Agile Test Project (30 study points)

Overview of project structure and vision

You must work as agile consultants in teams of 3-4 students on a project that has its "roots" in the database course. Until you have received the full project description from the database course, you can regard the Twitter Data project from the database course as your mission.

The Twitter Data project will serve the initial purpose of team activities such setting up a test strategy, deciding on coding standards, identifying suitable test tools, deciding on issue tracking system etc.

The overall functionalities of the system are described in the database project. The non-functional product requirements that relate specifically to the Test course are described in a separate section below.

In the Test course, the project has focus on high quality (working software) with low technical debt (good internal code quality that is testable and maintainable). Therefore, it is not only encouraged, but also expected that many resources be put into testing activities ☺ If it means a refactoring iteration every once in a while[1] that only focuses on adding tests, upgrading tools or reducing technical debt, that is perfectly agreeable with the business and test side of the project.

> *What do testers do? They provide information about the product under test, to expose risks for the team.*

The project must be run in an agile style and make use of the agile shift left testing paradigm with ATTD, TDD etc. The section of Agile Principles further down describes a number of principles you must follow.

It is the *future* goal of the company that "hires" you to have a continuous integration chain in Q3 2018 (grey section of figure 1) and a continuous delivery pipeline by the end of 2018 (orange section in figure 1).

If the team has the skills already, it is encouraged to practice Continuous Integration, i.e. set up a server to build and test any change submitted to a central repository, preferably several times a day[2].

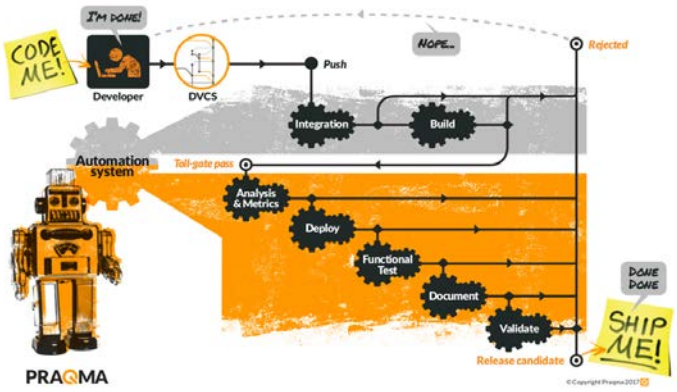


Figure 1Praqma: A Continuous Delivery storyline

[1] Probably every sixth month in a real project

[2] 1st and 2nd semester students have different prerequisites here and expectations are therefore not the same. 2nd semester students have knowledge of CI chain from Large Systems Development Fall 201f which may be included in this project, but only as secondary learning objective.

Nonfunctional requirements

There must be a graphical UI frontend that is not just an API, preferably a Web-based UI (since that has been our focus in class).

There are no constraints on programming language and testing tools as long as you do not go for something too exotic that makes hiring of future developers for the project complicated.

Agile Principles

An agile tester is a tester who embraces change, collaborates well with both technical and business people, and understands how tests can be used to both document requirements and drive development. Agile testers (or maybe any tester with the right skills and mindset) are continually looking for ways the team can do a better job of producing high-quality software.

In the project, your agile mind-set can be guided be ten principles for agile testers made by Lisa Crispin and Janet Gregrory[3]. The principles should be familiar to you as they are based on the XP values of feedback, communication, courage, and simplicity and have their roots in the agile manifesto:

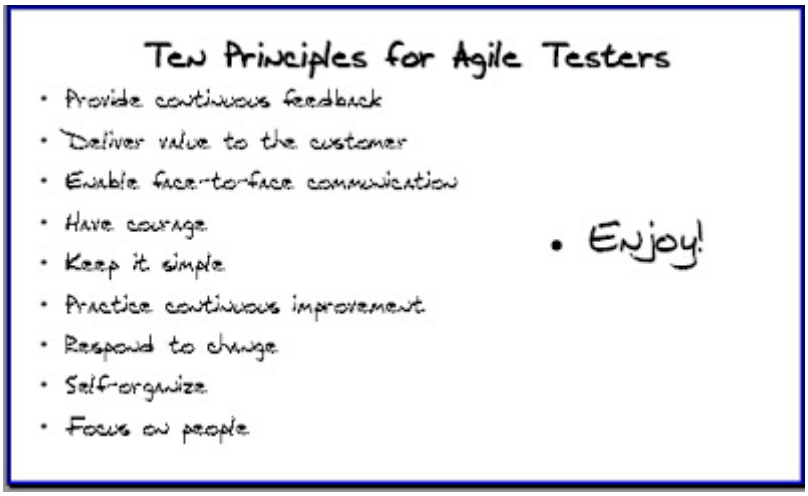


Figure 2http://agileinaflash.blogspot.dk/2009/03/ten-principles-for-agile-testers.html

Team logistics

Building a team

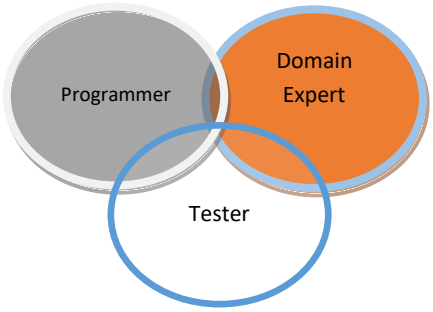
You build your team yourself (as long as we are able to find a team for all students in the class). You may work in distributed groups if you want to, i.e. across the Danish and International classroom borders☺

[3] Agile Testing by Lisa Crispin and Janet Gregrory is a practical guide for testers and agile teams - https://agiletester.ca/

Team structure

Agile teams are by definition cross-functional where you are meant to do pair programming, test-driven development and other agile practices with focus on high quality software. In reality, many teams have more specialized roles where some team members focus mostly on test activities and never do any programming, except from maybe assisting in designing automated tests.

In the present project, you must work as modern developers/testers with a holistic view to the whole-team effort, but with special focus on the Tester role:



Roles

Based on the above, you should act as both tester and programmer.

The teachers will by nature be the product owners of each our part of the project, i.e. the learning objectives for each exam respectively.

You must choose a "proxy" product owner for your team.

You decide yourself how you otherwise organize your team with roles such as architect, programmer, business analyst etc. You are all supposed to be testers.

Project Planning

There will be the following activities during the project period (i.e. the rest of the semester):

30.4 Project review 1 (focus on writing acceptance criteria/tests)

7.5 Lecture on BDD, Exploratory testing and non-functional testing

14.5 Project review 2

22.5 Project review 3

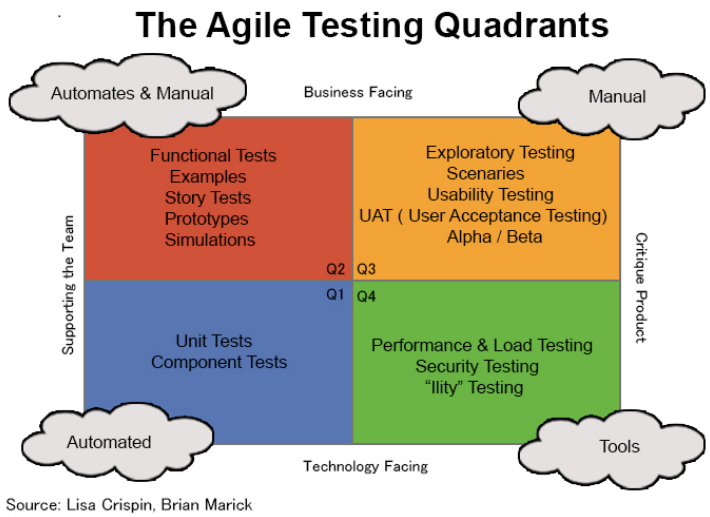
Test Strategy

You must come up with a test strategy for your project. I.e. some principles that can guide the project all along.

As a minimum, you must consider the following three elements: the Agile Testing Quadrants, the Test Pyramid, and a number rules as specified below.

Agile Testing Quadrants

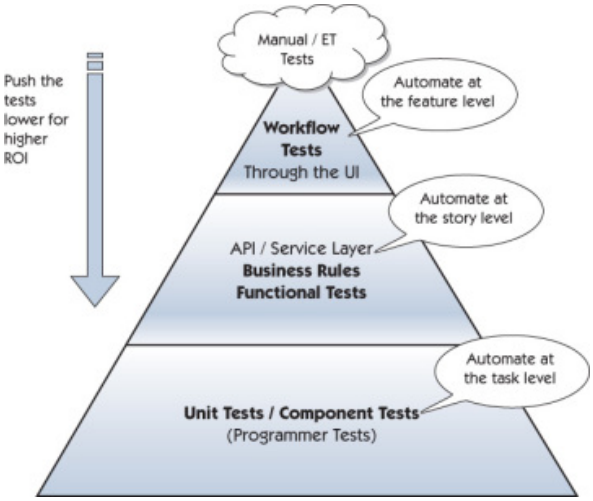
Make use of the Agile Testing Quadrants to make sure your team covers all the needed categories of testing:



Source: Lisa Crispin, Brian Marick

Test Pyramid

Make use of the Test Pyramid to get cheaper tests that run faster in order to get the best return of investment (ROI):



Simple Rules

The following principle of good code design apply to test code as well as production code. So pay attention to the design of your test code and follow these rules:

Rule	**Reason**
Single purpose	Easier to debug; easier to change if business rule changes
DRY	Ability to change tests in one place only
Abstract code out of the tests	Makes the tests business readable
Set up and tear down tests	Can run the tests repeatable
Independence	Tests can run independently
Avoid database access	Database calls slow down the tests
Tests must run green – all the time	Confidence in the tests; living documentation
Apply common test standards	Enables shared ownership and common understanding

Iteration metrics

In order to measure progress, a project needs a way to know how much work the team has completed at any point in the iteration and an idea of how much work is left to do. You need to know when it becomes obvious that some stories can't be completed and the team needs a plan B.

Burndown charts and estimated versus actual time for tasks are examples used to measure team progress. Story boards are good to visualize progress – or lack of it.

It is up to your team how you want to measure progress, but it must be presentable to each review meeting and documented in your final report.

In the end of each iteration, you must collect a number of metrics. Again, the metrics must be presentable to each review meeting and documented in your final report. The metrics are:

- Sprint deliverables are refactored and coded to standards
  - Use static analysis tool
- Sprint deliverables are unit tested
  - Look at the code coverage results each sprint
  - Count the number of packages with unit tests coverage falling into ranges of 0-30 % (low coverage, e.g. for legacy code), 31-55% (average coverage), 56-100% (high coverage, e.g. with TDD). An increase in the high coverage range is desirable.
- Sprint deliverables have passing automated acceptance tests
  - Map automated acceptance tests to requirements in system. Generate at the end of an iteration to show that all requirements selected as goals for the iteration have passing tests. Requirements that do not show passing test coverage are not done.
- Sprint deliverables are free of defects
  - Requirements completed during the iteration should be free of defects.
- Can the product ship in (5) days?
  - Proceed into the next iteration according to the answer.

Documentation, Rationale and Reflection

A main goal of this project is to reflect-and-adapt to enable you and your team members to find your way into testing in general, but agile testing in particular.

You must document how you have worked with:

- Requirements
  - User stories (the why)
  - Guide development with questions and examples[4]
- Architecture and design
  - Code metrics
  - Testable code
- Test
  - Unit tests, mocks and stubs
  - Automated Functional tests
- Organization
  - Team work
  - Agile process

Examples

Examples of elements you could include in your report:

- If you have chosen to use retrospectives or have a refactoring iteration during the project where you did not deliver any business value – why/rationale: tried reduced the technical debt of the project – did it work out well – if not, why not (reflection).
- If you are a Java team, you might have used Selenium for your UI tests, Cucumber for your API tests and JUnit for your unit tests. Why/rationale: they are standard tools that gives team autonomy, but allows consistency for the maintenance team and other teams working on the same product. Are there better alternatives you should have used instead (argumentation and reflection).

[4] Please consult the Test Planning Cheat/Chat sheet in appendix B that can serve as conversational starters in the beginning of a project and in the beginning of each sprint.

Hand-in

Date: May 25th at noon on Moodle.

The solution will be part of the examination. This means that it can be used at the exam for discussion. Also, exam questions can specifically refer to the assignment.

You must hand-in a URL to your project solution, i.e. the test code and production on your Github account. The documentation of your project work including the rationale and reflections can either be handed in as a pdf or a Markdown file on your Github account.

Appendix A

Assignment 2 - Analysis of Twitter Data

Your task is to implement a small database application, which imports a dataset of Twitter tweets from the CSV file into database.

Your application has to be able to answer queries corresponding to the following questions:

1. How many Twitter users are in the database?
2. Which Twitter users link the most to other Twitter users? (Provide the top ten.)
3. Who is are the most mentioned Twitter users? (Provide the top five.)
4. Who are the most active Twitter users (top ten)?
5. Who are the five most grumpy (most negative tweets) and the most happy (most positive tweets)? (Provide five users for each group)

Appendix B

Feature Chat Sheet: Conversation Starters

Value to users/business

- What's the purpose of the story?
- What problem will it solve for the user, for us as a business?
- How will we know the feature is successful once it is released? What can we measure, in what timeframe? Do we need any new analytics to get usage metrics in production?

Feature behavior

- What are the business rules?
- Get at least one happy path example for each rule, and ideally also one misbehavior example
- Who will use this feature? What persona, what particular job is a person doing?
- What will users do before using this feature? Afterward?
- What's the worst thing that could happen when someone uses it? (exposes risks)
- What is the best thing that can happen? (delighting the customer)
- Is the story / feature / epic too big? Can we deliver a thin slice and get feedback?
- Watch out for scope creep and gold-plating

Quality Attributes

- Could this affect performance? How will we test for that?
- Could this introduce security vulnerabilities? How will we test for that?
- Could the story introduce any accessibility issues?
- What quality attributes are important for this feature, for the context?

Risks

- Are there new API endpoints or server commands? Will they follow current patterns?
- Do we have all the expertise for this on our team? Should we get help from outside?
- Is this behind a feature flag? Will automated tests run with the feature flag on as well as off?
- Are mobile/web at risk of being affected?
- Are there impacts to other parts of the system?

Testability

- How will we test this?
- What automated tests will it have - unit, integration, smoke?
- Do we need a lot of exploratory testing?
- Should we write a high level exploratory testing charter for where to focus testing for this feature/epic?
- Do we have the right data to test this?
- Does it require updating existing tests, or adding new ones? If new ones, is there any learning curve for a new technology?

http://agiletester.ca © Lisa Crispin & Janet Gregory