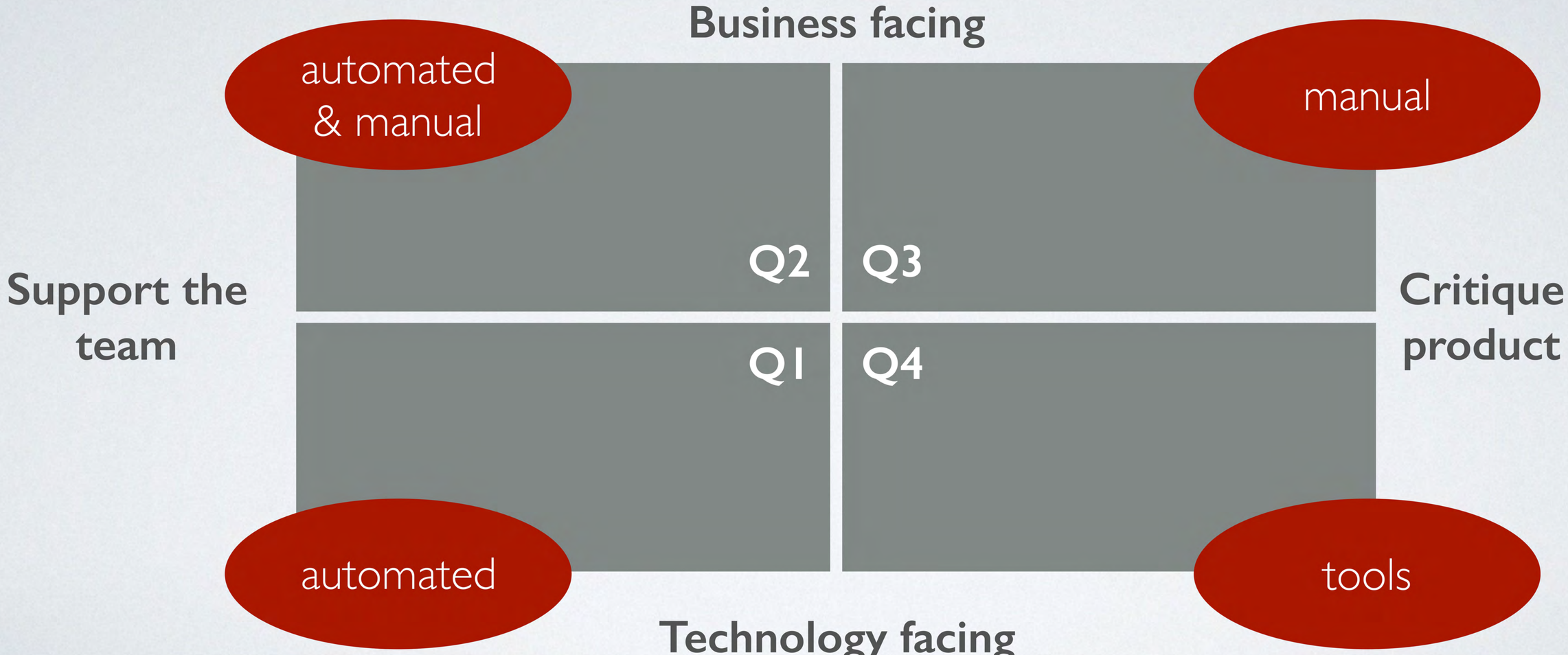
↳ users point of view

Technology facing

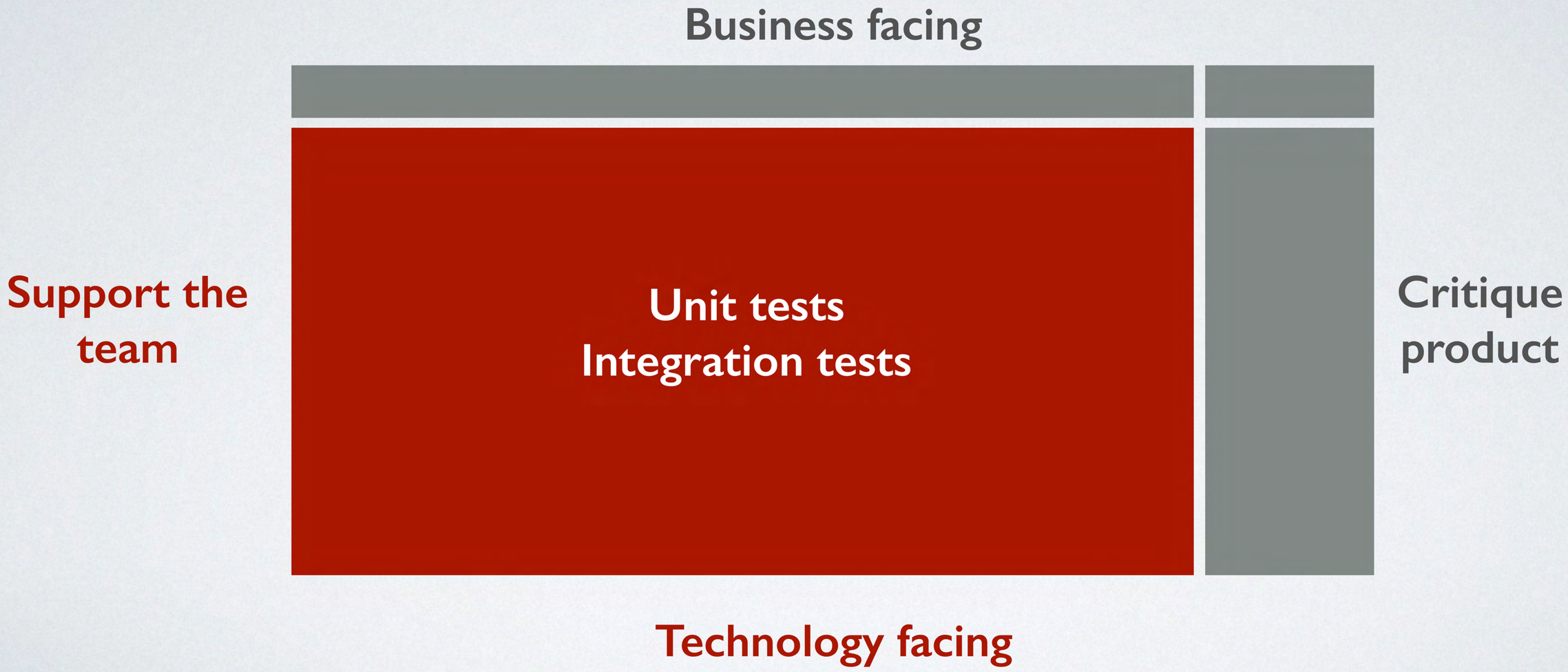
Are we building the product right?

Support the team	Some of these tests **help individual team members** while they do their job. Sometimes, creating a "test" helps me **specify and/or design** the product. Other tests **facilitate team collaboration**, especially between "business" and "technical" people (shared language).
Critique product	Some of these tests allow humans to evaluate the quality of a software from the **users point of view** (is it easy to use? is it easy to learn? does it solve the user's problem?). Other tests aim to detect issues with **non-functional (systemic) qualities**.
Technology facing	Some tests are created and executed by **technical team members**. They are highly automated. They relate to the "Are we building the product right?" question.
Business facing	Some tests are created by (or at least with) **business-oriented team members**. They also relate to the "Are we building the right product?" question.

AGILE TESTING QUADRANTS



AGILE TESTING QUADRANTS: Q1



UNIT TESTS

- Programmers use a **programming language** to write code in "source files". These "source files" are later translated in a language that can be understood by a machine.

- In a particular software system, there might hundreds or thousands of source files. **Every source file is made of "blocs"**. Depending on the programming language, we speak of "classes", "functions", "methods", etc.

- A **Unit Test** is a software function that validates one aspect of one specific "bloc" in the tested system.

Unit

```
public int add(int a, int b) {
  return a + b;
}
```

Unit

```
public int add(int a, int b) {
  return a + b;
}
```

Unit Test

```
@Test
public void testAdd() {
  int result = add(5, 3);
  assertEquals(8, result);
}
```

*When you write a unit test, you **invoke the unit** (call the function) and make a number of **assertions** on the result. By doing that, you **compare your expectations with the actual behavior** of the tested unit.*

Unit

```
public int add(int a, int b) {
  return a + b;
}
```

```
@Test
public void testAdd() {
  int result = add(5, 3);
  assertEquals(8, result);
}
```

Unit Test #1

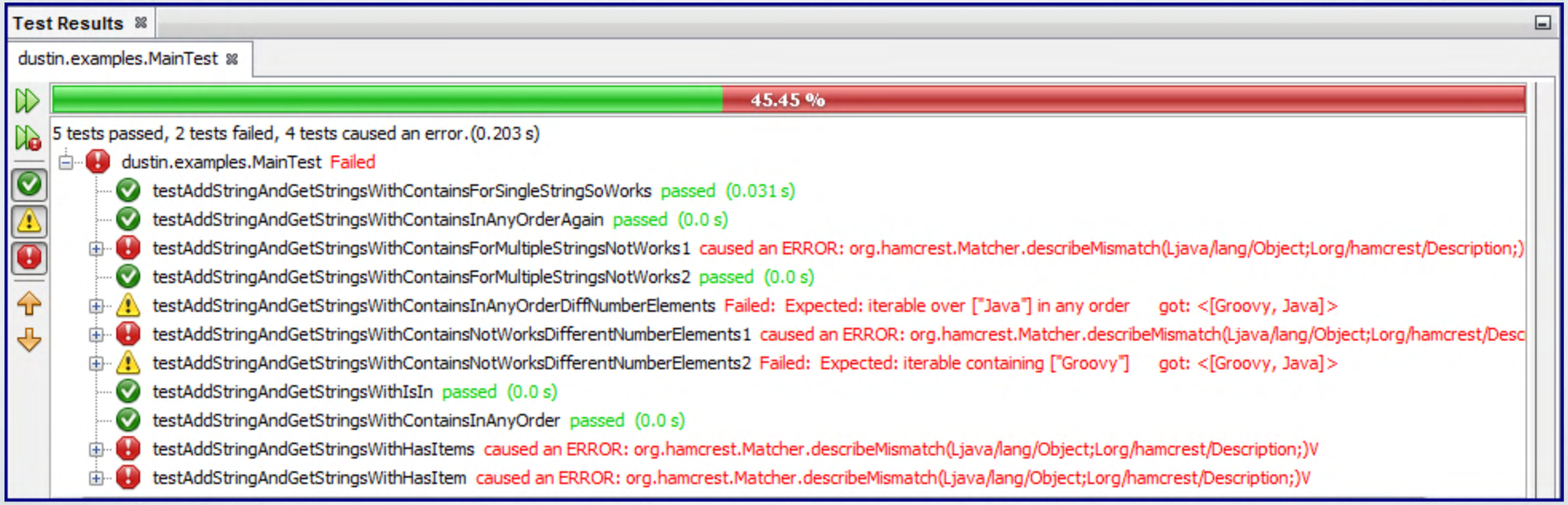
```
@Test
public void testAdd_2() {
  int result = add(-2, 2);
  assertEquals(0, result);
}
```

Unit Test #2

*Developers use **Integrated Development Environment** (IDEs), such as IntelliJ, Netbeans or Eclipse to write their code.*

*In general, they use the same tool to **code the application** and to **code the unit tests** that validate the application components.*

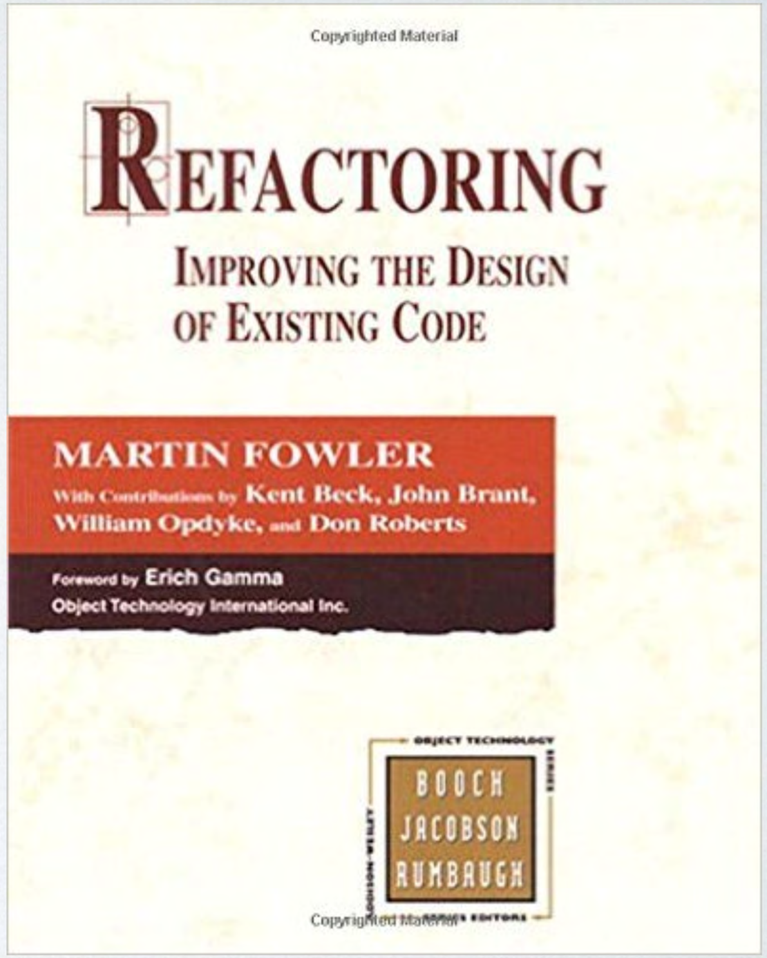
***Whenever** they modify their code, they can re-run the tests and get a "**green light / red light**" status in **milliseconds or seconds**. Even if there are thousands of unit tests.*



UNIT TESTS ENABLE REFACTORING

"Refactoring is a disciplined technique for ***restructuring an existing body of code****, altering its internal structure* ***without changing its external behavior****."*

http://www.refactoring.com/



*The **code works** (all tests are green) **but is a mess**. It is difficult to maintain and makes the development of new feature slow.*

- testArguments passed (3.592 s)
- testInputOutput passed (0.924 s)
- testWelcome passed (2.208 s)
- testQuote passed (8.346 s)
- testSubProjects passed (3.608 s)
- testPi passed (0.598 s)
- testFreeway passed (0.012 s)
- testFractal passed (15.9 s)
- testLexYacc passed (1.363 s)
- testMP passed (4.338 s)
- testHello passed (0.012 s)
- testHelloQtWorld passed (0.009 s)
- testProfilingDemo passed (0.025 s)

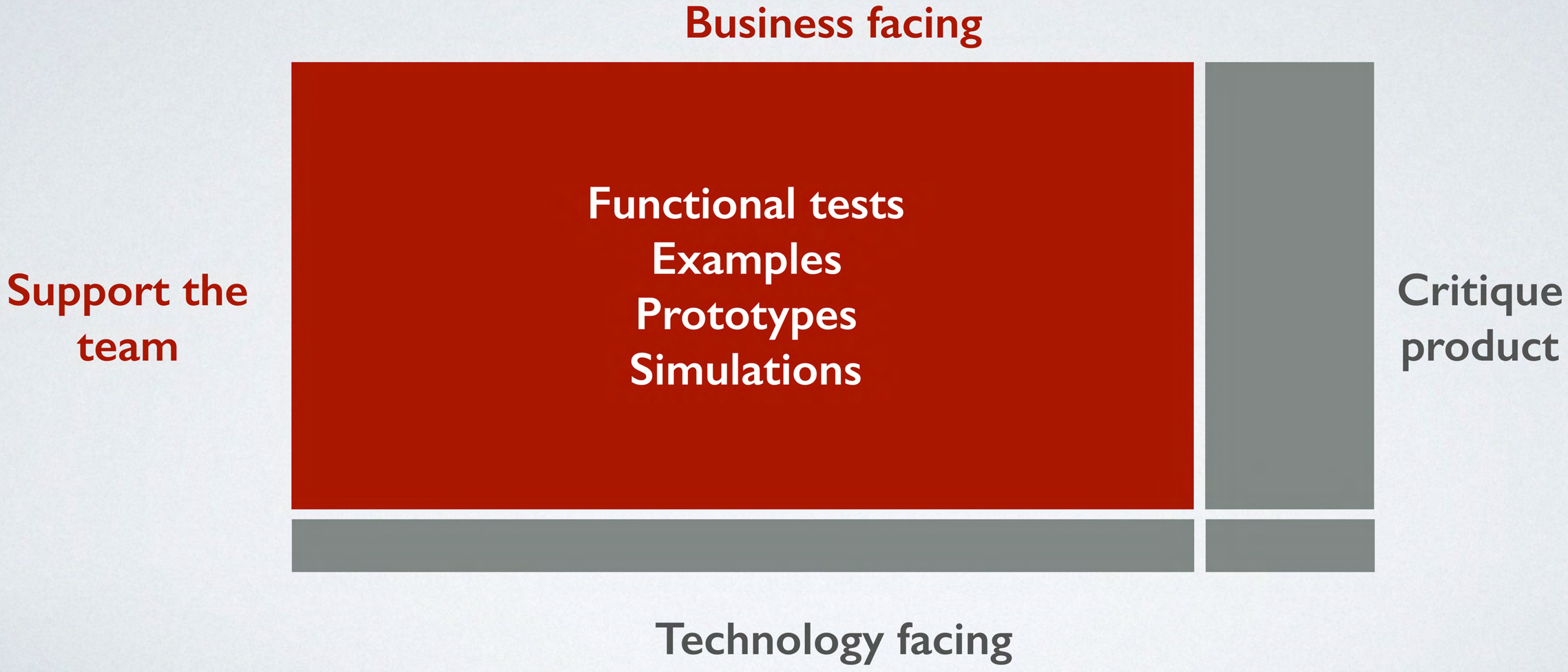
*The code is now **nicely structured**. It is meant to do the same thing as before (no behavior change). Because all my unit tests are still green, I have some confidence that this is true.*

- testArguments passed (3.592 s)
- testInputOutput passed (0.924 s)
- testWelcome passed (2.208 s)
- testQuote passed (8.346 s)
- testSubProjects passed (3.608 s)
- testPi passed (0.598 s)
- testFreeway passed (0.012 s)
- testFractal passed (15.9 s)
- testLexYacc passed (1.363 s)
- testMP passed (4.338 s)
- testHello passed (0.012 s)
- testHelloQtWorld passed (0.009 s)
- testProfilingDemo passed (0.025 s)

INTEGRATION TESTS

- Integration Tests are another type of **automated tests**.
- They validate the **interactions between several units**.
  - Does my "prediction algorithm" work with my "database service"?
  - Does my "web front-end" work with my "REST API"?
- Integration tests are **slower than unit tests**. We do not run them as often as unit tests (not to slow developers down), but we run them very often (at least before sharing one's code with the team).

AGILE TESTING QUADRANTS: Q2



FUNCTIONAL TESTS

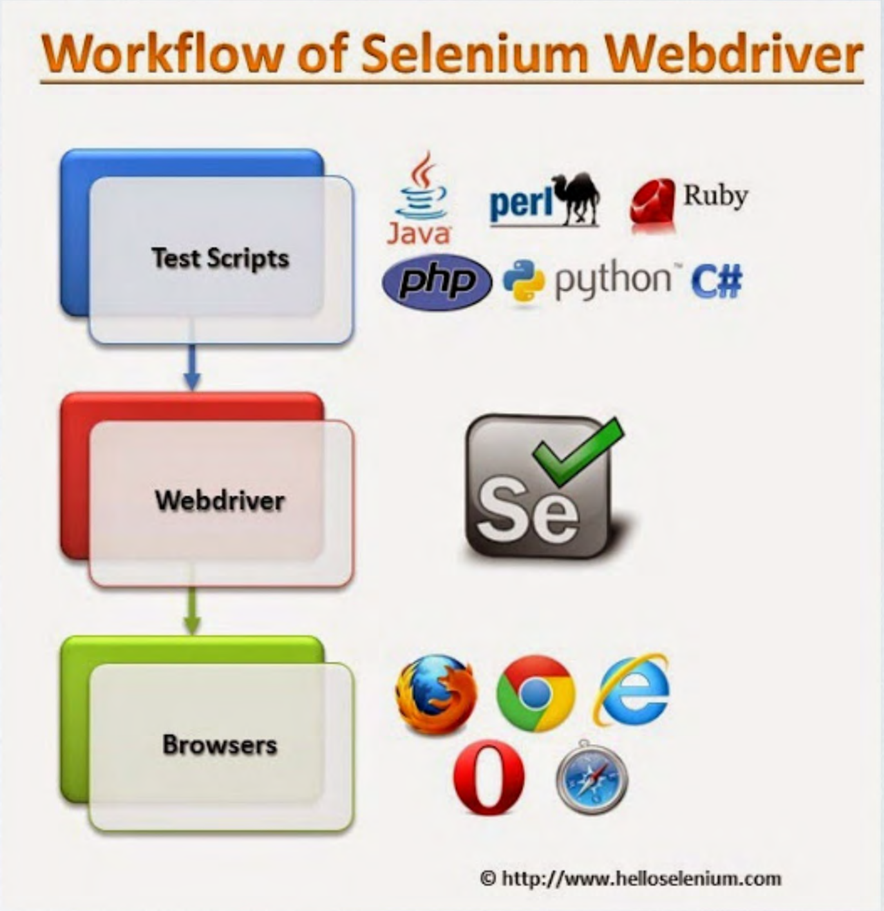
- With **functional tests**, we want to validate that the system does what it it supposed to do **from the users point of view**.
- Very often, this means **defining usage scenarios** (**test cases**). We describe the steps to be followed by users and the expected results.
- When we evaluate a software release, we can **check** whether the defined test cases can be executed with success.

MANUAL FUNCTIONAL TESTS

- In many organizations, test cases are documented in **test management software**. They are executed by **human operators**.
- This is a **repetitive process** with little added value.
- This is a **slow process**.
- It creates **overhead** and often gives a **false sense of confidence**.
- If you release every 3 months, it "might" work. If you release on a weekly basis, it is just not possible.

AUTOMATED FUNCTIONAL TESTS

- There are now **tools** that can be used to **simulate human users**.
- With these tools, you write scripts. When the scripts are executed, they **control a web browser** and check that the content of the pages is.
- **It is not a free lunch**. Writing these scripts takes time. Maintaining these scripts (when the UI changes) takes a lot of time.
- Integration tests are slower than unit tests. Automated functional tests are **a lot slower** than integration tests.
- For this reason, they are not executed as often (at a later stage in the continuous delivery pipeline).



BEHAVIOR DRIVEN DEVELOPMENT

- With Unit Tests, developers have a way to **specify and check** the behavior of a tiny piece of code.

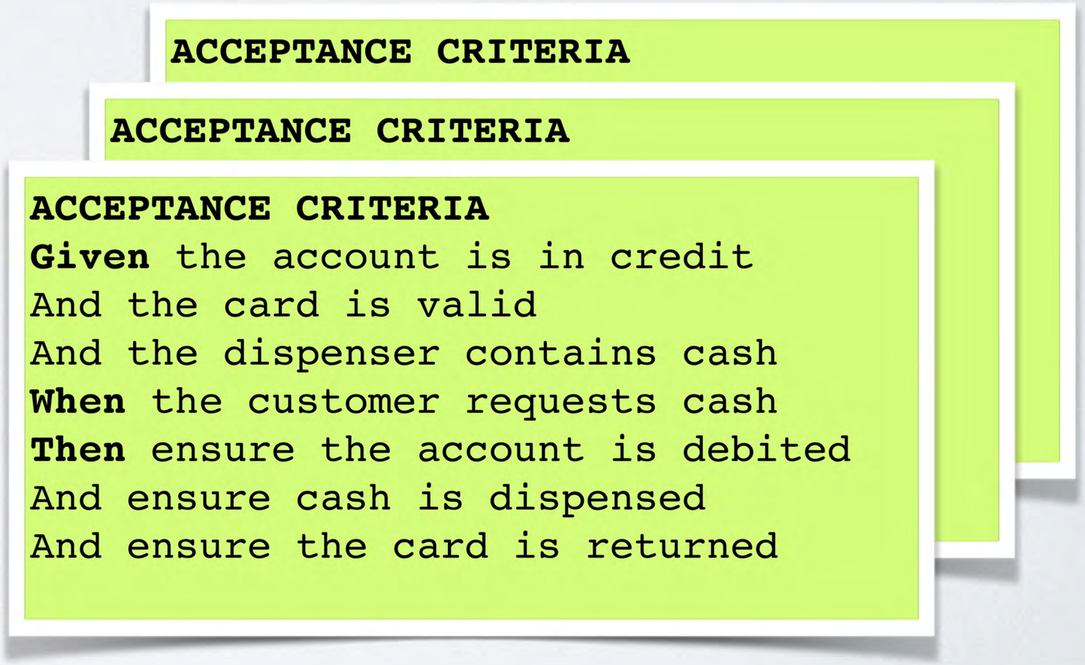
- The same principle can be applied with higher-level, **business oriented tests**. This is the idea of "behavior driven development" or BDD.

- BDD is a method that **facilitates the collaboration** between business analysis, developers and testers. It gives them a **common vocabulary**.

- BDD proposes a **template** to describe the intended behavior of a system. The template is used to specify the acceptance criteria for a given user story.

```
Given some initial context (the givens),
When an event occurs,
then ensure some outcomes.
```

```
USER STORY
As a customer,
I want to withdraw cash from an ATM,
so that I don't have to wait in line at
the bank.
```



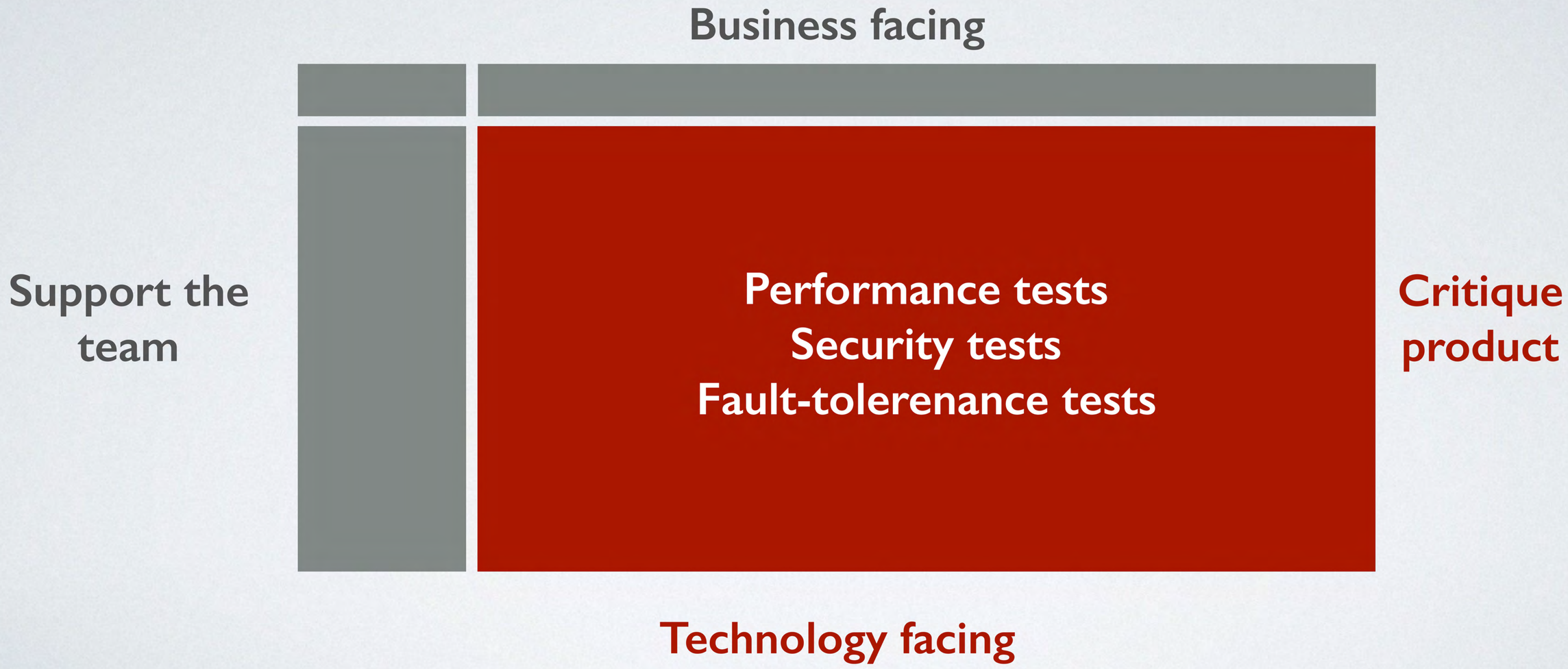
- The specification is **human readable**. It can be written, or at least read, by business analysis and product owners.
- But because we use certain keywords and conventions, the specification can also be **interpreted by machines**.
- We can therefore "**execute the specification**" and trigger automated tests for all stories and acceptance criteria.

```
Scenario:  trader is not alerted
below threshold

Given a stock of symbol STK1 and a
threshold of 10.0
When the stock is traded at 5.0
Then the alert status should be OFF
```

Executable Specifications

AGILE TESTING QUADRANTS: Q4



SYSTEMIC QUALITIES

- A common problem is that developers focus on the functionality (**what** the software does), but forget about the non-functional aspects (**how** it does it).
- Things like performance, availability, security, scalability, etc.
- We often talk about "**systemic qualities**" or "-ilities"

A SAD STORY...

Developer	I have **modified** the "search feature", so that it now ranks the results based on the user profile (e.g. preferences, past history, etc.)
Product Owner	Cool, let's **release** it!
User	Why is it search feature so **slow** today?
System Administrator	Why is all the **CPU** and **memory** in red on the server?

*An algorithm may work well if it is applied on a database that contains **1'000** entries. But it may break if it contains **1'000'000** entries.*

*A feature may be fast if it is used by a single user at the time. But it may extremely slow if is used by 10 **concurrent users**. It may even return wrong results in this case.*

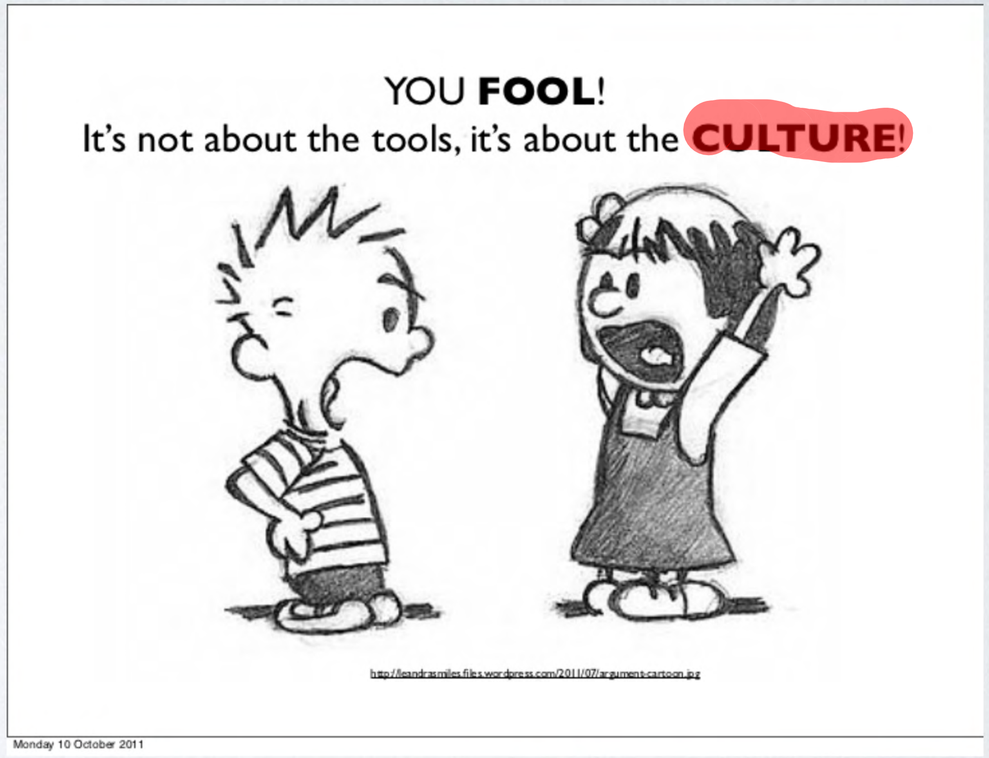
*A web app may appear to be fast when the browser, the server and the database are all on the same (developer) machine. But may appear very slow when the components are **distributed across the Internet**.*

IT WORKS

on my machine

HOW TO AVOID THESE ISSUES (1)

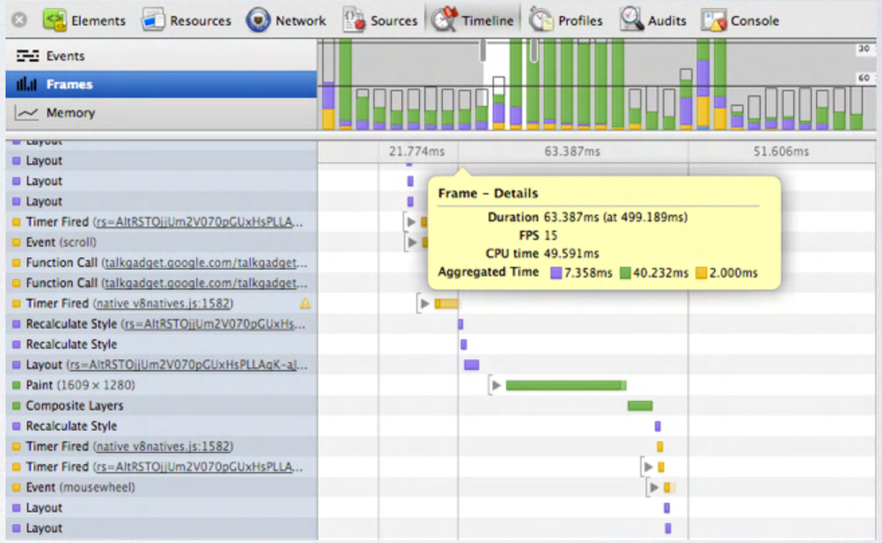
- Increase **team awareness**
  - Developers and product owners should be reminded about "**systemic qualities**" and the importance of **non-functional aspects**.
  - With a "**whole-team approach**", you are more likely to hear the point of view of system administrators very early. This will help a lot.

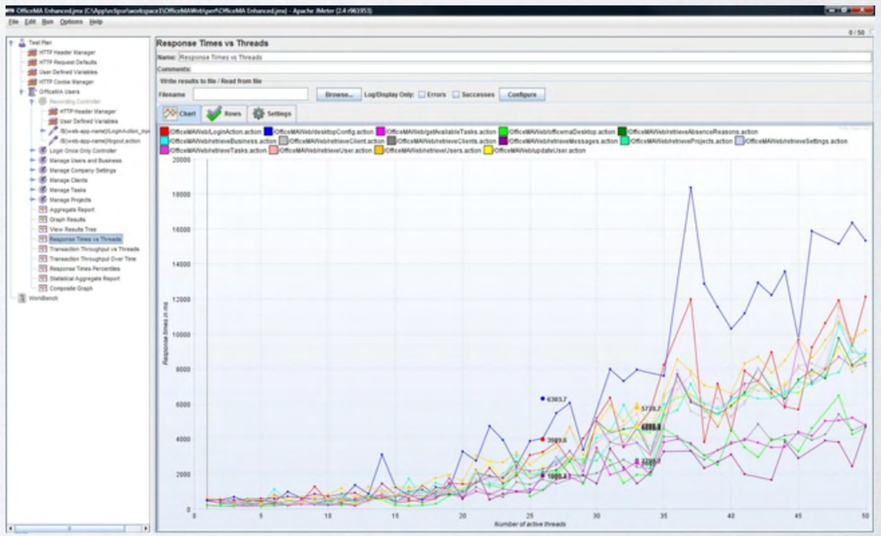


http://codeignition.co/blog/2015/09/17/dev_to_ops/

HOW TO AVOID THESE ISSUES (2)

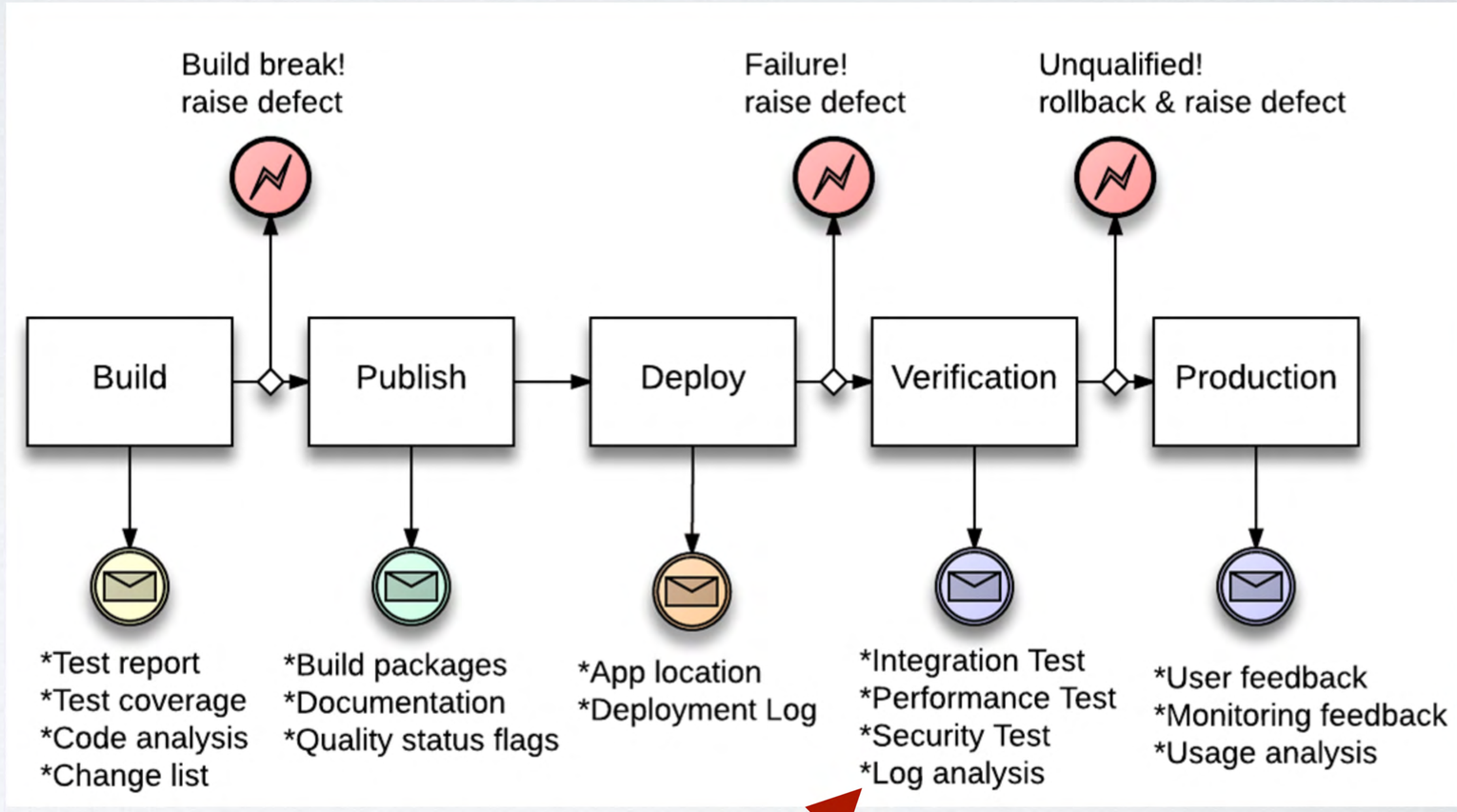
- **Measure** early, measure often
  - There are **many tools to measure performance** (response time) of a software system. There are also **many tools that simulate a large number of users** and to generate a heavy load on a test system.
  - Engineers can also **write custom programs and scripts** to do these experiments.
  - **Allocate time and resources** to support these activities.





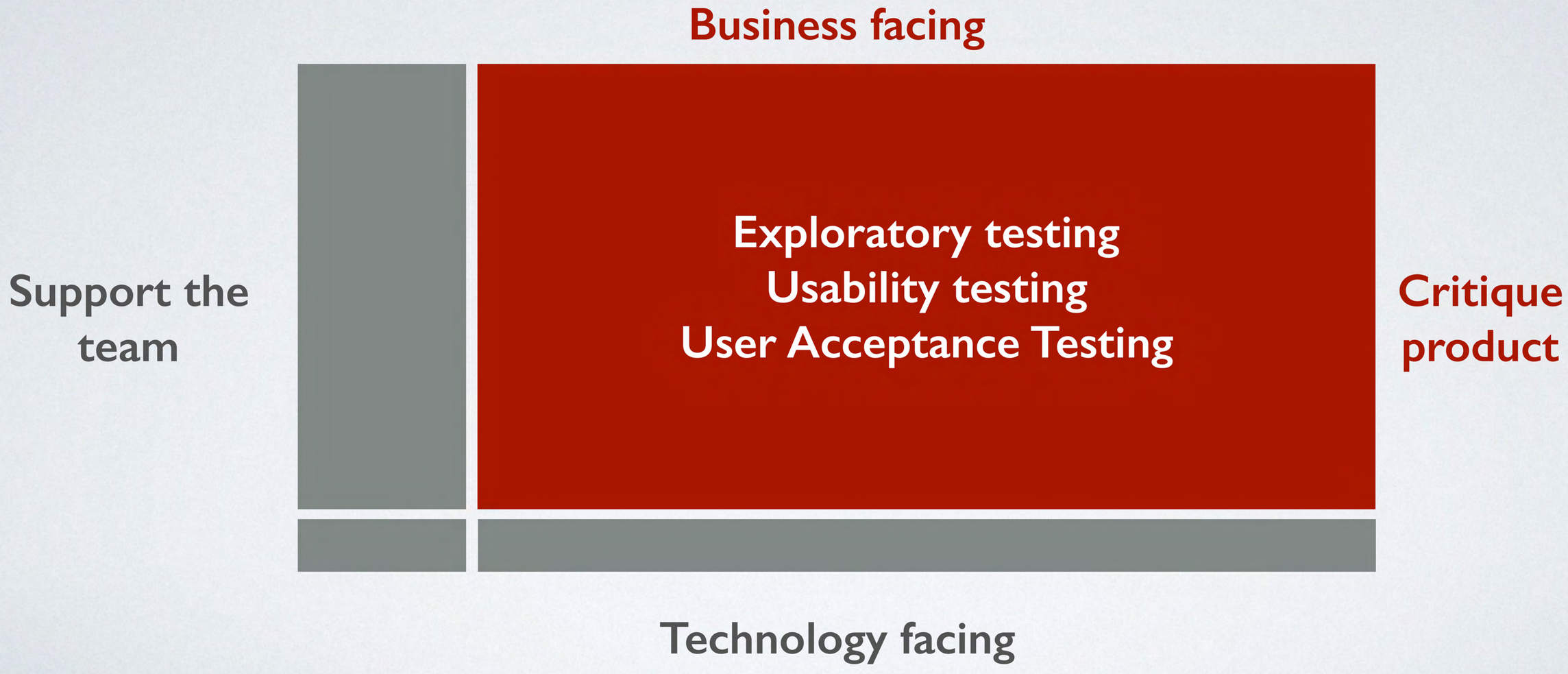
HOW TO AVOID THESE ISSUES (3)

- Integrate automated performance tests in your **continuous delivery pipeline**.
- Performance is only one aspect, you should also think about security, fault tolerance, etc.



https://www.ibm.com/developerworks/community/blogs/c914709e-8097-4537-92ef-8982fc416138/entry/devops_in_practice_best_practices_for_adopting_continuous_delivery?lang=en

AGILE TESTING QUADRANTS: Q3



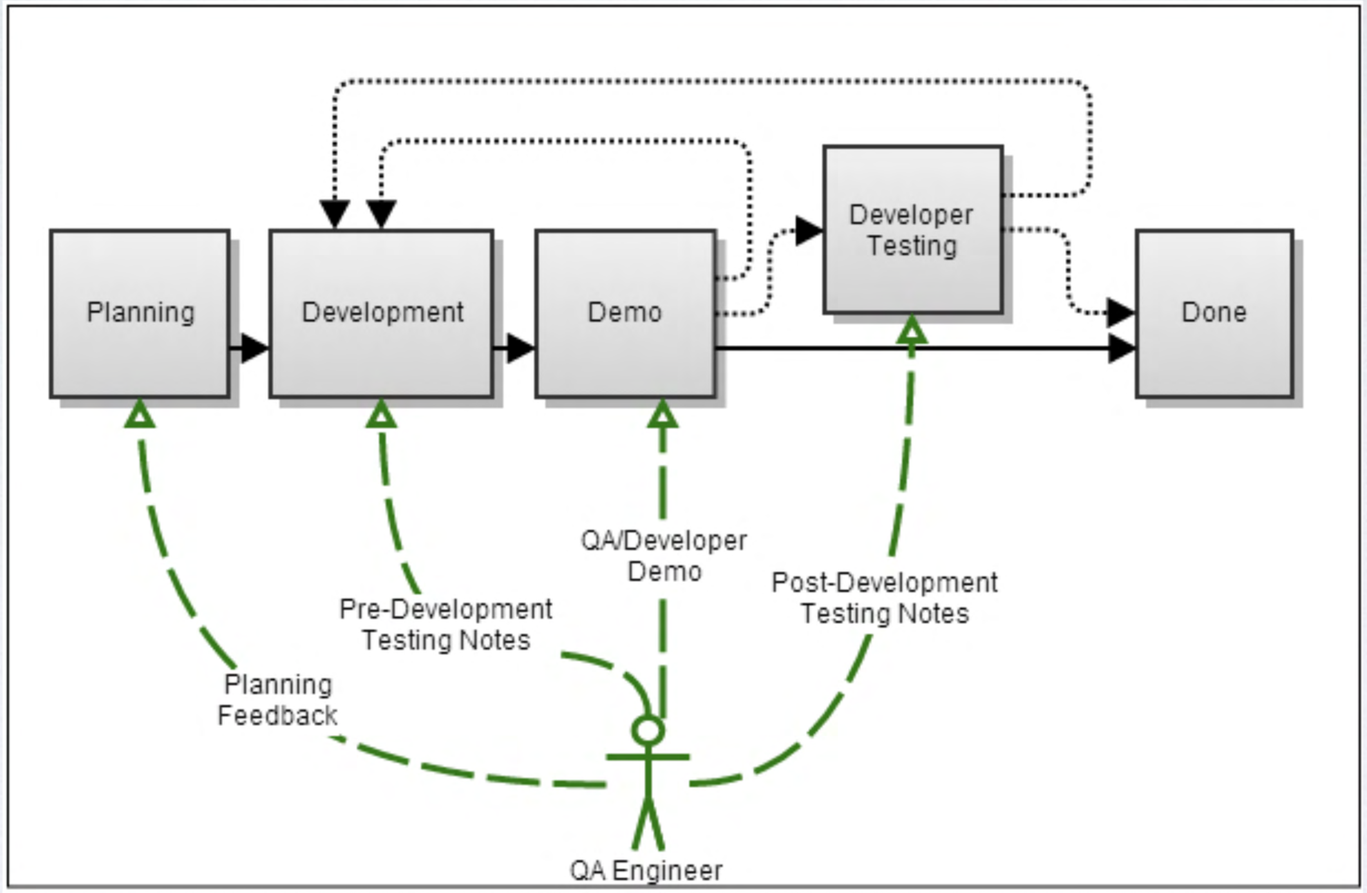
EXPLORATORY TESTING

- Exploratory testing is a **human activity** (which may involve the use of automated tests).
- Unlike scripted manual testing, it requires **strong skills and expertise**.
- It fits well with the "**whole-team**" approach (handing over the system to an external testing team is not efficient).

*"A style of software testing that emphasizes the **personal freedom and responsibility** of the individual tester to continually optimize the quality of his/her work by treating **test-related learning**, **test design**, **test execution**, and **test result interpretation** as **mutually supportive activities** that run in parallel throughout the project."*

Cem Kaner

JIRA DEVELOPMENT PROCESS



http://blogs.atlassian.com/2013/12/jira-qa-process/

JIRA DEVELOPMENT PROCESS

"Once a developer is happy that they have completed a story, he or she will call over their QA engineer for a ***demo session****.*

The demo is ***a discussion between equals****, and not a test that the developer or story can "pass" or "fail." However, sometimes concerns are noticed during the demo, and these get quickly noted as comments on the story.*

At the end of the demo, the QA engineer and developer will make a ***joint decision*** *on what should happen to the story next."*

JIRA DEVELOPMENT PROCESS

"The Atlassian QA strategy is more about prevention and trust.

As QA, we know the roads really well – we know what causes accidents, and where the accidents have been in the past. ***So instead of sitting by the road with a radar gun to catch speeding drivers, we sit in the cab with the drivers and say, "careful in this next bit, there's a huge rock in the middle of the road after that blind corner."*** *In safer stretches, we say, "There've been no nasty accidents in this stretch in the past, full speed ahead!""*

http://blogs.atlassian.com/2014/01/atlassian-qa-makes-development-faster/