Jeppe Holm Adnan Unal Jiří Vrbas Marcin Zelent

aka

ALPHA SQUADRON

3rd semester project report

1. Introduction

Author: Marcin Zelent

This is the documentation of Alpha Squadron's group project, mandatory activity on the 3rd semester of Computer Science AP programme at Zealand Institute of Business and Technology. Alpha Squadron consists of Jeppe Holm, Adnan Unal, Jiří Vrbas and Marcin Zelent.

The purpose of this document is to organize our group work, to describe the requirements and architecture of the system, we are creating, the methodologies and technologies we used and also the whole software creation process.

The document structure is based on agile methodology. The chapters are corresponding to each sprint and contain subchapters according to sprint components.
Because we have two roles, product owner and developer team, sprint chapters are split into sections for each of the roles.

There are also some chapters outside of sprints such as project establishment, theory and summary.

2. Project establishment

2.1. Our team roles

Author: Jeppe Holm

We will be rotating scrum master every sprint.
Below is the rotation of Scrum Master in the designated sprint.
SM = Scrum Master
PO = Product Owner

Names	Sprint 0	Sprint 1	Sprint 2	Sprint 3
Marcin	SM / PO	PO	PO	PO
Adnan		SM		
Jeppe			SM	
Jiří				SM

2.2. Working process theory

Author: Jeppe Holm

Before going into sprint 0 and establishing requirements, we want to start by explaining the theory behind our software development in this project. For the project we used agile methodology - Scrum for the process oriented part and Extreme programming for the product oriented part of the project.

Scrum[1]

There are 2 major things to talk about when working with Scrum: The parties of scrum and the Scrum lifecycle.
First we would like to talk about the parties of Scrum and then go into, what is actually happening in the components of the Scrum lifecycle.

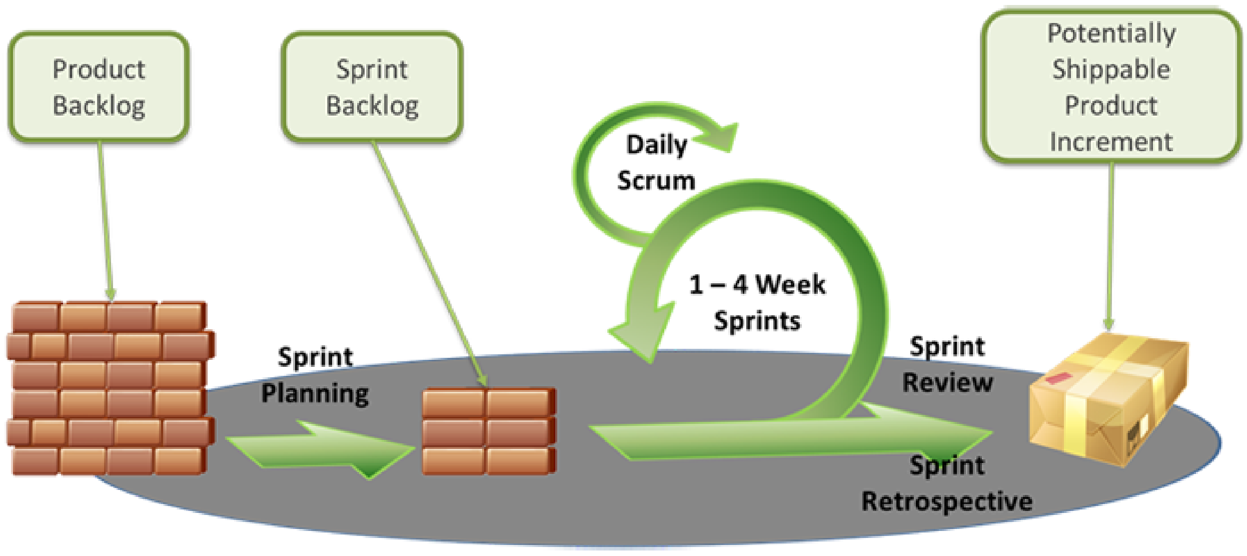
[1] Chap 8 SCRUM (in details) slides by Mohammed

Parties of Scrum

The people attending the Scrum meetings consist of: **product owner (PO), Scrum Master and the Scrum team (ST).**
PO is a business guy representing the group / company, that has requested a product from us.
Scrum master is a programming manager that makes sure, that the ST is working focused and efficiently and not encountering any issues.
ST is typically a team of 5-10 people (QA, programmers, UI designers etc.).

Scrum lifecycle

Scrum lifecycle has 4 components: **Sprint Planning, Daily Scrum meetings, Sprint review, Sprint retrospective.**
Below is an image showing the usual working lifecycle of Scrum.



Sprint 0: User stories and estimation

Before talking about the 4 components of the Scrum lifecycle, it would be better to start by explaining what happens chronologically. As shown to the left of the image is the product backlog, which is the first thing that should be created before Sprint Planning can begin.

This is done in a collaboration between the ST and PO (PO). When starting the Sprint 0, the PO creates the User Story or in other words defines the requirements for the product, the PO wants created. The PO prioritises these user stories by which PO finds to be the most important for them and then gives these user stories to the ST.

Here the first meeting is held between meeting between PO and ST, where they try to create a complete understanding of what the PO actually wants and writing any technical user stories if necessary.

When the ST has received and understood, what the PO wants, they can start estimating how difficult / time consuming each User Story is. This is done using relative estimation techniques such as a **Planning Game or Triangulation**

Planning Game

In a Planning Game each member of the ST gets a deck of cards, each card with a different number ranging from 0 and 100. This number is to define, how difficult / how time consuming the specific User Story is. This number is called **Story Points.** In a usual Planning Game story points range from 0 to 100, where 0 is done in no time and 100 is impossible. Each User Story gets introduced separately and each member gets to give that User Story a card with a number of story points on it and their opinion on why it should be exactly this number. This continous until everyone is in agreement of what number to give each User Story.

Triangulation

When using Triangulation you take a few reference stories and use those user stories to compare to the assigned user stories and give them story points based on experience.

Sprint 0: Velocity

After all the user stories are estimated, it is time for the ST to decide, how much they can actually make during a sprint and what they are actually going to make. This is done by creating another value called Velocity, which will define, how many story points the ST can make during a sprint. When both user stories have been estimated and Velocity is defined, this is given to the PO to decide, what the ST should actually be working on. The last step in Sprint 0 is the PO assigning the ST to work on one or more user stories, which will be the Sprint Backlog.

Sprint Planning

When the ST has received their assigned, user stories by the PO the sprint planning can begin. The first thing that happens during the Sprint Planning is the specification of User Stories. This is done by dividing the assigned User Stories into tasks and placing these tasks in the sprint backlog for all members of the ST to overview usually by making a Kanban Board etc. The Scrum members then pick a task, they want to work with and the development can begin.

Kanban Board

To get an easy overview of what tasks the ST has to do and who are currently making what tasks you usually use a Kanban Board.
The Kanban board consists of 5 columns: **Product Backlog, To Do, Test, Doing and Done**.

In our case where we use the Kanban Board for managing and assigning tasks we can use the **product backlog** to see which user story all these tasks originally came from. In **To Do** there is the tasks that needs to be made. In **Doing** you can see which person is currently / did the task by looking if the **done** field has been marked. The **test** field is used to check who is testing the task.

Daily Scrum Meetings

To make sure that the development process / Sprint is running smoothly Daily Scrum meetings are done every morning to review on how each ST member is progressing with the chosen task or in other words: how far they are with making the task, issues they encountered, what they will be doing today.

Sprint Review

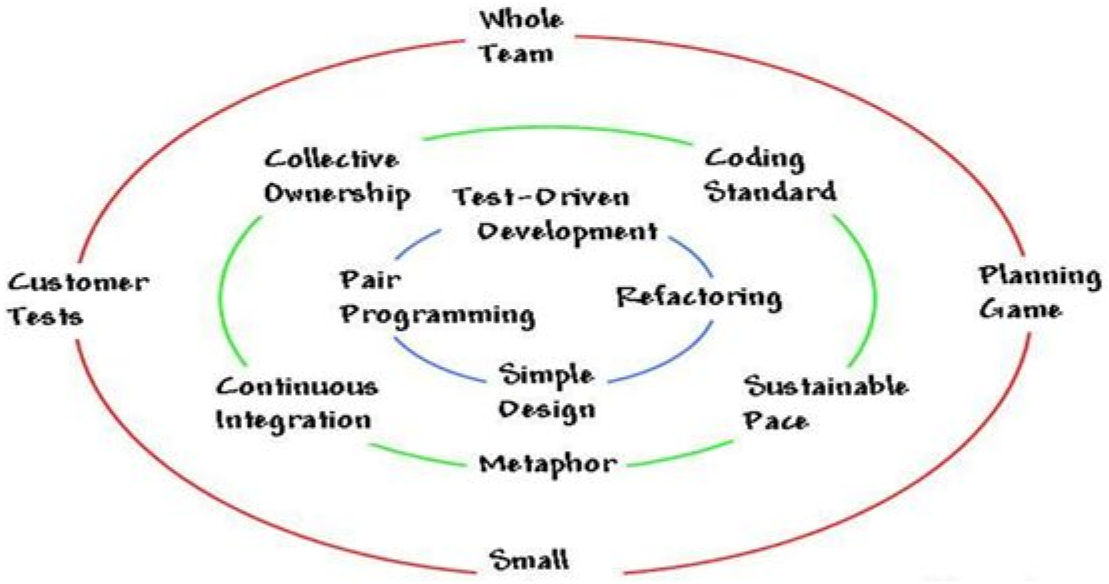
In the Sprint Review the ST shows the PO what they have made during the sprint usually through a demo.
PO checks if the acceptance criteria of the user stories are met in the program.
It is also during the Sprint Review that a reevaluation of the user stories' story points and the team's velocity and the PO assigns what to work on for the next sprint.

Sprint Retrospective

After the review the retrospective is made within the ST to discuss the feedback they got from the PO and what to improve on both the product- and process-oriented part.

Extreme Programming (XP)[2]

In this project we will be using some of the XP principles for doing the product oriented part. In other words, we will be using XP principles for writing the program, that will satisfy the customer. The easiest way to describe the XP principles are by looking at a summary of the principles seen on the image below:
These 3 circles are the core principles of XP and can be summarised into:



Red

The outer (red) circle is the project lifecycle. The project is kept going by first doing a **planning game** and producing **customer tests** to specify the requirements for creating the code. Then the **whole team** plans **small releases** of the program, that can pass those **customer tests**.

Green

The middle (green) circle is the circle with the supporting principles, that should be followed when programming in XP.

[2] Extreme Programming slides by Mohammed

The program should be kept **continuous integrated** with a specific **standard of coding** / **metaphor**, that the whole team should follow. The standard of coding / metaphor is mostly for the next principles collective ownership which means, that any qualified pair within the team should be able to modify the code, if changes are needed.

The programming team should follow a sustainable pace of deliverance, so there is a predictable "pace of deliverance". This predictable "pace of deliverance" is very important when working on projects like this when the goal is to meet the deadlines set by the PO.

Blue

The inner blue circle is the work practices in XP.
A Simple Design is important.
Always do pair programming.
Test-Driven Development is supported in XP.
Code is kept clean by refactoring.
In other words following these 4 work practices in XP will help creating a clean and usually easier to understand program where there's less redundancies, errors and bugs.

Test-Driven Development (TDD)[3]

TDD is a development practice where you, instead of going straight to coding the requirements and then test them, start by writing a test. TDD is known as a test first development where you start by writing the test and then start planning out and creating the class and methods, which are going to make this test pass. After the test passes, you write another test scenario and repeat until all the requirements have been met.

There are a lot of benefits from using TDD. For example it ensures code quality from the start because the developers are encouraged to ONLY write necessary code to make the test pass and therefore the requirements. When the developers are focused on only making the test pass, they are also going to write less unnecessary code which also makes less room for errors.

Since we are following Scrum it is the PO writing the user stories and acceptance criteria. This means that TDD encourages communication between developer and PO which is an important part of the agile methodology.

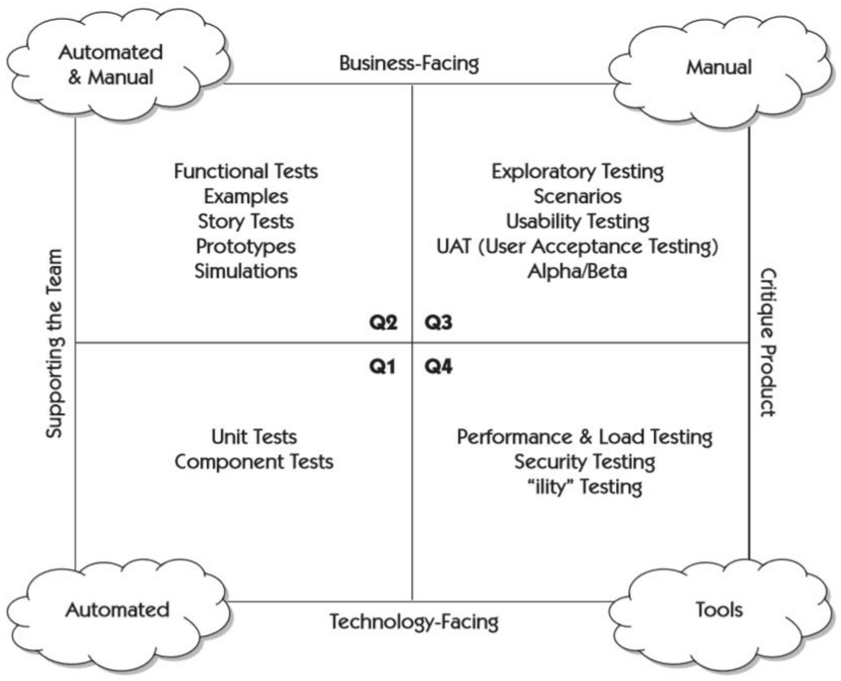
[3] TDD slides by Mohammed

Also since those acceptance criterias are written as tests that reflect the acceptance criteria there’s a higher possibility that the program meets the needs of the PO. At the same time it might also gives the PO a better understanding of what we are actually building for them and if the developer has misunderstood the requirements the PO can quickly help the developer come back on the right track.

TDD also provides built in regression test. The tests will always be the same but if changes are made to the system, which might result in the test failing, the developer can quickly find the issue and fix it.

Other kinds of testing: The Agile Testing Quadrant[4]

The Agile Testing Quadrant is used by the team / testers to ensure that they covered all categories of testing.
Below is a picture of the Agile Testing Quadrant and some examples of tests that could be in those quadrants.



On one axis we divide the quadrants by tests that support the team and tests that critique the product. On the other axis we divide the quadrants by the tests being business-facing or technology-facing.
A business-facing test could be defined as a test that could be of interest to a business expert / PO.

[4] Agile testing quadrant slides by Mohammed

A technology-facing test could be defined as a test that should be of interest to a developer and is to be explained in technical terms.

The Agile Testing Quadrant is divided into four quadrants: Q1, Q2, Q3, Q4.

Q1 contains the tests that define the internal quality of the code. The tests in this quadrant are usually created through TDD.

Q2 defines the external quality and the features the customer wants. In other words this quadrant is used for supporting the work of the development team.
While Q1 and Q2 are more about support the team to create a product Q3 and Q4 is about critiquing the product in different ways.

In **Q3** the testing done is an attempt to emulate the way a real user would use the product. This is all done manually through different kinds of tests that help create scenarios and in some cases find bugs that needs fixing.

Q4 is about the more serious issues that the end user might encounter such as performance, robustness or even security.

3. Sprint 0

3.1. As a PO

Security System X

Author: Marcin Zelent, Jeppe Holm

We need a security system that will be able to spot unwelcome guests. We need to have a sensor measuring noise levels and a motion sensor that if goes off take a picture of the invader. The data from the sensors should be stored in a database and should be viewable on a website.

We also need an email service that weekly sends an email to our account that reports stuff like: average weekly noise level, amount of times motion sensor has gone off and any pictures taken by the motion sensor.

We want this system because we are afraid that somebody might break in while we are not at home or sleeping. By having this system our house will be more secure, and we will not have to worry so much about burglars.

User Stories

Author: Marcin Zelent, Jeppe Holm

A user story is the equivalent of a use case in Unified Process. The user story is, as mentioned in the Scrum theory, used by the PO to help describe what the PO wants the ST to make and what order the PO wants the ST to make it in. A user story has the format “As an <x>, I want to do <y>, so that <z>”.
In short, a user story describes who it is for, what they want, and what they need it for.

After the user stories are made, some acceptance criteria describing the requirements for successfully having made, what the user story requested also needs to be made. When the PO thinks he has made all the user stories, he needs to fulfill his request, he gives them to the ST.

1) As a house owner
I need to see the data collected by the sensors
to see if there has been an intruder.

Acceptance criteria:

- The data of the noise level sensor should be displayed and stored when it reaches over a certain threshold.
- There should also be data for occurrences of motion sensor triggers.

2) As a house owner
I need to see a photo
to identify the potential intruder.

Acceptance criteria:

- The motion sensor taking the picture should not be so sensitive that it takes a picture of ANYTHING that triggers the sensor.
- The photo should be in JPG format.
- The motion sensor should not go off at nighttime.

3) As a house owner
I should receive an email once a week with a report of average weekly noise level
so I can have an overview of weekly noise levels.

Acceptance criteria:

- The average is calculated properly.
- The report should be a PDF document.
- It should be sent to the correct specified email address.

4) As a house owner
I want to filter the data by periods and type of record
to get a better overview of the data.

Acceptance criteria:

- The data is properly filtered.

5) As a house owner
I want to be able to sort the data by date or noise level
to get a better overview of the data.

Acceptance criteria:

- The data is properly sorted.

6) As a house owner
I want to see a graph of noise levels
to get a better representation of the data.

Acceptance criteria:
- The graph should properly reflect the data given by the noise level sensor.
- The graph must be a line diagram.

7) As a house owner
I want to see a graph over occurrences of pictures taken
to get a better overview of the data.

Acceptance criteria:
- The graph should properly reflect the number of occurrences of pictures taken.
- The graph must be a line diagram.

8) As a house owner
I want to put hours of when the security is active
to not go off all the time when I am at home.

Acceptance criteria:
- The security accepts input through a calendar showing the week and hours.
- The security system goes off only when I am not at home.

9) As a house owner
I would like to be able to delete some records from the database
so they would no longer exist there.

Acceptance criteria:
- The record should not exist in the database after deleting.

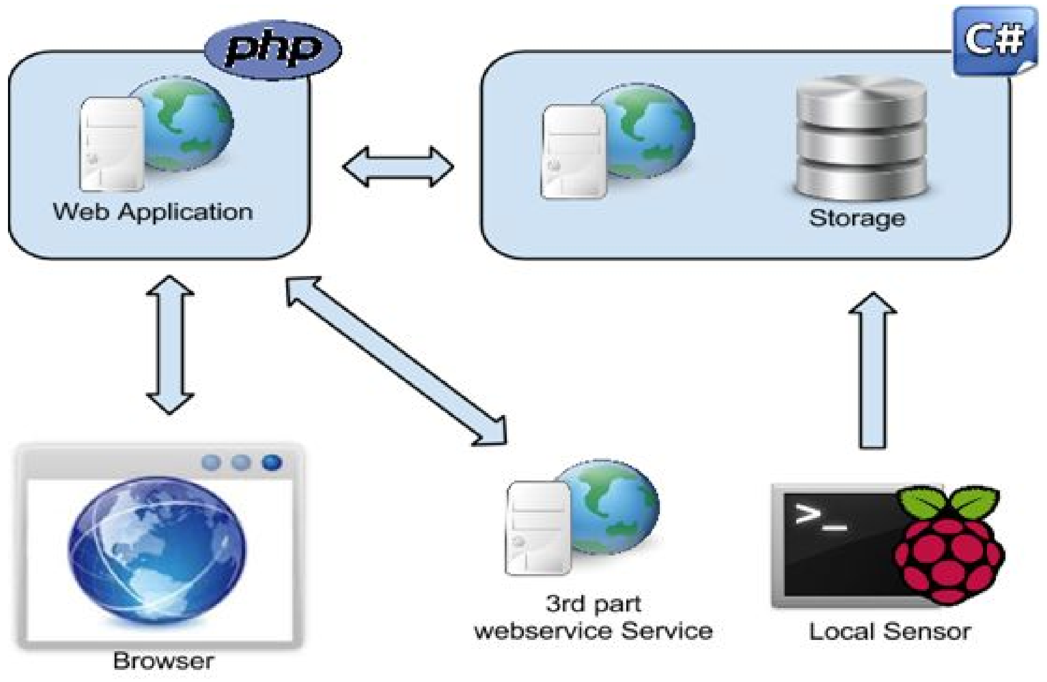
Priority order of user stories as a PO:
1 > 2 > 4 > 5 > 6 > 7 > 8 > 3 > 9

3.2. As a developer team

System architecture requirements

Author: Jeppe Holm

For this project we were given a project charter with a list of requirements the system were to contain. This can be seen on the image below.



In other words the application had to:

- Use *at least* one 3rd party web service
- Read and use data from *at least* one local sensor
- Store data from the local sensor in Microsoft Azure
- Access data from Azure using web services
- Have a middle-tier written in PHP
- Have a browser based user interface / website
- Be tested
- Be documented

System architecture plan

Author: Marcin Zelent

In the group we discussed the system architecture requirements and we created a plan of what technologies to use in order to fulfill them.

Local sensor

We decided, that we needed a sensor for Raspberry Pi which would measure the pollution in the air, especially the amount of CO, NO and SO. Using a Python code we could read the values every minute and then send them to a database.

As for sending the data, we had two ideas. One was to send the data using UDP broadcast to receiver, which would then post it to database using Web API or with SQL queries. The other one was to write Python code for sending HTTP requests directly to the Web API without a receiver acting as middle man.

We chose the second solution, because it is much simpler and it removes the necessity of having another computer running all the time. It is also more reliable, because there are less components and less possibilities that something can go wrong.

Storage

We did not have too many choices here as the requirement is to store the data in Azure. We are limited by the possibilities of Microsoft Imagine account, which means we could either use Microsoft SQL Server or create Web App with MySQL support. We decided to use the first one, because we have more experience with it and we did not want to have Web API and database as one component, because it would be harder to develop it locally.

Web service

We had many options for web service and it was not easy to choose one. The only requirement for it was that it should be written in C#. There are many web frameworks available for this programming language, for example: ASP.NET, ASP.NET Core, Windows Communication Foundation, ServiceStack, Nancy, Manos.

We narrowed the choices to three biggest and most popular ones, meaning the ones which are being developed by Microsoft. Then, we had to decide if we want to make REST web service or SOAP. We opted for REST, mainly because the only way to create a SOAP web service is to create a WCF project, which is available only in Visual Studio for Windows. We did not want to be bound to just one platform, we wanted to be able to develop our solution on different operating systems like GNU/Linux and Apple macOS. It is only possible with RESTful web service created either with ASP.NET or ASP.NET Core, which is a successor of the first one.

Apart from being cross-platform, REST has many more advantages over SOAP. It supports sending data not only in XML format, but also plain text and JSON, which helps save the bandwidth, it has better performance and it is easier to create. That is why it is more popular than SOAP among big companies such as Google, Yahoo, Amazon and eBay.

So, finally we decided to create RESTful Web API with ASP.NET, which we could send HTTP requests to in order to get, add, remove and update data in the database. We did not want to ASP.NET Core, because we believe it is not mature enough yet. Although, it is not a requirement, we thought that it would be a good idea to have it on Azure, so it would be easily accessible.

Web application

Web application is the last part of our system. It acts as a consumer of the web service and a frontend for the user. It is supposed to written in PHP, which seems to be a perfect language for this kind of tasks.

Because we wanted to save time and ease the development process, we started looking for a PHP framework which could help us. We found Symfony and decided to use it, because it has automated testing, good folder structure and it is well documented. It is also using Twig as a templating engine, which we have some experience working with.

3rd-party web service

One of the requirements is that we have to use a 3rd-party web service and we need to show the train schedule for Roskilde station. We searched online for APIs which we could use for that purpose and found only Rejseplanen API, so it was an obvious choice.

User stories

1) As an air quality inspector
I want to see the measured data at the current moment
so that I can see if it is at the acceptable level .

Acceptance criteria:
● Measure the values each minute
● Show the measured values in the browser

Estimation: 40

2) As an air quality inspector
I want to see the acceptable air pollution level
so that I can compare the measured data with the acceptable level.

Acceptance criteria:
● Show the acceptable air pollution level in the browser
● It should be constant value

Estimation: 0.5

3) As an air quality inspector
I want to see the maximum air pollution level allowed
so that I can compare the measured data with the maximum level.

Acceptance criteria:
● Show the maximum air pollution level in the browser
● It should be constant value

Estimation: 0.5

4) As an air quality inspector
I want to see the measured data from the last week
so that I can see the air pollution alterations.

Acceptance criteria:
- Show the saved measured values from last week in the browser
- Show data from each day of the week for last week and this week

Estimation: 8

5) As an air quality inspector
I want to get a warning when the levels of pollution are too high
so that I can react accordingly.

Acceptance criteria:
- Sends out a mail to the air quality inspector with the air quality level when the warning is triggered.

Estimation: 20

6) As an air quality inspector
I want to see if the warning is triggered by Carbon Monoxide, Nitrogen Dioxide or Sulfur Dioxide increase in the air
so that I can know what caused the warning.

Acceptance criteria:
- Include relative data in the mail if the warning is triggered by Carbon Monoxide, Nitrogen Dioxide or Sulfur Dioxide with the level of health concern.

Estimation: 3

7) As an air quality inspector
I want to see the Roskilde station train schedule
so that I can compare it with the measured data and see if and how they are related.

Acceptance criteria:
- Show the train schedule and the measured values for each train coming

Estimation: 33

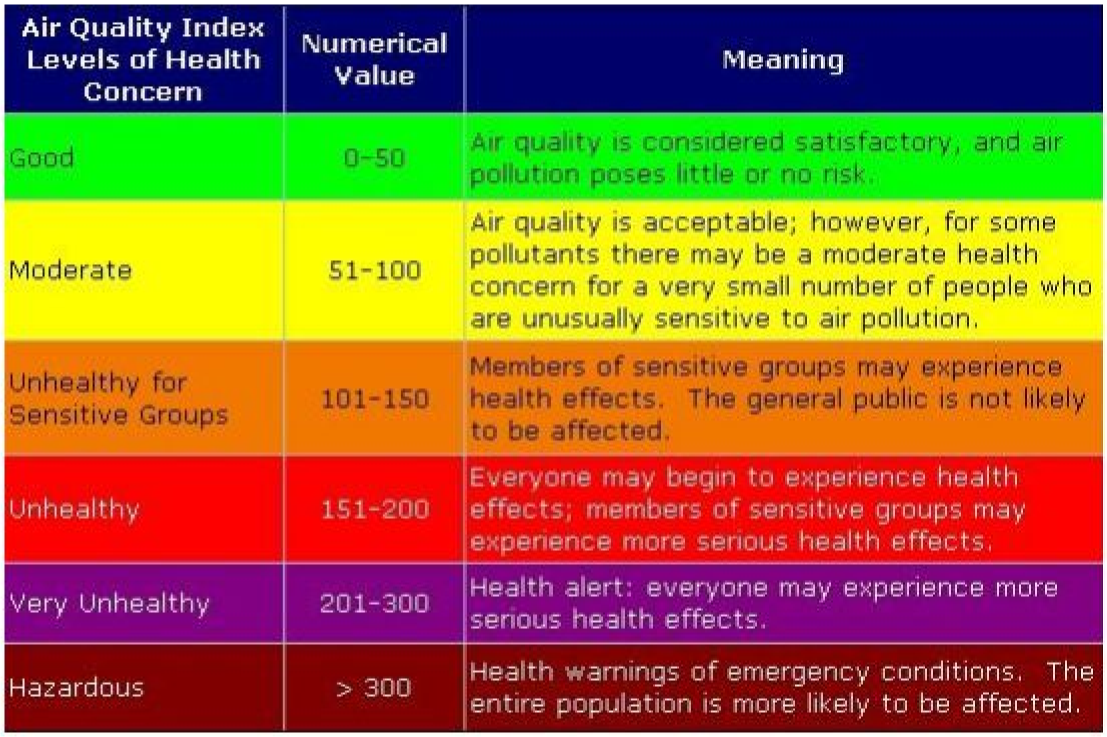
8) As an air quality inspector
I want to see charts with the measured data
so that it is easier to understand the levels of pollution.

Acceptance criteria:
- Implement and update chart/tables with detailed information (i.e time, level and so on..) about recent air pollution levels weekly

Estimation: 13

9) As an air quality inspector
I want to see the calculated Air Quality Index(AQI) based on the Co, No or So
so that it is easier to understand the level of pollution and the level of health concern.

Acceptance criteria:
- Calculate Air Quality Index based on the measured data of air pollutants



Air Quality Index Levels of Health Concern	Numerical Value	Meaning
Good	0-50	Air quality is considered satisfactory, and air pollution poses little or no risk.
Moderate	51-100	Air quality is acceptable; however, for some pollutants there may be a moderate health concern for a very small number of people who are unusually sensitive to air pollution.
Unhealthy for Sensitive Groups	101-150	Members of sensitive groups may experience health effects. The general public is not likely to be affected.
Unhealthy	151-200	Everyone may begin to experience health effects; members of sensitive groups may experience more serious health effects.
Very Unhealthy	201-300	Health alert: everyone may experience more serious health effects.
Hazardous	> 300	Health warnings of emergency conditions. The entire population is more likely to be affected.

Estimation: 5

4. Sprint 1

4.1. As a PO

Sprint planning

Author: Marcin Zelent

The development team told us they had a velocity of 40.
We assigned them to do **user story 1:**
As a house owner
I need to see the data collected by the sensors
to see if there has been an intruder.

Acceptance criteria:

- The data of the noise level sensor should be displayed and stored when it reaches over a certain threshold.
- There should also be data for occurrences of motion sensor triggers.

Sprint review

Author: Marcin Zelent

At the end of sprint 1 we met with the developer team to evaluate their work. Although they were supposed to finish user story 1, they did not manage to deliver.

However, they took some steps towards realizing it. They set up a Raspberry Pi with a noise sensor, wrote a Python script for reading its measurements and printing a message when noise is over a certain value. They have also created a simple database in Azure for storing gathered data and a basic HTML website for displaying it. Because they do not send readings to the database and they do not have any web service connecting the database and the website, so far they serve no purpose.

For the next sprint, our developer team decided that their velocity is 20. We told them to continue working on user story 1 and when they will finish it, they should contact us to ask what user story they should work on next.

4.2. As a developer team

Sprint Planning

Author: Jeppe Holm

Velocity: 40.
The PO has decided for us to work on **User Story 1:**
As an air quality inspector
I want to see the measured data at the current moment
so that I can see if it is at the acceptable level .

Kanban Board for Sprint 1

Product backlog	To do	Doing	Test	Done
User story 1	Design database	Marcin / Jeppe	Marcin / Jeppe	✓
	Set up REST API on Azure using the database	Marcin	Marcin	✓
	Set up routes on API to POST and GET data	Marcin	Marcin	✓
	Read data from Raspberry Pi sensor every minute	Jiří		✓
	Send read data to the Azure web service	Jiří		✓
	Set up PHP server with a framework	Adnan		✓
	Show real time data on webpage using websockets (Send ajax on azure database update to php > trigger websocket on route)	Adnan		✓
	Write Documentation	Marcin / Jeppe	Group effort	~

Sprint execution

Author: Jiří Vrbas, Adnan Unal

We have been assigned the task of displaying the data measured by our device in the sprint number one, so the user can see if data are in the allowed zone. We have decided to divide this task into several tasks as seen on the Kanban Board above.
It was necessary to:
Design a database in which the measured values would be stored.
Design a web service that would access database features.
Assemble and program a device for reading the actual values of the measured quantities.
Design a website to display these values.

Database

First we had to design the database where the stored measured values would be. It was important to decide how our database would look like to match the data, sent from the raspberry. We created the databases using SQL. We decided to keep the values in the database using a timestamp as an integer and the values of carbon dioxide, nitrogen dioxide and sulfur dioxide in float. Then we uploaded the entire database to the Azure Server to be able to access it from the web service and the web site. We also inserted several records to the database with which we could work with immediately.

Web service

Another thing we had to do was to design a web service. We had an option between Soap web service or REST web service. As mentioned earlier we decided to use REST because it was easier for us to implement. Because our web service is used to access or modify data in the database it was necessary to connect to the database first. Then we had to solve the routing our service would be using and decide, how the service will work. We also have this web service placed on our Azure server to access it at any time.

Sending data from sensor

Next, we had to create and program a device that would be measuring data and inserting them through a web service into a database. Unfortunately, we came across a problem here because the necessary sensors were not available for us and therefore we could not compile the necessary hardware. We had two ways of solving this problem. The first solution was to generate random data that may be near reality, or get this data from an external source. After some research we found a web service that provides this data for free and decided to use it. That meant however that it was necessary to write the code in Python that would call a third-party web service, read the data, and send it to our web service.

When designing this script, we encountered the problem that not all of the data produced by web service responded to the values we needed for calculating the air quality index. So we only needed to select data that suits our model.

Browser and real time updating

The last important part of Sprint one was to create a web application in which would be displaying measured values in real time. To simplify testing, we decided to use Symfony which is a PHP framework. To show our sensor readings in real time we had multiple options. We could have used websockets to get instant updates the moment a new reading is sent to the database or we could have sent a request a set interval to the server. We decided to send a request every minute to the server since it is easier to implement and the readings are updated in specified interval (every minute) so sending requests every minute is a valid solution. We have also placed this website on our Azure server.

Testing

An important part of the agile development of applications is testing. In our case we used unit testing, which means automatic testing and verification of the functionality and correctness of the system implementation. Unit testing including tools, methodologies and actions aimed at verifying the correct functionality of the parts of the source code. In our system, we wanted to test each class or its method. It is necessary to test it isolate the test part from other parts of the program.

In this sprint, we wanted to test the web service and our website. So we created a number of test classes that test each feature of the Web service. It was necessary to test the AQI calculator to handle different ranges of values.

Also, it was necessary to test the created databases. We tested the restoration of all the values from the database, editing records, adding new records, and deleting existing records.

In addition, it was necessary to test the web service itself, and most of all whether all service calls worked. So we tested getting all the data from the database by request, getting one result from the database, obtaining the last measured value in the database, obtaining the measurement values from the last week, successful insertion of measured values, insertion of null values and successful and unsuccessful editing and deleting of values.

Of course, not all of these methods were used in our system, but would certainly be suitable for future expansions.

In addition, we would need to test our python code. But our code was so short and simple, that we did not create any testing of this program. However it is possible to write automated tests using a tool called pytest. Testing is similar in this case as in C# itself.

The last important testing in this sprint would be dedicated to a website, but because we did not know which framework to use at this point, we did the testing for this in the next sprint.

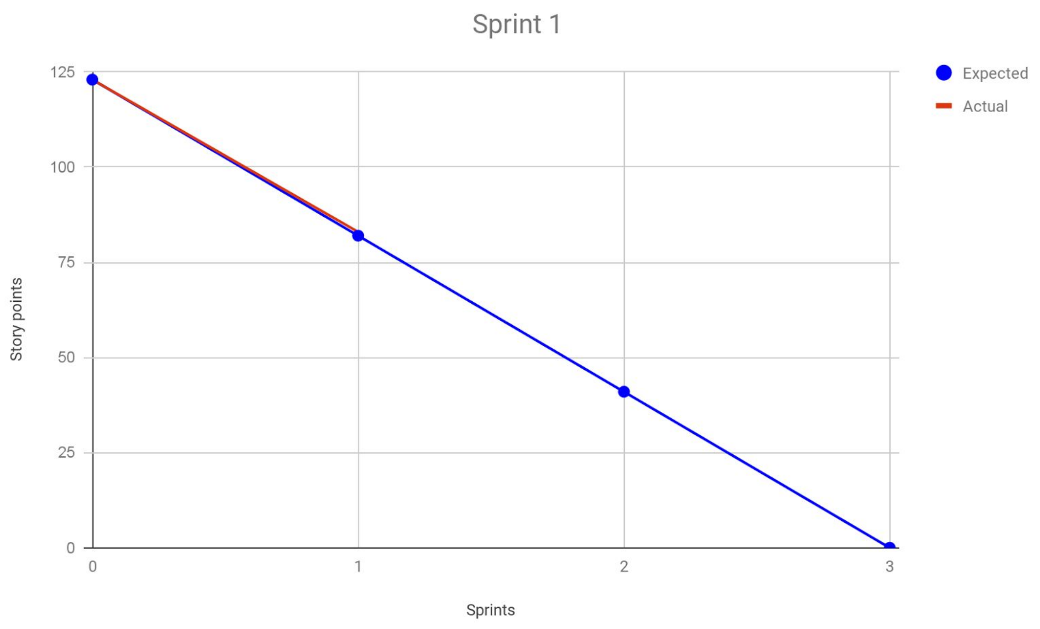
Sprint review

Author: Jeppe Holm

During the sprint review with the PO we showed the demo, we made for sprint 1. The PO had no issues with the demo and we passed the acceptance criteria for **User Story 1** which we had been assigned to do during this sprint.

We told the PO that our velocity was going to stay the same which was 40.
The PO assigned us to work on **User Story: 2, 3, 4, 5 and 6**.

Below is a burndown chart showing the expected and actual work progress. Actual will be updated accordingly at each sprint review. So, to summarise, we did 40 story points during this sprint which makes us nearly even with the expected schedule.



Sprint retrospective

Author: Adnan Unal & Jeppe Holm

To reflect on Sprint 1 a lot of things went well. Communication between members of the team was good and there was always something to do for every member of the team. Every member of the team was focused on the tasks and we finished a proper demo to show to the PO.

An issue, we encountered during this sprint, was that we did not have any sensor that could measure the air pollution described by the PO so we had to do the project without a sensor.

Looking back at the architecture choices we made in sprint 0 we feel that RESTful service was the right choice in this case because it worked for what we needed it to do.

The choice to send a request every minute instead of websockets has proven to be sufficient. The user might be a few seconds behind the newest readings but we did not need to implement a socket server, we saved time and resources for a very similar and sufficient result.

5. Sprint 2

5.1. As a PO

Sprint planning

Author: Marcin Zelent

The development team told us their velocity is 20.
We told them to continue working on **User Story 1:**
As a house owner
I need to see the data collected by the sensors
to see if there has been an intruder.

Acceptance criteria:

- The data of the noise level sensor should be displayed and stored when it reaches over a certain threshold.
- There should also be data for occurrences of motion sensor triggers.

Sprint review

Author: Marcin Zelent

Our developer team worked hard this sprint and managed to finish user story 1, as well as user story 2, thus exceeding their velocity almost twice.

They made the motion sensor work and automatically take a picture with a camera when the sensor was triggered. After that a notification of this occurrence was sent along with the photo to a receiver on a PC using UDP broadcast. Then, the receiver posted the data to the database through Web API and the result appeared on the website. Currently, it is only possible to see the latest reading, however in the future all gathered data should be visible.

We are content with what was delivered by the developer team and surprised that they did more than planned.

For the next sprint, developer team decided their velocity was 40 and we assigned them **user stories: 4, 5, 6, 7, 9.**

5.2. As a developer team

Sprint planning

Author: Jeppe Holm

Velocity: 40.
The PO has decided for us to work on **User Story: 2, 3, 4, 5 and 6**.
See user stories in 3.2.

Kanban Board for Sprint 2

Product backlog	To do	Doing	Test	Done
User story 2	Write HTML to show the acceptable air pollution level in the browser	Adnan / Jeppe		✓
User story 3	Write HTML to show the maximum air pollution level in the browser	Adnan / Jeppe		✓
User story 4	Write query to show data from last week in browser	Marcin		✓
User story 5	Send mail when acceptable air pollution level is exceeded.	Marcin	Marcin	✓
User story 6	Show what triggered the warning (gas).	Marcin	Marcin	✓
User story 9	Calculate AQI and show it on browser	Adnan / Jeppe		✓
	Write documentation	Group effort		~

Sprint execution

Author: Adnan Unal, Marcin Zelent

HTML

For our website backend we had to use PHP. We decided to use the framework Symfony since it provided us with a simple project structure with low cohesion, easy code splitting and refactoring, tools like the templating engine Twig and easy to write unit tests.

For every allowed request route we set up a controller in Symfony to handle the request and provide a response. There we usually make a request to our C# REST API to get the readings data, then we pass this data to our Twig template and render it on the screen.

For example when the user accesses the root route “/” in the controller we make a request to our C# REST API using cURL, afterwards we transform this data into an associative array and pass the associative array as an argument to our Twig template which transforms it into HTML that we send back to the user to see.

Emails

One of the features we had to implement was sending warnings by email about too high air pollution. To achieve this we created a web account at free, anonymous email service, cock.li and we tried to implement email sending in PHP service.

But then, we realized that it is not the right approach, because PHP service works only when user is interacting with it. So, instead we added a class in our ASP.NET web service designed for sending emails.

Every time a new reading is posted to the Web API, a special class is counting the Air Quality Index and if it is bigger or equal 151 (the level of pollution which is considered harmful for health) it is executing an email sender. The email is being sent using SMTP with credentials of the email we created before to the email address of our product owner. Every email contains information about the gas which triggered the warning, calculated AQI and the level of pollution.

Calculating Air Quality Index

The biggest issue with calculating the Air Quality Index (AQI) was figuring out the correct equation. The equation depends on a table of concentration of gases, and breakpoints of Air Quality Indices. The formula is as follows.

Formula for calculating Air Quality Index:

$I_p = [(I_{hi}-I_{low})/(BP_{hi}-BP_{low})] \times (C_p-BP_{low})+I_{low}$

We calculate the Air Quality Index or I_p in the formula for each gas measured and pick the highest value. That will be the final result.

We calculate the Air Quality Index by comparing our gas concentration (C_p) to the breakpoint-concentrations in the table we find the closest concentration value which is lower than our measured concentration I_{low}. Then we find the closest concentration which is higher than our measured concentration I_{hi}. We subtract them and divide them by the difference of the breakpoints corresponding to I_{low} and I_{hi} called BP_{low} and BP_{hi}.
The result of this whole operation is multiplied by the difference of our measured concentration C_p plus I_{low}..

We had some issues with choosing the right data structure to hold our readings and table data in PHP. We also had some issues with data types, we had to manually parse all values to floats because PHP as a dynamically typed language interpreted some values as string or int and it led to calculation errors.

Testing

In this sprint, we focused mainly on testing our website. After some discussion, we decided to use the Symfony framework to help us with testing.

It was necessary to first understand how a website was being tested and then we could implement our own tests. Symfony framework distinguishes between two kinds of tests: Unit tests and functional tests.

Unit tests are used to test your business logic, which should live in classes that are independent of Symfony. For that reason, Symfony does not really have an opinion on what tools you use for unit testing. However, the most popular tools are PhpUnit and PhpSpec.

Creating really good functional tests can be tough so some developers skip these completely. By defining some simple functional tests, we can quickly spot any big errors before we deploy them.

We have also implemented in this sprint the sending of the measured values by email, so it was also necessary to test this functionality. We therefore created another test which sent the user an email with the measured values.

Sprint review

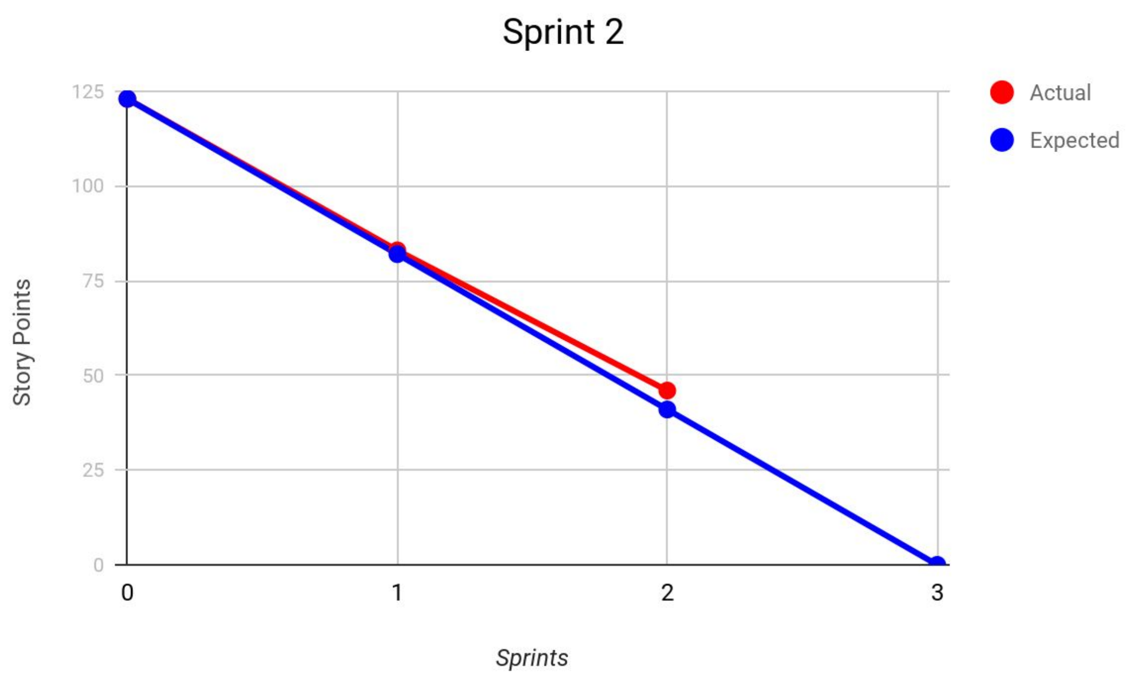
Author: Jeppe Holm

During the sprint review with the PO we showed the demo we had made for sprint 2. As in sprint 1 the PO had no issues with the demo and we passed the acceptance criteria set for the user stories we were assigned to do.

At the end of the sprint review we told the PO that our velocity was 46 story points. The PO assigned us to do **user story: 7 and 8**.

So, to summarise, we did 37 story points during this sprint which makes us fall a little behind the expected schedule.

Below is the burndown chart for sprint 2.



Sprint retrospective

Author: Adnan Unal, Jeppe Holm, Marcin Zelent

Sprint 2 went well, even though it looks like from the burndown chart, that we are little behind schedule we are 100% focused on finishing the program in time for the deadline given by the PO.

Using Symfony instead of another framework or just plain PHP has proven to save us time and effort in the end. We did have some problems with deployment but those were resolved quickly. Symfony provided us with a lot of reusable code, reusable templates and a good development experience.

Sending of emails by web service worked very well and was easy to implement. The only problem was that we wasted some time trying to implement this functionality in PHP web service, but in the end we managed to finish it as planned.

The way we calculated the Air Quality Index (AQI) proved to be correct. The return of the calculation is correct. We tested and compared the results to online calculators and they are nearly identical.

6. Sprint 3

6.1. As a PO

Sprint planning

Author: Marcin Zelent

For this sprint our development team decided their velocity is 40, so we assigned to them **user stories 4, 5, 6, 7 and 9**.

For more details check chapter 3.1.

Sprint review

Author: Marcin Zelent

Once again we met at the end of the sprint with our developer team to review their work in the past week. However, this was the last meeting.

We have been shown the final version of the system. The development team managed to complete all assigned user stories. They added sorting and filtering of data, graph of occurrences of picture taken and ability to delete some records from the database. Although they did not add functionality of setting the active hours, during which the security system is on, they have specified hours when it is dark so the camera will not take a picture on which nothing is visible.

We are generally content with the system they provided. It works well, it is easy to use and covers almost all of the user stories we wanted to have. The only thing that was not properly done is that the user can only see the photo which was taken most recently. We would like to see all the photos ever taken by the system. However, we believe it is not a big issue and can be quickly fixed.

6.2. As a developer team

Sprint planning

Author: Jeppe Holm

Velocity: 46
The PO has decided for us to work on the rest of the user stories specifically User story 7 and 8 see user stories in 3.2.

Product backlog	To do	Doing	Test	Done
User Story 7	Use Rejseplanen service to show train departures	Marcin		✓
User story 8	Write code to create charts	Adnan		✓
	Style page	Adnan		✓

Sprint execution

Author: Adnan Unal, Marcin Zelent, Jeppe Holm, Jiří Vrbas

Rejseplanen API

The biggest issues we encountered during this Sprint was the Rejseplanen API. Even though we did receive it in the end it took them around 10 days to actually send us permission to use the API which meant we received it around Monday the 11th of December. We had already thought about using alternatives such as DSB but when we finally received the API we tried it out and it worked smoothly for what we needed it for.

In our PHP service we send HTTP GET request to Rejseplanen API and receive a file in JSON format with all train departures from Roskilde station. We decode it and then display it as table on a page dedicated for it. Because the PO wants to see a correlation between train departures and pollution level, in the last column of the table, we put an AQI of the reading with time closest to the corresponding train departure.

Drawing charts

To draw charts of the change in gas concentration readings, we used the javascript library called Charts.js. We take the concentration readings for the past week of every gas and draw a multi-line chart (one line for each gas) showing the changes in concentration for the past week. All of this is done client side in javascript so the user needs to have javascript enabled to view the chart.

Testing

In the last sprint, we focused on testing all parts of the system to make sure the whole system works the way it does. In this test, we have identified several bugs in our system. The first bug we found was that the data on train arrivals in Roskilde did not appear correctly.

So it was necessary to review the code and correct these bugs. We also had to fix an algorithm for calculating average pollution per day. After removing these errors, we could be sure that our system works the way the customer requested. By testing most of the system's features, we were always able to show our customers that they meet the requirements for a given sprint.

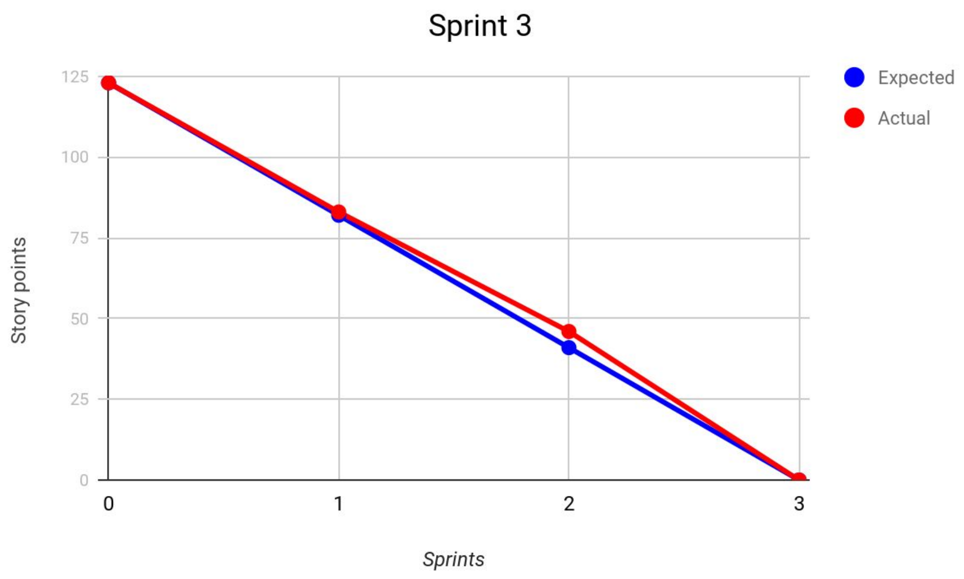
Sprint review

Author: Jeppe Holm

During the sprint review with the PO, we showed the demo, we had made for sprint 3. As in sprint 2 the PO had no issues with the demo and we passed the acceptance criteria set for the user stories we were assigned to do. In other words we managed to finish the project within deadline set by the PO.

To summarise, we did 46 story points during this sprint which made us finish all the requirements set by the PO.

Below is the burndown chart for sprint 3.



Sprint retrospective

Author: Adnan Unal, Jeppe Holm

During Sprint 3 we challenged ourselves by increasing our velocity to try and match the rest of the requirements the PO had set for us to see if we could finish in time. We successfully did so.

The decision to draw the chart client side in javascript was proven to be correct. Using the library drawing the chart was quite easy and fast. The library uses HTML5 canvas to draw charts instead of drawing it in SVG, so the performance is quite fast. If we need different types of charts or more charts in the future, we can use the same library without issue.

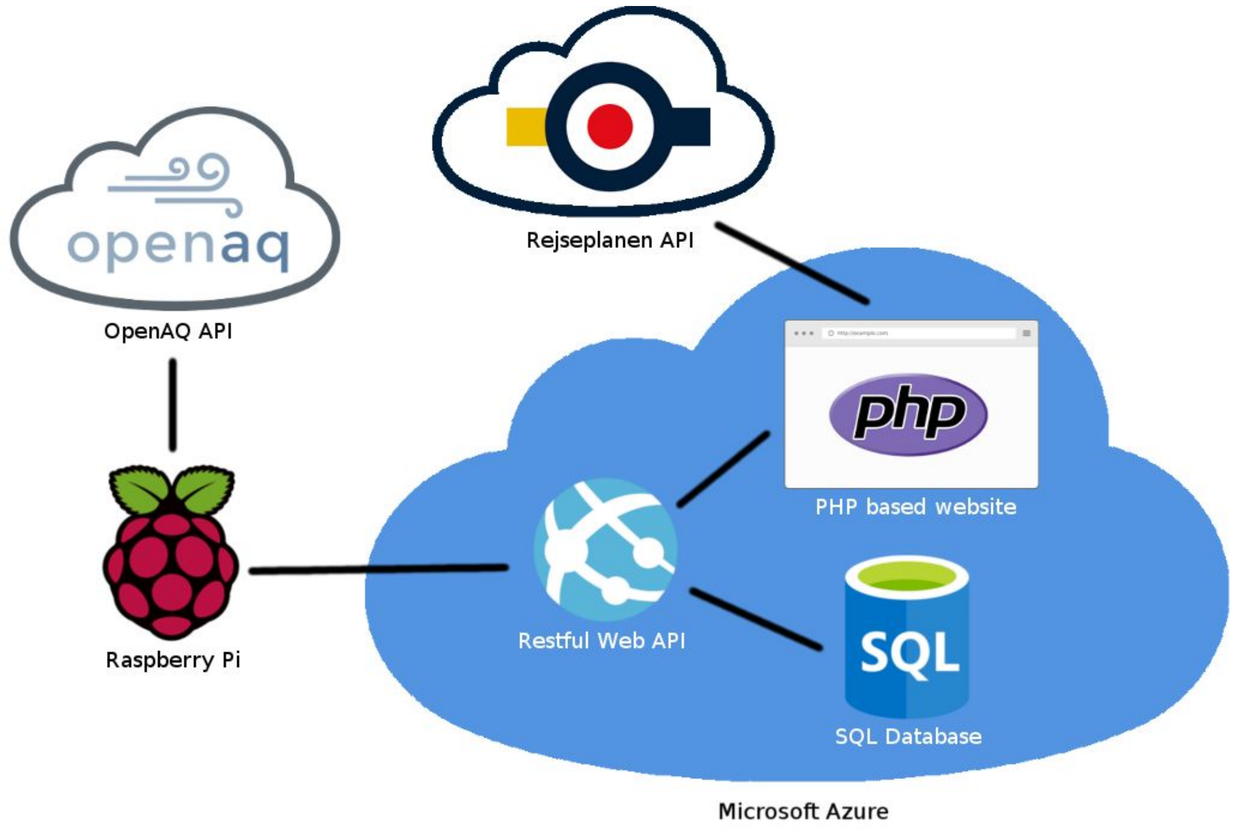
7. Conclusion

Author: Jeppe Holm, Marcin Zelent

Final product

In conclusion the project was a success since we finished the final product for the PO before the deadline and he was satisfied. The final product can be seen at: https://pollutometer.azurewebsites.net/

How everything is connected can be seen on the drawing below of our architecture.



What went well?

During the making of the project a lot of things went well, specifically focus and communication between both the ST and the PO. There were no issues following the work process of scrum neither since sprint planning and sprint review went smoothly.
We had an easier time staying focused because what we had to do was well documented through the tasks made by user stories written on the kanban board.

What did we learn from this project?

This project both made us better at efficiently using agile methodology to solve a problem requested by a customer. At the same time it also got us to work with different things within programming to give us more experience in different languages such as:
Python, C#, HTML, PHP, etc.

The project also gave us experience in working and communicating with a PO and working around their feedback to have a higher possibility of making them satisfied.

What could we have done better?

If everything had worked out i.e. we didn't encounter time delays with Rejseplanen API.
It might have been possible for us to finish before the deadline and maybe then the PO had some more requirements for us that we could do.

We could also have bought a pollution sensor ourselves to be able to actually read the data ourselves instead of using 3rd party sensors.

What would we have done if we had to continue with this project?

Since we finished the project in time it would mean, that either the PO is 100% satisfied with our project and will start selling and using it or he has something extra, he wants us to implement. In other words, all we would be able to do on this project would be to refactor, improve, test and bug fix each component included in the final product. These improvements could include the right side of the Agile Testing Quadrant in other words Q3 and Q4 which contain the product critique factors. These topics would be stuff like usability, security, performance etc.

Amount of Characters: 48019
Amount of Pages in Characters: 20.0079