Agile Test Project (30 study points)

Overview of project requirements

You must work as agile consultants in teams of 3-4 students[1] on a project that has its "roots" in the database course.

The overall requirements of the application are therefor found in database project description, but the parts that are of specific interest to the Test course are repeated in appendix A. The non-functional product requirements that relate specifically to the Test course are described in a separate section below.

From Test course perspective, 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 project resources be put into testing activities ☺ If it means a refactoring iteration every once in a while[2] that only focuses on adding tests, upgrading tools or reducing technical debt, that is perfectly agreeable with the business 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 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 in 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[3].

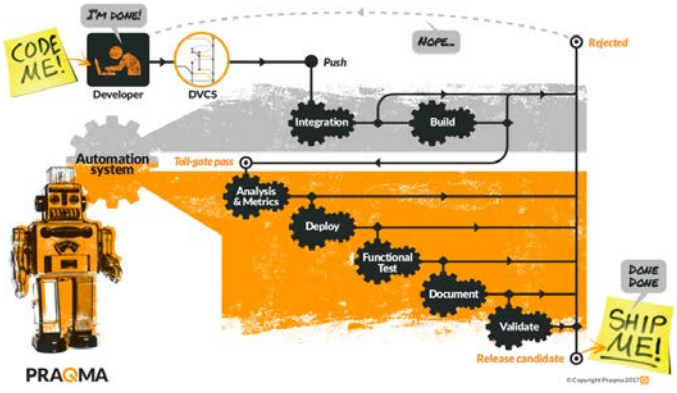


Figure 1Praqma: A Continuous Delivery storyline

[1] You are supposed to be groups of 4 in database project – and to database exam. Test exam is individual.

[2] Probably every sixth month in a real project

[3] 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 2017 which may be included in this project, but only as a secondary learning objective.

Nonfunctional requirements

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

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.

Queries must run "fast enough".

Agile Principles

The team is expected to work according to the agile manifesto and its 12 principles of agile software.

An agile tester 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 test mind-set may be guided by ten principles for agile testers made by Lisa Crispin and Janet Gregrory[4]. 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:

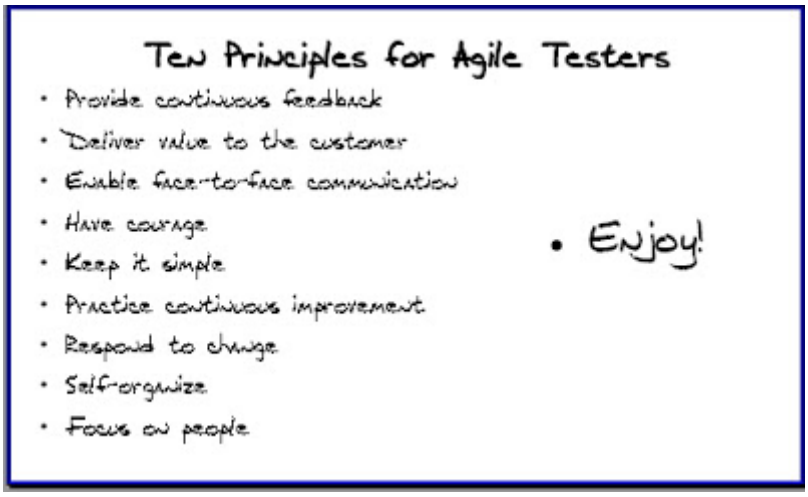


Figure 2http://agileinaflash.blogspot.dk/2009/03/ten-principles-for-agile-testers.html

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

Team logistics

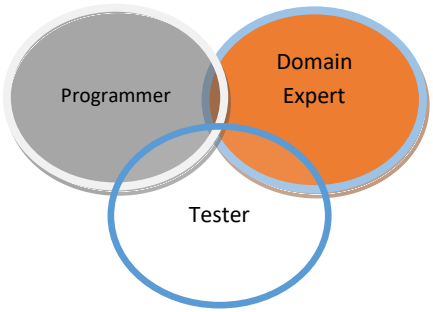
Building a team

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☺

Team structure

Agile teams are by definition cross-functional where each team member is meant to be equally fit to work with any type of activity. In reality, most team members only cover a subset of roles (e.g. being the tester).

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:



Based on the above, you should act as both tester and programmer. The teachers will be the product owners of each our part of the project, i.e. the learning objectives for Database and Test 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

Project startup involves team activities such setting up a test strategy, deciding on coding standards, identifying suitable test tools, experimenting and agreeing on style of writing for automated tests etc.

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

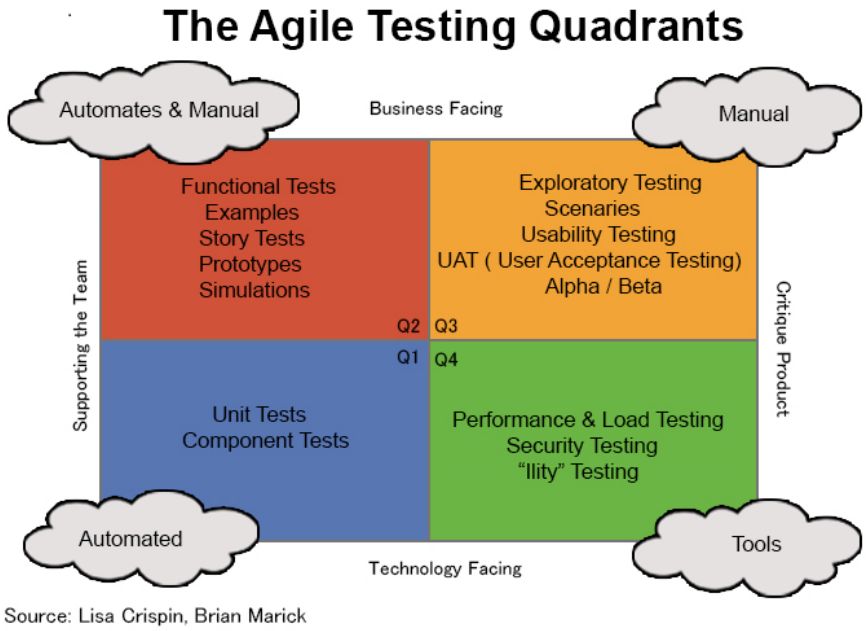
30.4 Project review 1 (focus on writing acceptance criteria/tests + test strategy)
7.5 Lecture on BDD, Exploratory testing, and non-functional testing
14.5 Project review 2
17.5 Project review 3 (OBS date changed as I am not available on the original review date, 22.5)
You also have database review this date, but we will coordinate the time schedule.

Test Strategy

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

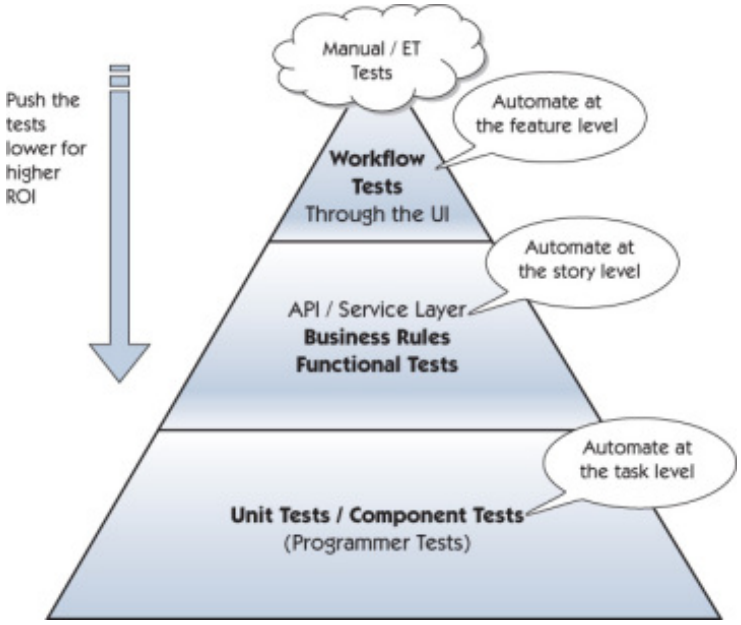
For inspiration in that work, you should consider the Agile Testing Quadrants and the Test Pyramid.

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



Source: Lisa Crispin, Brian Marick

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



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 some stories cannot 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. The metrics must be presentable to each review meeting and documented in your 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.

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

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 (including rationale and reflection):

- Requirements
  - User stories (the why)
  - Guide development with questions and examples[5]
- 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 related to the above bullets:

- 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 to 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 allow 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).

[5] 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.

Other evaluation criteria/question you can discuss in the report are (the list is for inspiration and not final):

- What design techniques did you use to design your test cases? Which do you find the most useful?
- How did you work with code reviews? Was it useful? Why /why not?
- How did you handle incident management (issue/bug tracking)
- Did do run exploratory workshops? Why /why not?
- What test level did you find the most useful? Why?
- What test level did you find most complicated? Why? How did you handle this complexity?
- What is the cyclomatic complexity of your code? Have you tried to keep it as low as possible?
- Do you consider your code testable? If yes, what techniques and rules have you followed to obtain this. If no, why do you think it is so, and how can the code design be improved?

Hand-in

Latest hand-in is on 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 and production code on your Github account. The documentation of your project work can either be handed in as a pdf or a Markdown file on your Github account.

The document must be 10 – 15 pages.

Appendix A

The Project

The task for the semester project in the database course is to build a small application that allows to figure out which cities are mentioned in which English books from Project Gutenberg, given a city which books mentioned it, and given a location, which books mention cities in vicinity.

Build an Application with Various Databases

The goal of the project is that you build such an application in a way, so that you can exchange the underlying database engine.

Types of End-user Queries

In essence, your application will support the following basic end-user queries:

1. Given a city name, your application returns all book titles with corresponding authors that mention this city.
2. Given a book title, your application plots all cities mentioned in this book onto a map.
3. Given an author name, your application lists all books written by that author and plots all cities mentioned in any of the books onto a map.
4. Given a geolocation, your application lists all books mentioning a city in vicinity of the given geolocation.

Consequently, your databases store author names, book titles, names of cities, their geolocations and their occurences in texts, and any other information that you deem important.

Measure Application Behavior

Subsequently, you define a set of end-user test queries with at least five queries per type, see section above. That is, in total your end-user query test set contains at least 20 queries.

Now, you measure the responsiveness of your application with respect to the end-user queries.

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