UNIVERSITEIT TWENTE.

Data & Information (201300180)

Project Manual 2014-2015

Djoerd Hiemstra
Luís Ferreira Pires

Version 0.8 – 21 April 2015

Preface

Most of the extremely successful companies of the last 15 years, such as Google, Facebook, Spotify and Twitter, have been started by people with an excellent idea, knowledge on web technology and a small but handy and motivated team of (agile) software developers. In the Data & Information module you will learn how to design and implement web applications for desktop computers and/or mobile phones, working in on a team of 5 people. We focus on requirements capturing, web and database technologies, semi-structured data and cooperation by means of agile software development techniques. By learning these techniques you will be able to unleash your ideas!

Enschede, April 2015
Klaas Sikkel (module coordinator), Luís Ferreira Pires, Maurice van Keulen

Contents

1 Introduction

In the Data & Information module you will learn and practise with the design, implementation and testing of complex layered software systems, by using de facto standard tools such as database management systems, CSS and scripting frameworks and web services. Data play an important role in this module, in terms of classical structured data stored in databases, semi-structured data and data retrieval from social media and other web resources.

We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following a plan.

That is, while there is value in the items on the right, we value the items on the left more.

Manifesto for Agile Software Development (http://agilemanifesto.org)

In this module, a big project is performed according to the agile development principles by using Scrum, which is a modern and currently popular way of organising a (software) project. This means that (i) you will get experience with handling stakeholders (more specifically the client who has ordered the system), (ii) the teams manage themselves (there is no single manager who tells you what to do), (iii) the teams are multidisciplinary, and (iv) every two weeks a new or improved product is delivered (teams work in short sprints of 2 weeks). Scrum is explained during the lectures, so that you will learn and be able to apply the following concepts: ***product backlog***, ***sprint***, ***sprint backlog***, ***user stories***, ***sprint planning meeting***, ***sprint review meeting***, ***sprint retrospective***, ***task board***, ***daily standup***, ***scrum master***, ***development team***, ***product owner*** and ***burn down chart***. More information on Scrum can be found in Section 4 of this manual and in the handout on Blackboard. IBM has a special page on Agile development at http://www-01.ibm.com/software/rational/agile/, where all sorts of resources can be found.

The project groups will be allowed to choose from the realistic cases that have been proposed by our commercial partners (see Section 1.1) or define their own case for the project (see Section 1.2). The groups have to make their choice known to the module coordinator by e-mail (k.sikkel@utwente.nl) before the deadline (see below). If you choose to define your own project, one of the Scrum mentors will play the role of product owner, and act on behalf of the (fictitious) client. If you choose one of the projects for external clients, one of the teachers will act as product owner. He can act as intermediate between the scrum team and the client, but you also will have an opportunity to talk to the actual client to get more information on the proposed case.

Deadline: 22 April 2015. Make your choice of project known to the Scrum mentors.

1.1 Projects for external clients

This year three projects have been proposed by our 'external' clients, namely Concordia (http://www.concordia.nl/), OVSoftware (http://www.ovsoftware.nl/) and our own student grade management. Detailed descriptions of these projects will be made available on the Blackboard site of this module.

1.2 Own case

The groups are also free to define their own case for the project. Design and implement a web application for your sport club, your theatre group, your uncle's business, your church or any other (legal ☺) organisation that you prefer. The web application has to fulfil a set of criteria and requirements, which can be found in Sections 2 and 4.

2 Assessment criteria

I never teach my pupils. I only attempt to provide the conditions in which they can learn.
Albert Einstein (http://www.einsteinssecret.net/about-the-secret/)

In order to assess the project for the purpose of grading the module team will use the following criteria, which have been grouped in 4 main categories:

Project management

1.1 Correct use of Git and Git branching.
1.2 Project performed according to the defined sprints and global milestones.
1.3 Product backlog and sprint backlog in Trello (user stories + tasks).
1.4 Scrum 'daily' standup meetings each Wednesday, Thursday and Friday morning.
1.5 Five presentations with demos during the sprint review meetings and the product delivery.
1.6 Team work: all group members contribute to Git, Trello, presentations, etc.
1.7 Profession communication with project owner and the customer involved in the development process.

Design

2.1 A case chosen and approved for the project.
2.2 Adequate UML use case diagrams defined for the system.
2.3 Adequate UML class diagrams defined for the system.
2.4 Adequate SQL Database design defined for the system.
2.5 Representative mockup for the application built with a CSS framework.

Programming

3.1 Code compiled using Maven to a Java Web Archive (WAR file).
3.2 Code adequately tested with unit tests.
3.3 Adequate use of Java Servlets.
3.4 Adequate use of RESTful services with XML or JSON.
3.5 Adequate use of JDBC and stored procedures.
3.6 Uses a framework for HTML, CSS and Javascript.
3.7 Concurrency: adequate use of database transactions.

Project finalisation and delivery

4.1 Working system: the application can be deployed and runs on the Tomcat server.
4.2 Functionality of the application systematically tested with (automated) test cases.
4.3 Security issues properly analysed.

3 Work procedures and tools

When working on a big project, it is impossible (or at least far too expensive) to write all the necessary programming code from scratch. Therefore it is common practice to make use of available software libraries, frameworks, applications and tools. The choice of work procedures and tools depends on the working environment and is often arbitrary. The Agile Manifesto proclaims 'individuals and interactions over processes and tools' (see Section 1). However, many work procedures and tools for software development are known to be helpful. Furthermore, team work is only effective if all the team members agree to follow the same procedures and to use the same tools. All the team members have to use the same tools for a team to be productive, even if some individual team member has some other preferences, as illustrated by the words of Guido van Rossum, the driving force behind Python:

> You can make me use Eclipse but you can't make me like it.
>
> *Guido van Rossum (https://twitter.com/gvanrossum/status/424595747397332992)*

Big successful software companies often force their projects to follow standard procedures, so that new project members can be added much more easily to projects than if each project had its own special procedures. Actually, all software projects share some similarities: source code and unit tests can be found at some location, software is produced according to more or less standard steps (e.g., compile, test, package, verify, deploy), documentation should comply with a certain pre-defined structure, etc. In short, without standard procedures and specific tools it takes a lot of time to understand a project and contribute to it, and more importantly, without these agreements it would take the teachers of the Data & information module much more time to assess the project results.

Therefore each project team has to use the following tools and libraries:

- Trello (for planning the work).
- VisualParadigm (for drawing UML models).
- Eclipse (for software development).
- Git and GitHub (for cooperation and version management).
- Maven (for defining software dependencies and facilitating software building).
- Java and Java Servlets / JSP (for implementing web applications).
- Jersey (for implementing RESTful web services in Java).
- JUnit (for unit tests).
- PostgreSQL (database management system).
- Apache Tomcat (application server and web container).

> My goals were different and I wanted to make an opinionated piece of software and I preferred the notion of convention over configuration. I wanted a project's infrastructure to look the same and work the same so I continued to pursue my own model for a project (...) I wanted to save people time by being able to find things in the same place.
>
> *Jason van Zyl* (http://maven.apache.org/background/history-of-maven.html)

Some tools, like Maven, support rigid standard procedures that should only be ignored if there are very strong reasons to apply dissimilar procedures. This practice is often termed 'convention over configuration'. Many software frameworks force developers to follow such rigid procedures. Here, a framework is considered to be a reusable software component, which can be configured so that it can be used for some specific purpose. The most general the framework, the more configuration is necessary, and therefore the less time is saved and the less useful is the framework for a specific project. The project teams are advised to make additional agreements concerning procedures and tools, such as, for example, about the framework to be used for HTML5, CSS and JavaScript.

4 Project management

This section discusses the criteria concerning project management that will be used to assess the project results.

4.1 Git and GitHub

Make sure you become a member of a project team consisting of 5 people (see information on Blackboard). Each member of the team should get a free account on https://github.com/. Ask one of the Scrum mentors for your group number and request him to add all the members of your team to your GitHub team. For example, if your group is number 1, then your team name will have the account "di01", see http://github.com/utwente-di/. After that add a `README.md` file to this account.

Important: Make sure that the HEAD always compiles to a working product, branches are created for each new feature and unit tests are ready (and committed) before new functionality is implemented. Useful information on Git can be found at http://git-scm.com/book/en/Getting-Started

> Deadline: 20 April 2015. All students are subscribed to a team. Teams will be formed during the Introduction session. The teams are then definitive and will be published on Blackboard.
>
> Deadline: 22 April 2015. All students have a GitHub account.

4.2 Sprints and global milestones

In order to make the whole project manageable (also for the teachers) we fixed the number and duration of the sprints, and the global milestones that should be delivered in each sprint.

Sprint 1: weeks 1-3

The first week is dedicated to the preparation, namely finding out which project will be performed and getting some general idea of the envisaged product. By the end of week 1 you should have an (initial) product backlog, so that sprint 1 can properly start in week 2. By the end of sprint 1 you should have a demo of what the product could look like, and, if things go according to plan, a very small application with some bits of functionality that you can demonstrate.

Sprint 2: weeks 4-6

Working system is demonstrated at the end of this sprint, in which more functionality has been implemented. The high-level design of the application (requirements and database models) will also be delivered at the end of this sprint.

Sprint 3: week 7-8

Working system is demonstrated at the end of this sprint, in which even more functionality has been implemented.

Sprint 4: week 9-10

At the end of sprint 4 you have a working application that satisfies the criteria mentioned in Section 2. This version will be reviewed by the teachers.

Sprint 5 (consolidation) week 11

Sprint 5 lasts only one week and is intended for consolidation, rather than developing additional features. You can improve the code or improve some features that you are not completely satisfied with. Sprint 5 ends with a final presentation to the client.

4.3 Sprint backlog in Trello / planning meeting

In this module we use Trello, which is (free) project management application. All students are requested to create a free account on http://trello.com. Each group should create a project in Trello, add tasks to this project as 'user stories' and create a sprint backlog. Make sure each task has an estimate of time it requires to be completed. Allow both Scrum mentors to become members of your Trello project, so that they can follow the progress of your project team. Add the URL of the Trello project board (for example, https://trello.com/b/0tcjABX8/foobar, if your team is called 'foobar') to the `README.md` file in the GitHub repository of your team. Do no forget to ask the Scrum mentor to give a pack planning poker cards to your team.

> Suggested dates for planning meetings: 30 April, 13 May, 3 June, 17 June, 30 June
>
> Deadline: 30 April 2015. All students have a Trello account and can access the project board. Project backlog is in the Trello project board.

4.4 Scrum standup meetings

Every Wednesday, Thursday and Friday morning between 8:45 and 9:45 there is a Scrum standup meeting that takes not longer than 10 minutes. Each team has two Scrum mentors and (whenever possible) one of the mentors is present during this meeting. During the meeting all team members stand up (hence the term 'standup meeting'), and each team member answers the following three questions:

1. What did you finish since the last meeting?
2. What are you going to finish today?
3. What are the impediments (problems) that prevent you from making progress?

Each Scrum standup meeting is relatively short, therefore the problems should be further discussed in the team after the meeting. Team members work together (after the Scrum meeting) to solve the problems that prevent the group from making progress. Each team can use a project room from 8:45 to 12:30 on Wednesday, Thursday and Friday. On Blackboard you will find information about when the Scrum mentors are going to visit each team for the Scrum meeting.

4.5 Sprint review meeting

Five review meetings take place each two weeks on Friday at 13:45. During this meeting the team demonstrates the product delivered at the end of the sprint to the product owner. The team also explains which planned tasks have not been performed. Upload the material used for the presentation (e.g., slides) to the GitHub repository of your team in a directory called `meetings`, where each file name starts with `review`, such as, for example, `review20150508.odp`.
Check http://en.wikipedia.org/wiki/Scrum_%28software_development%29#End_meetings for information on review meetings.

Review meetings: At 13:45 on 8 May, 29 May, 12 June, 26 June, 3 July

Sprint retrospective

The team should evaluate each sprint once it is finished under the guidance of the scrum master. In this evaluation two questions should be answered:

1. What went well during this sprint?
2. What can we improve in the next sprint?

Copy the report on the sprint retrospective to the `meetings` directory in the GitHub repository of your team.

Sprint retrospectives (suggested dates): 13 May, 3 June, 17 June, 30 June

4.6 Team work

Each team member actively participates in the project. Evidences of this participation are the contributions to Git, Trello, the presentations, etc. Team members should distribute the work evenly among themselves. Each team member should change his/her role after a sprint. In particular, each team member should get an opportunity to act as a scrum master.

4.7 Customer and project owner

The customer is the person or organisation that ordered the system. The project owner is the person who represents the customer for the development team. One of the scrum mentors (a 'teacher') plays the role of project owner in a project.

5 Design

This section discusses the criteria concerning the design that will be used to assess the project results. The steps of the design are well defined and have strict deadlines, which are also indicated in this section.

5.1 Choose a case/project

Each team should write a document of around one A4 page that describes the case chosen by the team. This document may contain text fragments from this manual (e.g., from Section 1). Convert the document to PDF format, call it `project.pdf` and upload it to the GitHub repository of your team in a directory called `design`.

> Deadline: 24 April 2015. Document `project.pdf` with project description is copied to the `design` directory in the GitHub repository of the team.

5.2 Mock-up

Select a CSS framework (see http://en.wikipedia.org/wiki/CSS_frameworks) and learn how to use it. Build a mock-up of your application by using the CSS framework you chose. This mock-up should consist of one or more HTML pages that use the stylesheets available in the CSS framework. Generate an archive file `mockup.tar.gz`, `mockup.zip` or `mockup.war` (see …; in this case do not add any Java or JSP file) that contains the files of your mock-up and copy this file to the `design` directory of the GitHub repository of the team. Alternatively you can create your mock-up with the help of a drawing tool such as Sketch or Paint, or with pen and paper. In this case you should create a `mockup.pdf` file by saving the drawing file as PDF or scanning your drawing.

Useful information can be found at http://en.wikipedia.org/wiki/Mockup#Software_Engineering (for mock-ups in general), http://www.bohemiancoding.com/sketch/ (for Sketch), http://en.wikipedia.org/wiki/Pen (for pen) and http://en.wikipedia.org/wiki/Paper (for paper) ☺

The terms 'HTML5', 'Responsive' and 'mobile-first' will make you sound 'cool' during the first sprint presentation, but make sure you know what they mean before using them.

> Deadline: 8 May 2015. Mock-up file is uploaded to the `design` directory.

5.3 UML diagrams

Using VisualParadigm or any other UML tool of your preference, draw a UML use case diagram and a UML class diagram for the web application that you are going to develop. Export your UML diagrams to some picture format (e.g., `.jpg` or `.pgn`) and copy the files to the `design` folder of the GitHub repository of your team.

> Deadline: 29 May 2015. Picture files of the UML use case and class diagrams to folder `design` in the GitHub repository.

In this module we use UML class diagrams for a different purpose than in module 2, where class diagrams were meant to be a model of the software system (Java code) and contained a lot of details, like methods, constructors, abstract classes, etc.. In our case the class diagram is a model of the domain in which the system operates, and is meant to be used as input for the database design. Therefore this class diagram can be much simpler (has less details) and is independent of the programming language being used in the system.

5.4 SQL Database design

Create the file `schema.sql`, which should contain the database design that has been derived from the UML class diagram of Section 5.3. Make sure that the primary keys and the foreign keys are properly defined in this schema. Upload this schema to `design` directory of the GitHub repository of the team.

> Deadline: 29 May 2015. File `schema.sql` containing the database definitions is uploaded to `design` directory.

6 Programming

This section discusses the criteria concerning the programming work that will be used to assess the project results by defining tools, libraries and components that are mandatory. You will probably only be able to fully understand these criteria after following the lectures and doing the exercises that cover these tools, libraries and components.

6.1 Compilation to a WAR file with Maven

The team must follow the Maven standard directory structure for Java projects (see http://maven.apache.org/), as explained in the first laboratory session. Make sure your web application compiles 'out-of-the-box' by declaring all the project dependencies in the `pom.xml` file located in the root of the Maven project directory. The build process (compilation) should generate a `.war` file that can be deployed in the Apache Tomcat Application Server (see http://tomcat.apache.org/). Copy the Java code and the Maven `pom.xml` file to the GitHub repository of the team.

6.2 JUnit Unit tests

The code (Java classes) should be tested with Junit (see http://en.wikipedia.org/wiki/Junit) unit tests. Make sure that the unit tests that use the database work on an empty PostgreSQL database, so that the unit tests can add and remove data from the database accordingly. The JUnit annotations `@BeforeClass` and `@AfterClass` can be quite useful and are recommended.
Tools for 'code coverage', such as, e.g., EclEmma (http://www.eclemma.org/) can be also useful.

Test-driven development [recommended]

When applying test-driven development the unit tests are developed and committed to the GitHub repository before the code to be tested is implemented (ee http://en.wikipedia.org/wiki/Test-driven_development). The unit tests play the role of the requirements for the code, by describing what the code is supposed to do (or how the code is supposed to behave). Use a new Git branch for code that fails the tests, and make sure the JUnit tests always succeed in the HEAD. Never check in code that does not compile.

6.3 Java Servlets

In this module we use Java Servlets to implement web applications (see http://en.wikipedia.org/wiki/Java_Servlet). The code produced by the groups should therefore make correct use of Java Servlets. The Tomcat Application Server is the reference implementation of the Java Servlet and JSP standards (APIs), and that is one of the main reasons why we use Tomcat in this module.

6.4 RESTful web services and XML

In this module we also introduce the students to RESTful web services (http://en.wikipedia.org/wiki/RESTful). The application should therefore make use of RESTful services and XML serialisation, possibly using JSON (http://json.org/). Jersey (https://jersey.java.net/) is the standard implementation of the JAX-RS standard, which defines how RESTful services can be implemented using Java. This is one of the main reasons why we use Jersey in this module.

RESTXQ [recommended]

In the SEMI-part of this module, you will also learn to work with RESTXQ, which is a standard for developing RESTful web services using XQuery only (XQuery is a standard query language for XML data). You are recommended to try out this technology for one or more of your web services to experience its strengths and weaknesses as compared to developing them in Java.

6.5 PostgreSQL and JDBC

In this module we use PostgreSQL as database management system (DBMS). In order to decouple the Java code from the specific DBMS chosen in this module the code should interact with PostgreSQL through JDBC (see http://en.wikipedia.org/wiki/Java_Database_Connectivity). It may be desirable to develop certain functionality using stored procedures.

6.6 Framework for HTML, CSS and Javascript

The code should make use of a CSS framework to implement the user interface (also called the 'front-end'). The more complete the mock-up built in the initial design (see Section 5.2), the easier it will be to reuse it in the implementation of the front-end.

6.7 Concurrency

The web application correctly uses database transactions where they are required by the application. You may use lower isolation levels to improve performance, but be sure to use it correctly.

7 Project finalisation and delivery

This section discusses the criteria concerning the final product and the project finalisation.

7.1 Working system

The final product (the web application) should be deployed and run in the central Tomcat Application Server (AS) at http://datainfo.ewi.utwente.nl. You should ask the scrum mentor to give you a userid and a password to sign in to this server. Add the URL of your application to the `README.md` file in the GitHub repository of your team. For example, the URL should be http://datainfo.ewi.utwente.nl/di10_app if your application is called 'di10_app'.

Since all users of the central Tomcat AS can manage all applications, you have to make sure your application has a unique name. Therefore we use the convention that the application name starts with 'diXX_', where XX is the team number. Furthermore, you should only deploy the application to the central Tomcat AS after the application has been thoroughly tested using Eclipse and your local Tomcat installation, since this central Tomcat AS is shared by all groups and it takes a while before it can be restarted in case you manage to make it crash. Making the central Tomcat AS crash will not make you popular with your fellow students.

Some ambitious groups may assign a separate domain name for their applications. In this case you should inform the domain name and the application name to the Scrum mentors, so that the central server can be properly configured. A separate domain name does not result in a higher mark, so this is only 'for fun'.

Deadline: 26 June 2015. Delivery of final product (all updated artefacts and running web application).

Deadline: 3 July 2015. Final presentation.

7.2 Application testing

In addition to testing the Java classes using JUnit as mentioned in Section 6.2, you also have to test the your application systematically as a blackbox. Web application can be tested by (Java) applications that simulate browsers, or by using browser plugins. For example, Selenium (http://docs.seleniumhq.org/) is a popular environment that offers many tools for facilitating the systematic testing of web applications, including an environment to define test suites and a Firefox Add-On for automatically performing Selenium test cases. Your web application should be systematically tested using one of these advanced tools.

7.3 Security analysis

The team has to analyse the security aspects of the system, and make sure the application is properly protected against SQL-injection and cross-site scripting, amongst others. Each team has to write a security analysis document and upload it to the GitHub repository of the team

More information will be given during the Security days on 8 and 9 June 2015.

8 Conclusion

If you have gone through all the chapters and arrived at this point, congratulations! This means that ***either*** you are ready to start your project ***or*** even ready with the project itself, because who nowadays reads a manual from cover to cover? In case you have finished the project, you have probably noticed that some parts of the project were easy, but some others were difficult or even frustrating. The installation of some software packages, for example, may have costed many hours of your sleep. The use of some software libraries force you to think like the programmer of these libraries and may require some time and effort.

… somewhat unique to programming is that it consists of near constant failure. Unlike learning other skills where one can expect to be reasonably competent after sufficient practice, programming largely consists of constantly failing, trying some things, failing some more, and trying more things until it works. One of the biggest differences between experienced and novice programmers is that experienced programmers know more things to try.

Alicia Liu, Overcoming the Imposter Syndrome (https://medium.com/tech-talk/bdae04e46ec5)

8.1 The myth of the 'real programmer'

In case you are worried about your own programming skills, or if you are not sure whether you are a 'real programmer', or if you think that the lecturers of this module have left you to your fate, read Alicia Liu's blog post mentioned above. A good programmer knows a couple of tricks that are worth mentioning (and we do not mean naming all characters of Stark Trek):

1. Be patient and systematic, and try one solution at time.
2. Read the … manual (http://en.wikipedia.org/wiki/RTFM).
3. GIYF: Google is your friend! Or Stackoverflow, or Duckduckgo (if you are paranoid about your privacy). In this manual we often use Wikipedia as an entry point for information on some subjects, running the risk of being criticised for not being 'scientific' enough.
4. Explain your big problems to the other members of the team, and if they are not around, to your rubber duck (see http://en.wikipedia.org/wiki/Rubber_duck_debugging).

If you want to play with computers, what do you need…? To be patient!
Luís Ferreira Pires (to his children when they started using computers)

8.2 Module evaluation

This module is being offered for the second time this academic year. We collected all the feedback we received last year (the first run of the module) and we tried to process this feedback when preparing the module this year in order to make the module more challenging and stimulating. We believe that feedback is the key to improvement, thus we encourage you to participate in the module evaluation.
Evaluation and monitoring will take place in different forms. Short online questionaries will be available in weeks 4 and 8 in which the students will be asked to give their feedback about the module. A panel of 4-5 students will also have regular feedback meetings with the module staff. If there is anything that requires immediate action, feel free to directly contact the module coordinator.

Appendix A Deadlines

I love deadlines. I like the whooshing sound they make as they fly by.
Douglas Adams

For your convenience, the deadlines set for the activities or deliverables described in this document are summarised in chronological order in Table 1.

Table 1 Deadline of project deliverables / activities.

Deliverable/ Activity	Deadline
All students are subscribed to a team.	20 April 2015
Make your choice of project known to the Scrum mentors.	22 April 2015
All students have a GitHub account.	22 April 2015
Document project.pdf with project description is copied to the design directory in the GitHub repository of the team.	24 April 2015
All students have a Trello account and can access the project board.	30 April 2015
Mock-up file is uploaded to the design directory.	8 May 2015
Picture files of the UML use case and class diagrams to folder design in the GitHub repository.	29 May 2015
File schema.sql containing the database definitions is uploaded to design directory.	29 May 2015
Delivery of final product (all updated artefacts and running web application).	26 June 2015
Final presentation.	3 July 2015