Design and implementation of a web interface for Axis global testing Live Sites

Aron Söderling & Niklas Grahl

2015

Master's Thesis

Department of Design Sciences
Lund University

Abstract

Designing and developing a software product is a difficult process. The product must be usable and solve the correct problem. At the same time, the underlying code must be well written. Many projects fail to deliver or exceed their budget. This thesis explores a practical approach to software design and development that tries to adhere to both user centered design and agile development. The process follows Google Ventures' design sprint model and also takes inspiration from Jacob Nielsen's discount usability methods. This approach is applied to a real project at Axis Communications.

The goal of the project was to design and implement a web application for monitoring and analyzing data from Axis global weather testing "Live Sites". The data was collected and analyzed manually which was a very time consuming process. It was difficult to interact with the data in order to see correlations between the weather and the camera images. We were able design a solution to this and implement it during four iterations. Each iteration consisted of a design sprint, an implementation sprint and an evaluation phase. The design sprints were fast and effective, which meant we could spend more time on building the actual product while still being confident that we were building something that would actually work. Through continuous usability evaluation and regular stakeholder meetings we were able to validate our design. The project resulted in a web application consisting of a number of interactive dashboards. Our conclusion is that the resulting interface solves the problem of interacting with the "Live Site" data and should provide a good foundation to build upon. We also conclude that Google Ventures' design sprint is a powerful and effective model which could be of great benefit to software development projects.

Acknowledgements

We thank Joakim Eriksson, our supervisor at LTH, for his support and guidance. Martin Kvistholm, for his help, feedback and trust. Niclas Hörnquist and Andreas Olsson for their input and support. The HW Test team at Axis for their feedback and help by taking part in user tests. And finally Axis Communications for letting us do this master thesis project.

Table of Contents

Glossary

affordance Describes how easy it is to discover the possible actions of a system.

back end The part of a system that the user does not interact with directly. The interface between the user and the back end is called the front end.

chartjunk Visual elements in a graphic that do not contribute to the overall understanding of the data presented.

conceptual model The user's mental model of a system. It helps the user understand the inner workings of the system it represents.

CSS Cascading Style Sheets.

data-ink The ink in a graphic that represents data.

data-ink ratio A data visualization term that equals the data-ink divided by the total ink used in the graphic.

DOM Document Object Model.

groupthink Groupthink occurs when a group is discussing possible solutions and ideas without critical evaluation. The effort to avoid controversy within the group leads to loss of individual creativity. The group often overrates its decision-making abilities.

hi-fi high fidelity.

internal locus of control The feeling of being in control of a system.

lie factor A data visualization term that is used to describe the size of effect shown in a graphic, divided by the size of effect in the underlying data.

lo-fi	low fidelity.
mnemonic	Something that is intended to help remembering things.
QA	Quality Assurance.
refactor	Change the existing code without changing its external behavior.
refactoring	See refactor.
signifier	An indicator that signifies critical information and can be interpreted meaningfully. It applies both to indicators in the physical and in the social world.
TeamViewer	A remote control software used for desktop sharing.
tooltip	A graphical user interface element containing information that is usually shown when hovering another element with the cursor.
UCD	User-Centered Design.
UI	user interface.
user interface	The boundary between human and machine in human-machine interaction.

Chapter 1

Introduction

1.1 Background

Axis Communications is a company based in Sweden that provides physical security solutions, primarily in network video surveillance. To make sure Axis' cameras work well they are thoroughly tested by a hardware Quality Assurance (QA) team. Since surveillance cameras are often placed outdoors a large part of this testing consists of weather testing. The weather tests are split into two parts, lab tests and long term tests.

The long term tests are conducted to see how well the cameras cope with real weather conditions over a longer period of time. To achieve this, Axis global weather testing "Live Sites" (from here on referred to simply as Live Sites) are installed around the world at locations with extreme climate.

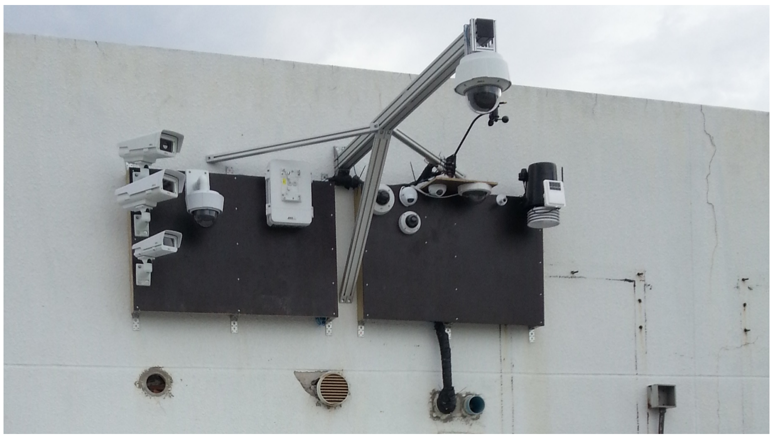


Figure 1.1: Example Live Site

A Live Site consists of a weather station and several Axis cameras (see figure 1.1). Every day the weather station and the cameras collect and store data on a local PC at the site. This data is left there until someone decides to take a look at it. This user goes through the images and the weather data to find correlations between climate patterns and different degradation attributes of the cameras such as condensation and corrosion. This process of working with the data is tedious, repetitive and slow. This is a problem. The Live Site concept is interesting and many different stakeholders at Axis see potential gains and ways of using it, but to unlock the potential the way of interacting with the Live Sites must change.

1.2 Purpose and scope

Our task was to find a good solution of interacting with the Live Sites and implement it as a web interface. To complete this task we applied a design process that we believed would produce well designed software at a fast pace. Since data visualization was a vital part of the problem, this area of design was also be discussed.

The project conducted a full software design process; from initial research to design prototyping to implementation. The main reason for having such a wide scope rather than focusing on a single step in the software development process was to investigate what happens when the entire process is used in practice.

1.3 Related work

In [1] Da Silva et al. presents a framework for combining user experience design and agile development methods. This framework is based on the assumption that UX work is done by one team and development by another. The process model proposed by this framework suggests starting the project with and iteration 0, in which every member of the project team should take part. This iteration 0 is based on the agile concept of “Little Design up Front", meaning that you perform a little design work before implementation begins. When this iteration is concluded the project team divides itself into two teams, the design team and the development team. The development team starts implementing the result from iteration 0, while the design team inspects and improves the design. Usability evaluation is performed by the design team after every implementation sprint. The framework also stresses the importance of continuous communication and feedback between the two teams. Furthermore, da Silva et al. performs a field study from which they conclude that a major issue in commercial projects is that UX designers work on too many projects simultaneously. This leads to the UX designer not being able to collaborate closely with development, nor being able to do little design up front.

In 2008, Fox, Sillito and Maurer [2] conducted a study with ten participants who had previous experience in combining agile methodologies and a user-centered design practices. They noticed three different approaches taken by the participants: the Specialist, the Generalist and the Hybrid approach. The teams that used the Specialist approach consisted of users/customers, developers and UCD specialists. The Generalist approach team lacked a UCD specialist and this responsibility was put on the developers without any formal UCD training. The Hybrid approach describes a team where at least one of the members have both formal UCD training and software development experience.

Blomkvist [3] concludes that the best approach to combine agile development and user centered design to be a balanced integration of both models, a hybrid, instead of trying to pepper one or the other with parts of the other. Blomkvist argues that the integrated model would preserve the core values of both methods The non-integrated way would lack coordination, leading to methods being adapted and undermined to a degree where they become useless. Blomkvist also suggests a set of principles that could help in integrating UCD and agile.

1.4 Design approach

In this thesis we explored a combined process that tries to employ the best of user centered design and agile into a practical software development process. In this process we used Google Venture's design sprint and Jacob Nielsen's discount usability methods.

1.5 Work distribution

The workload has been distributed fairly evenly between the two of us. Almost all design work was done together on one computer or one whiteboard while coding were done separately. The search for the articles and books required for the project was done on separate computers, but studied by both of us. Niklas had more previous experience with web development and implemented some of the trickier parts of the design while Aron spent a bit more time on writing the report and the test cases. When testing, Aron was test leader and Niklas observed the sessions and took notes.

Chapter 2

Theory

The first and main part of this chapter, sections 2.1 to 2.4, revolves around the design and development process. This includes sections on agile development, User-Centered Design (UCD), combining agile and UCD, and Google Ventures' design sprint.

The second part of this chapter, sections 2.5 to 2.7, presents interface design theory that have been used during the project to guide the design of our product. This includes sections on data visualization, usability, and design principles.

2.1 Agile development

Agile development methods are very popular in today's software engineering industry. The more traditional waterfall methodology is generally outclassed by agile. The slow speed and step-by-step nature of the waterfall almost always result in poor user experience[4]. According to usability consultant Jacob Nielsen the reason is simple, requirement specifications are always wrong. Years might pass between the design phase and product delivery, and during this time the users' needs change. Working with a fixed design specification is also an issue. Problems might arise in implementation, which are then resolved by the programmer without any involvement of the design team since the design has already been set in stone[4].

Agile development methods are completely different. The "Agile Manifesto" presents the core of all agile methods[5]:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

This manifesto came to be as a response to the shortcomings of the waterfall model and its overzealous emphasis on process and documentation[6]. The agile manifesto and the methods derived from it are very different to the waterfall model. The methods are lightweight, iterative processes that involves users in the requirements analysis and trusts the knowledge in the team over documents[7]. This means problems are found faster and new features reach users at a much quicker rate[8].

To implement agile in a development team is not always a simple task since there is no definitive version or process of agile that fits any and all. Every team is different and every project is different. Today there are a number of agile methods such as Scrum, Crystal Clear, Extreme programming, and Kanban. Which method that is best suited to a specific team or project, usually comes down to team composition and how the team members prefer to interact with each other and the users.

2.2 User-centered design

User-Centered Design (UCD), sometimes called human-centered design, is a design process where the needs of end users are put in center. The model of designing with the users in center was first outlined, in an article by Gould and Lewis, in 1985[9]. In this article they declared three principles which a design process should follow:

1. Early focus on users and tasks: To solve a design problem, the designer must understand the users. The designer should study the characteristics of the users by observing and interacting with them. The designer must also understand the tasks which the users are expected to perform.

2. Empirical measument: The design should be tested early in the development process by observing users using a prototype to perform tasks.

3. Iterative design: Continuous empirical measurement means that problems will be found throughout the project. These problems must be fixed. This leads to a cyclic process of design.

The term user centered design was originally defined by Norman and Draper in 1986[10]. The concept of UCD has since then been re-defined and disputed several times[11]. In this thesis we used Norman's definition from his latest revision of *The Design of Everyday Things*.

Norman defines UCD as the process of *design thinking*[12]. Norman believes that the correct way to approach a design problem is not to immediately seek the solution, but instead to first discover the correct problem. Solving the wrong problem would not result in something useful, something good. The solution to the wrong problem could in fact be worse than no solution at all. This means that the designer should see the initial problem as a suggestion, and think broadly about it to discover its true source. When the correct problem has been found, work can begin on finding a solution[12].

This process has resulted in the *diamond design process model* (see figure 2.1), first introduced by the British Design Council in 2005[12]. This process divides the design process into two diverge-converge phases, diamonds. It is only by first diverging in many different suggestions and directions, the correct problem and solution can be found. The first diamond consists of the phases *discover* and *define* and the second *develop* and *deliver*. To make this double-diamond model work the designer employs an iterative cycle of UCD[12].

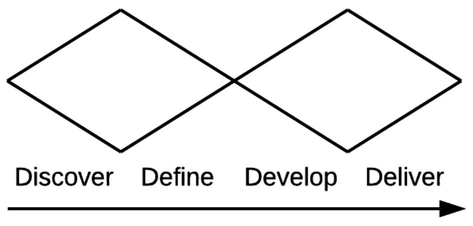


Figure 2.1: Illustration of the double-diamond model

The design process is too based on iterative improvement. Each iteration consists of the following activities:

1. observation
2. idea generation
3. prototyping
4. testing

These activities are then repeated, resulting in more and more insights and eventually a converging solution[12].

2.2.1 Observation

In the first activity, observation, the designer researches the users by watching and asking. The designer must go to the users and observe their behaviors, interests, motives, issues and true needs. It is of utmost importance that it is indeed the true user that is being observed, and not someone trying to imitate the user or someone from the wrong population. Only from observing the users can the designer identify their true needs, the true problem and eventually the correct solution[12].

2.2.2 Idea generation

After observation comes idea generation (ideation). This is the most creative activity in the design process and is usually performed by doing some kind of brainstorming activity. Norman declares three rules that this brainstorming session should follow[12]:

- Generate numerous ideas.
- Be creative without regard for constraints.
- Question everything.

Focusing on too few ideas early in the process is dangerous and limits the project. Only by exploring many ideas can the true problem be discovered. Idea generation without regards for constraints and criticism will result in many crazy ideas. Crazy ideas, even those that seem obviously wrong, are the ideas that lead to creative insights that can later be put together to form excellence. Norman's favorite of these three rules is the last one about which he says:

> *"A stupid question asks about things so fundamental that everyone assumes the answer is obvious. But when the question is taken seriously, it often turns out to be profound: the obvious often is not obvious at all. What we assume to be obvious is simply the way things have always been done, but now that it is questioned, we do not actually know the reasons. Quite often the solution to problems is discovered through stupid questions, through questioning the obvious"[12].*

2.2.3 Prototyping

Prototyping is the process of taking the ideas from idea generation and make them testable. The prototype can be really simple low fidelity (lo-fi) pen and paper creations or more sophisticated high fidelity (hi-fi) mock-ups. As long as the prototype conveys your design in a clear and testable way, it should work[12].

2.2.4 Testing

Testing is crucial in UCD. To make sure the solution truly solves the problem and satisfies the users' needs you need to test it on actual users. The testing should be done in a similar way to observation. You should watch users actually trying to accomplish something with your prototype. This is called task based testing. You should not ask test participants what they think about your design, since they might say it is good out of politeness. Instead you should instruct them to do specific tasks[13].

2.2.5 Constructing a task based test

When constructing a task based test the following hints are useful to keep in mind:

- The tasks should be put into a scenario to make the user more invested. E.g. “What would you do if you needed assistance?” instead of instructing someone to find the help button[14][15]. Giving users tasks they would not normally perform makes the test feel superficial and you would not get the feedback you needed[14].
- To avoid influencing the user, the instructions should give hints and not exact actions to perform[15].
- Do not use words that appear directly in the user interface (UI), as this would give too clear instructions. Use synonyms instead[15].

Conducting a task based test

While your test participant tries to complete the tasks you can ask them to think aloud while you observe and take notes. To keep the participant talking and reflecting you can ask them why they do things and what they are thinking[13]. You should also ask follow-up questions after the user completes a task. These questions should be designed to ease your participant into talking about their actions. They should be simple and not too vague or intimidating. Michael Margolis, user researcher at Google Ventures, lists the following as good questions[15]:

- What is this? What is it for?
- What did you think of that?
- So what happened there?
- Was that what you expected? Why or why not?
- So what goes through your mind as you look at this?
- Did you find what you were looking for?
- What would you do next? Why?
- Is there anything else you would do at this point?
- Is there any other way to do that?
- In what ways would you want this changed to make it better for you?
- What additional information would have helped?

When you have gone through every task the test is complete. But there are still a few things you can learn from your participant. These include reflections, opinions and thoughts. To get this information out of your testing you should end the interview with a few broader debriefing questions such as “What did you like?”, “What did you dislike?”, “How would you make this better?”[15].

2.3 Applying UCD in an agile development process

As stated earlier, agile development brings many improvements to usability and user experience, but there are still problems. A big concern with agile is that it is created by programmers and aimed at addressing programming issues. The fast pace of agile does not make space for the extensive user research of UCD. Interaction design is instead left to the side and only happens as a bi-product of coding[4].

The UCD process works well and creates wonderful user experiences, but it too has its flaws. In agile, the developers want to create working software instead of documentation, but the UCD process wants to thoroughly explore the problem and solution in two diverge-converge diamonds only delivering an implementable design towards the last few iterations. This means that if both methodologies are to work together, both need to adapt.

2.3.1 Discount usability

The rapid pace of development is central in agile. To make sure usability is not neglected in such a process, it must also be fast. Jacob Nielsen has done much work in this area. Already in 1989, long before agile became a thing, Nielsen coined the term *discount usability.* The core idea of discount usability is to make usability fast and cheap so that any project can include it. Discount usability consists of three main components[16]:

- Simplified user testing
- Narrowed-down prototypes
- Heuristic evaluation

Simplified user testing

Nielsen has long been advocating that usability testing should not be such a big deal. In fact testing on no more than five users and running many small tests gives the best results. Nielsen bases this claim on a mathematical equation derived from a study on a large number of projects. This equation says that the number of usability problems found, N, in a test with n users is[17]:

$$N = (1 - (1 - L)^n)$$

Where L represents the proportion of usability problems found while testing on a single user, with typical value 31%, which results in the curve in figure 2.2:

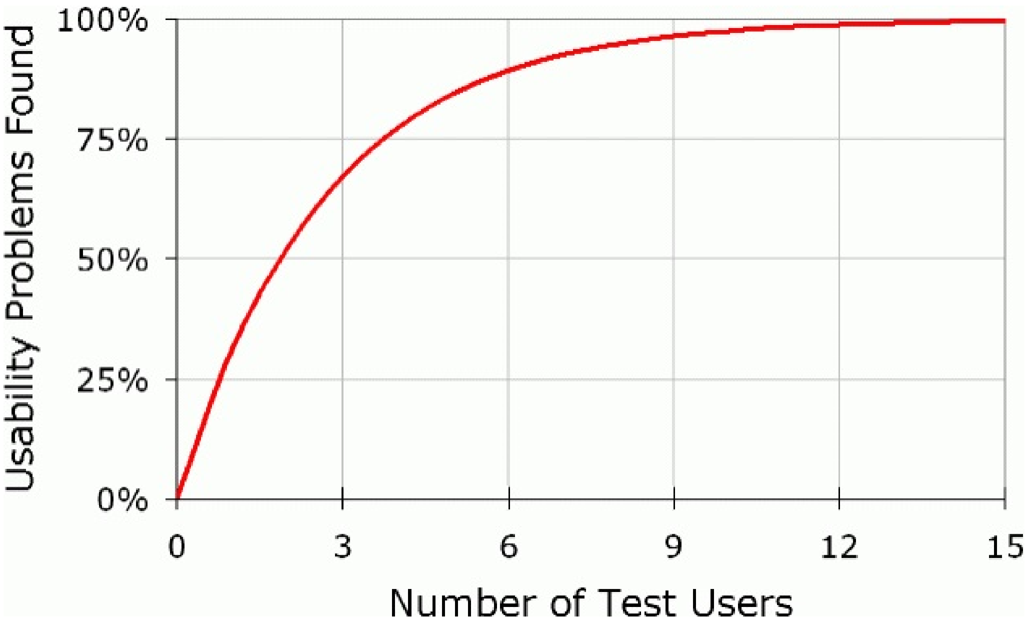


Figure 2.2: The relation between the number of test users and the percentage of usability problems found. Image from [17].

As seen in the curve usability problems found spikes up on lower number of test users until reaching 85 % at 5 users. The number of new problems found then slowly increases and approaches 100% at 15 users. The curve looks like this since many users will have the same problems, hence increasing the number of users will yield redundant problems. Nielsen concludes that 85% is enough to get many new insights and start a new iteration, and testing on 15 users every iteration would be a waste of resources. In the next iteration the designer fixes the problems and tests again on five new users.

Most of the remaining problems are then found while the proposed solutions to the previous problems are tested. Doing additional smaller tests also means that as the prototype evolves the users can help provide deeper insights. The initial design will almost definitely contain usability problems that prevent the user from testing deeper issues such as information architecture and task flow. Every redesign will also lead to new, hopefully smaller, problems that need to be discovered. Usability expert Steve Krug takes this even further and recommends testing with only three users per iteration. He bases this on the argument that three is enough to uncover any major problems and will let you iterate even faster[13].

Preferably testing should be done with users that match your target audience[12][14]. But this does not mean you should not test if you cannot find such users. Krug puts it this way[13]:

> ***Testing one user is 100 percent better than testing none.*** *Testing always works. Even the worst test with the wrong user will show you things you can do that will improve your site.*

Krug also argues that testing early and often is far more important than testing with users that represent your target audience[13].

Narrowed-down prototyping

To quickly produce prototypes that convey your design Nielsen advocates the use of paper prototypes. These prototypes should support a single path through the user interface. This way you can get feedback on that specific interaction. Prototyping a whole application is too complex and would take too much time and effort.

Jake Knapp, designer at Google Ventures, however has a different opinion. Paper prototypes are great for making you think about your design, Knapp says, but these prototypes should not be seen by your testers. Knapp compares paper prototypes with macaroni art, whatever the kid shows you will respond positively. "You are so creative", "Great effort" are things you might say. The same is true for paper prototypes. Testers will think they look good and commend your creativity, but you will not get real feedback[18].

Instead Knapp promotes limiting your paper prototypes to the idea generation part of the design process and instead focus on quick hi-fi prototypes in the prototyping phase. When you show testers something that looks real you will get real feedback. Hi-fi prototyping does not have to take a lot of time. By using simple tools such as Keynote or Microsoft PowerPoint you can create a good enough looking prototype in a few hours[18].

Heuristic evaluation

Heuristic evaluation is a method, introduced by Jacob Nielsen, where a usability expert inspects an interface and evaluates it according to a set of usability principles. The evaluation can be conducted in more or less formal ways. The fastest and easiest way is to let the expert navigate through the interface giving feedback on the fly while you take notes. A more formal way would be to let several experts use the interface on their own, then report and discuss their findings. It is important to note that heuristic evaluation can never replace user testing, but instead work as a complement. Heuristic evaluation is generally faster and easier than user testing, but the feedback you get is not as valuable as that from real users[19].

2.4 Google Ventures' design sprint

Another method that aims to make UCD and design thinking fast is the design sprint developed by Jake Knapp at Google Ventures. This method takes design thinking and combines it with processes from business strategy, innovation and behavior science into a compressed one week package. It is based on the "launch and iterate" model of startups (see figure 2.3) that is popular in Silicon Valley[20].

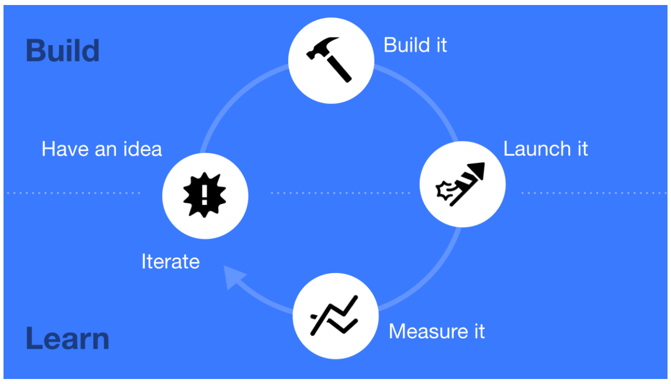


Figure 2.3: The "launch and iterate" model. Image from [21].

Knapp found the "launch and iterate" model too slow and risky. If the initial idea is bad, you will not know until you test it and by that time you might have already spent months building it. This problem is illustrated in figure 2.4.

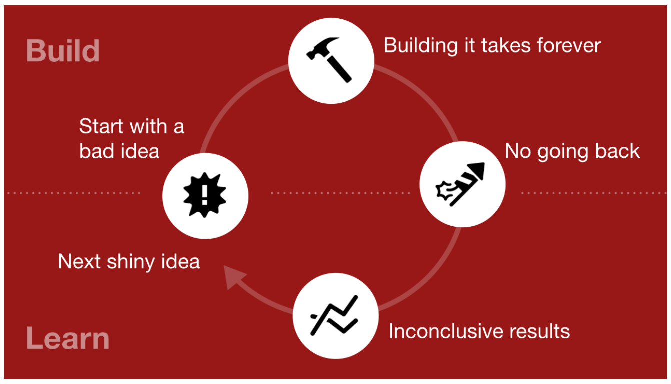


Figure 2.4: The problem with the "launch and iterate" model. Image from [21].

To address this problem the design sprint is instead based on the UCD idea of doing user research before moving on to building the solution, validation before commitment. This way the bad ideas dies fast and you end up saving time and resources.

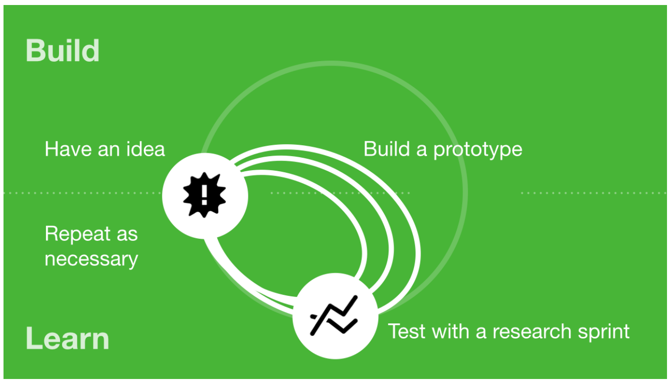


Figure 2.5: The design sprint model. Image from [21].

The design sprint is however not designed to answer everything, it does not feature extensive market or user analysis, but instead answers basic, yet very important questions such as what problems, needs and motivations people have, in a very short amount of time. To do this the design sprint emphasizes individual thinking over groupthink, rapid prototyping over detailed mock-ups, fast decisions over design by committee and testing early over building the wrong thing. This is illustrated in figure 2.5.

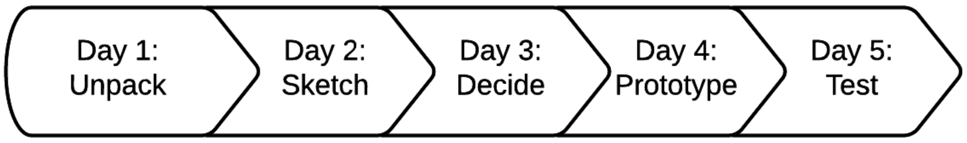


Figure 2.6: The 5 days/stages of the design sprint

Google Ventures' design sprint consists of five stages, each one designed to be completed within one day (see figure 2.6)

2.4.1 Preparation

Before doing a design sprint there are certain things that need to be prepared. Knapp lists the following[22]:

1. **Pick an important idea:** Running a design sprint requires there to be

something to investigate. This could be anything from defining a new product or redesigning an existing one, to improving a detailed feature. As long as it is an important problem the design sprint works.

2. **Gather the team:** The ideal team is between four and eight people, but smaller or bigger team sizes also work as long as the following roles are present:

    (a) Designer: Someone who knows design principles.

    (b) Decider: To make the hard decisions, e.g. the CEO.

    (c) Product manager: The one who will be responsible for implementing the solution.

    (d) User expert: The person who knows the most about your customers.

3. **Schedule the user study:** Setting the deadline even before starting is important to help make the process faster.

4. **Find a facilitator:** Someone has to lead and manage the whole sprint which does require a lot of work, hence this should preferably be someone who is not expected to do any actual design work. Often, it is easiest to put someone from outside the project in this position.

5. **Clear the calendar:** In order to perform the sprint every team member must be completely committed to the sprint for five consecutive days. This will create a sense of urgency which helps in making sure you do not get stuck.

6. **Gather supplies:** To run a design sprint you need certain physical objects. These include things such as sticky notes, drawing pens, whiteboards, stickers, timer and snacks.

2.4.2 Day 1: Unpack

The first day of the design sprint revolves around building a team understanding of the problem at hand. This is very similar to the UCD idea of first exploring the problem, although not as thorough. Knapp does however mention it might be a good idea to do some research ahead of the sprint e.g. by internal interviews at your company or a survey.

The goal of this day is to get everyone in the team to share their knowledge and to reach common ground regarding what the problem is. During the day you should write down thoughts, ideas and anything else relevant on sticky notes and put on the wall/whiteboard. To get the team started Knapp suggests starting at the beginning by going through a set of 10 minute exercises/presentations[23]:

Business opportunity: The business knowledgeable person in the team goes over the business opportunity and market to get the team up to par on why this is worth investigating.

Lightning demos: Take a quick look at competitor products to see how others have solved similar problems. Here it could even be useful to look at products from other markets that solve something resembling your problem.

Success metrics: Discuss how the design's success will be measured.

Existing research: Present earlier user research studies conducted concerning your customers.

Team interviews: Interview other people at your company. Often times there is a lot of knowledge about the problem in different parts of your organization. This could be valuable information you should gather for your sprint.

Analytics: Present what data you have concerning customer behavior such as web site visits and conversion rates.

Basic user story

To follow up on the shared knowledge from the exercises the team should map out the important user story. The facilitator stands at a whiteboard and sketches the flow while the team gives instructions through a discussion. Knowing which user story is important might be difficult and it all depends on what problem you are trying to solve. Knapp gives three different examples in his guide[23]:

- Helping people getting started with your product: Focus on the first encounter experience.
- Creating a new product: Imagine the value proposition and the user going through the product's core features.
- Improving conversion rate from a landing page: Sketch how people land on your page, how they think and what their goals are.

Scope

During the previous discussion you probably uncovered a lot of ideas, but since the sprint is on a tight deadline some ideas must be put on the shelf. Start a discussion on the following questions:

1. What do we hope to learn?
2. What do we need to prototype in order to learn those things?

In this discussion you should practice Norman's principle of asking stupid questions (see section 2.2.2), but remember to keep track of time and interrupt too lengthy discussions.

2.4.3 Day 2: Sketch

Day two is the day of divergence. During this day the goal is to explore as many ideas as possible and generate a lot of possible solutions. To avoid groupthink, this day is centered on individual work instead of group brainstorming. The idea behind this, Knapp says, is that from his experience great ideas come from individuals not groups, hence individuals should be given time to work on their ideas[24].

It is also important to not only focus on new ideas, but to also look back and take those old ideas you never really worked on and give it some more work. Knapp also argues prototyping on paper first is a must. Paper prototyping is fast, you do not get caught in details and everyone can contribute. It is also important to note that even though this is a fast process, you should give individuals enough time to really think through and work on their solutions before you start discussing in the team. To do this effectively Knapp suggests the following a prototyping cycle[24]:

1. **Choose what to work on:** Take the user story from day 1, look at the different steps and divide the story into natural parts and choose which part to work on. Knapp says it is usually best to have everyone in the team working on the same part at once to enhance collaboration at the end of each cycle. If you do choose to split up and work on different parts there is a risk of missing out on things.

2. **Take notes:** Everyone should prepare themselves by quickly writing down anything they deem important from the day 1 notes.

3. **Mind map:** Look through your notes and add in your own ideas. Try to organize everything so you can find the connections.

4. **Crazy eights:** This is a fast sketching technique that aims to get you thinking about the whole interaction flow. Take a paper and fold it in half four times, and then unfold it to get an eight panel paper. You then set the timer for five minutes and during those five minutes, sketch one idea per panel. This leads to you to think really quickly about your ideas and get every variation on paper fast. Keep in mind these sketches will not be shared so they do not need to be pretty. Knapp suggests doing two sets of crazy eights since it can be a bit tricky to get the feel and timing right the first time.

5. **Storyboard:** A storyboard is a sequence of illustrations with the purpose of visualizing the interaction. To be able to explain your best idea to the team you should now create such a storyboard. This storyboard will later be shared and critiqued anonymously by the team. To create the storyboard look back at your mind map and crazy eights and choose what you think is your best idea. Knapp gives three rules to make your storyboard better:

    - Make it stand alone: Your sketch should be self-explanatory. The other team members will be looking at your storyboard later and you will not be there to explain it, hence it has to make sense. To make sure it does, write down the story, where the users click, what info they enter, how they think, etcetera.
    - Keep it anonymous: To make sure the team does not get distracted and influenced by who made what, everything should be without names.
    - Give it a name: Having names for the storyboards helps a lot when you open up for discussions and comparisons.

6. **Silent critique:** Put up all the storyboards up on the wall and let everyone in the team look at each one for a minute or two and put votes on the ideas they like. Voting is done through the use of dot stickers. There is no limits or regulations on how this voting is done, anyone can use however many votes they please. Once everyone has voted there will be a kind of heat map on your storyboards and some ideas will already be starting to stand out.

7. **3-minute critiques:** Up until this point everything has been done individually, but here the process opens up for discussion. Every storyboard is given three minutes of discussion where everyone in the team will say what they liked about it. The person who drew the sketch will then be asked if anything was missed and to fill in the gaps. Knapp argues that letting everyone present their own storyboard instead almost always uses up a lot more time.

8. **Super vote:** This step is designed to make sure you avoid design-by-committee. Here the members of the group are given “special” stickers. These special stickers are not given out equally, but instead in regards to the individual’s decision making level. For instance the CEO might be given double or triple amount of votes. This makes sure the people in charge truly support the winning idea.

Repeat

After going through this cycle you have two options: repeat the cycle or cement focus on the section you worked on before and move on. If you choose to repeat you go back to step 1 and choose a new part to work on. From experience, Knapp notes, he will often realize at this point that the scope of the sprint was too big and the team should focus on the same section. Either way you should take some time to discuss and choose where to go next. Normally a team is able to do this cycle two or three times in a day[24].

Recruit participants

During day 2 you also need to start thinking about your upcoming user study. Depending on what you are designing there are different approaches you need to take to get the right users. Preferably you should try to get users that represent your target audience. Send out a recruitment screener in a forum where you are likely to find many of these target users. Getting people to actually sign up can be a challenge. Margolis suggests a good way to solve this is to include a small gift card for participating in the study[25].

2.4.4 Day 3: Decide

This day covers the converge model of design thinking. Day two leaves you with lots of different ideas, which is good, but there is no way you can evaluate each and every one. Here you decide which of these ideas you are going to prototype on day four. Making these decisions is no simple task. The group effect can lead to a democratic feeling that does not exist outside the sprint. After leaving the sprint you could end up with a design the people in charge do not support, which, as stated under day two, is not good. Super voting could be used here as well, but in the end there is no easy way around it. The decision makers has to make the call[26].

Conflicts

To start the decision making process Knapp recommends that you search for conflicting solutions, ideas that solve the same problem in different ways. The conflicts, says Knapp, are like little gold mines and a big advantage of running this design sprint. Instead of a designer choosing an approach these conflicts makes you think twice before moving on.

The next step in the process builds on the conflicts you uncovered earlier, you must now decide what kind of user study you will conduct on day five. Here you can choose two different paths to follow, *“best shot”* or *“battle royale”*[26].

- **Best shot.** Choosing the best shot option means you pick one solution to prototype and test. This lets you put more effort into making the prototype which opens up for even better feedback from your testers. Testing only one solution also means the user study will be fairly simple, which allows you to explore other areas during the tests such as asking about competitors or general interviewing.

- **Battle royale.** Best shot only works if you really only have one good solution, which is not always the case. Battle royale instead lets you pick two or more conflicting solutions. Doing this means you will have less time to spend on crafting each prototype and the user study will be more complex, but the results might be worth the extra effort. Results of battle royales can often be surprising, Knapp says, reminiscing about earlier experiences where the long shot design have turned out to be the users' preferred solution.

Assumptions

Testing your prototype is not the only thing you should test during your user study. Testing your assumptions is equally important. At this point it is good to list all your assumptions and how you can test them. These assumptions can be anything that concerns your users, business or technology. Thinking about your assumptions is good argues Knapp, it takes you back to reality. Not all assumptions can be tested in a user study you should try to find other ways to test these in parallel. One example of this is where you ask the engineers to spend a few hours hacking away at an algorithm that is assumed to be possible to implement efficiently[26].

Detailed storyboard

Having selected which kind of user study to run and which assumptions you need to test, you are ready to move forward. Now you should make a detailed storyboard on a whiteboard that shows precisely how the user will progress through your prototype, frame by frame. This storyboard is going to be your specification when making your prototype on day four. The whole team should work together making this storyboard. This will lead to more discussions and conflicts since you are adding more detail to your solution. It is important to keep the undemocratic decision process. It is better to open up for a "battle royale" than to try and find some kind of compromise[26].

Start planning interviews

Here you also need to start preparing for your interviews. You should select suitable users from the recruitment screener you posted earlier and check back with them and try to work out a schedule. You should also think about what you should say to your testers when you interview them on day five.

First write up an introduction that introduces the user to the test. You might want to mention things like why you are doing this test, that you will be using a prototype, what type of questions you will ask, etcetera. This will help the interviewee feel comfortable[27].

Second you should think of some context questions you will ask before jumping into the actual test. This will further enhance the participant's probability to give honest feedback and reactions. These questions should be somewhat related to the context of your product, but still on a general level, such as asking about someone's work, interests or habits[27].

2.4.5 Day 4: Prototype

The next challenge of the sprint is to build your prototype. To get good results from in the upcoming study this prototype must look and feel like a real product, it must suspend disbelief. Alas, you need to remember you only have the one day to do this, and that is 8 hours not 24. This might sound, as Knapp puts it, crazy, but this is by design. The deadline will make sure you keep it simple and do not spend time trying to perfect the parts of your prototype that do not affect your user study[28].

Tools

To achieve this seemingly impossible feat, Knapp suggests using a slideshow tool such as Keynote or PowerPoint. These tools are fast and easy. You can make things look pretty good with hyperlinks, transitions and animations. At the same time it is impossible to make things look perfect, which means not getting stuck in unimportant details. The storyboard also naturally translates into a slideshow and makes it easy to divide the workload[28].

Process

Even with the right tool, making a realistic prototype is still a big challenge. You have to be organized and make sure you stay productive and effective throughout the day. To do this Knapp lists a few pointers[28]:

- Divide and Conquer: Put everyone in the team to work. Pretty much anyone can contribute when working in a slideshow tool. Assign different parts of the storyboard from day three to every team member. This will also require you to assign someone to be responsible for putting it all together at the end, a “stitcher” as Knapp calls it.

- Build an asset library: Creating a template slide at the beginning with included standard assets such as menus and logos helps the team work consistently.
- Use a timer to maintain focus: It is important to keep everyone productive. One way to achieve this is by using a timer and take collective breaks. This time-slotting will reduce the work into smaller achievements and help deflect the temptation to open up your favorite news site or email.
- Appoint an email sheriff: Another way to stop the email allure is to appoint a sheriff who will out anyone who starts time wasting on the internet to the group.
- Lightning critique: When several team members work in parallel it is important to make sure everyone is on the same page. Knapp recommends taking a break midday to critique each other’s work. It is important to note that these critiques should be lightning fast, Knapp recommends setting the timer to 5 minutes per design.
- Review with an outsider: Having someone who is not working on the design come in and have a look helps making sure that you do not adhere to bad ideas.
- Check for consistency: Changing details and texts such as names is very distracting in a prototype, make sure you get everything consistent. Also make sure you use real text and names as this helps giving the impression of realism.

Prepare interviews

When you have a complete prototype you should review it and decide what tomorrow’s tests will look like. Finish the interview guide you started on during day 3 by adding tasks and follow up questions. How to construct a task based test is described in section 2.2.5.

2.4.6 Day 5: Test

You should start this day by making sure your setup works. One person will be conducting the interviews and the rest of the team will be observing. You can either observe by sitting behind the test participant or from another room using video conference software. Prepare your minds by going over how the interviews will be conducted and review your conflicts and assumptions from day 3[29].

Observation

During the interviews your whole team should be active in observing. To get the most out of this Knapp lists a few pointers[29]:

- **Do not diss the user:** Remember that your participants are not the ones being tested, your design is. If the users fail at something it is not their fault, it is your design that is flawed.
- **Everybody should take notes:** Everyone in the team should take notes of their observations and thoughts during the test. Knapp forbids the use of laptops for the note-taking, lest you be tempted to check your emails.
- **Designate a court reporter:** It is important to have a quick way of skimming over the interviews again, and going over video recordings will take a long time. You can solve this by assigning one person to transcript the entire interview. You should alternate this role throughout the day since it is a quite demanding task.
- **Make a scoreboard:** To make it easier when you review everything at the end of the day you should prepare a scoreboard. This scoreboard should consist of a column for every interviewee and a row for each part of the interview. At the end of each interview you should add the highlights from the team notes, here Knapp recommends the use of color coding. Using green for good and red for bad will help you spot patterns easily.

Analysis

When all the interviews are done you should take some time and look over your notes and the scoreboard. There is a pretty big chance you already started to see some patterns during the last interviews, but you must make sure not to miss anything. You also need to prepare for what comes next. Write a list of everything that worked in your prototype and another with the problems that need to be solved. With these two lists complete the sprint is concluded[29].

Moving on

At this moment you need to evaluate the outcome of the sprint and decide what to do next. According to Knapp most sprints can be put in one of three categories[29]:

1. **Most stuff worked:** This outcome usually only happens after a few sprints. Since most of your prototype worked you probably only need to make small adjustments before testing again. You should start your next sprint from Day 3: Decide.

2. **Some big questions:** This is the most normal outcome, a mixed result. Some things worked, some did not. This means you need to rethink the solutions, but also that you have a pretty clear understanding of the problem. Start the next sprint from Day 2: Sketch.

3. **Everything exploded:** It might also be the case that almost nothing in the prototype worked. This is actually great since it means you did not spend months building something that does not work, and it gives you the opportunity to start over with lots of new knowledge. Start the next sprint from Day 1: Understand.

2.5 Data visualization

Data visualization is the creation and study of the visual representation of data, and borders the fields of statistics and design. Its goal is to communicate information in the most effective and clear way possible using statistical graphics e.g. tables and charts.

The importance of good data visualization is growing with the increasing rate of data generation and propagation we see today. The term “Big Data”, used to describe data sets too large and complex for traditional data visualization software, presents new challenges. In order to be able to process, analyze and communicate all this data in a meaningful way, the field of data visualization is put to the test.

2.5.1 General guidelines

One of the big names in data visualization is Edward R. Tufte and in his book “The Visual Display of Quantitative Information”, he states that in order to achieve graphical excellence, graphical displays should[30]:

- show the data
- induce the viewer to think about the substance rather than about methodology, graphic design, the technology of graphic production, or something else
- avoid distorting what the data have to say
- present a lot of numbers in a small space
- make large data sets coherent
- encourage the eye to compare different pieces of data

- reveal the data at several levels of detail, from a broad overview to the fine structure.
- serve a reasonably clear purpose: description, exploration, tabulation, or decoration
- be closely integrated with the statistical and verbal descriptions of a data set

Above all else, great graphical displays tells the truth about the data. To achieve this Tufte lists the following principles[30]:

The representation of numbers on surface needs to be proportional to numerical quantities. This first principle can be measured using the term "lie factor"[30].

$$\text{Lie factor} = \frac{\text{Size of effect shown in the graphic}}{\text{Size of effect in the data}}$$

This means the optimal lie factor value is 1 and the further away you are from this value, the more dishonest your data representation is.

Use clear and detailed labeling. Clear and detailed labeling should be used to reduce graphical distortion and ambiguity. Explanations should be put in the graphic itself, especially for the important events in the data.

Show data variation, not design variation. A variation in design suggests variation in the data and deceives the user. If a scale uses regular intervals, it is expected to continue in a consistent fashion towards the edges of the scale. If this is not the case, the graphic shows a skewed version of the data measures. Decoration should be avoided where it might distort the data measures to enable the data to speak for itself[30].

It is almost always better to display money using deflated, standardized units rather than nominal units. If you do not account for inflation and population growth when plotting money, the chart easily gets biased and communicates a misleading message[30].

Use the same or less amount of dimensions in the graphic than in the data. Using more dimensions in the graphic than in the underlying data is never a good idea since it exaggerates the data variation. This connects to the first principle that states that the representation of numbers needs to be proportional to numerical quantities. One common error that falls into this category is using areas to show one-dimensional data[30].

Do not quote data out of context. The context of the data is essential and something graphics should always provide. Enable comparisons with adjacent states by adding the preceding and following data points to the plot.

Great designed graphical displays give the user the greatest number of ideas in the least amount of time. Through clarity, precision and efficiency it brings understanding to complex ideas[30].

2.5.2 Data-ink and non-data-ink

Tufte also introduces the notion of data-ink. A graphical representation of data is made up of two types of ink: data-ink and non-data-ink. Data-ink represents the actual data in the graphic and any other ink included in the graphic is non-data-ink. To create good graphical displays of data Tufte lists the following set of rules regarding data-ink[30]:

Maximize the data-ink ratio, within reason. Strive to have every piece of ink in a graphic represent information.

Erase non-data-ink, within reason. Any ink that does not represent information should be removed to avoid cluttering the data.

Erase redundant data-ink, within reason. Multiple pieces of ink representing the same information will clutter the graphic.

Chartjunk

Chartjunk is a term originally coined by Edward Tufte in his 1983 book, “The Visual Display of Quantitative Information”. This term refers to non-data-ink that is added to a graphic with only the purpose of decoration. Tufte gives three examples of chartjunk:

Unintentional optical art: The use of patterns can cause distracting moiré effects, effects that makes the pattern appear to be vibrating or moving.

Grids: The grid is a common artifact in graphics. A grid takes up a lot of ink without actually saying much. Sometimes a grid can be useful, but it should be muted enough to not distract or compete with the data-ink.

Graphical ducks: This is the most extreme case of chartjunk. A graphical duck is Tufte’s name for art that is added to a graphic with the single purpose of decoration, e.g. unnecessary 3D effects.

2.5.3 Small multiples

Small multiples is when the same data is repeated in slices with small or no variation in the graphic design (see figure 2.7). They allow the viewer to focus on the changes in the data instead of changes in design.

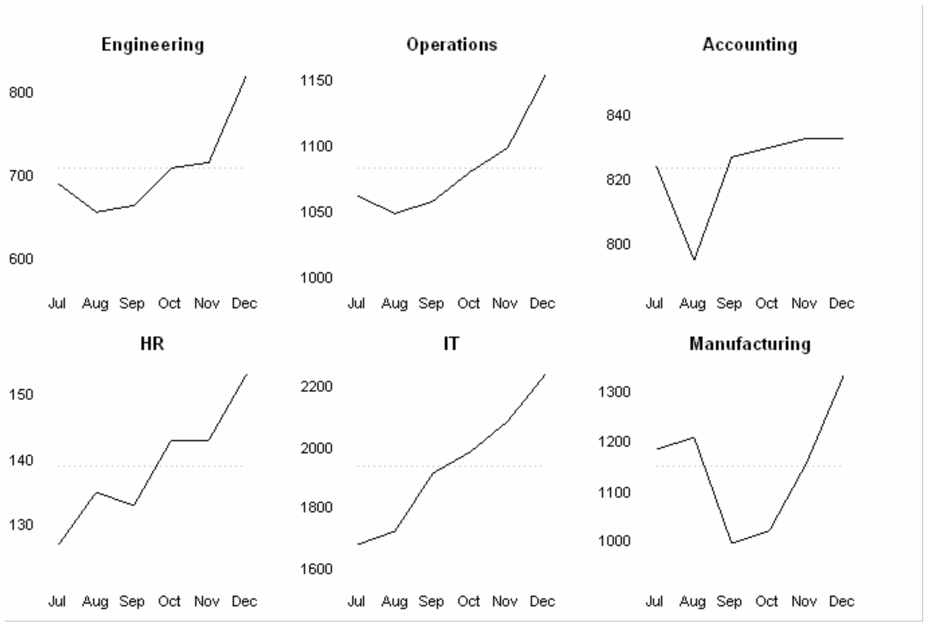


Figure 2.7: Salary expenses visualized using small multiples.

Well-designed small multiples are: *a*) inevitably comparative, *b*) deftly multivariate, *c*) shrunken, high-density graphics, *d*) usually based on a large data matrix, *e*) drawn almost entirely with data-ink, *f*) efficient in interpretation, *g*) often narrative in content, showing shifts in the relationship between variables as the index variable changes (thereby revealing interaction or multiplicative effects) [30].

2.5.4 Sparklines

Sparklines is another term coined by Tufte in his book "Beautiful Evidence" from 2006[31]. Sparklines are condensed graphics that can be used anywhere a word could be used, e.g. in sentences, tables or spreadsheets (see figure 2.8 and 2.9). Where other graphics are typically limited to separate figures with captions, sparkline graphics are meant to be placed together with their respective data to provide context and show overall trends in the data.

Sparklines are naturally applicable to time-series. Because of the small size of the sparklines, all non-data ink is removed such as axes and grids. This makes

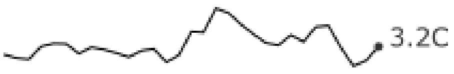


Figure 2.8: A sparkline showing temperature changes over time.

them less ideal for reading the exact value of the data.

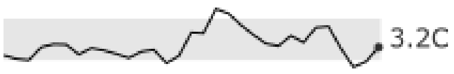


Figure 2.9: To provide additional context and help the reader to find anomalies and areas of interest in the data quickly, the normal range can be shown as a gray box.

Achieving a good aspect ratio in a sparkline makes a big difference. They are generally greater in width than in height. A rule of thumb is to try and keep the hills and slopes averaging at 45 degrees. This yields a result that looks *lumpy*, rather than *spiky* (see figure 2.10).

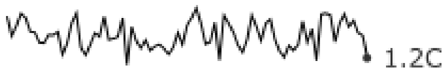


Figure 2.10: A lumpy and spiky sparkline is hard to read.

2.5.5 Micro/macro readings

In his second book, "Envisioning Information", Tufte further explores the qualities of good data visualization design. One concept introduced in this book is micro/macro readings. The idea is simple. Allow one graphic to show both high level structure and intricate detail at the same time. By having both high level and detail in the same graphic the reader can avoid context switching which would interrupt information reasoning. In a computer interface this graphic could even be interactive, allowing the reader to control the data. Tufte does however raise concern over computer interfaces that constantly forces context switching through menus, pop-ups and other distractions. The reader should not have to jump back and forth between graph and menu, but should be able to control the graph directly.

Further on Tufte continues by arguing that having too little detail in the data in fact damages the clarity of the data. Too thin data does not induce the reader to reason about it, since there would be nothing to reason about. To understand the complex world one needs complex data. Confusion does not come from data complexity, Tufte says, but from failure of design. The design should not be dumbed down, but instead help the reader by making it easy to do comparisons[30].

2.5.6 Pre-attentive processing

Pre-attentive processing occurs on a subconscious level and is adapted to detect specific visual attributes. It gives certain objects a “pop out” effect and makes finding them among other objects much easier. A demonstration of this effect can be seen in figure 2.11.

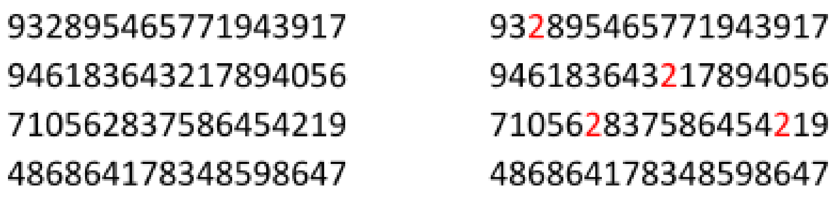


Figure 2.11: Pre-attentive processing makes it far easier to find the twos in the right image compared to the left image because of the color difference

Colin Ware divides the pre-attentively processed features into the following categories: line orientation, line length, line width, size, curvature, spatial grouping, blur, added marks, numerosity, color, hue, motion, blinking, spatial position and convex/concave shape from shading[32].

The effects vary in strength. Color, orientation, size, contrast, motion and blinking are in general stronger than e.g. curvature. The “pop out” effect does however not solely depend on the strength of the attribute, but also the contrast of the element in its context.

Color

Human color vision has been a great evolutionary advantage that has helped us break camouflage and judge material properties of objects. This helped us in life-or-death decisions such as finding and determining the condition of our food and aiding us when making tools. Color blindness might have been a big disadvantage in our history but in today’s society, where many of these decisions are no longer dependent on our color vision, it might go unnoticed until very late in life.

Color is however extremely useful in data visualization, and knowledge about the strengths and limitations of our color perception helps us design usable user interfaces. This does not only mean we should avoid colors that are difficult to distinguish for people with color blindness, but also know where, when and how to use them in the best way.

Human perception of color is heavily influenced by the surrounding context. A gray object on a black background appears lighter than the same gray object on a white background. It is not limited to the intensity but applies to hue as well. This means the context must be taken into consideration when we want colors to appear the same and when we want them to appear different.

Colin Ware mentions the following guidelines regarding color use in data visualization in his book “Information Visualization”[32]:

- Use color for categorization and labeling.
- When coloring larger areas, less saturated colors should be used. To be able to distinguish smaller or thinner areas, more saturated colors need to be used.
- Do not use colors with the same luminance as the background since chromatic difference is not enough to easily distinguish small shapes.
- Larger areas with equiluminous colors can be seen more clearly with a thin dividing line with a large luminance difference.
- Color code small symbols using unique hues (e.g. red, green, yellow and blue).
- Use symbol colors with enough variation in the yellow-blue direction since these can be distinguished by most color blind people.
- Use no more than five to ten colors to code symbols where dependable identification is necessary.
- Larger areas should use low-saturation colors.

Form

Form encompasses a lot of the pre-attentive attributes. Line orientation, line length, line width, size, curvature, added marks and the effects from shading all fall into this category.

Line length is normally used in dashboards for encoding quantitative data, e.g. in bar graphs and line graphs. The thickness or width of the line is however not as often used for quantitative data but more for highlighting. Line orientation is not as commonly used.

Marks are often added for highlighting purposes.

Position

The position of a data point is the most common way to encode quantitative data in a graph. The position of the graph itself indicates importance. A graph in the center and upper left parts of a user interface is ranked as more important.

Motion and blinking

Motion and blinking are powerful attention-getters and some of the trickier data visualization attributes to get right, since large amounts of motion and blinking easily gets overstimulating and annoying. One of the most common occurrences is the flickering of the cursor when you are typing text, helping you find where you are currently typing. It can be used to highlight extremely important information or indicate rare events.

2.5.7 Dashboard

The term dashboard has been defined on several occasions, but in our report we stick with Stephen Few's definition originally published in an article in the magazine "Intelligent Enterprise" in 2004[33]:

> Visual display of the most important information needed to achieve one or more objectives which fits entirely on a single computer screen so it can be monitored at a glance

To that definition he later added the following supporting attributes: "Dashboards have small, concise, clear and intuitive display mechanisms" and "Dashboards are customized".

When used correctly, dashboards have the potential to identify trends, patterns and anomalies in large amounts of underlying data by effectively using the visual capabilities of the human perception. They help the user reason about the current situation and guides future decisions.

History

Dashboards were around as early as the 1980s, but known under a different name, Executive Information Systems (EISs). EISs were a bit ahead of its time and enabled to evolve with the help of the Key Performance Indicators (KPIs) of the 1990s. The demand for dashboards increased after the Enron Scandal in 2001 that forced companies to look for ways to help the managers to monitor what was going on within the company[34].

Principles

Well designed dashboards are well organized, condensed, customized to the target audience and their objectives, and offers a clear communication of data using small concise media[35].

Common mistakes

When designing dashboards, it is of great importance to make sure it communicates its data clearly and effectively. Stephen Few explains in his book “Information Dashboard Design” the problems with most dashboards to date. One of the fundamental points he makes is to apply design principles and practices that play to the strengths of our visual perception instead of focusing of flashy designs. This might seem obvious, but we still see a lot of dashboards that are more about flash than effectively communicating information.

Few lists the following thirteen common mistakes in dashboard design and explains why they do not play to our strengths[35].

Exceeding the boundaries of a single screen. Since humans can only hold a small amount of information in short-term memory, it is important that all necessary information is within eye span. When separating information into different screens or needing to scroll to access it we risk losing critical information.

Supplying inadequate context for the data. In order to make sense of the data presented, we need to put them in a relevant context. We cannot easily find out if a certain sales total is good or bad if we do not offer an easy comparison with e.g. previous years’ sales total or a forecast.

Displaying excessive detail or precision. Displaying too much detail or precision can be just as bad as supplying inadequate information. Using several decimal digits might be necessary in certain contexts but on a dashboard where the goal is to get a fast understanding of the current state and too much precision slows the user down.

Choosing a deficient measure. If is important to choose the measure that most effectively communicates the intended message to the user. Choosing a deficient measure makes it more difficult for the user to get the desired understanding since they might need to calculate a more efficient measure manually.

Choosing inappropriate display media. The choice of display media (bar chart, line graph, table etcetera) is very important and needs to be made so it plays to our visual perception strengths. This is considered by Few to be one of the most common mistakes in data visualization. The pie chart is often put in this category since it seldom communicates its measure effectively. This because pie charts do not display quantitative information in a way that is easily analyzed and compared by the human mind.

Introducing meaningless variety. Using different display mediums the items on the dashboard for the sake of variety is not a good idea. The best strategy is to use the most efficient display medium for each item on the dashboard. This consistency helps users since they can use the same strategy for interpreting the data.

Using poorly designed display media. The components of the display medium must be designed so it communicates its data effectively. We need to consider ordering and avoid the need for unnecessary eye movements, overuse of bright colors, visually prominent grid lines, 3-D effects etcetera.

Encoding quantitative data inaccurately. Displaying quantitative data in a way that distorts the communicated value is obviously not wanted in a dashboard. One example of this is when a bar chart is used and its value axis does not start at zero. The comparison between bars suggests there are greater differences between the measures than what is supported by the data.

Arranging the data poorly. In order to avoid a design that feels cluttered and improvised, the content needs to be organized properly. The content should be divided into meaningful groups and the data of greatest importance should appear in a prominent place.

Highlighting important data ineffectively or not at all. The dashboard should immediately attract attention to the most important information on the dashboard. As stated previously, this can be achieved by putting the most important data in the more prominent areas of the screen. But even when this is not the case, the important information can be made visually prominent using e.g. color intensity or size. Avoid making all or only the less important data visually prominent.

Cluttering the display with useless decoration. Commercial dashboards are often filled with useless decoration, which distracts the user and makes it more difficult to read and analyze the data. Decoration should be kept to a minimum and not added thoughtlessly.

Misusing or overusing color. The use of color in a dashboard should not be made without an understanding of how color is perceived (see 2.5.6).

Designing an unattractive visual display. The user experience will be reduced if the display is visually unattractive. Often, by striving for simplicity, an unattractive dashboard can be avoided.

2.6 Usability

In order to design a usable user interface, we need to define what we mean with the term usability. We will use the most well accepted and wide spread definition of usability, defined in ISO 9241-11[36]:

> The effectiveness, efficiency and satisfaction with which specified users achieve specified goals in particular environments.

Effectiveness describes if the user is able to complete the tasks and goals with the product.

Efficiency describes how much effort the user needs to put into the task in order to complete it.

Satisfaction describes the comfort and subjective feeling of the system that is transmitted to the users and other people affected by its use.

This means we have to figure out who our users are, what their goals are and in what environment the product will be used. Are they expert users or novices? What are they trying to achieve with the product? Where is the product used and how?

2.6.1 User experience

User Experience (UX) emerged as a reaction to what some saw as a narrow mindset of the term usability. It extends the field of usability to include a more comprehensive perspective of a user's emotions, behaviors and attitudes towards a product. It encompasses the feelings a user gets when using a product and focuses on the human interaction the product enables and invites to. This helps us create products that meet the needs of the customer and at the same time are more than only what the users say they want[37].

UX can be considered more subjective than usability. It is enabled by the merging of multiple disciplines. UX can however, be a slippery term since there have been many attempts to properly define it and the definition is still evolving.

In ISO 9241-210 it is defined as[38]:

> a person's perceptions and responses resulting from the use and/or anticipated use of a product, system or service

Jakob Nielsen and Donald Norman summarizes as[37]:

> "User experience" encompasses all aspects of the end-user's interaction with the company, its services, and its products.

Experience Design or User Experience Design (UXD) is the practice of achieving good user experience.

2.7 Design principles

This section presents a list of recognized design principles that we used in the heuristic evaluations of our design. They also influenced the decisions we made in the design sprints.

2.7.1 Donald Norman's design principles

In Donald Norman's book The Design of Everyday Things he lists 7 fundamental principles of design[12]. They are not specific for human-computer interfaces, but also apply to other everyday items, and are as follows.

Discoverability: The first principle states it should be made easy to the user to establish what the possible actions are and what state the device is currently in.

Feedback: It should be made clear the user when an action has been made and what the results of said action were. The new state of the device should be easily discovered by the user.

Conceptual model: The design must provide the necessary information for the user to be able to create a good conceptual model of the system. A good conceptual model helps the user to form an understanding of the system and enhances the feeling of control.

Affordances: Good affordances are needed in a design since they make the necessary actions possible to the user.

Signifiers: Proper signifiers makes for good discoverability and ensures that the feedback is communicated and intelligible.

Mappings: A strong mapping between the controls and their effects eases the interaction with the system. Spatial correspondence and natural grouping can be used to effectively create these mappings.

Constraints: Constraints limits and guides the possible interactions with a system. Constraints can be physical, logical, semantic or cultural.

2.7.2 Ben Shneiderman's interface design rules

In 1986, Ben Shneiderman published his book "Designing the User Interface: Strategies for Human-Computer Interaction." and in it he lists eight golden rules of interface design[39]. The book is, at the time of writing, on its fifth edition and each edition has introduced minor changes to the original list.

This list is taken from the fifth edition[40].

Strive for consistency. Provide a consistent sequence of actions for situations that are similar. User similar terminology in the different parts of the user interface e.g. menus and help screens.

Cater to universal usability. With more frequent use, the user seeks to reduce the number of interactions needed to perform his/her tasks. This can be accomplished by providing shortcuts such as abbreviations, macros, and hidden commands.

Offer informative feedback. All actions should result in a response from the system so the user can verify that the action was acknowledged and performed properly. More frequent actions do not need as substantial responses as less frequent actions.

Design dialogs to yield closure. Divide lengthy dialogs into steps so there is a clear beginning, middle and end. Provide feedback after each step has been performed.

Prevent errors. Try to prevent the user from performing any major errors and make sure to detect any errors and provide simple error handling for when errors do occur.

Permit easy reversal of actions. The user must be able to undo his/her actions, data entries and entire groups of actions. This promotes exploration and reduces anxiety.

Support internal locus of control. More experienced users like to feel in control of the system and not the other way around. Make sure the user is the initiator of actions rather than merely responding to actions.

Reduce short-term memory load. By using simple and multiple page displays, and providing sufficient training time for codes, mnemonics and sequences of actions, the stress on the short-term memory is reduced.

2.7.3 Jacob Nielsen's 10 usability heuristics for user interface design

Similar to Ben Shneiderman, Jacob Nielsen has also created a list of usability principles to follow. In contrast to Shneiderman, Nielsen calls his principles heuristics to further emphasize that these principles are not exact rules. The heuristics are as follows[41]:

Visibility of system status. This relates to Norman's principle of discoverability (see section 2.7.1). The state of the system should be visible and discoverable. This is achieved through good feedback within reasonable time.

Match between system and the real world. To make communication with the user simple, the interface should speak the same language as the user. This is an application of Norman's principle of natural mappings. Information shown by the interface should be natural to the user. Using system-oriented terms is usually a bad idea.

User control and freedom. The user should be free to make mistakes and easily reverse these mistakes. It should be easy to go back and forth.

Consistency and standards. The interface should be consistent and follow standard conventions. Using different words or unconventional interaction might confuse the user. Very similar to Shneiderman's first rule (see section 2.7.2).

Error prevention. In accordance with Shneiderman Nielsen also counsels error prevention to limit user mistakes.

Recognition rather than recall. Nielsen also recognizes the weakness of the human short-term memory and hence recommends keeping important information visible.

Flexibility and efficiency of use. This heuristic is similar to Shneiderman's rule of internal locus of control. Expert users should be empowered by allowing shortcuts to be tailored by the user.

Aesthetic and minimalist design. Every piece of information visible on the screen competes for the user's attention. Visible information should thus be limited to what is truly necessary.

Help users recognize, diagnose, and recover from errors. When errors cannot be prevented, the user should be aided by the system in resolving these errors. The interface should clearly communicate what is wrong and offer help.

Help and documentation. Some systems can be too big or complex to be used without documentation. This kind of documentation should be easy to find and read.

Chapter 3

Process workflow

We decided to follow the process model illustrated in figure 3.1. This process was inspired by the ideas of agile, UCD, discount usability and Google Ventures' design sprint.

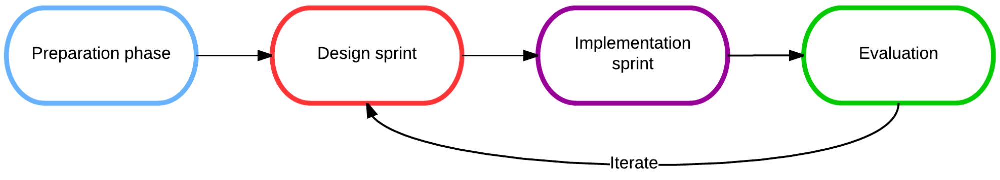


Figure 3.1: The outline of our iterative process

3.1 Preparation phase

This was the startup phase of the project. The purpose of the background study was to form a base for the upcoming design work. Here we examined the current system, studied the additional literature needed for our specific project, and categorized potential users.

3.2 Design sprint

To create our design we worked according to Google Ventures' design sprint. We adapted the model to fit our small team, but the core ideas of individual thinking, rapid prototyping, fast decisions and testing before building stayed the same. We also used the same five steps, understand, sketch, decide, prototype and test. A design sprint was allowed to loop directly into a new design sprint if we felt the need to work further on our prototype before we started building it. Our plan was to let a design sprint last somewhere between two to five days.

3.3 Implementation sprint

After the design sprint followed an agile implementation sprint. This sprint took the results from the design sprint as input and formed a development plan. We split the result from the design sprint into small implementable parts. These parts was ranked and then implemented according to this ranking. Each implementation sprint lasted around two to three weeks.

3.4 Evaluation

At the end of each iteration we applied the discount usability ideas (see section 2.3.1) of heuristic evaluation and simple user testing to evaluate the result. We, when possible, enlisted the help of usability professionals at Axis to help with the heuristic evaluation. The usability tests looked exactly the same as the one performed in the design sprint, but with the implemented product instead of a prototype. The results of the evaluation fed the next design and implementation sprints. Each evaluation lasted one to two days.

3.5 Iteration

After evaluation the process iterated. This followed the UCD and agile model of iterative improvement and refinement. Every iteration was different depending on the current stage of the product. Depending of the results of the evaluation we also allowed skipping the design sprint. If the evaluation ended up with very small or few usability problems or if we encountered larger programming issues we simply jumped right into an implementation sprint.

Chapter 4

Preparation phase

The following five chapters gives a thorough recount of our design process. Throughout these chapters we describe our work and account every decision we made. These decisions are discussed directly in this text as to help the reader better follow our line of thought. In the discussion chapter, that follows the process chapters, we instead discuss larger issues, decisions and their implications.

4.1 Previous system

The previous system was based on many manual administrative steps with different workflows for each of the sites. Because of the differences in site setup we start by explaining a typical setup (see figure 4.1) and thereafter describe how the implemented sites differ from the typical setup.

4.1.1 A typical Live Site

The Live Site consists of a main camera, the cameras to be tested, a switch, a weather station and a PC.

The cameras that are being tested are mounted on a wall. The main camera is positioned facing the wall so that all the other cameras are in view. It is used to observe outside defects on the cameras such as discoloration or if snow or condensation is covering the lens. The PC and all cameras are connected through the switch. A weather station is mounted to record the weather conditions at the site.

The main camera is setup to take pictures of each of the cameras at regular intervals and save them to the PC. The weather conditions are recorded in a similar way and sent to weather server.

The folder containing the images are synced to a PC in Lund automatically using a cloud storage and synchronization service (see figure 4.2). Roughly once a month, the Live Site manager looks through the images one by one looking for defects. If a defect is detected, a record containing the date and product name

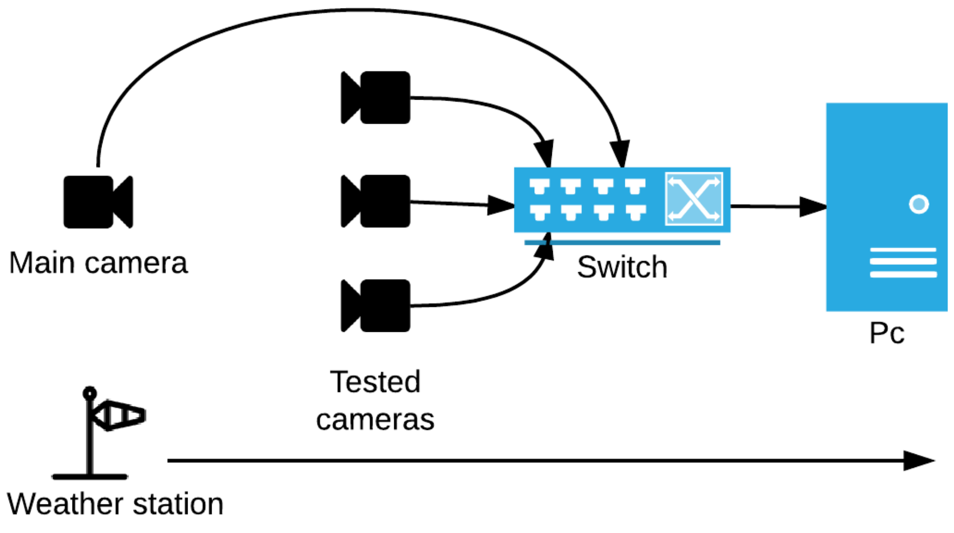


Figure 4.1: The typical setup of a Live Site

is added in an Excel spreadsheet together with a description and comments, and the image is copied to a separate directory.

TeamViewer is used to connect to the Live Site PC for maintenance tasks and is also used to manually copy images if firewall settings or other policies prevent the use of cloud storage synchronization.

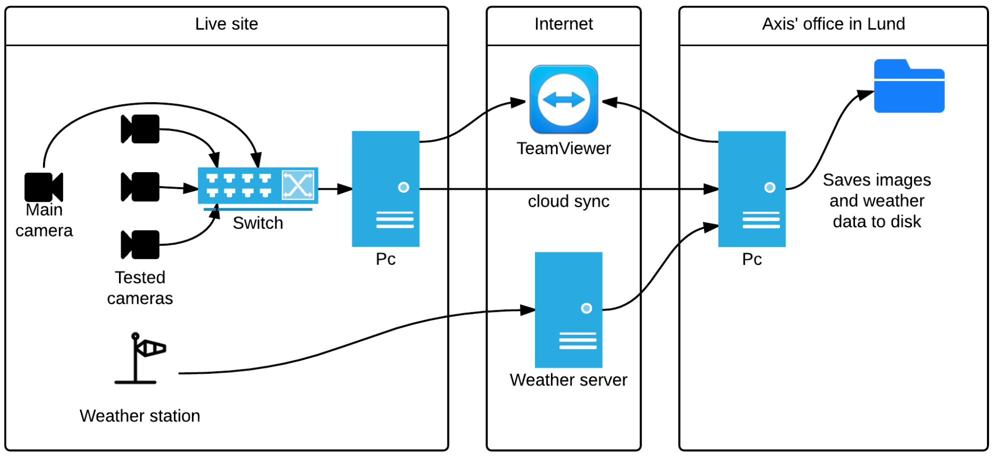


Figure 4.2: The data flow from a Live Site and the storage in Lund.

Lund: Lund is the location of the first Live Site. It is placed on the roof of one of Axis' buildings and is the simplest of the sites. It differs most from the typical setup and because of strict surveillance laws has no main camera. Pictures are instead taken manually using hand held cameras when deemed necessary. The tested cameras also need to be dimmed so that they would not record anything outside the property.

Novosibirsk: In Russia's third most populous city, Novosibirsk, the winter is cold with temperatures sometimes dropping to -40 degrees Celsius while the summers are dry with temperatures reaching 35 degrees Celsius. It is not possible to use cloud synchronization here so TeamViewer has to be used to copy the images.

Bangkok: The Live Site in Bangkok, Thailand, did not differ from the typical setup. Unfortunately, it burned down before our project started but is to be rebuilt in the near future.

4.2 User categorization

In order to create a product with good usability we had to know who the user would be. To stay within the scope of the project we decided not to do any extensive research on who might be interested in this product and instead discuss and list our own ideas on who might want to use it. This resulted in the following user categories.

4.2.1 Live Site manager

This is the primary user who will be using the product as part of his/her daily job. The Live Site manager is responsible for the running of the Live Sites and the analysis of the data collected. The analysis this user wants to perform includes:

- Physical impact of weather on the cameras.
- Temperature performance.
- Specific hardware performance such as fan and heater operation.

4.2.2 Axis employee

This category includes any other user such as sales, product maintenance, product introduction and product managers. These users are more interested in seeing how a specific camera performs and comparing it with other cameras. Example use cases could be:

- Product maintenance investigating a specific issue that only appears in a certain climate.
- The sales department matching products to a specific climate.

Chapter 5

Iteration 1 - Discovery and understanding

This first iteration was themed by discovery and divergence (see figure 5.1). We started with a full design sprint and then moved on to an experimental implementation sprint. The iteration was concluded with a short evaluation phase.

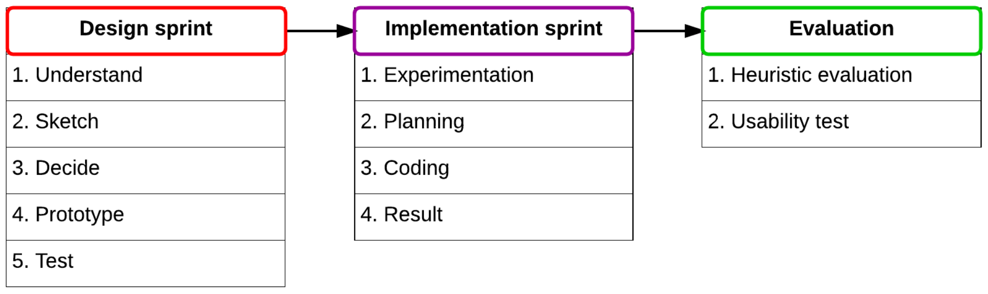


Figure 5.1: Iteration 1

5.1 Design sprint

With the knowledge gained from our background study we were now prepared to launch our design process. We started as planned by running a design sprint. This initial sprint differed a bit from the upcoming ones since we had no previous design to improve upon. We defined the goals of this sprint as:

1. *Form an understanding of the problem.*
2. *Prototype and test a solution to this problem.*

5.1.1 Understand

The sprint started with an exploration of the problem. This meant revisiting the original goals of the Live Sites project:

- Control of long term effects of environment.
- Hardware design output - how does the camera look in a real environment?
- Hardware quality test design output - find defects that were not discovered during development.

We had a brainstorming session where we split these goals into small, concrete parts. It resulted in the following items:

- Visualizing the collected data.
- Finding defects in cameras.
- Find the connection between climate patterns and camera defects.
- Automation of the process.
- Reporting.

To help us rank these items in order of importance, we performed an observation session. In this session we observed how the current system was used at the time. We instructed the user to use the system as he normally would have, while we observed. By doing this we avoided getting influenced by the user's opinions and could instead base our analysis on the observed reality. The study led us to the following conclusion:

1. Automation - The current system demanded the user to do a lot of manual steps: logging in, moving data, copying images, clicking through images and so on.
2. Visualization - The user had no idea what the data actually looks like or what is actually going on at the site.
3. Reporting - There was no easy way for the user to build a report to share with others.

5.1.2 Sketch

After discovering and defining the problem, we moved on to developing a solution. To generate many diverging ideas we started by doing individual rapid prototyping with pen and paper. These early paper sketches were then combined on a whiteboard to form three coherent prototypes. These three prototypes were based on three different ideas of what should be the base of the user interface: a map, the weather data or the camera images.

The map

In the first idea, based on the map, the user interface starts with a geographical map where the different Live Sites are marked and clickable (see figure 5.2). The user could click on a Live Site to get to a detailed overview of all the data for that Live Site. This view should offer a lot of interaction with the data in the form of filters and tooltips with camera images that appear when hovering the data with the cursor.

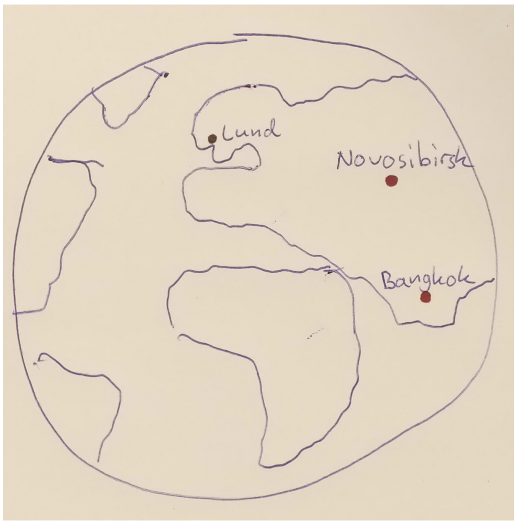


Figure 5.2: The map idea

The weather data

The second idea based the interaction on the weather data (see figure 5.3). The interaction starts by immediately showing the weather data. The user can select different weather parameters to draw for the different sites and then view camera images as a secondary graph at the bottom of the screen.

The images

The third prototype was based on a completely different idea. The interaction was instead based entirely on the automation problem and tried to solve it by removing as many manual steps as possible. This let the user focus entirely on analyzing the images. The interaction started with the user selecting a specific camera for a specific site (see figure 5.4). This launched a new view where the latest images of and from that camera were shown. The user could view individual images in the search for camera defects. When a defect was found the user could

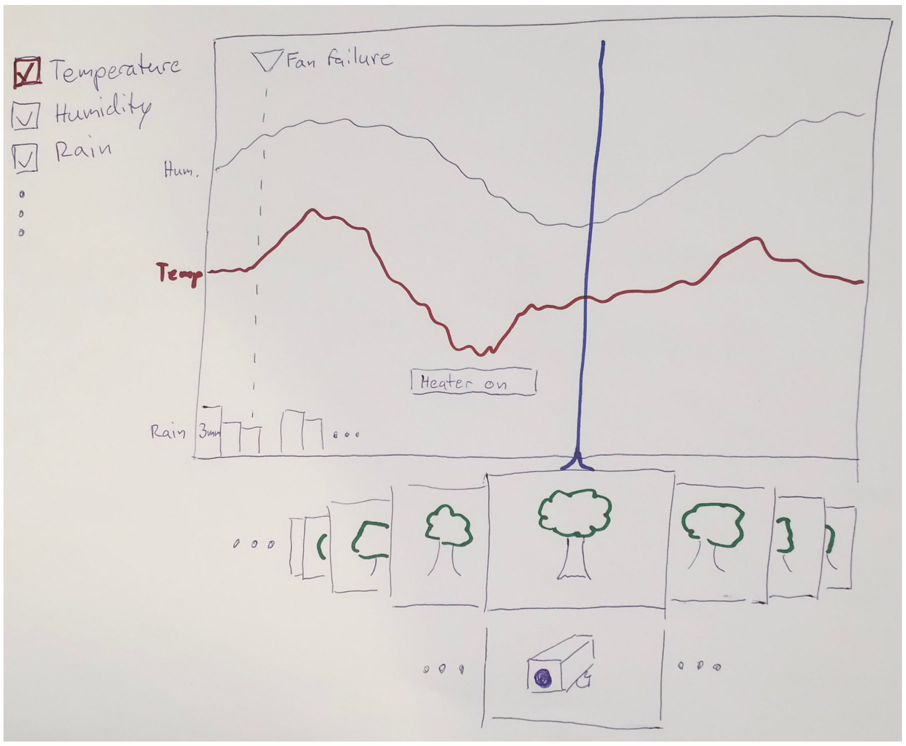


Figure 5.3: The weather data idea

hit the tag button which displayed a pop-up dialog where a tag description and comment could be entered and saved.

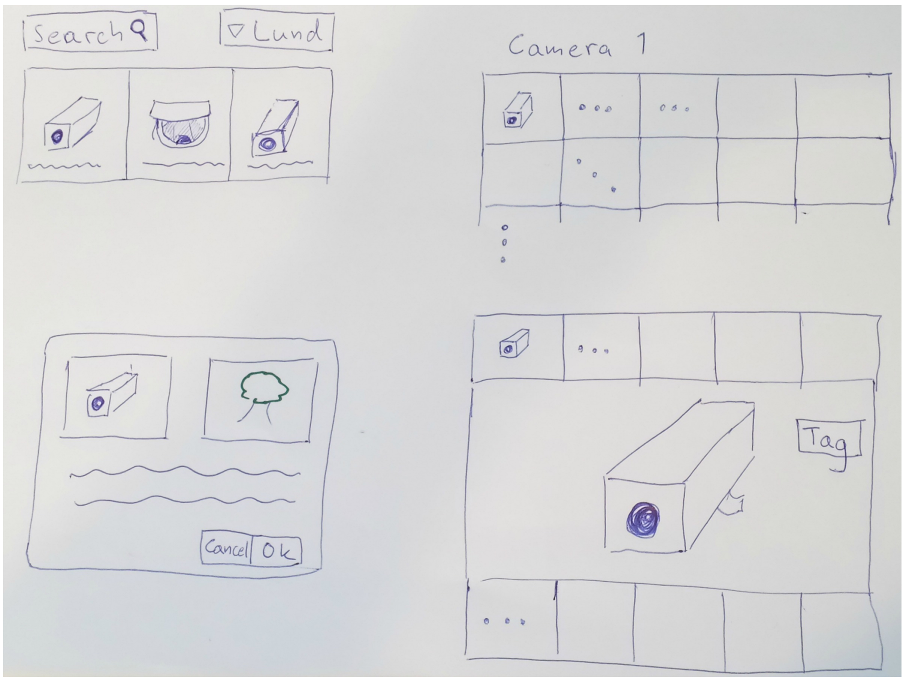


Figure 5.4: The image based idea

5.1.3 Decide

By this stage we had diverged into three core concepts, now we needed to converge on a single solution. To do this we ran a silent brainstorming session where we scored our different ideas. We then presented our critiques and praises and followed up with a discussion.

We started by scrapping the map idea. Although it would be aesthetically pleasing, we felt it did not fit in and was simply distracting. This decision is also supported by Stephen Few's theory on choosing the correct display media. The map would in this case not offer any additional insights.

Moving forward, ideas two and three received mixed judgments. Basing everything on the weather data shows the user the data right of the bat, but it also disconnects too much from the task of finding actual defects in cameras. The solution would lack an easy way of viewing the images. It could also be intimidating to get launched straight into this detailed data and the user might get completely thrown off.

The image based idea would solve the defect finding problem, but it lacked a connection to the weather data. We also felt it was a bit too boring. Being an effective and usable tool cannot entirely make up for a bland experience. There was also a lack of user empowerment. While it would give power over the images, it left the user completely powerless against the weather data.

The discussion led us to the whiteboard where we stitched the second and third ideas together and ended up with a new concept. This user interface instead started in a dashboard that showed an overview of all the sites, with both weather and camera data. The user could then move into two different analysis views, data analysis and image analysis. We really liked the dashboard idea and decided to move forward with it. The dashboard gave an overview the Live Sites. It did not completely overwhelm the user with data, while still retaining the important data. Combining the dashboard with the analysis tools gave the user complete access to all the data the Live Sites have to offer and hence power over it.

5.1.4 Prototype

After combining our thoughts and ideas into a single solution we now needed to build a prototype to be able to test it. To do this we followed Knapp's advice and used PowerPoint (see section 2.3.1). This decision turned out to be a great one, and we managed to build a good-looking prototype in a few hours. The prototype started with the previously discussed dashboard (see figure 5.5).

We intentionally made the "event bar" (see figure 5.6) cover the whole width of the screen and placed it on top to emphasize its importance. We felt the "event bar" offered the most value in the context of monitoring the Live Sites.

To provide a simple and effective view of the weather data we looked to Tufte's

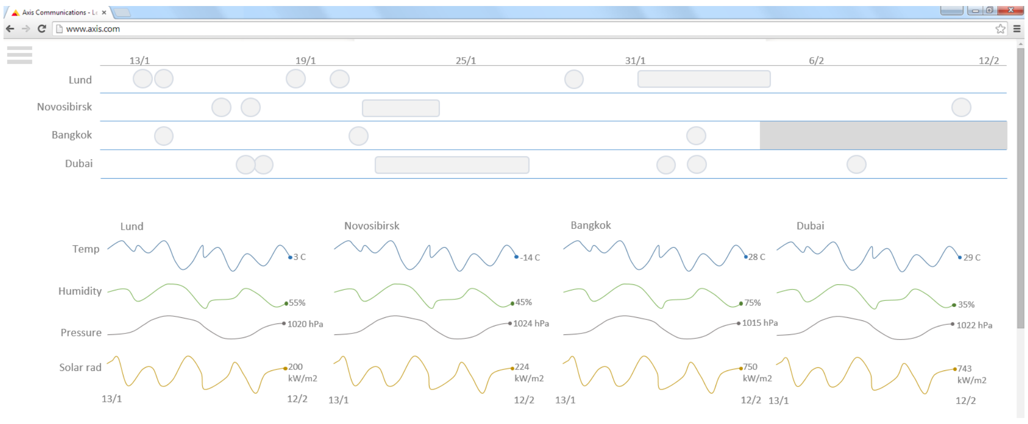


Figure 5.5: This was the dashboard. The "event bar" at the top showed every event that had occurred on a Live Site. Below the "event bar" there were sparkline charts of the weather data from each Live Site.

ideas of small multiples (see section 2.5.3) and sparklines (see section 2.5.4). The weather station provided a lot of different data parameters, e.g. temperature, wind speed, air pressure, humidity, solar radiation etc. To give the user a quick overview of different parameters we created a table of sparklines. Every site has a column and every weather parameter has one row. The sparklines (see figure 5.7) works well to show the weather data without going into the details. The user could see if there had been extreme dips or fast changes in weather. At the same the sparklines were not detailed enough to distract the user from the more important "event bar". The prototype also had a small menu button in the top left corner (see figure 5.8). This goes against Stephen Few's recommendation to reserve the top left to the most important data. We did however feel that the placement of the button was enforced by today's standard and was sufficiently silent to not distract from the data. We managed to make the menu work the way it is was supposed to and can be used to navigate through the prototype.

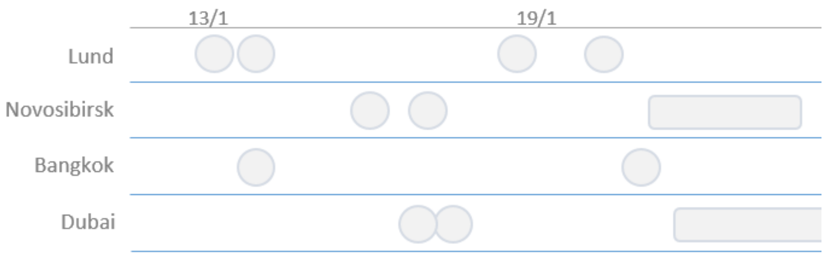


Figure 5.6: Close-up of the "event bar". The dots represented momentary events, such as "failed to save image" and the rectangles represented events that lasted over time, such as "heater on".

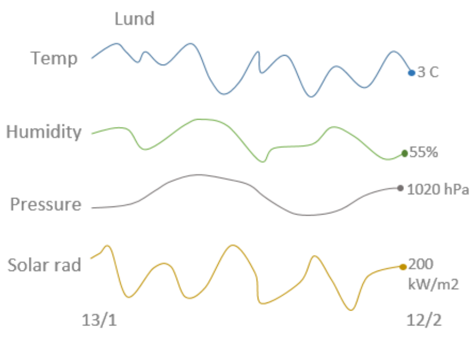


Figure 5.7: Each Live Site got a column with sparklines showing the latest weather conditions at the site.

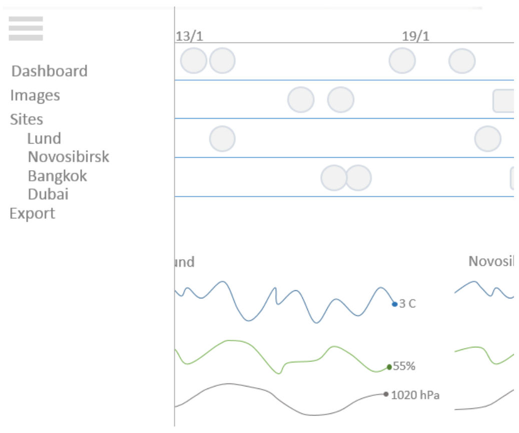


Figure 5.8: When clicking the menu button a menu appeared on the left hand side.

We created a page that gives the user a more detailed view of the data gathered from a selected Live Site (see figure 5.9 and 5.10). The "event bar" was used in this view as well, but only shows the one row for the specific site.

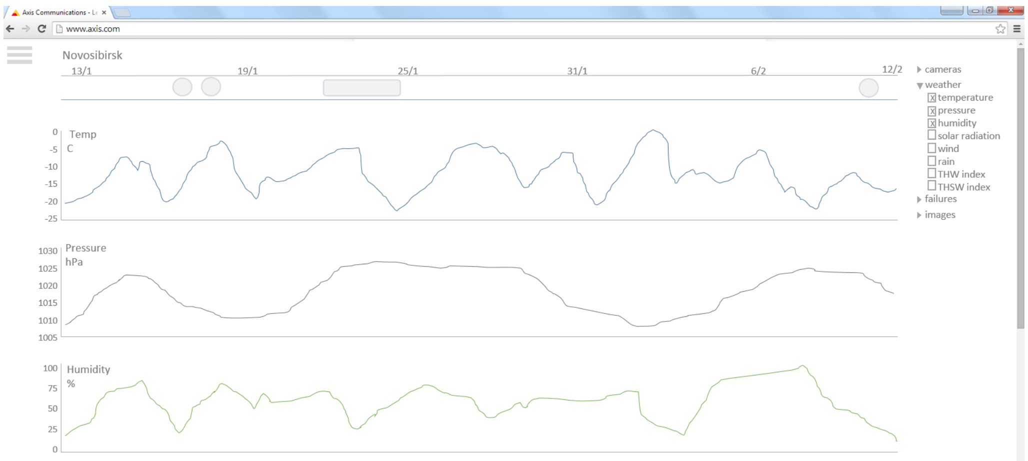


Figure 5.9: The site view showed detailed weather data collected from a single Live Site over the last month. It was also possible to plot sensor data from the cameras or mark where failures were detected.

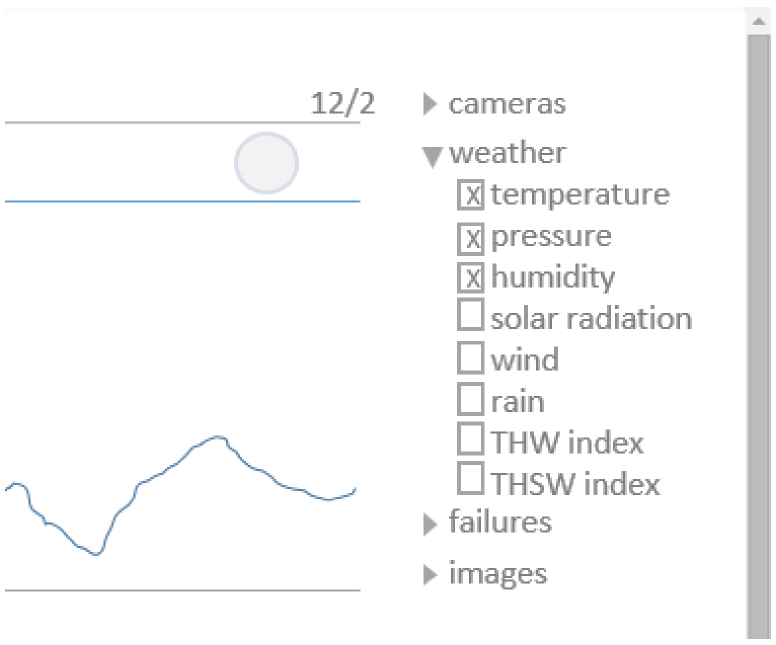


Figure 5.10: To the right hand side of the site view, the user could choose which data to display

The image based idea resulted in the prototype shown in figures 5.11 to 5.14.

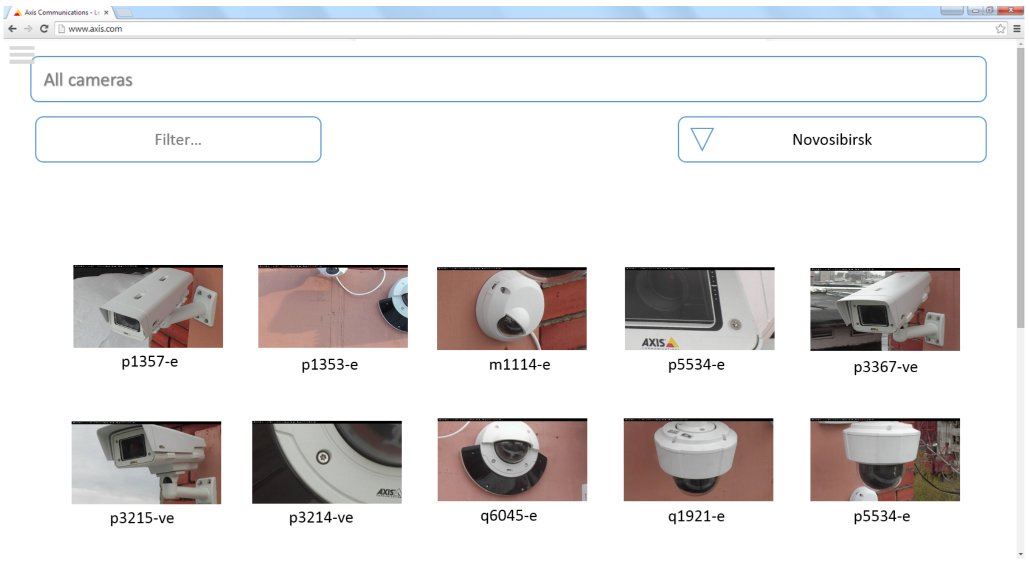


Figure 5.11: The user started by selecting the site and camera they wanted to examine, optionally filtering the results using a search string.

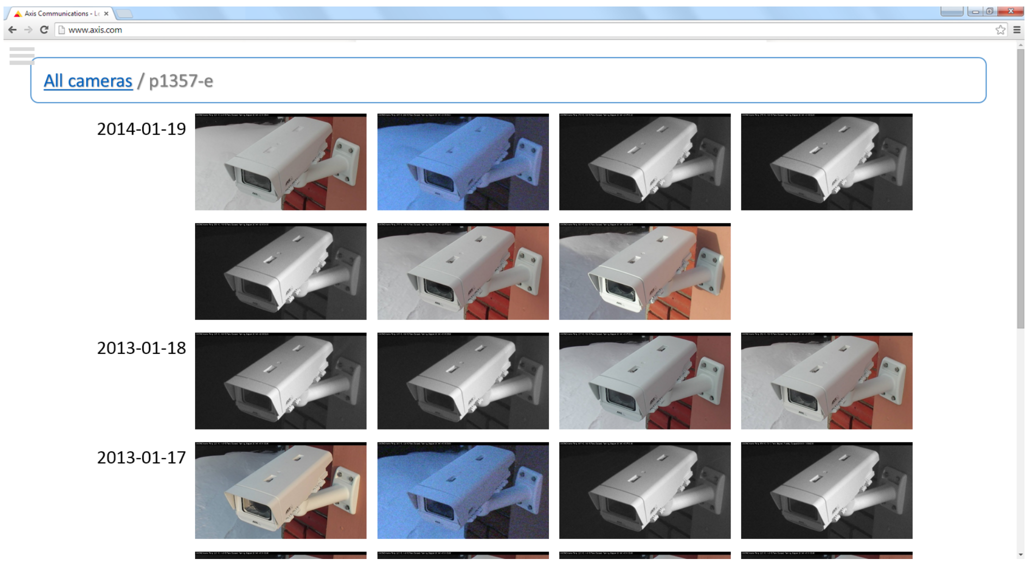


Figure 5.12: This took the user to a list of the images from the selected camera sorted by date.

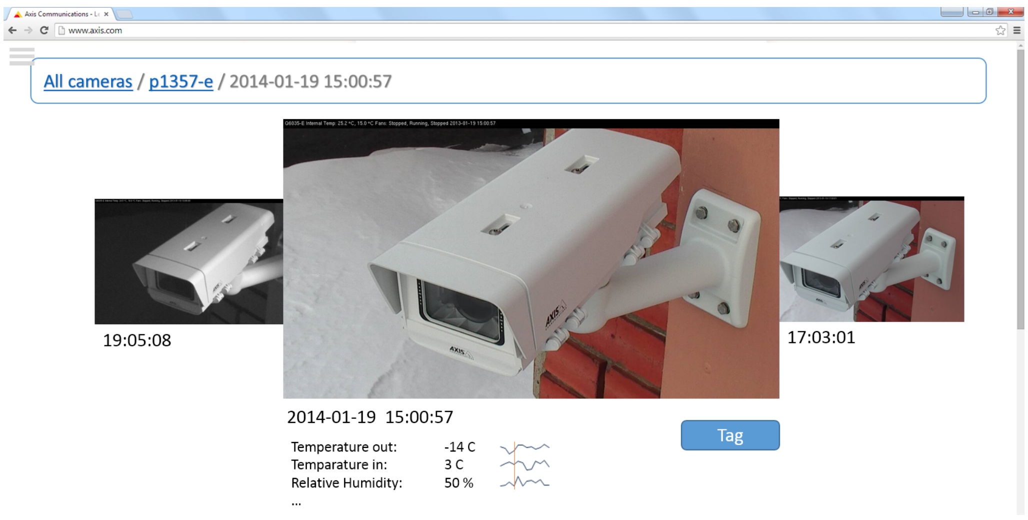


Figure 5.13: Clicking on one of the images in figure 5.12 showed an enlarged version of the image as well as the previous and next image. These were clickable and would enlarge the clicked image instead. Navigation could also be done using the arrow keys.

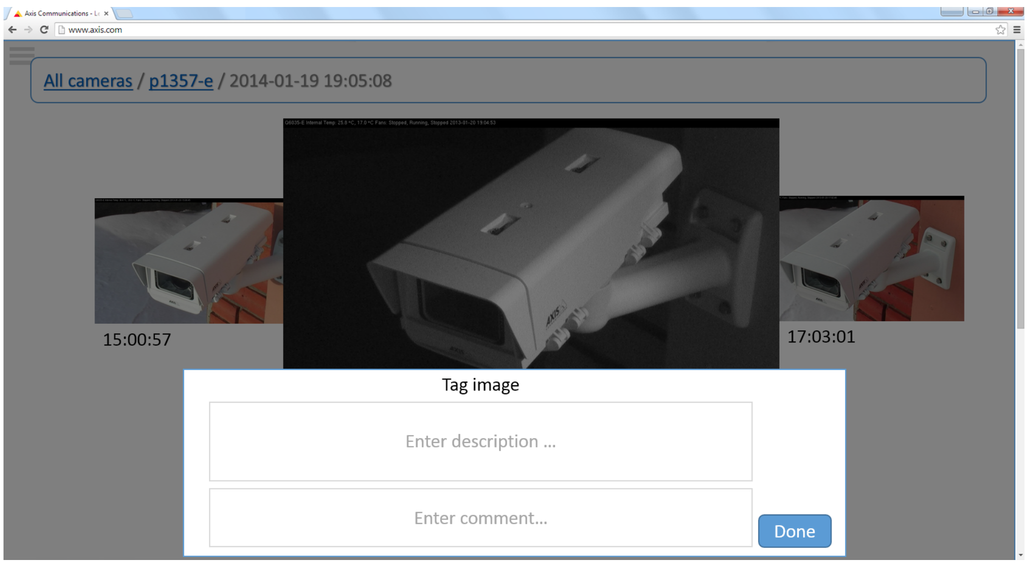


Figure 5.14: When a defect had been found, the user clicked the tag button and a tag image dialog appeared. It saved a description and comment for the image in the database.

5.1.5 Test

To evaluate our first prototype we performed a minor exploratory user test with two users. The reason for only including two users was a lack of available users. We needed quick feedback and did not have the time to recruit more testers, hence two would have to suffice. During the tests we observed the users moving through the user interface and listened to their thoughts and feedback. After the test we also had a discussion with the testers to further develop their comments and ideas.

5.2 Implementation sprint

After the initial design sprint we now had the choice to either follow up with a second design sprint or move on to implementation. At the time we felt our solution was a good start, but we lacked an understanding of what was technically possible to build. Following this reasoning we decided to move forward into implementation to learn more and to explore the technical aspects of the solution. The decision to delve so quickly into implementation might seem risky, but we deemed it to be the best way forward. After our own analysis of our solution and the input of our testers we felt assured that we were on the right track. We were also confident that by exploring the technical aspect we would uncover more ideas and background going into the next design sprint. Moving fast into implementation did however expose us to the risk of building the wrong solution or even that we would be solving the wrong problem. We accepted this risk on the premise that we had to gain technical knowledge either way. Gaining this knowledge while also implementing our solution so far felt like the most time effective way going forward.

5.2.1 Experimentation

Before moving on to actually implementing our prototype we started by experimenting with different technologies. During this experimentation phase we built several simple graph-drawing web applications. By doing this we increased our understanding of the technology, especially D3.js.

The following technologies were investigated during the experimentation:

Node.js	Web server
Apache	Web server
D3 (Data-Driven Documents)	Data visualization library
dc.js (Dimensional Charting)	Charting library
MySQL	Relational database
MongoDB	Document oriented database
HTML5 Canvas	Scriptable 2D rendering element
SVG (Scalable Vector Graphics)	XML-based vector image format

Web server

When choosing web server our priorities were getting a fast, scalable, and reliable server. We ended up choosing between an Apache server running PHP and a Node.js server, since we had previous experience working with them.

PHP on Apache: Running PHP on an Apache web server is the most popular choice for websites at the time of writing[42][43][44].

Pros: Stable, reliable.

Cons: Cannot run the same languages as the web browser, hence we cannot share code between the browser and the server. Thread and process based instead of asynchronous and event driven which generally makes it slower than Node.js[45].

Node.js: Node.js is a newer option that is based on the JavaScript V8 engine and can be extended using the web application framework Express to easily serve web pages.

Pros: Runs JavaScript which can also be used in the browser. Faster than running PHP on Apache and is rapidly increasing in popularity.

Cons: Is still relatively rarely used and only serves about 0.1% of today's websites[42].

Verdict: The fact that Node.js uses JavaScript means we can easily experiment with processing the gathered data both on the server and client. This together with a need for only one programming language made Node.js the web server we chose.

Data visualization

The choice of data visualization libraries was of great importance to our project.

D3.js: D3.js (Data-Driven Documents) is a JavaScript library for manipulating objects based on data. It does not take care of the rendering of the graphs but rather the updating of the Document Object Model (DOM).

DC.js: DC.js (Dimensional Charting) is built upon D3.js and has native crossfilter (a fast multidimensional filtering library) support.

SVG: SVG (Scalable Vector Graphics) is an XML-based vector graphics format. It has support for animations and interaction similar to the DOM, and can be styled using Cascading Style Sheets (CSS).

HTML5 Canvas: The HTML5 Canvas is an element that allows low level rendering in the browser.

Verdict: We decided to use D3.js together with SVG since we found this to be the simplest and most straight forward solution. During our early experiments, we used DC.js but it did not have the necessary graph-types built in and it felt easier to use D3.js directly. DC.js' close integration with crossfilter, which is not that well suited for time series, made the graphs slow when filtering on dates. Using the HTML5 Canvas instead of SVG would probably increase performance, at the cost of not being able to style the graphs using CSS, but the rendering did not seem to be a bottleneck.

Database

How the data collected from the sites should be saved and accessed was one of the dilemmas we were faced with. Our previous experience with databases was limited to MySQL and MongoDB.

MySQL: MySQL is an open-source relational database that uses the query language SQL.

MongoDB: MongoDB is a NoSQL database i.e. not a relational database. It is data oriented and stores JSON-documents with dynamic schemas.

Verdict: When starting to model the database structure we realized that we would end up with quite a few relations between our entities. Because of this, MySQL felt like the best choice.

5.2.2 Planning

We listed the following items to build in this sprint:

- Back end groundwork.
- Overview dashboard.
- Explorative data visualization view for each site.
- Image tagging view.

5.2.3 System architecture

We needed to rethink the underlying architecture and data flow to be able to build the solution we wanted. This meant adding a MySQL database for storing data and simplifying the weather data flow by syncing them via FTP to the Live Site PC. This meant all data from the Live Site is stored in a folder that is synced to the office in Lund using a cloud storage service (see figure 5.15).

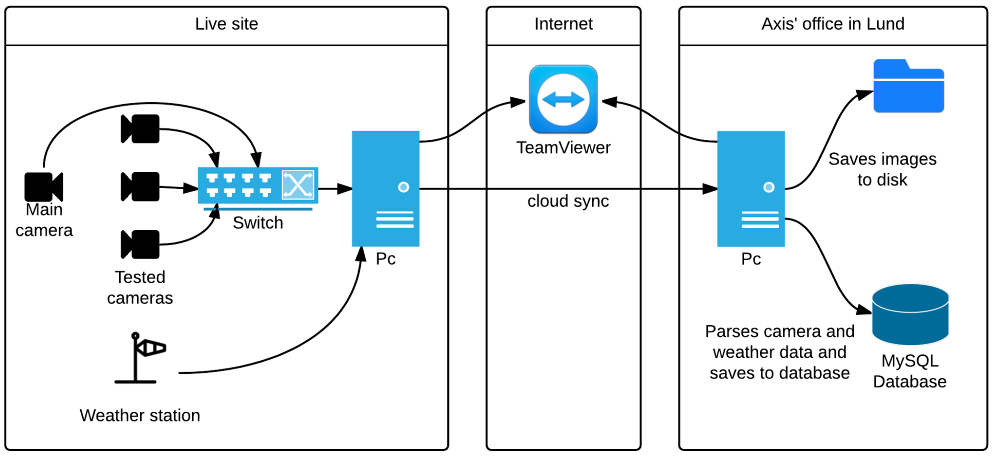


Figure 5.15: The new data flow from a Live Site to the office in Lund.

5.2.4 Coding

When implementing the elements from the prototype we realized that many of them were unnecessarily large and decided to shrink them. This meant we had space to add some additional elements to the dashboard. These were not added solely because there was space available, but because we believed they would provide a deeper understanding of the state of the live sites. During coding we also did some other adjustments to the user interface. They did not add a significant amount of complexity to the implementation, and did therefore not slow down the process too much. List of changes and additions:

Added tables to the dashboard showing latest events. Relying entirely on the “event bar” felt insufficient. Using a table to view this information is the best option since every piece of data represents detailed information regarding singular events.

Added uptime to the dashboard. Adding uptime to the dashboard felt natural. It is a single number that can directly convey if an error has occurred recently.

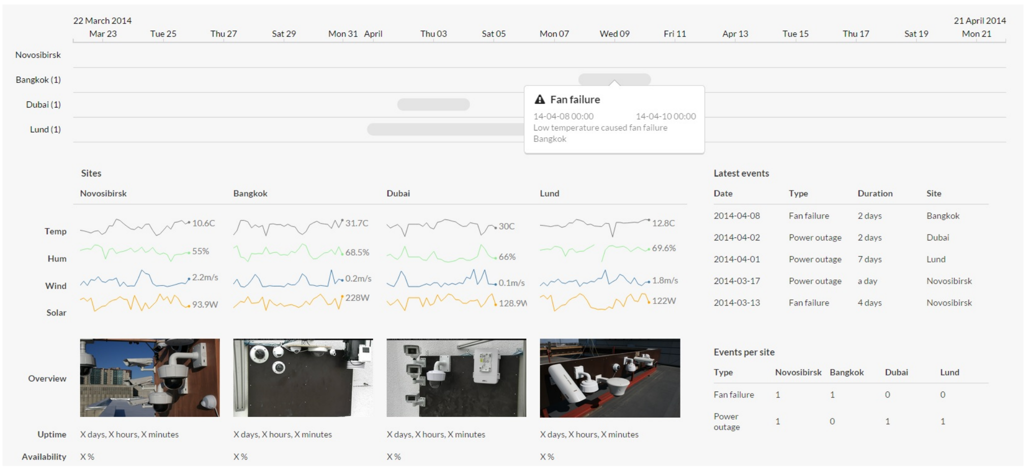
Background color. The background was set to a slightly off-white color since we felt this would be easier on the eyes than a pure white background.

Added overview images on the dashboard. We got the idea to add an image of an overview of all cameras at the site on the dashboard. This image should in a quick and easy way convey to the user if something major has gone wrong with any of the cameras, e.g. a camera is missing. This image can also be used to quickly tell if the master camera is working or not.

Re-designed the image tagging view. We were not satisfied with our initial design for the image tagging view and we decided to look for inspiration on the web. One we liked was of was Google images. Most users should be familiar with this view and so if we make our view look similar, our design should hopefully be a success. Hence we changed our design to somewhat match Google images (see figure 5.19). Images appear in rows and when clicked that image is enlarged and centered. Images can then be navigated by using the arrow buttons.

5.2.5 Result

The implemented prototype is shown in figures 5.16 to 5.20.



Fan failure
14-04-08 00:00 14-04-10 00:00
Low temperature caused fan failure
Bangkok

Sites

	Novosibirsk	Bangkok	Dubai	Lund
Temp	10.6C	31.7C	30C	12.8C
Hum	55%	68.5%	66%	69.6%
Wind	2.2m/s	0.2m/s	0.1m/s	1.8m/s
Solar	93.9W	228W	128.9W	122W
Overview				
Uptime	X days, X hours, X minutes	X days, X hours, X minutes	X days, X hours, X minutes	X days, X hours, X minutes
Availability	X %	X %	X %	X %

Latest events

Date	Type	Duration	Site
2014-04-08	Fan failure	2 days	Bangkok
2014-04-02	Power outage	2 days	Dubai
2014-04-01	Power outage	7 days	Lund
2014-03-17	Power outage	a day	Novosibirsk
2014-03-13	Fan failure	4 days	Novosibirsk

Events per site

Type	Novosibirsk	Bangkok	Dubai	Lund
Fan failure	1	1	0	0
Power outage	1	0	1	1

Figure 5.16: The dashboard was the first screen presented to the user. It contained all the elements that were in the prototype and added some additional ideas. Putting the cursor over an event in the "event bar" showed additional information about the event.

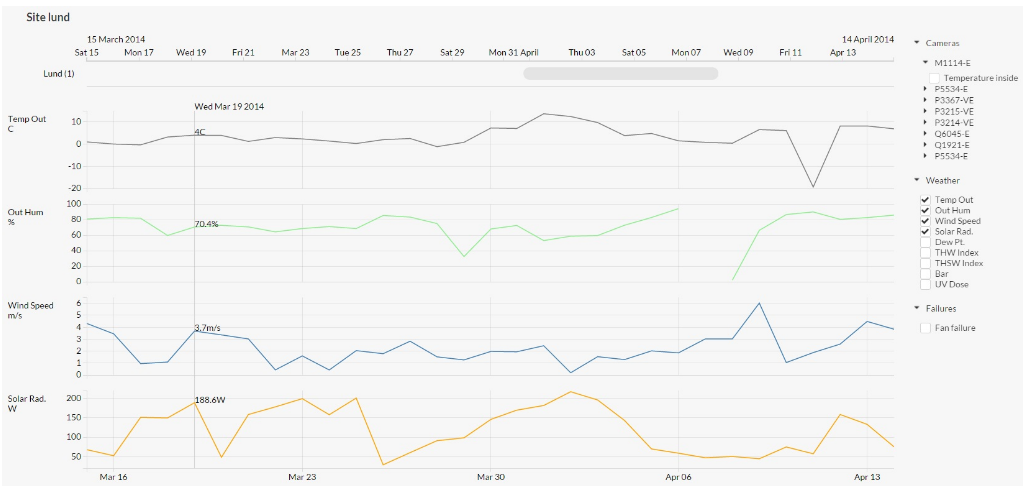


Figure 5.17: The site view was almost exactly the same as the prototype but with an added grid for easier reading. Moving the mouse over the graphs showed the measured value of all the plotted data at that point in time.

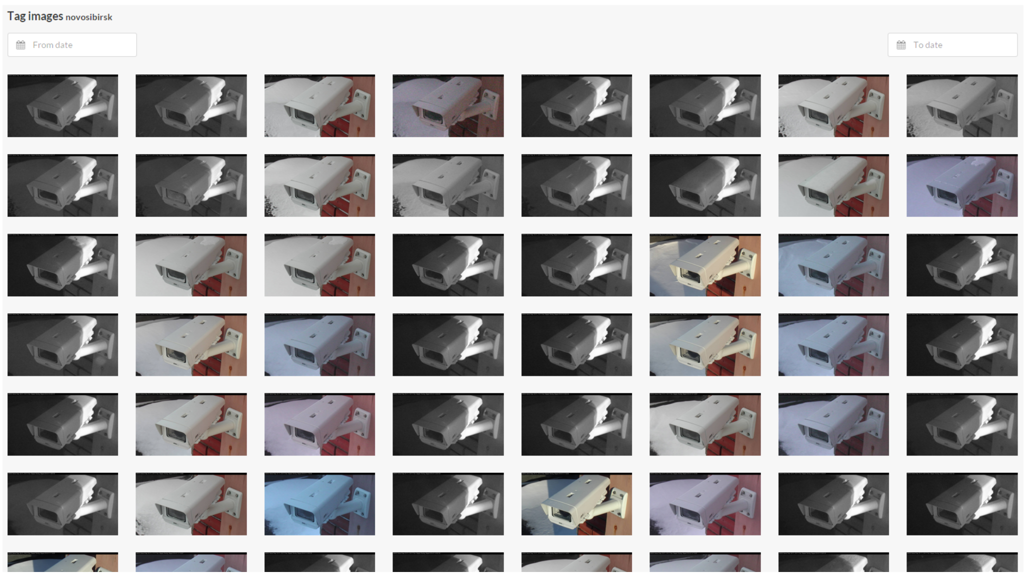
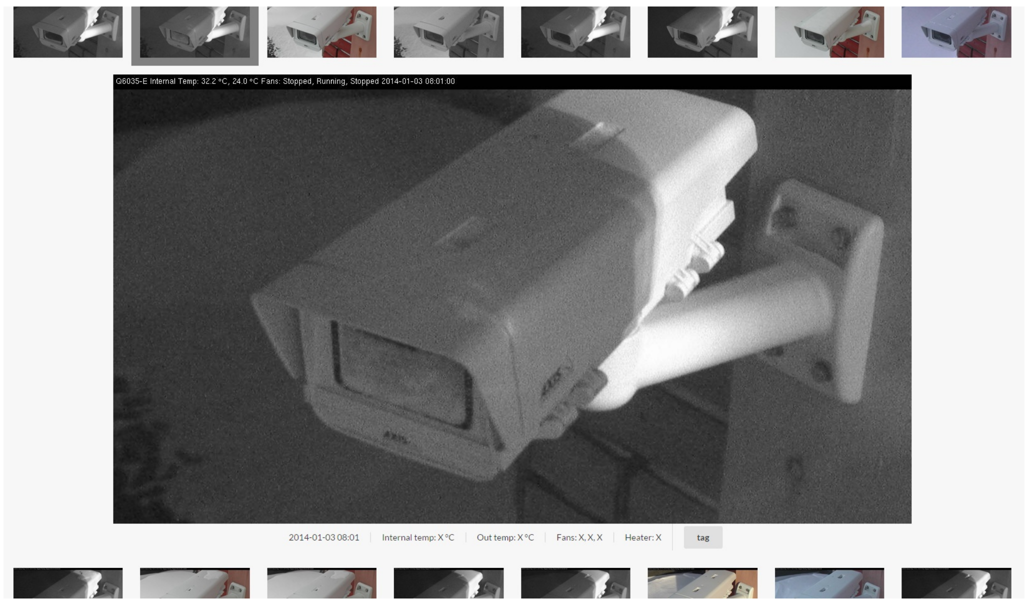
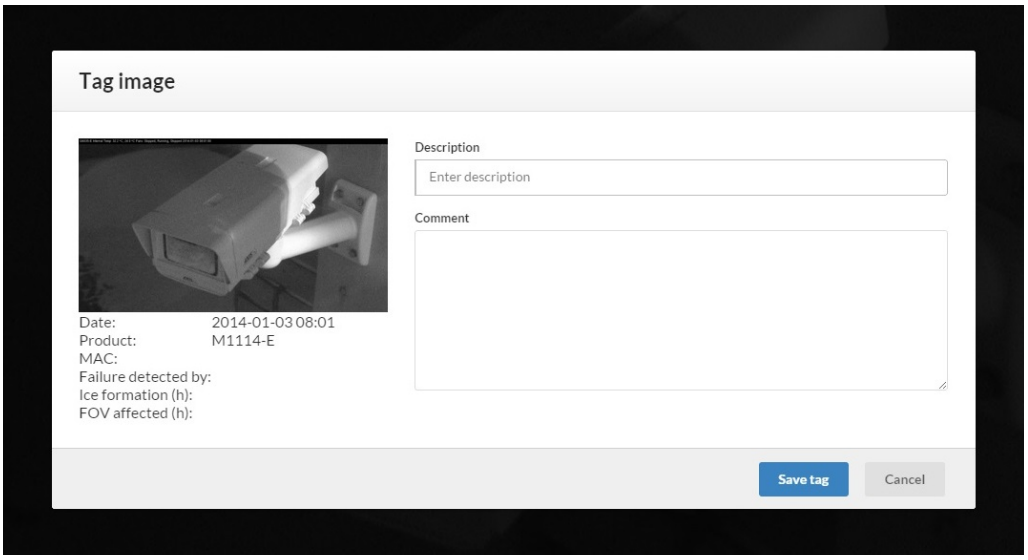


Figure 5.18: When a camera had been selected a grid appeared with the pictures from said camera.


Q6035-E Internal Temp: 32.2 °C, 24.0 °C Fans: Stopped, Running, Stopped 2014-01-03 08:01:00

2014-01-03 08:01 | Internal temp: X °C | Out temp: X °C | Fans: X, X, X | Heater: X

Figure 5.19: Clicking one of the images highlighted it by adding a gray border and showed an enlarged version of it in center of the screen. Pressing the left or right arrow keys selected the previous or next image in the list.


Tag image

Date:	2014-01-03 08:01
Product:	M1114-E
MAC:	
Failure detected by:	
Ice formation (h):	
FOV affected (h):	

Description

Enter description

Comment

Figure 5.20: Pressing the tag button or the letter "T" on the keyboard brought up the tag dialog.

5.3 Evaluation

To evaluate our implementation we decided to perform both a usability test and a heuristic evaluation.

5.3.1 User test

We started by conducting a user test. We created a task based interview guide and booked a one hour session with one potential user. The tasks were defined by going through our user stories and adding some context. The full interview guide can be found in the Appendix. We only ran one test during this testing phase. We felt we got enough feedback from this single participant and did not see the point in trying to chase down more users. We reasoned that we would rather do many small tests throughout the project than test with many users few times. The following problems were found during the test:

- The weather graphs on the dashboard needed to be more clearly grouped by column than by row.
- The user wanted to click on things directly on the dashboard instead of going through the menu.
- There was too much focus on the weather data.
- The user did not discover the accordion checkbox menu.
- The user wanted to export directly from the view where the data was shown.
- The icon on tagged images was confusing, the user interpreted the exclamation mark to mean failure instead of tag.
- The user wanted to have a personal view of his/her tags/images.
- The user felt that information about the different cameras was lacking.

5.3.2 Heuristic evaluation

Before addressing the issues we uncovered in the user test, we performed a heuristic evaluation where we looked at all the design principles and guidelines that we presented in our theory chapter (see sections 2.5.1, 2.5.7 and 2.7.1 to 2.7.3). Here follows a summary of the problems we found during this evaluation.

General usability

- The “event bar” did not offer clues that the event drops show additional information when hovered with the cursor.
- The “event bar” could be zoomed and panned indefinitely.
- The hand cursor made the “event bar” seem clickable instead of draggable.
- The checkbox menus in the site view used an “accordion”-like show and hide effect that was not easy to discover.
- There was a lack of navigation arrows in the tagging view and it is not clear that the keyboard arrows could be used to navigate.
- The dashboard sparklines did not update until the mouse was released.
- The interface did not communicate the correct conceptual model. Where was the data coming from? Where was it stored? Was it updated in real-time? How often was it updated?
- The flat design made the buttons appear less clickable.
- It was not clear what happened to the data when you dragged the “event bar”. The graph did not animate, but instantly updated and changed shape. It should instead move in the same direction as the “event bar” does.
- The images view used a vertical time orientation while the other views used a horizontal. This could be confusing.
- The charts in the site view had very limited interaction.
- Image tagging was limited to one image per tag which could be annoying.
- The user could not enlarge several images at once to compare.
- There were too many shortened labels in dashboard.
- Delayed updates of charts made it less apparent when changes had been made. Actions that trigger a server request did not provide any feedback until a response arrived, unless they triggered a page redirect that showed a spinner in the browsers address bar.
- There was no clear difference between an event and a failure.
- The terms availability and uptime were not clear.
- Help documentation was lacking.
- Lackluster error control. You could for example crash the browser by adding/removing graphs too quickly in the sites view.

Data visualization

- The dashboard did not support more than four Live Sites.
- It was not easy to see if the weather data is out of the ordinary.
- The number of decimals were not consistent in the sparklines.
- We needed to further investigate which data is most important.
- The "event bar" was not very data-ink effective.
- The y-domain was misleading in some cases. For example the humidity bar often set 90 % as the maximum value, when 100 % should be the maximum value.
- The site grouping was not separated enough on the dashboard. This led the user to group the sparklines by row instead of by column.
- The overview images on the dashboard were distracting.
- The user could not resize, change color, plot multiple graphs in the same chart, show means, variance, min-max, distributions etcetera. These were all important parts in making the user reason about the data.

Chapter 6

Iteration 2 - Conveying a better conceptual model

This iteration (see figure 6.1) followed our idea of a standard iteration. It started with a design sprint that aimed to solve the problems found in the previous sprint. These solutions were then implemented and finally evaluated.

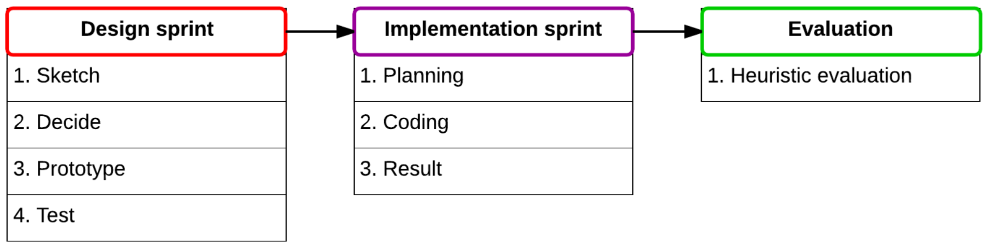


Figure 6.1: Iteration 2

6.1 Design sprint

Following the evaluation phase of iteration 1 we formulated three main problems that we wanted to explore in this design sprint:

1. *How might we connect the weather data and the camera images in a better way?*

2. *How might we give the user more control over the data?*

3. *How might we communicate the correct conceptual model?*

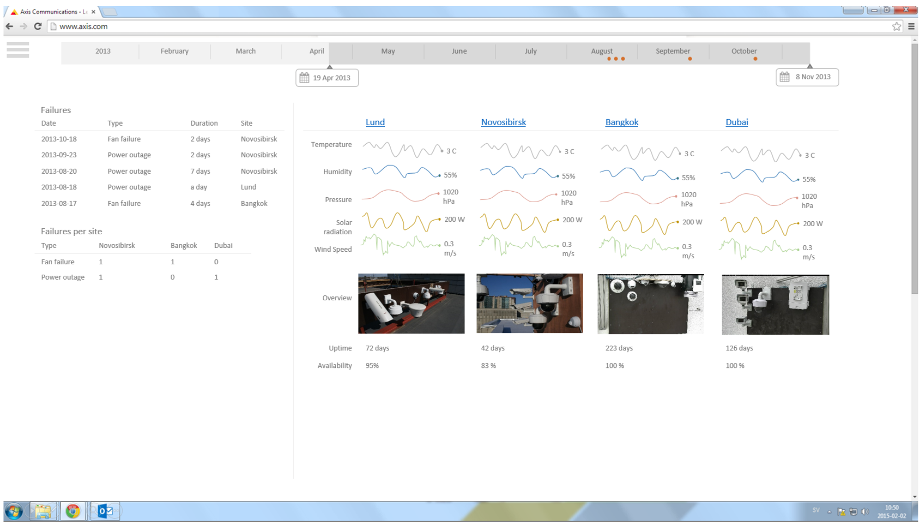
6.1.1 Sketch, Decide and Prototype

Since we already had an understanding of the problem at hand we could skip the first step of the design sprint and move directly into sketching. We sketched individually for a few minutes and came up with several new ideas. After discussing our ideas we emerged with two new concepts:

- **Multiple levels of information:** This idea is an extension of the old dashboard. The concept builds on the natural hierarchy of the Live Sites: all sites, individual sites, and individual cameras. Each of these levels has their own dashboard with different data.
    1. Top level: System overview (see figure 6.2). The purpose of this dashboard was to see the overall state of the different sites. Data includes uptime, failures, weather sparklines, and overview image.
    2. Middle level: Site view (see figure 6.3). This dashboard showed details for a specific site. Data includes a table of cameras, more detailed weather graphs and events.
    3. Low level: Camera view (see figures 6.4 and 6.5). The lowest level of the dashboard hierarchy was the individual camera. The user could view detailed information about the camera such as temperature at each sensor, fan and heater performance, uptime and availability.
- **Analytics view:** This view (see figures 6.6 to 6.8) replaced the old sites view. The concept was simple: allow the user to decide which data to plot and how to plot it. Since our users were experts in the field of camera test analysis, they knew what data they want to see, but lacked an easy way of visualizing it. This view included the option to add/remove graphs, customize graphs, plot different kinds of graphs e.g. bars, lines, scatter plots etcetera. A graph setup could be saved as a template so that the user could easily go back and view the same setup again.

We both recognized potential in these two ideas. The analytics view should give the user more control over the data and the multiple dashboards should help bridge the gap between the weather and camera data. We were still slightly concerned about the conceptual model conveyed by the interface. The dashboards might help communicate a better model. After a short discussion we agreed that we were confident enough in these ideas to move on to prototyping. User testing would have to answer our concerns on the conceptual model problem.

The prototyping was done using PowerPoint in the same way as in design sprint 1.



Failures

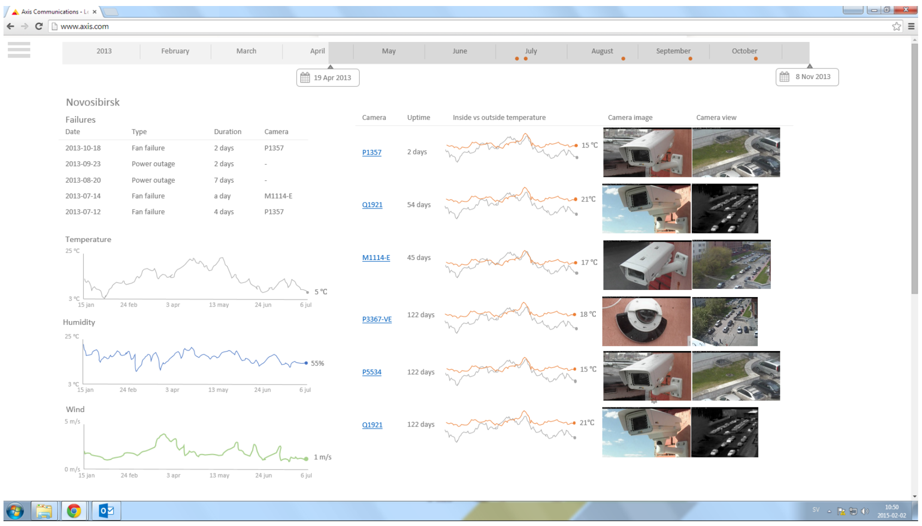
Date	Type	Duration	Site
2013-10-18	Fan failure	2 days	Novosibirsk
2013-09-23	Power outage	2 days	Novosibirsk
2013-08-20	Power outage	7 days	Novosibirsk
2013-08-18	Power outage	a day	Lund
2013-08-17	Fan failure	4 days	Bangkok

Failures per site

Type	Novosibirsk	Bangkok	Dubai
Fan failure	1	1	0
Power outage	1	0	1

	Lund	Novosibirsk	Bangkok	Dubai
Uptime	72 days	42 days	223 days	126 days
Availability	95%	83 %	100 %	100 %

Figure 6.2: The overview dashboard. The "event bar" had been replaced with a smaller version we called "time bar" and the events were instead described in tables on the left side.



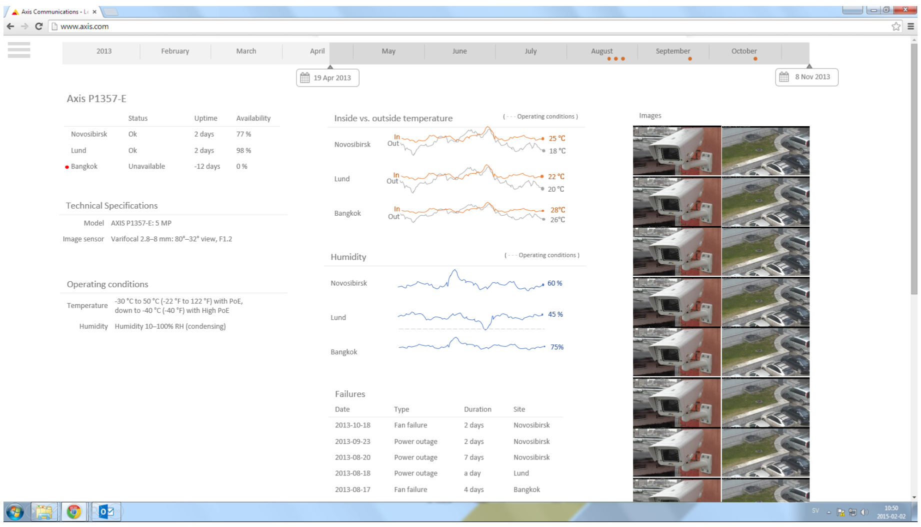
Novosibirsk

Failures

Date	Type	Duration	Camera
2013-10-18	Fan failure	2 days	P1357
2013-09-23	Power outage	2 days	-
2013-08-20	Power outage	7 days	-
2013-07-14	Fan failure	a day	M1114-E
2013-07-12	Fan failure	4 days	P1357

Camera	Uptime	Inside vs outside temperature	Camera image	Camera view
P1357	2 days	15 °C		
Q1921	54 days	21°C		
M1114-E	45 days	17 °C		
P3367-VE	122 days	18 °C		
P5534	122 days	15 °C		
Q1921	122 days	21°C		

Figure 6.3: The site dashboard was a view where the data from a specific site was shown. It contained a "time bar" and an event table as in the overview dashboard, but with the events filtered to this site. Graphs displaying weather data and a table listing the cameras installed at the site could be found on the right hand side. Each row showed the latest image of, and from that camera.



Axis P1357-E

	Status	Uptime	Availability
Novosibirsk	Ok	2 days	77 %
Lund	Ok	2 days	98 %
● Bangkok	Unavailable	-12 days	0 %

Technical Specifications

Model	AXIS P1357-E: 5 MP
Image sensor	Varifocal 2.8–8 mm: 80°–32° view, F1.2

Operating conditions

Temperature	-30 °C to 50 °C (-22 °F to 122 °F) with PoE, down to -40 °C (-40 °F) with High PoE
Humidity	Humidity 10–100% RH (condensing)

Inside vs. outside temperature

Novosibirsk

Lund

Bangkok

Humidity

Novosibirsk

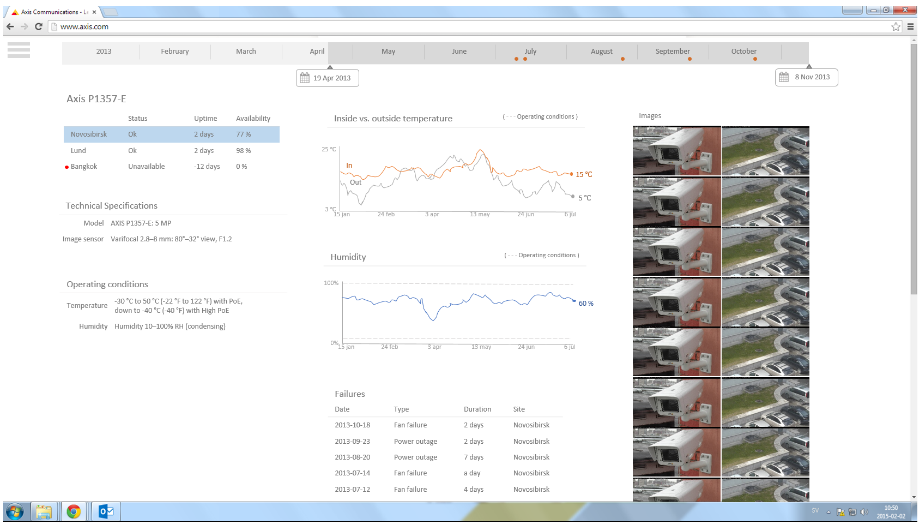
Lund

Bangkok

Failures

Date	Type	Duration	Site
2013-10-18	Fan failure	2 days	Novosibirsk
2013-09-23	Power outage	2 days	Novosibirsk
2013-08-20	Power outage	7 days	Novosibirsk
2013-08-18	Power outage	a day	Lund
2013-08-17	Fan failure	4 days	Bangkok

Figure 6.4: The camera dashboard showed information about a single camera model. To the left there was a table showing which sites the camera was installed at and information about the camera model. The middle column displayed temperature and humidity graphs and a table containing recorded failures for this camera. To the right there was a list of all images of, and from the camera.



Axis P1357-E

	Status	Uptime	Availability
Novosibirsk	Ok	2 days	77 %
Lund	Ok	2 days	98 %
● Bangkok	Unavailable	-12 days	0 %

Technical Specifications

Model	AXIS P1357-E: 5 MP
Image sensor	Varifocal 2.8–8 mm: 80°–32° view, F1.2

Operating conditions

Temperature	-30 °C to 50 °C (-22 °F to 122 °F) with PoE, down to -40 °C (-40 °F) with High PoE
Humidity	Humidity 10–100% RH (condensing)

Inside vs. outside temperature

Humidity

Failures

Date	Type	Duration	Site
2013-10-18	Fan failure	2 days	Novosibirsk
2013-09-23	Power outage	2 days	Novosibirsk
2013-08-20	Power outage	7 days	Novosibirsk
2013-07-14	Fan failure	a day	Novosibirsk
2013-07-12	Fan failure	4 days	Novosibirsk

Figure 6.5: When clicking on one of the sites on the left hand side the camera dashboard filtered the view to only show data from that site. This enlarged the graphs to provide more details.

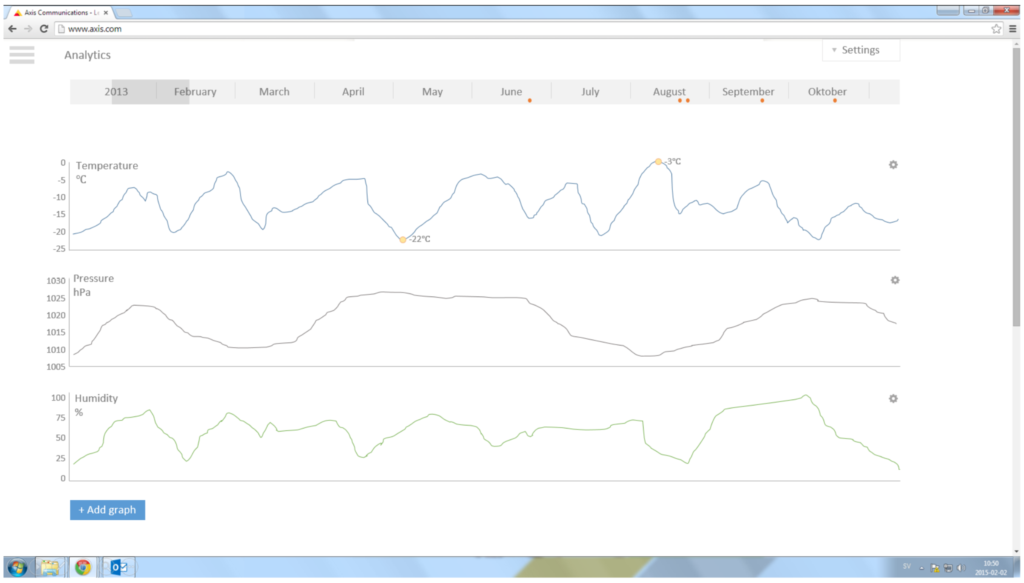
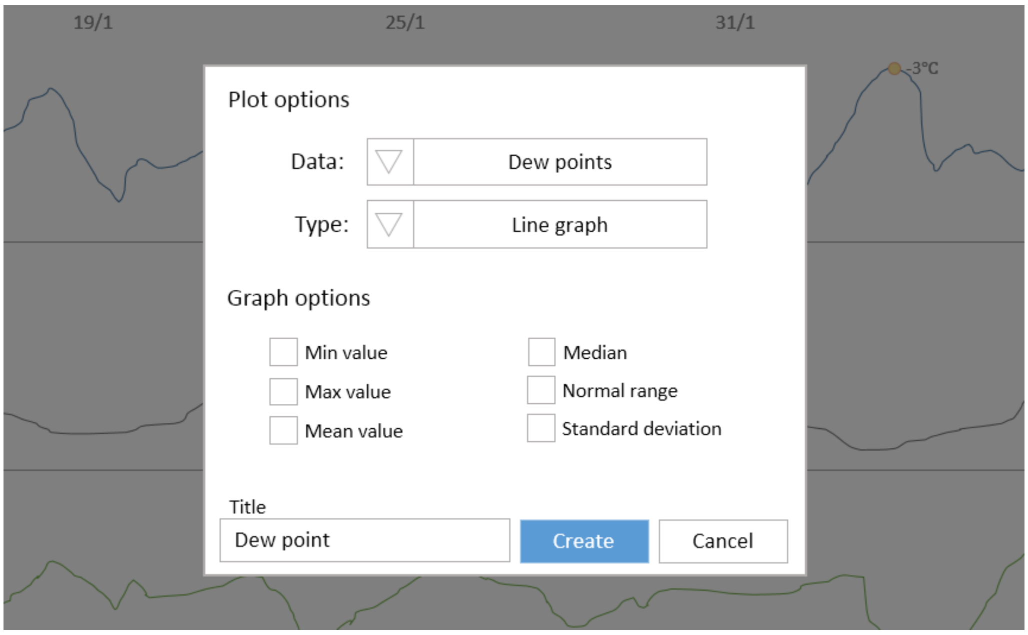


Figure 6.6: The analytics view was very similar to the site view from iteration 1 (see figure 5.9), but with the new "time bar" and without the right side checkboxes. Instead we had a button for adding graphs and cogwheel icon for modifying existing graphs. The upper graph also displayed the minimum and maximum values during this time period.



Plot options

Data: Dew points

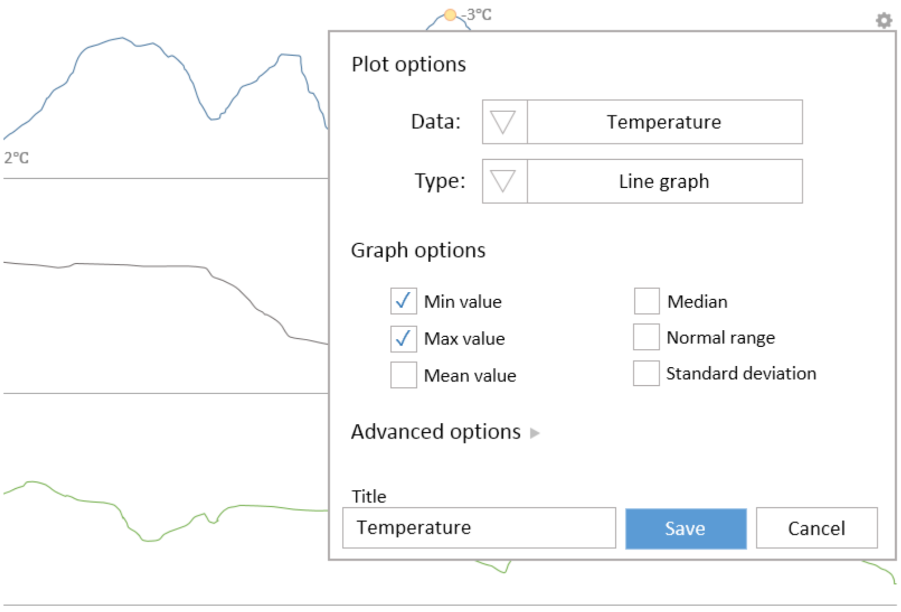
Type: Line graph

Graph options

- [ ] Min value
- [ ] Max value
- [ ] Mean value
- [ ] Median
- [ ] Normal range
- [ ] Standard deviation

Title: Dew point

Figure 6.7: Pressing the add graph button brought up a add graph dialog in the center of the screen.



Plot options

Data: Temperature

Type: Line graph

Graph options

- [✓] Min value
- [✓] Max value
- [ ] Mean value
- [ ] Median
- [ ] Normal range
- [ ] Standard deviation

Advanced options ▸

Title: Temperature

Figure 6.8: Clicking on the cogwheel icon brought up a dialog where graph modifications can be done.

6.1.2 Test

When the prototype was done we performed a user test with one participant (not the same user as in the previous test). The same interview guide was used as in the previous test. The following problems were found:

- There were some confusion regarding the meaning of the term “availability”.
- It was unclear as to what was filtered by the “time bar”.
- The user did not realize that filtering on site was possible on the camera dashboard.
- The user did not use the menu, but instead tried clicking directly in the interface.
- The user wanted to plot solar radiation in comparison to temperature in the camera dashboard.
- It was unclear when and how often the images were captured.
- The user tried to click directly on a row in the failure table to see further details about that particular failure, such as camera information.
- The preset label in analytics was unclear. The user confused this word with camera viewing presets.
- Information was lacking on when the site/camera was installed.
- The user wanted to write notes/comments directly in the interface to view on the different dashboards, e.g. “maintenance performed 14-02-17”.

Another idea that came up during the interview was the notion of a “start” or “about” view. This view would be an introduction to the Live Sites, how they work, where they are located, where the data is stored etcetera. We quite liked this idea. It could be a good way to explain the conceptual model and introduce the Live Sites to a new user. We added this idea as the problem for the next design sprint.

6.2 Implementation sprint

6.2.1 Planning

We listed the following goals of this sprint:

- Change the “event bar” to the more concise “time bar”.

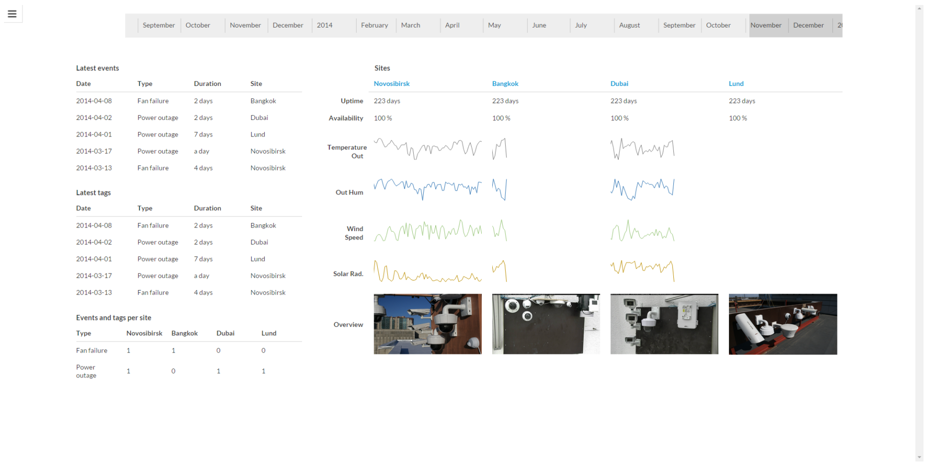
- Build the site dashboard.
- Build the camera dashboard.
- Change site view to analytics.
- Add graph options in analytics view.

6.2.2 Coding

At this early stage of the project parts of our code had already started to become messy, especially the graph drawing components. Thus we decided to refactor and re-write parts of it. This took a significant amount of time, but did not impact the interface except for minor performance gains. Code quality did however improve which should make it easier to make future changes and customizations to the code. The time spent on this should save us time when we get further into the project.

6.2.3 Result

The results of the second implementation iteration is shown in figures 6.9 to 6.14.



Latest events

Date	Type	Duration	Site
2014-04-08	Fan failure	2 days	Bangkok
2014-04-02	Power outage	2 days	Dubai
2014-04-01	Power outage	7 days	Lund
2014-03-17	Power outage	a day	Novosibirsk
2014-03-13	Fan failure	4 days	Novosibirsk

Latest tags

Date	Type	Duration	Site
2014-04-08	Fan failure	2 days	Bangkok
2014-04-02	Power outage	2 days	Dubai
2014-04-01	Power outage	7 days	Lund
2014-03-17	Power outage	a day	Novosibirsk
2014-03-13	Fan failure	4 days	Novosibirsk

Events and tags per site

Type	Novosibirsk	Bangkok	Dubai	Lund
Fan failure	1	1	0	0
Power outage	1	0	1	1

Sites

	Novosibirsk	Bangkok	Dubai	Lund
Uptime	223 days	223 days	223 days	223 days
Availability	100 %	100 %	100 %	100 %

Figure 6.9: The overview dashboard.

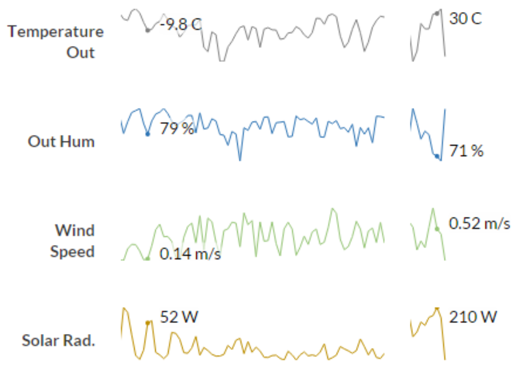
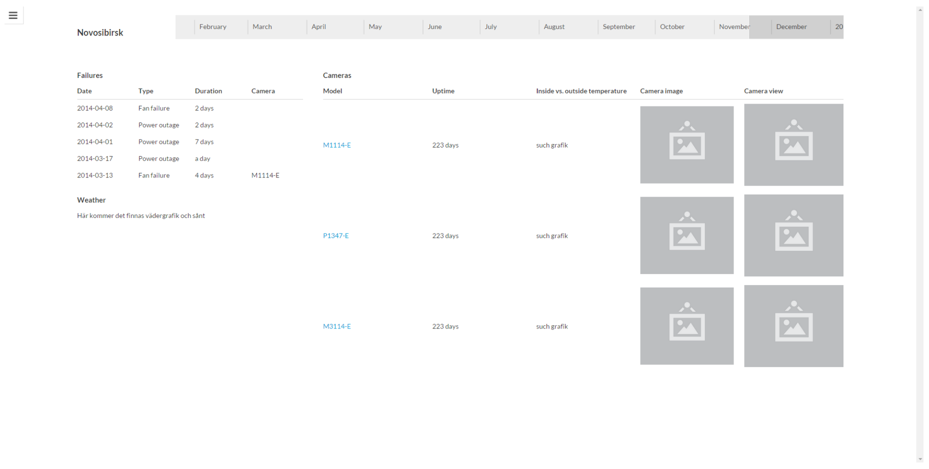


Figure 6.10: Hovering the sparklines with the cursor showed the values for the data at that point in time.



Novosibirsk

Failures

Date	Type	Duration	Camera
2014-04-08	Fan failure	2 days	
2014-04-02	Power outage	2 days	
2014-04-01	Power outage	7 days	
2014-03-17	Power outage	a day	
2014-03-13	Fan failure	4 days	M1114-E

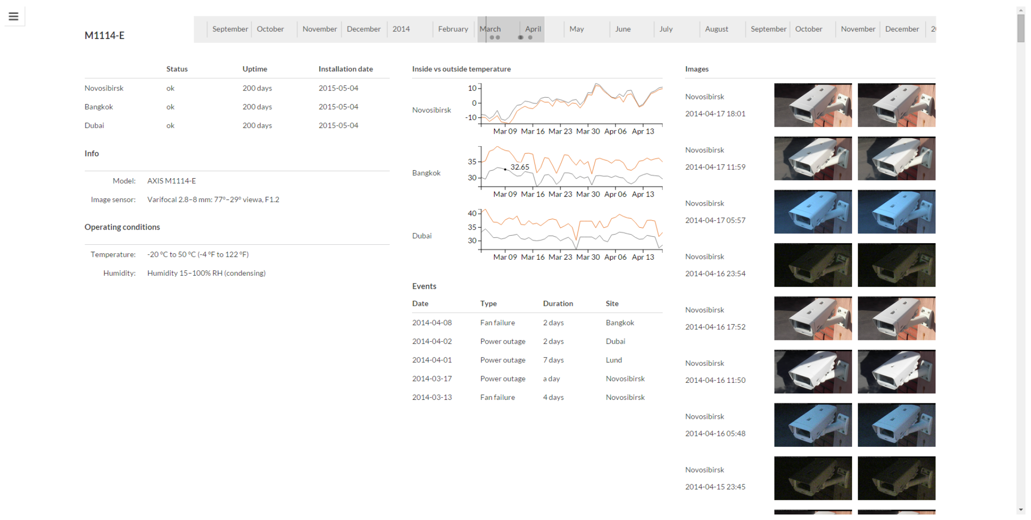
Weather

Här kommer det finnas vädergrafik och sånt

Cameras

Model	Uptime	Inside vs. outside temperature	Camera image	Camera view
M1114-E	223 days	such grafik		
P1347-E	223 days	such grafik		
M3114-E	223 days	such grafik		

Figure 6.11: The site dashboard lacked graphs and showed placeholders instead of the most recent image from each of the cameras. This would be corrected in following iterations.



M1114-E

	Status	Uptime	Installation date
Novosibirsk	ok	200 days	2015-05-04
Bangkok	ok	200 days	2015-05-04
Dubai	ok	200 days	2015-05-04

Info

Model:	AXIS M1114-E
Image sensor:	Varifocal 2.8–8 mm: 77°–29° viewa, F1.2

Operating conditions

Temperature:	-20 °C to 50 °C (-4 °F to 122 °F)
Humidity:	Humidity 15–100% RH (condensing)

Events

Date	Type	Duration	Site
2014-04-08	Fan failure	2 days	Bangkok
2014-04-02	Power outage	2 days	Dubai
2014-04-01	Power outage	7 days	Lund
2014-03-17	Power outage	a day	Novosibirsk
2014-03-13	Fan failure	4 days	Novosibirsk

Novosibirsk
2014-04-17 18:01

Novosibirsk
2014-04-17 11:59

Novosibirsk
2014-04-17 05:57

Novosibirsk
2014-04-16 23:54

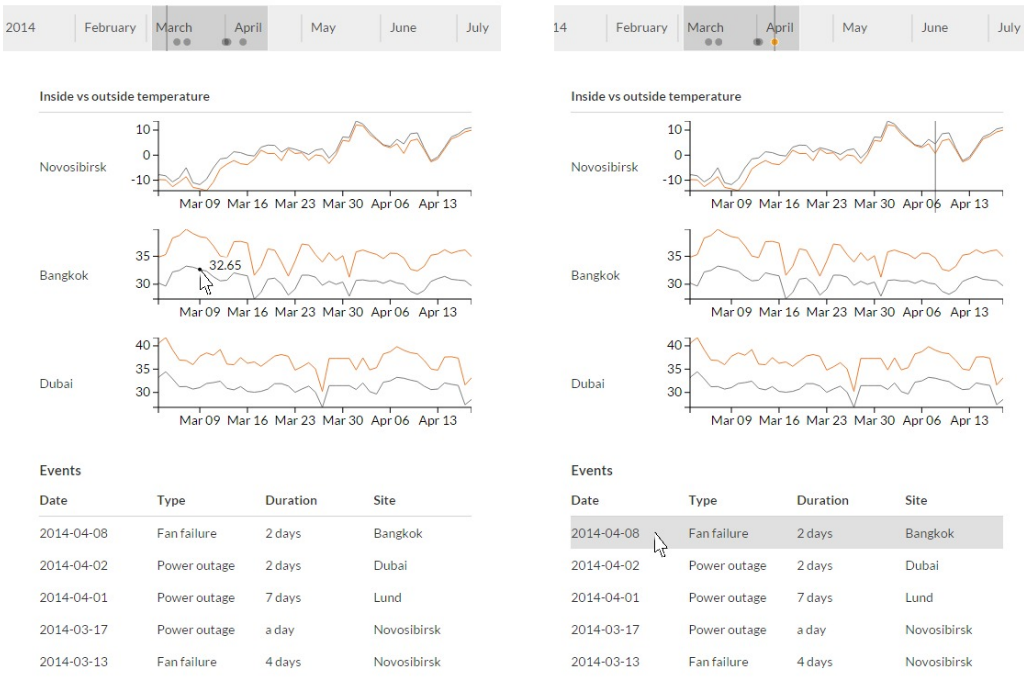
Novosibirsk
2014-04-16 17:52

Novosibirsk
2014-04-16 11:50

Novosibirsk
2014-04-16 05:48

Novosibirsk
2014-04-15 23:45

Figure 6.12: The camera dashboard was already quite functional. It contained all the elements from the design prototype except for the humidity graphs. It was however not possible to filter the data to show only one site.



Events

Date	Type	Duration	Site
2014-04-08	Fan failure	2 days	Bangkok
2014-04-02	Power outage	2 days	Dubai
2014-04-01	Power outage	7 days	Lund
2014-03-17	Power outage	a day	Novosibirsk
2014-03-13	Fan failure	4 days	Novosibirsk

Figure 6.13: When moving the cursor over a chart (left image) or an event (right image) that date was meant to be highlighted in the surrounding elements. It did not work yet for all elements.

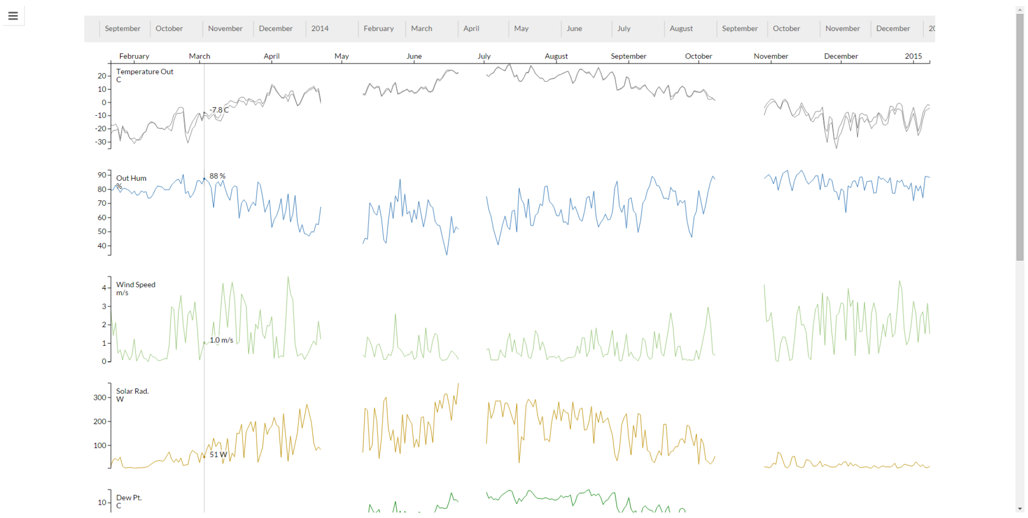


Figure 6.14: We were able to build a mostly functioning analytics view. The grid lines were removed when refactoring the code but are planned to be added again.

6.3 Evaluation

6.3.1 Heuristic evaluation

To help us evaluate and find new problems in our design we decided to enlist the help of a user experience expert from Axis. During this evaluation we let the expert use the interface and talk aloud while we took notes. The following problems were found:

- The images were too distracting and interrupt the flow. The user's eyes would be drawn to the images, therefore the images should mark the starting point of each entry of information.
- The font used in the charts was too prominent. Make it smaller.
- The start of a year in the "time bar" should be printed above or below the graph to be found easily.
- Give every image the same aspect ratio (16:9) by adding black background around those who have a different aspect ratio.
- The interface could guide the user's interaction flow better by using carefully chosen font sizes and adding shaded boxes.

Chapter 7

Iteration 3 - Refactoring

In this iteration we still had a long list of implementation issues left from the previous iteration. We therefore decided on a very short design sprint and instead focus our time on solving code related problems (see figure 7.1).

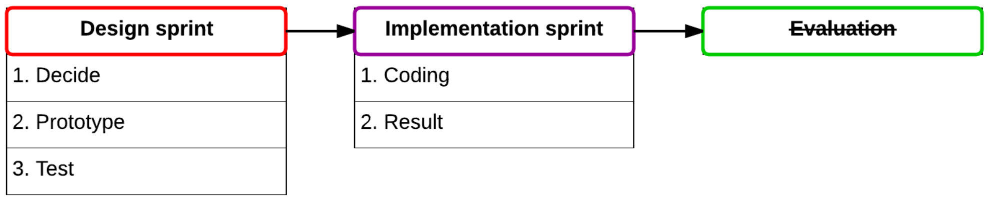


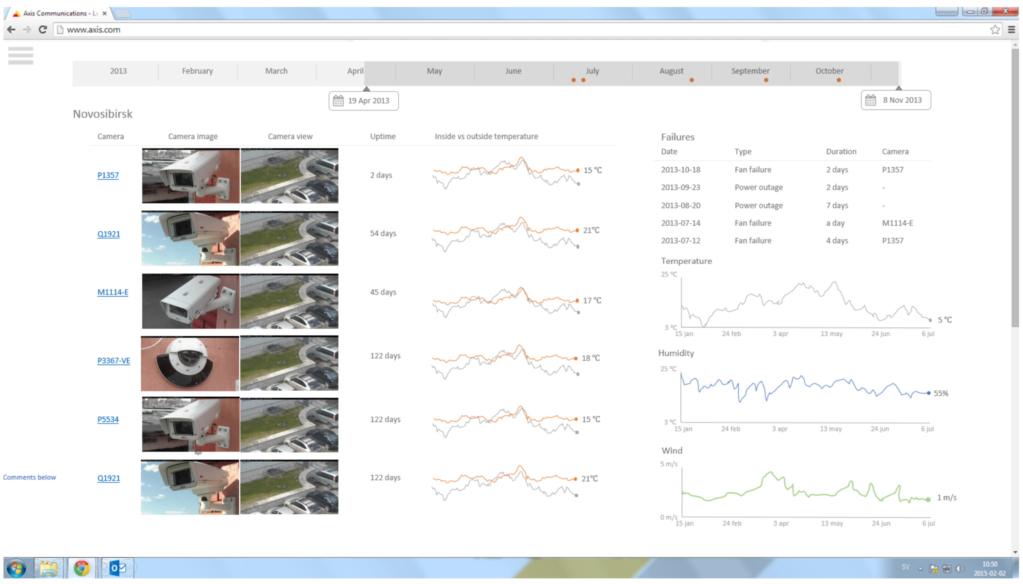
Figure 7.1: Iteration 3

7.1 Design sprint

To shorten this design sprint we decided to postpone any major design questions for a later design sprint and instead just make small adjustments to the prototype.

7.1.1 Prototype

The aim of this very short sprint was to address the issues of interaction flow found in evaluation phase 2. A user typically scans web interfaces in an F-shaped motion. Starting in the top left corner and moving to the right, followed by a sweeping motion from the middle left to the right. Then from top left to the bottom left corner[46]. Our previous design interrupted this normal flow by having distracting images to the right. To solve this we simply moved all images in the different dashboards to the left (see e.g. figure 7.2). Graphs were moved to the center and the tables were moved to the right. To make the tables less emphasized we put them in a grey segment box.



Novosibirsk

Camera	Camera image	Camera view	Uptime	Inside vs outside temperature
P1357			2 days	15 °C
Q1921			54 days	21°C
M1114-E			45 days	17 °C
P3367-VE			122 days	18 °C
P5534			122 days	15 °C
Q1921			122 days	21°C

Failures

Date	Type	Duration	Camera
2013-10-18	Fan failure	2 days	P1357
2013-09-23	Power outage	2 days	-
2013-08-20	Power outage	7 days	-
2013-07-14	Fan failure	a day	M1114-E
2013-07-12	Fan failure	4 days	P1357

Figure 7.2: The camera dashboard when the image column has been moved to the left.

7.1.2 Test

We chose not to run any test to validate these adjustments since we felt that they were clear improvements. Trying to improve the interaction flow might sound like a major undertaking, but our solution was a quick fix. Hence we decided to skip testing for now, to not waste time. We would have preferred to run tests to evaluate our solution. However, since we had bigger problems to take care of regarding implementation we decided to prioritize coding. It does not matter how well designed and tested an interface is if the implementation does not work. These changes would instead be tested in a later usability test.

7.2 Implementation sprint

We started refactoring large parts of the code during iteration 2, but were able to finish. The majority of this implementation sprint was spent finishing up the work started in iteration 2 and doing minor updates from design sprint 3.

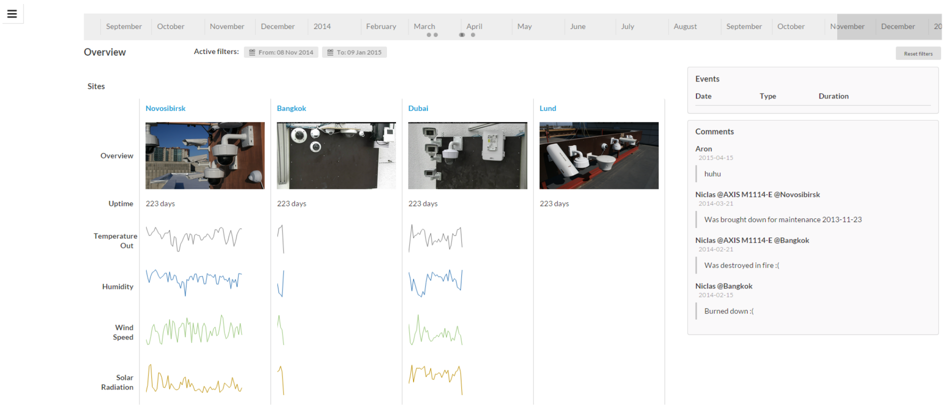
7.2.1 Planning

- Refactor database.
- Refactor back end.

- Continue to refactor the graph drawing component.
- Implement graph adding functionality in the analytics view.
- Update the different views to match the prototype.
- Create a program that can get images and logs from the site cameras.
- Create a program that can parse the camera images and logs into the database.
- Add commenting functionality to the dashboards as presented in section 6.1.2.

7.2.2 Result

The results of the implementation phase is shown in figures 7.3 to 7.14



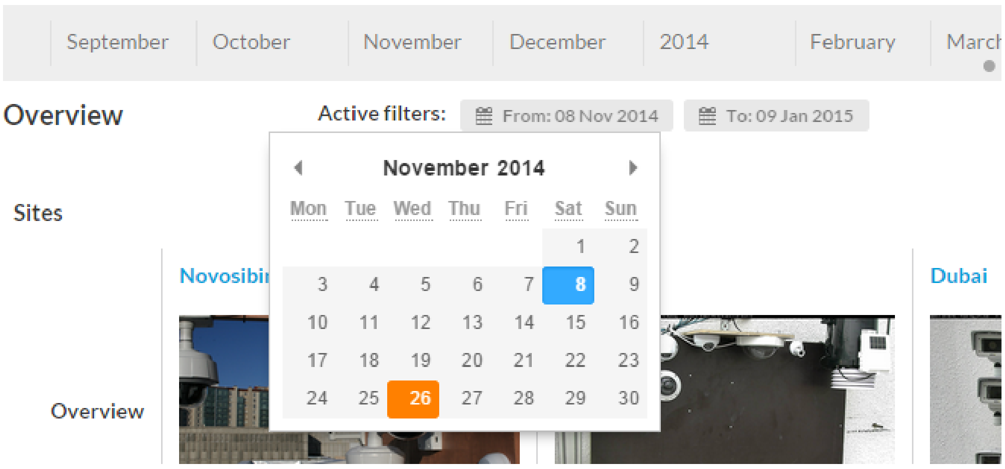
Aron
2015-04-15
huhu

Niclas @AXIS M1114-E @Novosibirsk
2014-03-21
Was brought down for maintenance 2013-11-23

Niclas @AXIS M1114-E @Bangkok
2014-02-21
Was destroyed in fire :(

Niclas @Bangkok
2014-02-15
Burned down :(

Figure 7.3: The events were moved to the right in the overview dashboard.

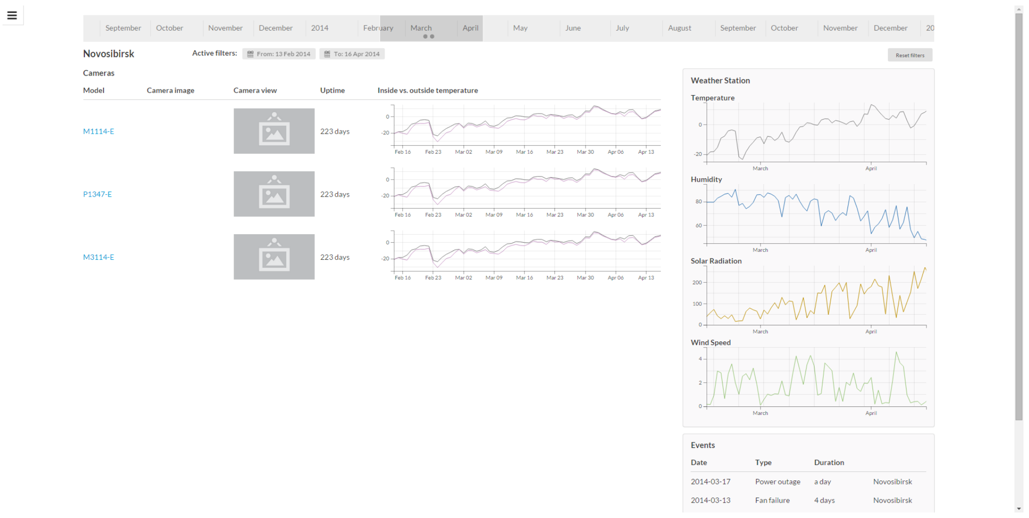


Active filters: From: 08 Nov 2014 To: 09 Jan 2015

November 2014

Mon	Tue	Wed	Thu	Fri	Sat	Sun
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

Figure 7.4: To know what filters were active, we added an active filters bar. Clicking the dates opened a date picker.



Novosibirsk

Active filters: From: 13 Feb 2014 To: 16 Apr 2014

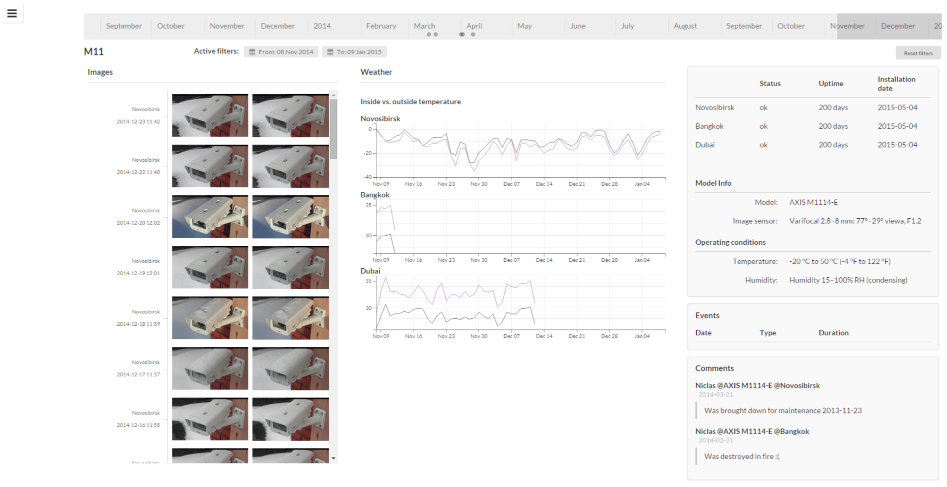
Cameras

Model	Camera image	Camera view	Uptime	Inside vs. outside temperature
M1114-E			223 days	
P1347-E			223 days	
M3114-E			223 days	

Events

Date	Type	Duration	
2014-03-17	Power outage	a day	Novosibirsk
2014-03-13	Fan failure	4 days	Novosibirsk

Figure 7.5: The layout of the site dashboard was also flipped so the camera table is on the left.



M11

Active filters: From: 08 Nov 2014 To: 09 Jan 2015

	Status	Uptime	Installation date
Novosibirsk	ok	200 days	2015-05-04
Bangkok	ok	200 days	2015-05-04
Dubai	ok	200 days	2015-05-04

Model Info

Model: AXIS M1114-E

Image sensor: Varifocal 2.8–8 mm: 77°–29° viewa, F1.2

Operating conditions

Temperature: -20 °C to 50 °C (-4 °F to 122 °F)

Humidity: Humidity 15–100% RH (condensing)

Events

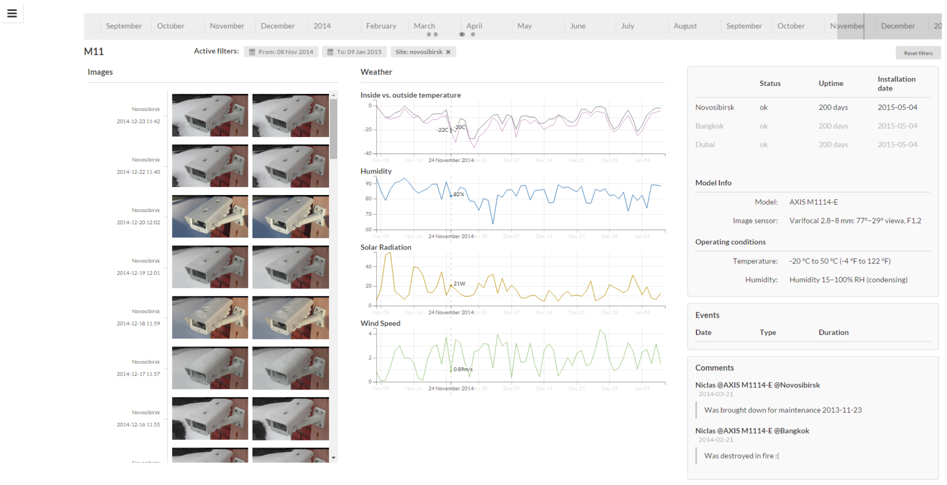
Date	Type	Duration

Comments

Niclas @AXIS M1114-E @Novosibirsk
2014-03-21
Was brought down for maintenance 2013-11-23

Niclas @AXIS M1114-E @Bangkok
2014-02-21
Was destroyed in fire :(

Figure 7.6: The camera dashboard gained a commenting box in the lower right corner.



M11

Active filters: From: 08 Nov 2014 To: 09 Jan 2015 Site: novosibirsk

	Status	Uptime	Installation date
Novosibirsk	ok	200 days	2015-05-04
Bangkok	ok	200 days	2015-05-04
Dubai	ok	200 days	2015-05-04

Model Info

Model: AXIS M1114-E

Image sensor: Varifocal 2.8–8 mm: 77°–29° viewa, F1.2

Operating conditions

Temperature: -20 °C to 50 °C (-4 °F to 122 °F)

Humidity: Humidity 15–100% RH (condensing)

Events

Date	Type	Duration

Comments

Niclas @AXIS M1114-E @Novosibirsk
2014-03-21
Was brought down for maintenance 2013-11-23

Niclas @AXIS M1114-E @Bangkok
2014-02-21
Was destroyed in fire :(

Figure 7.7: Filtering on a specific site was now possible. Here showing only the camera in Novosibirsk.

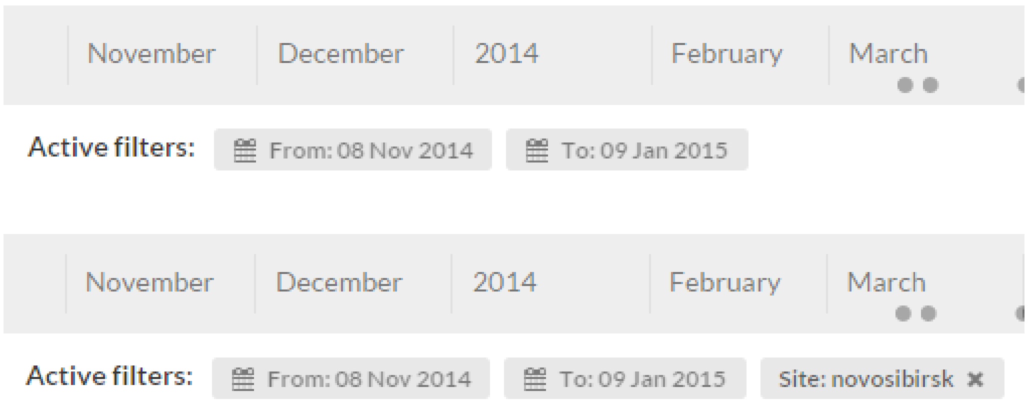


Figure 7.8: A site filter was added to the active filter bar when filtering on a site. It could be removed by pressing the X-button.

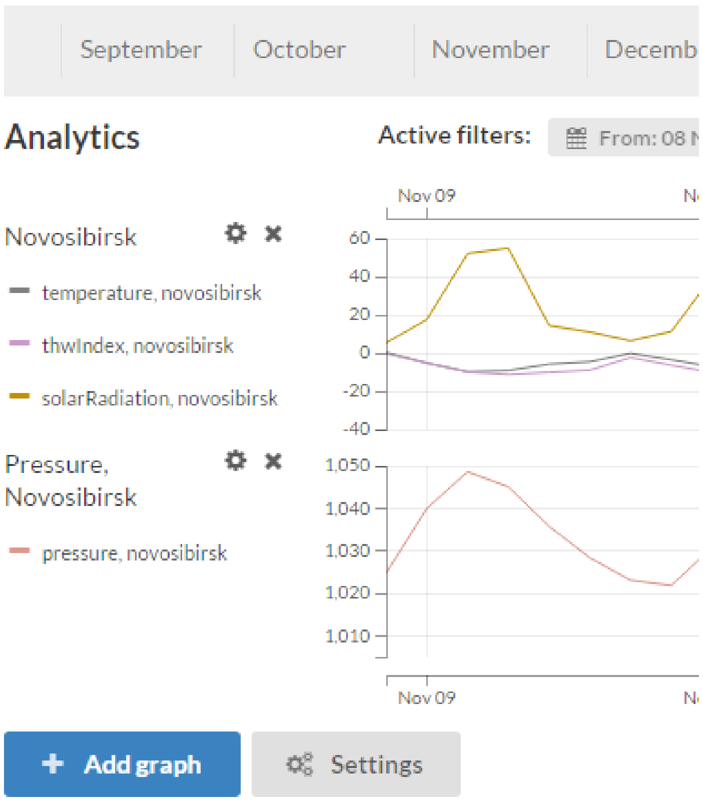


Figure 7.9: On the analytics view, each graph got an info box to the left that shows what measures are plotted and their colors.

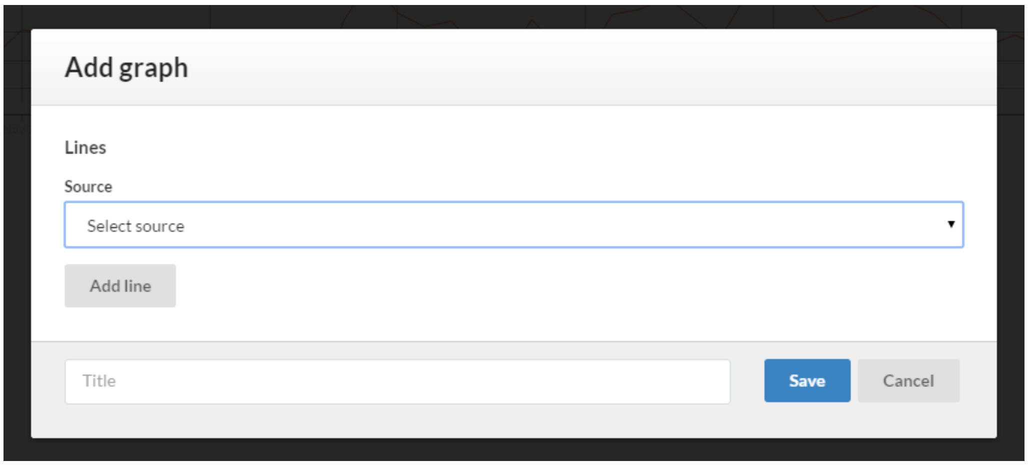


Figure 7.10: Clicking the add graph button displayed an add graph dialog. The user started by selecting the source of the data they wanted to plot (e.g. weather station or camera). Currently, only line graphs are supported.

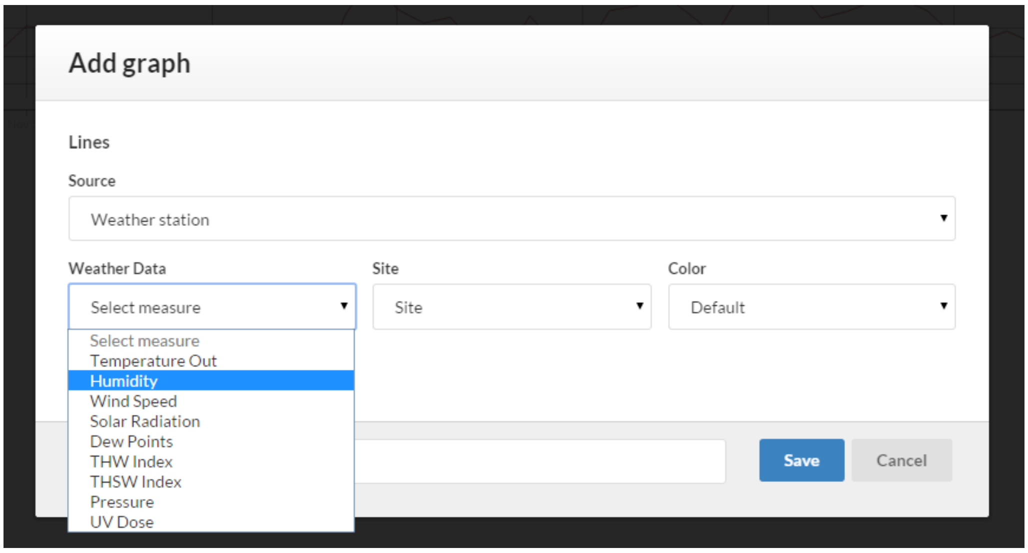
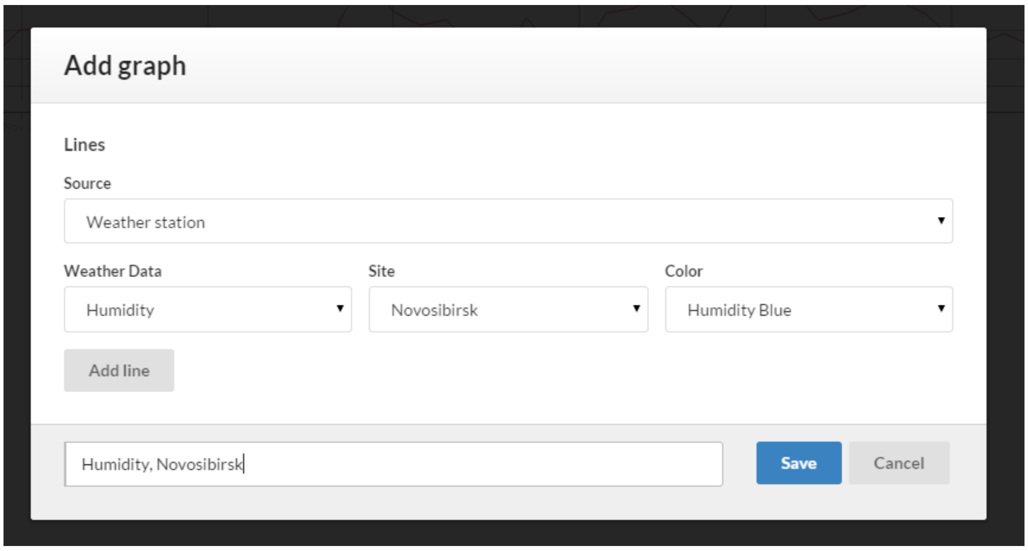


Figure 7.11: Selecting weather data from the source dropdown list displayed measure, site and color dropdowns.


Add graph

Lines

Source: Weather station

Weather Data: Humidity

Site: Novosibirsk

Color: Humidity Blue

Humidity, Novosibirsk

Figure 7.12: The user completed filling in the selections, gave the graph a title and clicked save.

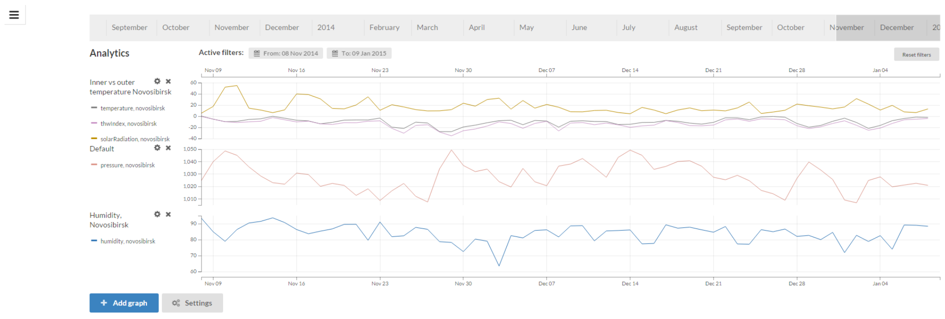


Figure 7.13: The new graph was then added to the charts.

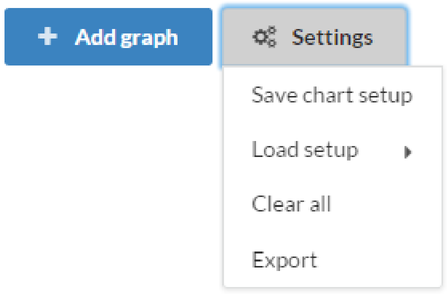


Figure 7.14: In the settings menu, the user would be able to save the current graph setup, load presets or previously saved setups, remove all graphs and export the currently displayed data. However, these features are not yet fully functional.

7.3 Evaluation

Since most of the work done during this iteration did not impact the user interface we decided to skip evaluation in this iteration. The changes that were done to the interface, moving images and such, would instead be evaluated as part of design sprint 4.

Chapter 8

Iteration 4 - Provide help to users

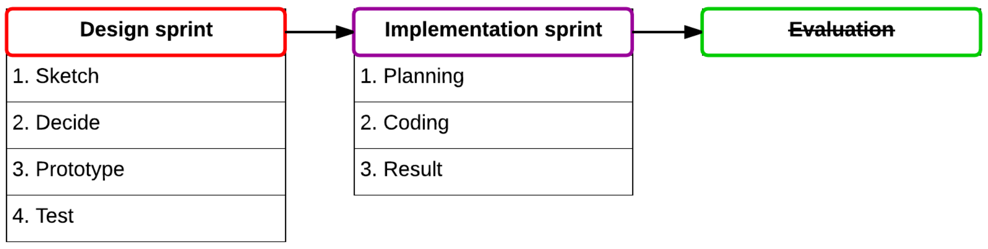


Figure 8.1: Iteration 4

8.1 Design sprint

In our previous testing and heuristic evaluations we recognized a pattern. The dashboards themselves was not enough to introduce new users and there was no help documentation. We came up with the idea of a start page during a user test in iteration 2 and decided to explore this idea. Earlier testing had also revealed that users were very hesitant towards opening the menu and instead chose to click directly on the dashboards. To solve these issues, we formulated these three questions to answer in this sprint:

1. *How might we provide a smoother introduction for new users?*
2. *How might we provide assistance for users who need help?*
3. *How might we improve navigation?*

8.1.1 Sketch and decide

Since the deadline of the project was fast approaching we decided to focus on really simple solutions to these three problems.

The introduction problem

During sketching we initially thought of solving this by adding a first-time guide. First-time users would be greeted by a guide/tutorial overlay on every page. This overlay would explain the different parts, and by doing so letting the user know exactly what every piece of the interface could do. Unfortunately, we felt this would require a lot of work and simply did not have the time.

After some sketching we came up with a much simpler solution. We could add a new start page to our application that would provide introductory information and help the user get started. Adding a new start page would be technically simple and could help alleviate the problem. The start page would not explain in detail how everything works. We reasoned that, while this simple idea might not be the best solution, it could be good enough. We chose to go for this simple option.

The menu

Our initial design had the menu hidden behind a "hamburger" button. This abides by the dashboard design principle of allowing the dashboard to take up all the screen space to keep the user focused on data analysis. However, throughout our user testing sessions the testers did not find the menu. Navigation often became a problem, e.g. when a user was given the task to analyze and compare different climate parameters to camera data, the user would only search for the solution on the dashboard instead of navigating to the "Analytics" view. Here the idiom "out of sight, out of mind" applied. The user did not directly see the option to navigate to the "Analytics" view and opening the menu was not the first thing the user thought of. An expert user might learn to open the menu, but this puts more strain on the user's memory. The user has to remember to open the menu. This might not sound like a big issue, maybe one could expect a user to know how to open a menu, but it is one more step, one more thing to remember.

The other option would be to have the menu visible at all times. This would solve the "out of sight, out of mind" problem, but this could harm the data analysis part. Always seeing the options would evoke the user to explore the different views instead of staying focused on the task of analyzing what the dashboard has to say. Following this reasoning we decided to try the always visible menu, but made it small and concise to avoid stealing the users attention (see figure 8.2).

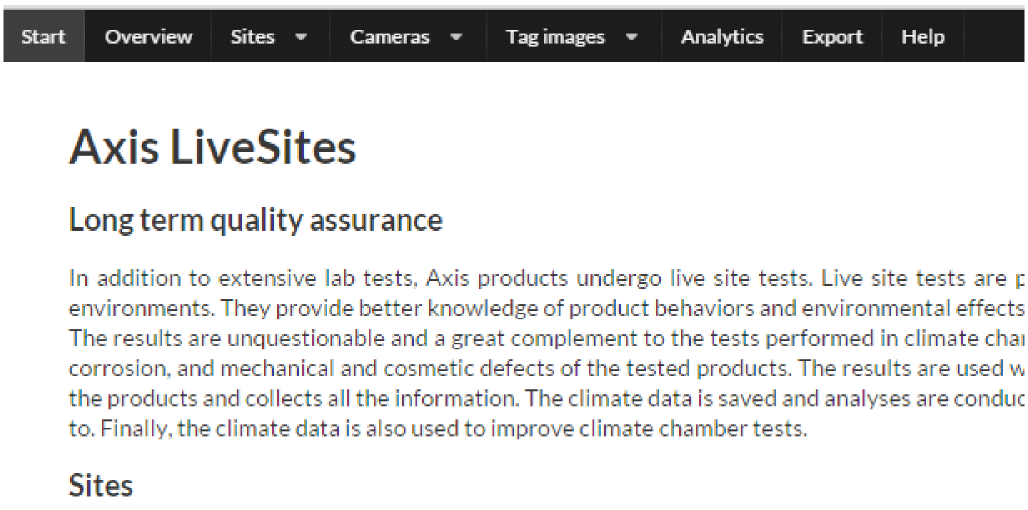
The help problem

The easiest way we could think of to add help functionality was to add a help page to the application. The help page would contain short explanations of the different views and inner workings of the Live Site system. The problem with having a single help page was that it forces the user to leave her current task and enter a completely new view. This was not optimal since the user would lose the connection to what he/she needed help with. The simplest way to solve this would be to force the help view to be opened in a new window or tab in the browser, but this was not optimal either. Forcing a new tab or window could be annoying for the user who wants to feel in control, and it would not follow the consistency of the other links in the menu. To solve this we decided to keep the static help page, but to also add a question mark button to the top right of the menu that would show the help information for that specific view when clicked. This provided the user with help when and where he/she needs it.

8.1.2 Prototype

We chose to skip the PowerPoint prototyping and add the pages directly to the product, to not waste any time. Since these pages were technically simple, this was the easiest and fastest way. We also knew that we needed to run usability tests on the additions from iteration 3. These usability tests would give better results if we used the actual product instead of the PowerPoint prototype.

The results of the prototyping can be seen in figures 8.2 to 8.5.



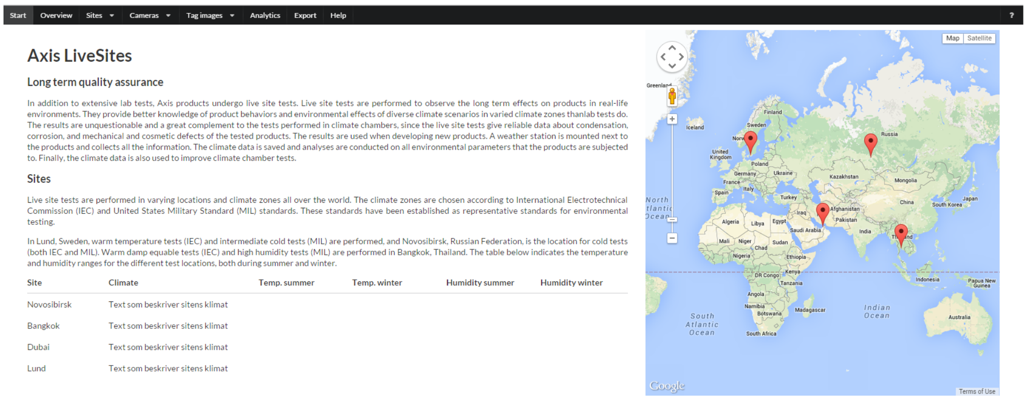
Axis LiveSites

Long term quality assurance

In addition to extensive lab tests, Axis products undergo live site tests. Live site tests are p
environments. They provide better knowledge of product behaviors and environmental effects
The results are unquestionable and a great complement to the tests performed in climate chai
corrosion, and mechanical and cosmetic defects of the tested products. The results are used w
the products and collects all the information. The climate data is saved and analyses are conduc
to. Finally, the climate data is also used to improve climate chamber tests.

Sites

Figure 8.2: The new menu. When a page exceeded the scroll height of the screen the menu sticks to the top of the page and was visible when scrolling.



Axis LiveSites

Long term quality assurance

In addition to extensive lab tests, Axis products undergo live site tests. Live site tests are performed to observe the long term effects on products in real-life environments. They provide better knowledge of product behaviors and environmental effects of diverse climate scenarios in varied climate zones thanlab tests do. The results are unquestionable and a great complement to the tests performed in climate chambers, since the live site tests give reliable data about condensation, corrosion, and mechanical and cosmetic defects of the tested products. The results are used when developing new products. A weather station is mounted next to the products and collects all the information. The climate data is saved and analyses are conducted on all environmental parameters that the products are subjected to. Finally, the climate data is also used to improve climate chamber tests.

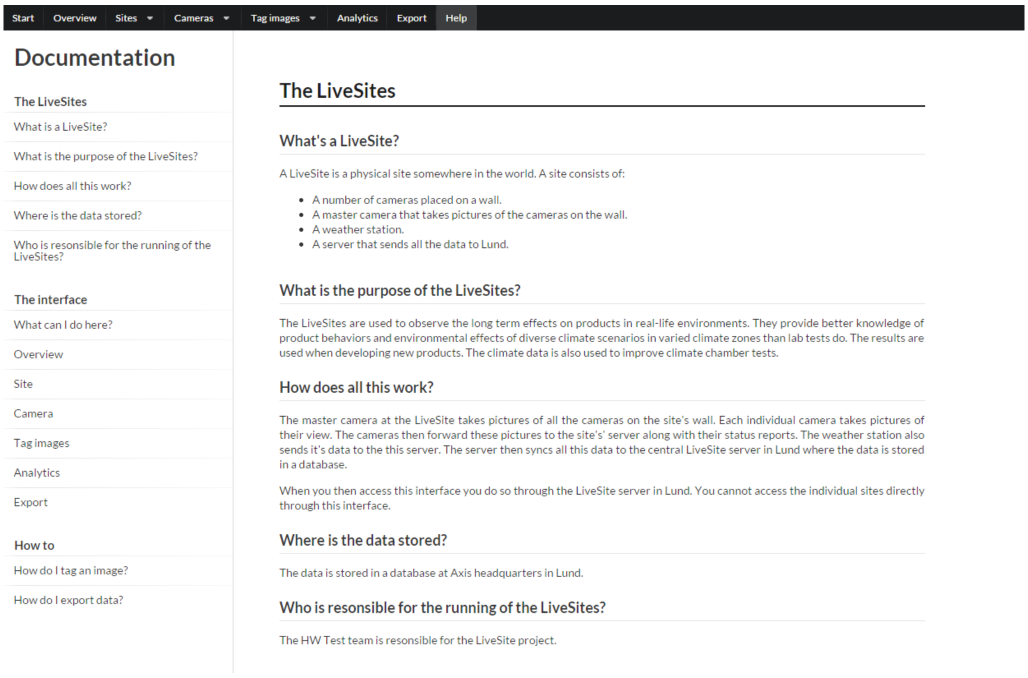
Sites

Live site tests are performed in varying locations and climate zones all over the world. The climate zones are chosen according to International Electrotechnical Commission (IEC) and United States Military Standard (MIL) standards. These standards have been established as representative standards for environmental testing.

In Lund, Sweden, warm temperature tests (IEC) and intermediate cold tests (MIL) are performed, and Novosibirsk, Russian Federation, is the location for cold tests (both IEC and MIL). Warm damp equable tests (IEC) and high humidity tests (MIL) are performed in Bangkok, Thailand. The table below indicates the temperature and humidity ranges for the different test locations, both during summer and winter.

Site	Climate	Temp. summer	Temp. winter	Humidity summer	Humidity winter
Novosibirsk	Text som beskriver sitens klimat				
Bangkok	Text som beskriver sitens klimat				
Dubai	Text som beskriver sitens klimat				
Lund	Text som beskriver sitens klimat				

Figure 8.3: The start page. The view consisted of an introductory text explaining the concept of the Live Sites. Below this followed a table where every site was described. On the right was a map that marked the geographical locations of the Live Sites.



Documentation

The LiveSites
- What is a LiveSite?
- What is the purpose of the LiveSites?
- How does all this work?
- Where is the data stored?
- Who is resonsible for the running of the LiveSites?

The interface
- What can I do here?
- Overview
- Site
- Camera
- Tag images
- Analytics
- Export

How to
- How do I tag an image?
- How do I export data?

The LiveSites

What's a LiveSite?

A LiveSite is a physical site somewhere in the world. A site consists of:

- A number of cameras placed on a wall.
- A master camera that takes pictures of the cameras on the wall.
- A weather station.
- A server that sends all the data to Lund.

What is the purpose of the LiveSites?

The LiveSites are used to observe the long term effects on products in real-life environments. They provide better knowledge of product behaviors and environmental effects of diverse climate scenarios in varied climate zones than lab tests do. The results are used when developing new products. The climate data is also used to improve climate chamber tests.

How does all this work?

The master camera at the LiveSite takes pictures of all the cameras on the site's wall. Each individual camera takes pictures of their view. The cameras then forward these pictures to the site's' server along with their status reports. The weather station also sends it's data to the this server. The server then syncs all this data to the central LiveSite server in Lund where the data is stored in a database.

When you then access this interface you do so through the LiveSite server in Lund. You cannot access the individual sites directly through this interface.

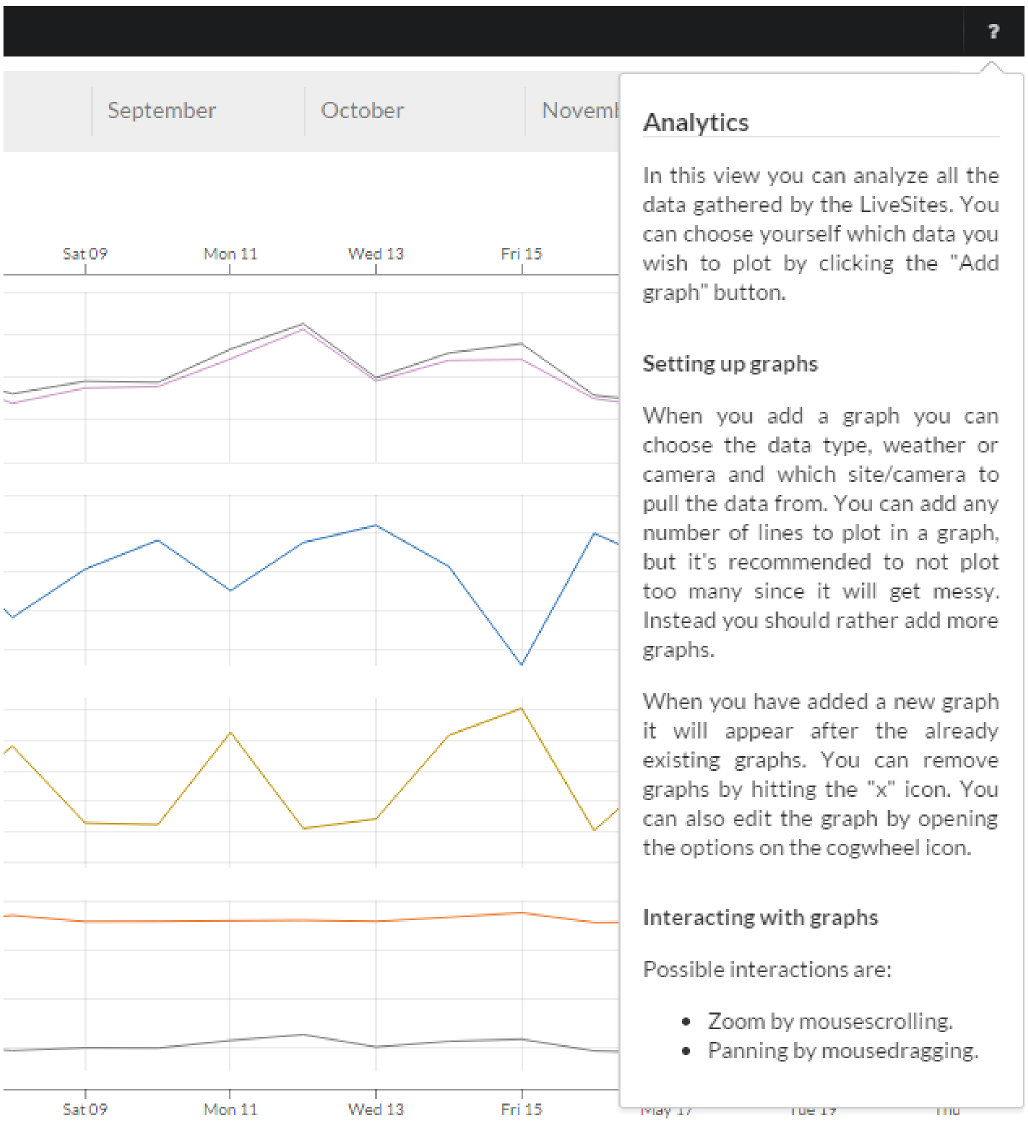
Where is the data stored?

The data is stored in a database at Axis headquarters in Lund.

Who is resonsible for the running of the LiveSites?

The HW Test team is resonsible for the LiveSite project.

Figure 8.4: The help view. To the left was a menu containing all the help entries. The rest of the view was occupied by the help texts. When an item was clicked on in the menu the view animated that item to the top of the view.



Analytics

In this view you can analyze all the data gathered by the LiveSites. You can choose yourself which data you wish to plot by clicking the "Add graph" button.

Setting up graphs

When you add a graph you can choose the data type, weather or camera and which site/camera to pull the data from. You can add any number of lines to plot in a graph, but it's recommended to not plot too many since it will get messy. Instead you should rather add more graphs.

When you have added a new graph it will appear after the already existing graphs. You can remove graphs by hitting the "x" icon. You can also edit the graph by opening the options on the cogwheel icon.

Interacting with graphs

Possible interactions are:

- Zoom by mousescrolling.
- Panning by mousedragging.

Figure 8.5: The contextual help popup as viewed in the analytics view. The popup appeared after clicking the question mark button.

8.1.3 Test

Since a few weeks had gone by since our last usability tests, we chose to run three general usability tests this session. Our reasoning was that we knew the last implementation sprint was coming up and we wanted to get input on as many things as possible. Doing the tests like this would however take focus away from the actual problem we were trying to solve in this design sprint. We recognized this, but went ahead with the tests since we needed feedback to plan for the next implementation sprint. The tests resulted in the following conclusions:

- The map on the start page was appreciated, but the other information presented could be more interesting. Adding live data could make it more interesting and connect it to the rest of the application.
- Navigating with the "time bar" was a recurring problem. The users did not realize it was interactive, but chose to use the date-picker instead.
- A user got confused when the "time bar" did not start at the start of the year but instead shows the last 365 days.
- All test participants tried to click on the images on the dashboard to enlarge and tag them.
- It took a long time for one user to find the menu.
- One user wanted to filter the images by certain weather parameters, e.g. when the temperature was above a certain threshold.
- Unclear if a black camera image meant that the image was missing or was taken during the night.
- None of the users realized the camera view could be filtered by clicking on the rows in the camera table.
- The "Add graph" dialog in the analytics view had some issues. The user clicked on "Add line" instead of "Save" when they felt they were done, which led to much confusion.
- Users seemed more aware of the different navigational options, but were still hesitant to open the different views. The naming of the menu links might have been the cause of this. Users seemed to be unsure what the different words meant and preferred to stay on the familiar pages.

8.2 Implementation sprint

In this last implementation sprint most work was spent on refactoring and fixing bugs. These issues were prioritized to make it easier to hand over the code after concluding this master thesis. The theme of usability not only applies to the interface, but also to the code itself. The customer must be able to improve and continue work on our code after we hand it over.

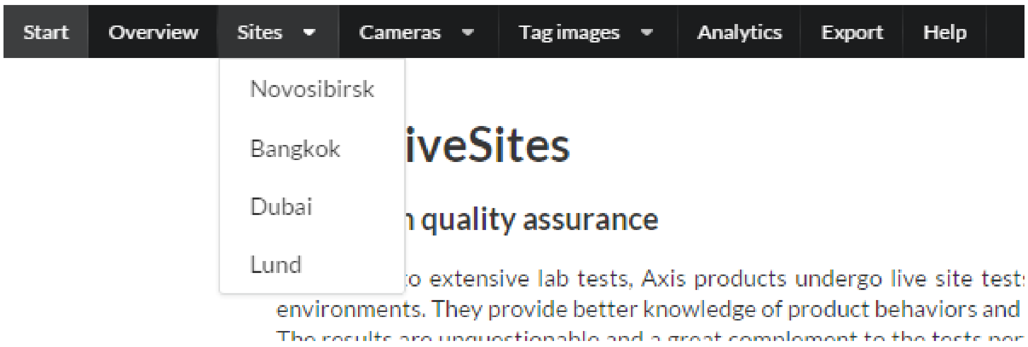
8.2.1 Planning

We planned to complete the following tasks in this sprint:

- General refactoring.
- Fix bugs.
- Refactor data parsing.
- Automate data parsing.
- Implement the data export view.
- Change the “time bar” to make it easier to discover the possible interactions.
- Move the “Add line” button in the “Add graph” module.
- Move the camera filtering in the camera view.

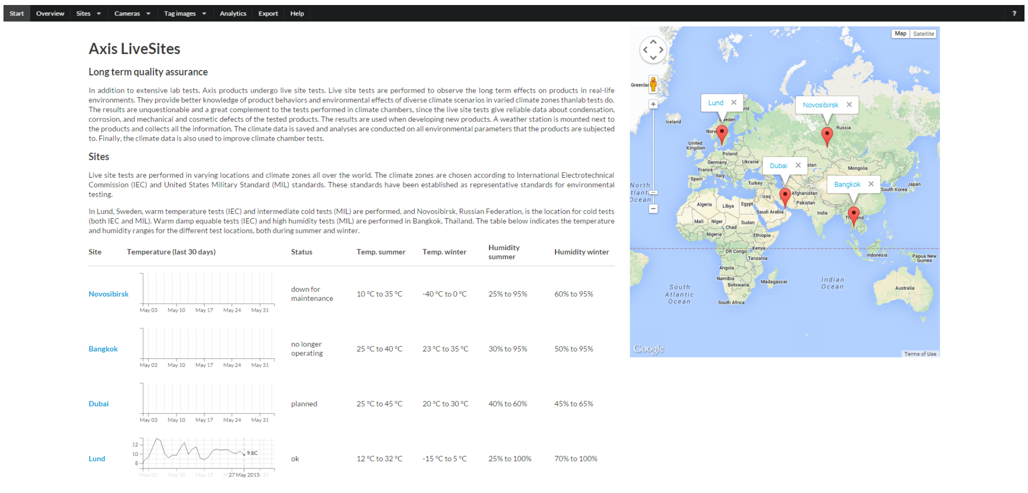
8.2.2 Result

The final result of our project can be seen in figures 8.6 to 8.21.



o extensive lab tests, Axis products undergo live site test
environments. They provide better knowledge of product behaviors and

Figure 8.6: The menu



Axis LiveSites

Long term quality assurance

In addition to extensive lab tests, Axis products undergo live site tests. Live site tests are performed to observe the long term effects on products in real-life environments. They provide better knowledge of product behaviors and environmental effects of diverse climate scenarios in varied climate zones thanlab tests do. The results are unquestionable and a great complement to the tests performed in climate chambers, since the live site tests give reliable data about condensation, corrosion, and mechanical and cosmetic defects of the tested products. The results are used when developing new products. A weather station is mounted next to the products and collects all the information. The climate data is saved and analyses are conducted on all environmental parameters that the products are subjected to. Finally, the climate data is also used to improve climate chamber tests.

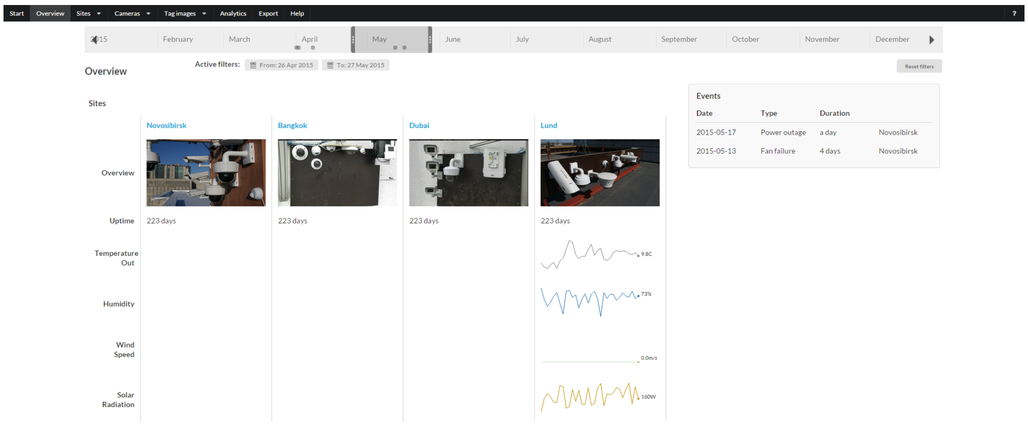
Sites

Live site tests are performed in varying locations and climate zones all over the world. The climate zones are chosen according to International Electrotechnical Commission (IEC) and United States Military Standard (MIL) standards. These standards have been established as representative standards for environmental testing.

In Lund, Sweden, warm temperature tests (IEC) and intermediate cold tests (MIL) are performed, and Novosibirsk, Russian Federation, is the location for cold tests (both IEC and MIL). Warm damp equable tests (IEC) and high humidity tests (MIL) are performed in Bangkok, Thailand. The table below indicates the temperature and humidity ranges for the different test locations, both during summer and winter.

Site	Temperature (last 30 days)	Status	Temp. summer	Temp. winter	Humidity summer	Humidity winter
Novosibirsk	May 03 May 10 May 17 May 24 May 31	down for maintenance	10 °C to 35 °C	-40 °C to 0 °C	25% to 95%	60% to 95%
Bangkok	May 03 May 10 May 17 May 24 May 31	no longer operating	25 °C to 40 °C	23 °C to 35 °C	30% to 95%	50% to 95%
Dubai	May 03 May 10 May 17 May 24 May 31	planned	25 °C to 45 °C	20 °C to 30 °C	40% to 60%	45% to 65%
Lund	12 10 8 9.8C 27 May 2015	ok	12 °C to 32 °C	-15 °C to 5 °C	25% to 100%	70% to 100%

Figure 8.7: The start page



Overview

Active filters: From: 26 Apr 2015 To: 27 May 2015

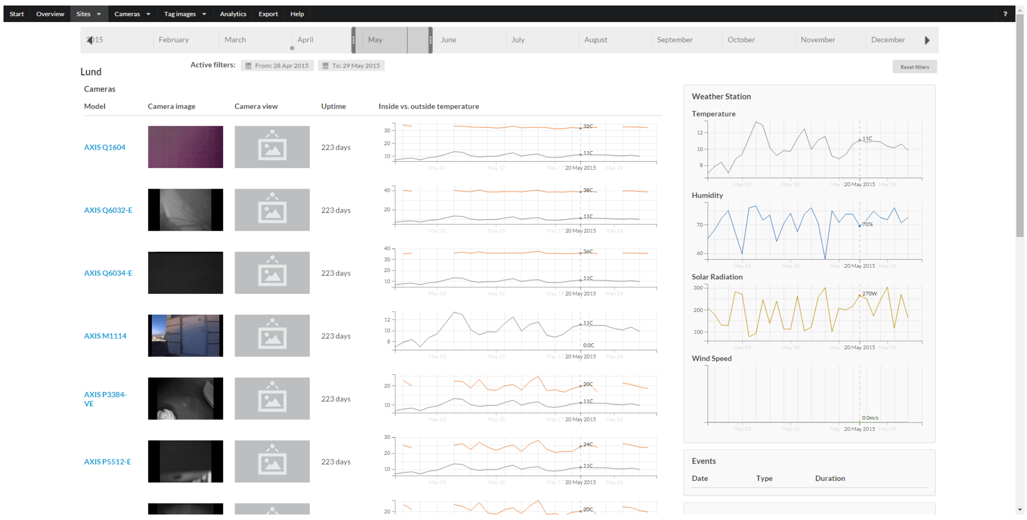
Sites

	Novosibirsk	Bangkok	Dubai	Lund
Overview				
Uptime	223 days	223 days	223 days	223 days
Temperature Out				
Humidity				
Wind Speed				
Solar Radiation				

Events

Date	Type	Duration	
2015-05-17	Power outage	a day	Novosibirsk
2015-05-13	Fan failure	4 days	Novosibirsk

Figure 8.8: The overview dashboard



Active filters: From: 28 Apr 2015 To: 29 May 2015

Lund

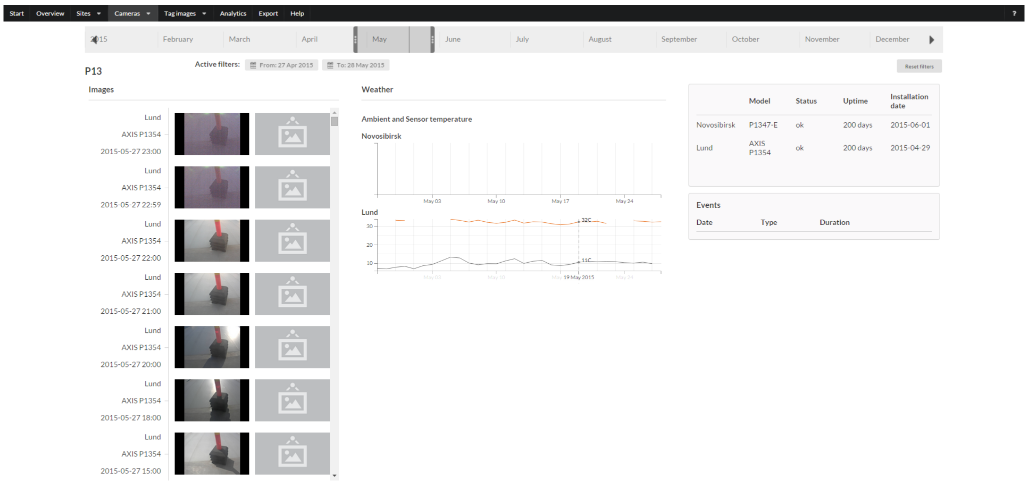
Cameras

Model	Camera image	Camera view	Uptime	Inside vs. outside temperature
AXIS Q1604			223 days	
AXIS Q6032-E			223 days	
AXIS Q6034-E			223 days	
AXIS M1114			223 days	
AXIS P3384-VE			223 days	
AXIS P5512-E			223 days	

Events

Date	Type	Duration

Figure 8.9: The site dashboard



Active filters: From: 27 Apr 2015 To: 28 May 2015

P13

Images

Lund AXIS P1354 2015-05-27 23:00

Lund AXIS P1354 2015-05-27 22:59

Lund AXIS P1354 2015-05-27 22:00

Lund AXIS P1354 2015-05-27 21:00

Lund AXIS P1354 2015-05-27 20:00

Lund AXIS P1354 2015-05-27 18:00

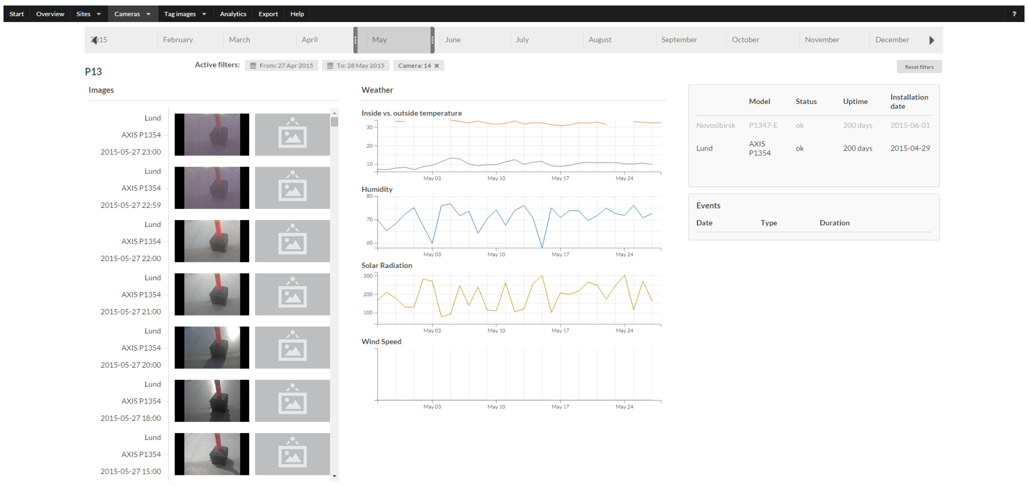
Lund AXIS P1354 2015-05-27 15:00

	Model	Status	Uptime	Installation date
Novosibirsk	P1347-E	ok	200 days	2015-06-01
Lund	AXIS P1354	ok	200 days	2015-04-29

Events

Date	Type	Duration

Figure 8.10: The camera series dashboard



	Model	Status	Uptime	Installation date
Novosibirsk	P1347-E	ok	200 days	2015-06-01
Lund	AXIS P1354	ok	200 days	2015-04-29

Figure 8.11: The camera series dashboard filtered to only show the camera in Lund

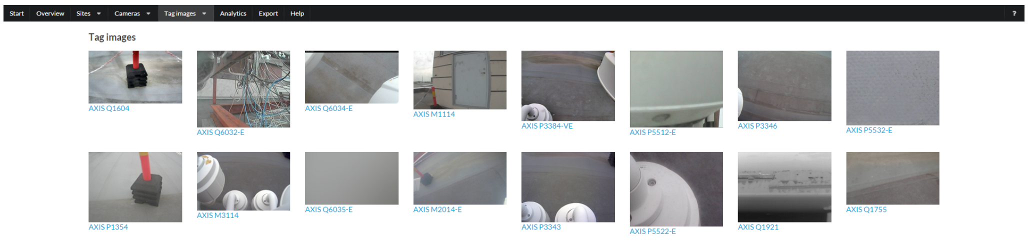


Figure 8.12: The tag images view showing cameras in Lund

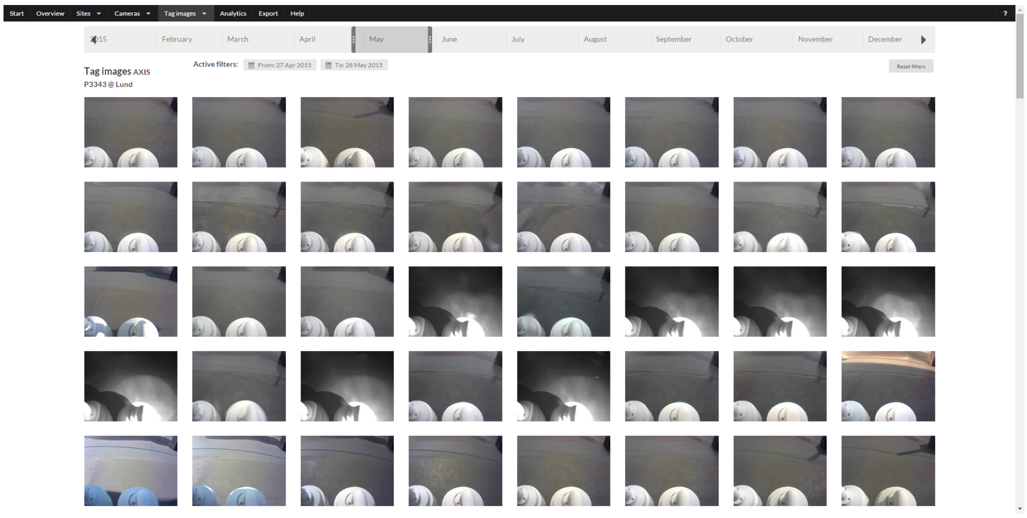


Figure 8.13: The tag images view when a camera has been selected

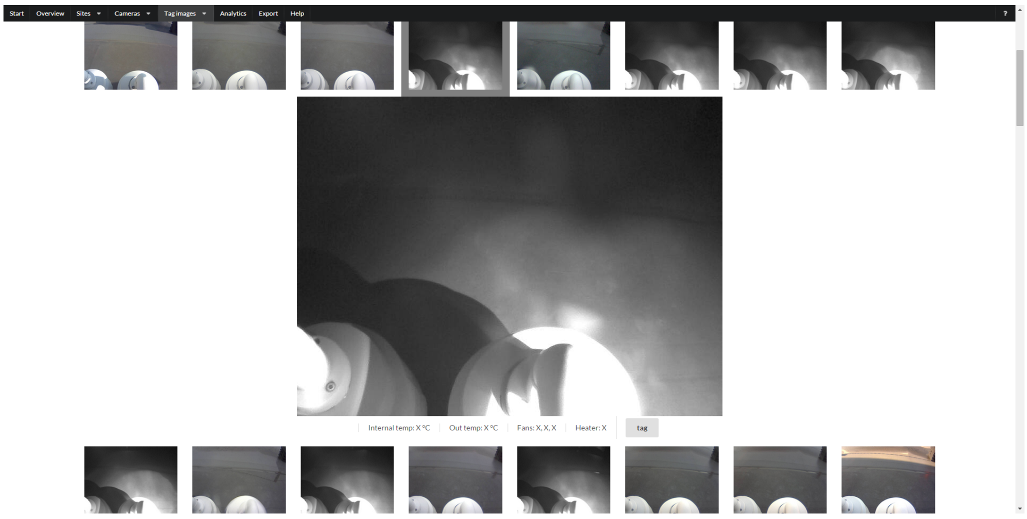
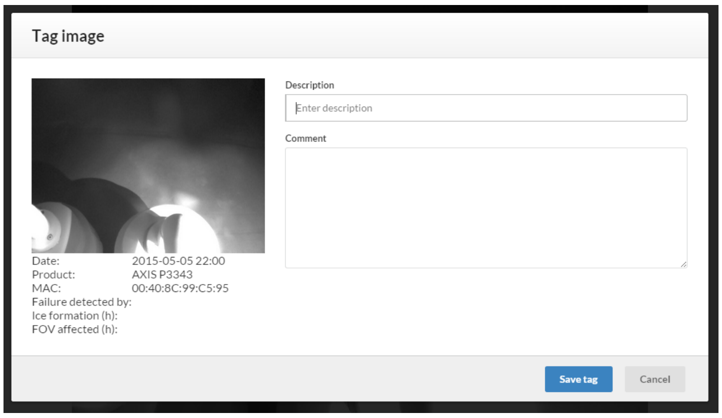


Figure 8.14: The tag images view when an image has been selected



Tag image

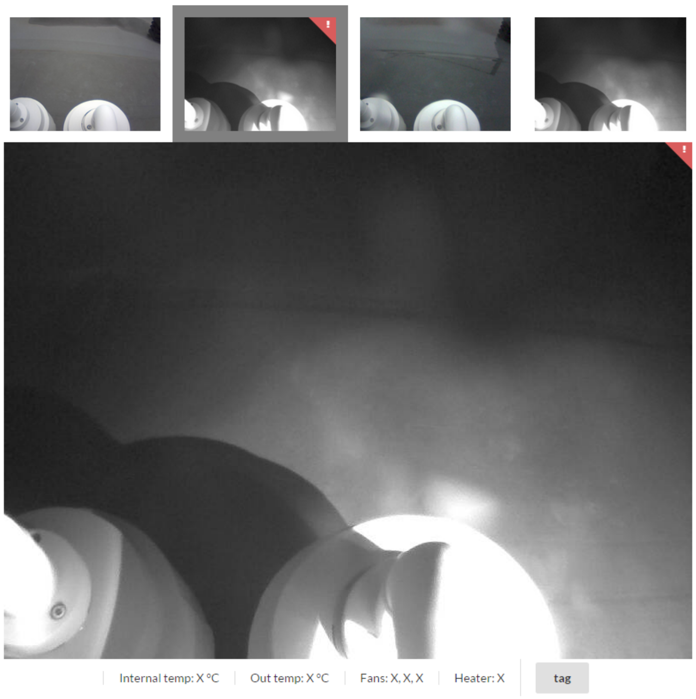
Description

Enter description

Comment

Date: 2015-05-05 22:00
Product: AXIS P3343
MAC: 00:40:8C:99:C5:95
Failure detected by:
Ice formation (h):
FOV affected (h):

Figure 8.15: The dialog that appears when clicking the tag button in figure 8.14



Internal temp: X °C | Out temp: X °C | Fans: X, X, X | Heater: X

Figure 8.16: The tagged image gets an icon.

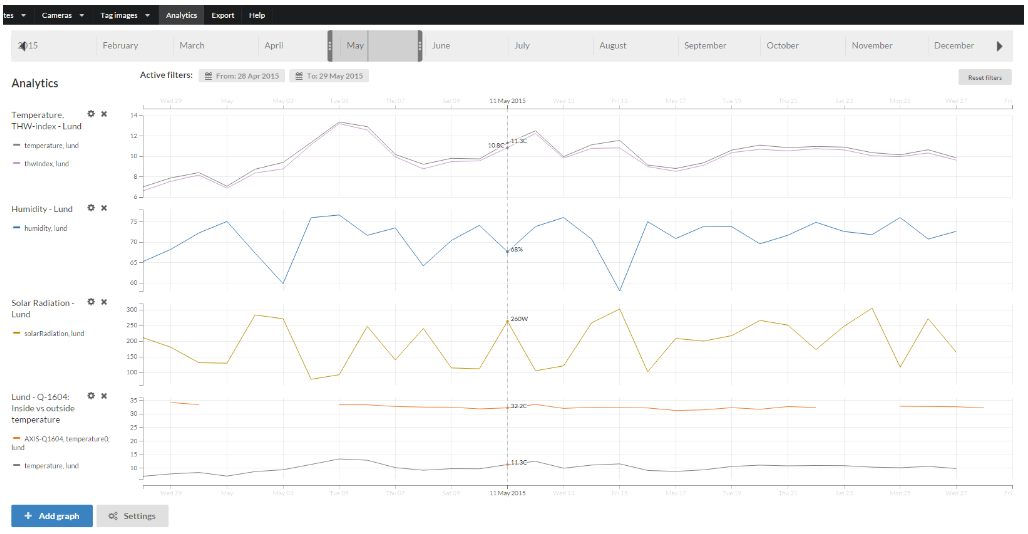


Figure 8.17: The analytics view showing average day values

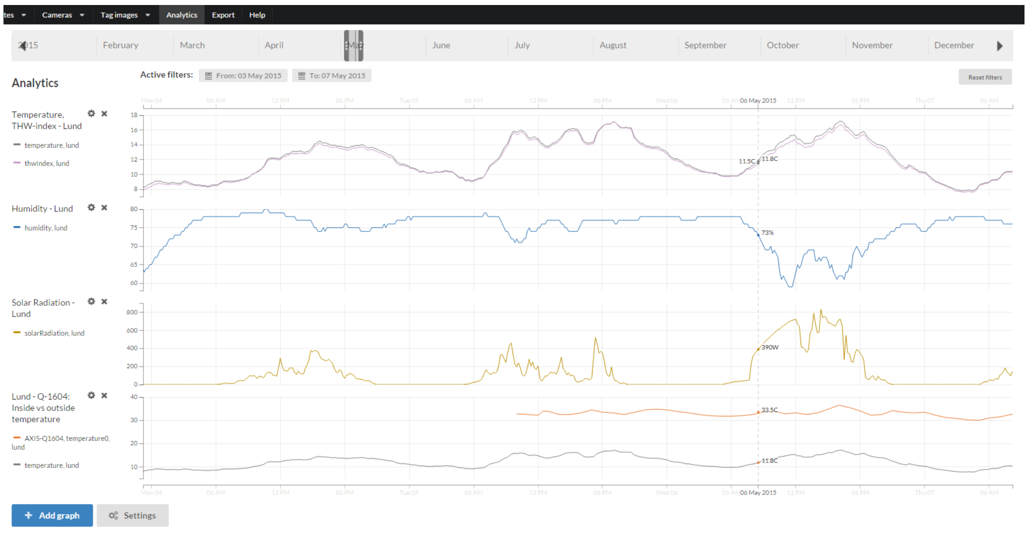
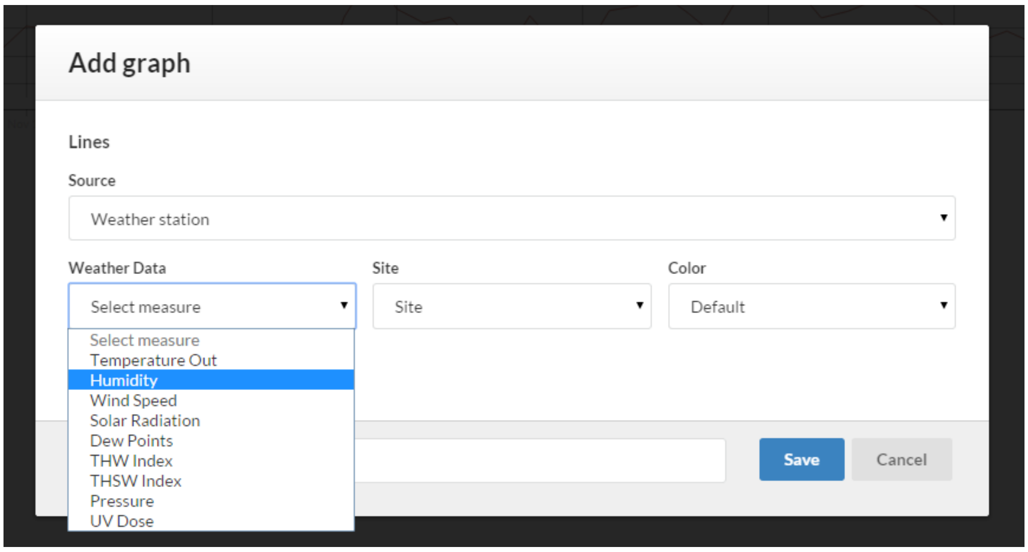


Figure 8.18: The analytics view when zoomed in to show detailed data


Add graph

Lines

Source

Weather station

Weather Data

Select measure

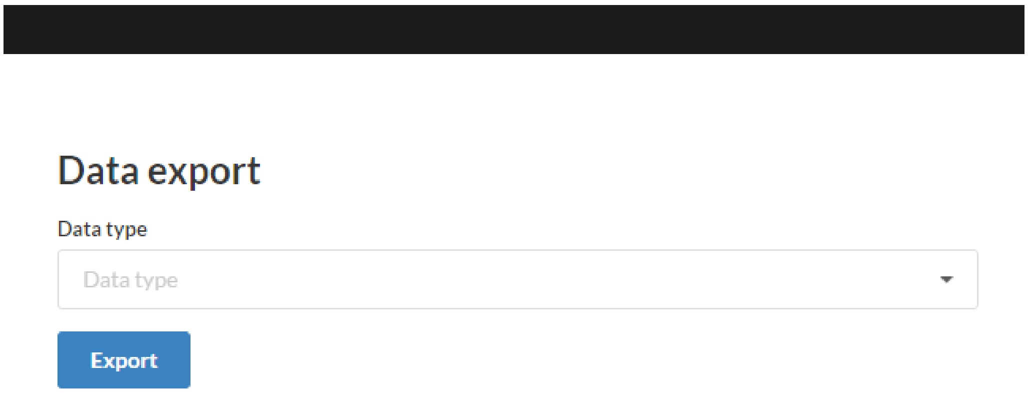
Site

Site

Color

Default

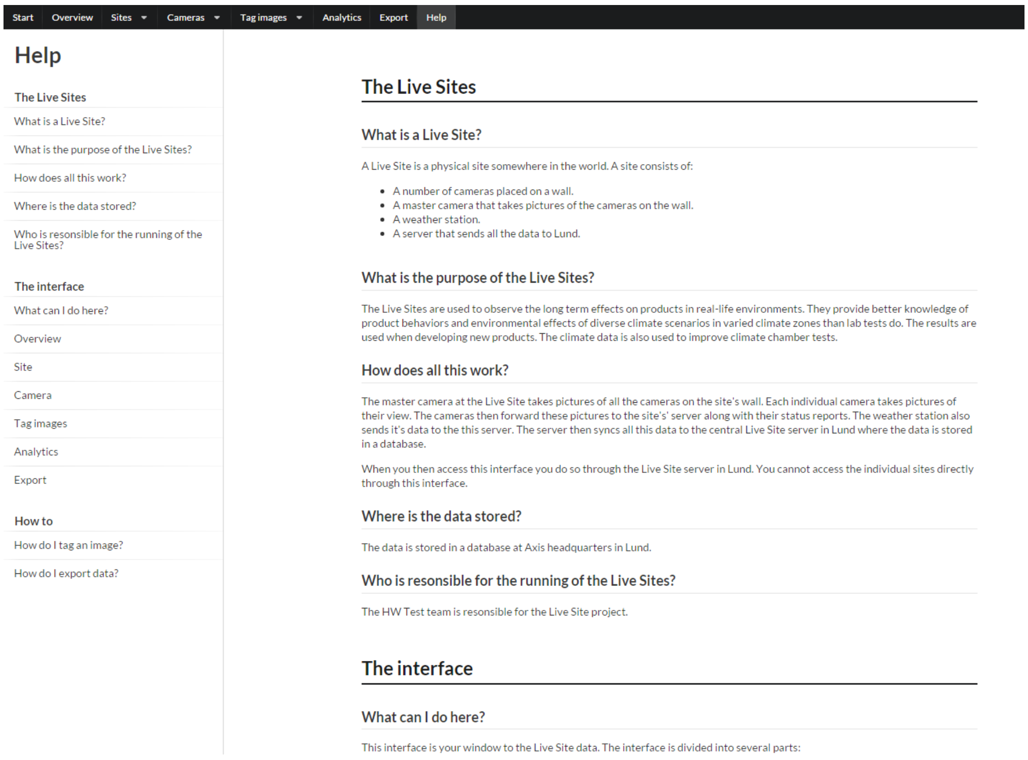
Figure 8.19: The add/edit graph dialog for the analytics view


Data export

Data type

Data type

Figure 8.20: The export view



The Live Sites

What is a Live Site?

A Live Site is a physical site somewhere in the world. A site consists of:

- A number of cameras placed on a wall.
- A master camera that takes pictures of the cameras on the wall.
- A weather station.
- A server that sends all the data to Lund.

What is the purpose of the Live Sites?

The Live Sites are used to observe the long term effects on products in real-life environments. They provide better knowledge of product behaviors and environmental effects of diverse climate scenarios in varied climate zones than lab tests do. The results are used when developing new products. The climate data is also used to improve climate chamber tests.

How does all this work?

The master camera at the Live Site takes pictures of all the cameras on the site's wall. Each individual camera takes pictures of their view. The cameras then forward these pictures to the site's' server along with their status reports. The weather station also sends it's data to the this server. The server then syncs all this data to the central Live Site server in Lund where the data is stored in a database.

When you then access this interface you do so through the Live Site server in Lund. You cannot access the individual sites directly through this interface.

Where is the data stored?

The data is stored in a database at Axis headquarters in Lund.

Who is resonsible for the running of the Live Sites?

The HW Test team is resonsible for the Live Site project.

The interface

What can I do here?

This interface is your window to the Live Site data. The interface is divided into several parts:

Figure 8.21: The help view that describes how to use the interface

8.3 Evaluation

With one week left of the project we had the option to either perform final validating user testing or spend the remaining time finishing as much coding as possible. After a discussion we reasoned that we had done enough user testing earlier in the project and should thus focus on coding. In total throughout the project we performed six user tests and two heuristic evaluation sessions. We also had continuous feedback meetings with different stakeholders. With the combined usability input from these sources, along with our own design intuition and support from data visualization theory, we deemed the solution to be validated.

Chapter 9

Discussion

9.1 Process

9.1.1 Balancing coding and design

Designing and developing a usable software product is no simple task. The product must be well designed both from the users' perspective and programmatically. There is no standard process one can follow that is guaranteed to reach this result. User centered design works well for designing a usable product, but takes no consideration of coding issues. Agile software development methods work well for producing good code, but does not focus on usability. Throughout the project we balanced this by following an iterative design process divided in design sprints and implementation sprints. This worked well overall. The process was both agile and user-centered, and the resulting product is usable.

Being agile

We did not follow any specific agile method, but simply tried to adhere to the agile manifesto. We strove to always have a working prototype to show our customer (Axis hardware QA team) and our customer was included throughout the process by taking part in usability tests and regular meetings. This regularity made the process fast, and made sure we did not ponder too long on insignificant problems. The process was flexible and we did not follow a fixed plan of requirements. Since we were responsible for both design and implementation we were able to not only respond to changes, but drive changes through continuous usability evaluation. This forced us to write good code that evolved throughout the project.

Being user-centered

We used Google Ventures' design sprint and discount usability methods to integrate agile and UCD. By not following any specific agile method we were able to form our own development process around the design sprint that kept the ideals of both agile and UCD. This way we followed Blomkvist's [3] advice on integrating agile and UCD.

By always preceding development with a design sprint we were able to follow the framework presented by da Silva et al. [1], but in a sequential manner instead of in parallel. The sequential manner enforced that design work was always one iteration ahead of development, this way we could counter the problem of design work not being done ahead of development. We also incorporated the idea of little design up front by conducting a preparation phase followed up by the first design sprint.

The design sprints made sure we never got stuck too long in the coding phase. Regularly taking a step back and looking at new problems made sure we did not get too attached to our current design. These regular sprints also made sure the design kept changing and evolving. This might sound annoying from a programming perspective, but in reality it is something positive. When you know that there will be upcoming changes, you will put more effort into coding the underlying structure instead of focusing on intricate design details. You also put in extra effort to make changes easier in upcoming sprints. This also adds the risk of over engineering your code. You might spend much time making sure things can be changed easily, but then the thing never changes and that time was wasted. It is however generally considered good practice to write code that can be modified easily[6].

The design sprints also made sure we were truly working on things that were important. Most of the ideas got scrutinized, prototyped and tested before moving on to implementation. This way we could be confident that what we were building would actually work and be usable. Using the design sprint also made sure that we never got too into design details, but instead stayed focused on the big problems.

Towards the end of the project some ideas would have been difficult or even impossible to test using the PowerPoint prototype and were therefore implemented directly and then tested. One example of this was the more detailed interaction patterns that would have been impossible to implement in the PowerPoint prototype.

Problems

Using an iterative process works most of the time but also has some drawbacks. During the project we seldom had clear goals that we aimed to reach by the end of the project. Instead we iterated and figured that the project would converge to a finished product in the end. Having clearer goals would probably have led to a process where even less time would have been wasted. However, setting these goals could also have led to impairing the design by not allowing rapid changes.

9.1.2 Design and development in a two-man team

Being a team of two forced us to do some adaptations to the design sprint. One of the major issues the design sprint aims to solve is decision making in a design process. The design sprint solves this by forcing the true decision makers to actually make the design decision instead of being overly democratic. Being only two members in our team made decision making quite simple. Generally a short discussion would conclude if an idea was good or bad.

The design sprint also puts a lot of focus on counteracting groupthink by focusing on individuals and their ideas. When the team consists of two members it can be hard for those two individuals to come up with good ideas on their own. In a bigger team, the chances are greater that someone will come up with a good idea. Being two limits the pool of possible ideas. Throughout the project we adapted to this and allowed discussions to fuel our idea generation.

Of the approaches described by Fox, Sillito and Maurer in 2008[2], our process is most similar to the Hybrid approach since we both had academic training in UCD as well as software development experience. We did however lack real work experience in UCD so it does not fit this category perfectly.

If one were to adapt this process to a larger team it would make sense to run the design and implementation phases in parallel, as suggested by da Silva et al. [1]. Then the programmers could focus more on the implementation part and designers could focus on the design prototypes. In accordance with da Silva et al. [1] as well as Blomkvist [3] we would however recommend programmers to be somewhat involved in the design process and vice versa so that communication between the groups is not lost. By not dividing the work into a development team and a design team we were however able to prevent the problems listed by da Silva et al. [1].

9.1.3 Usability evaluation

Over the course of the project we continually evaluated the usability of our product. We chose to use two methods to do this: heuristic evaluation and task based usability testing. The major reasons behind choosing these two methods exclusively was their simplicity and speed. These methods fit well together with our focus on fast paced development while still being effective in finding usability issues. In retrospect, we feel that this worked well in this project. We got enough feedback to always have things to work on and we felt assured that we were working on things that mattered. These methods do however rely on the designers' ability to evaluate the test results based entirely on insight and there are no numbers to back the results. However, this can be seen as irrelevant since the aim of running these smaller usability tests and heuristic evaluation often is to gain precisely that, insight. During our tests we were mostly looking for large recurring problems, not user preference on different pages, colors, fonts etcetera.

The number of test participants

The number of users to test with is also a matter of discussion. During our process we ran our tests with quite a small number of users, 1-3 per test phase. You could argue that that this is too few, but we would disagree. In our particular case, we could not see any reason to test with more users. If we had tested with more users we would probably had found more usability problems, but our fast paced development sprints would not have allowed time for fixing all these problems. During our tests we would often realize what the big problems were after only one or two tests. This was enough to feed our upcoming design sprint with problems that would need solving. To make up for the small number of users per test phase, our process instead contains more test phases. Every iteration includes one to two test phases, one in the end of the design sprint and one in the evaluation phase. Since our product changed and evolved so much during the process, it would have been a waste to try and find every usability problem in every version.

Running the tests with this limited number of users allowed us to spend more time on technical problems. However, one must take into consideration that the number of users to test with is very case specific. For our particular case, 1-3 was enough, but this might not be true for other projects. Following Nielsen's recommendation of testing with 5 users is probably better in the general case. Our initial plan was to follow this recommendation, but in the end we never felt that it was necessary to try and hunt down 5 users for every test phase. The number of users was never important for us, we instead valued the number of found problems. If we had enough problems that we felt were big enough after only one test, we saw no reason to run more tests in that phase.

Having the customer on site made user testing a lot easier. This made it possible to perform quick and simple tests on short notice. Otherwise, we might have opted for fewer usability tests with more people since that would be easier to manage.

9.2 Design

Most of our design is discussed directly in the process chapters of this report. This section covers areas that we thought about, but never got to explore in detail during our development process.

9.2.1 Micro/macro readings

While using our analytics view we realized that it was quite easy to get lost in the data. When you zoom to the detailed level you can no longer see the data average view. This forces you to zoom out again to get the overview, and then zoom back in again. Tufte's thoughts on micro/macro reading applies here (see section 2.5.5). Zooming in and out forces unnecessary context switching which, as Tufte says, interrupts information reasoning. Being able to view both the zoomed in and zoomed out data at the same time would solve this. This way you could use the zoomed out view to look for critical areas and then navigate there to look into the details without zooming in and out. Transition between micro and macro would be easier since both are visible at the same time. We did some sporadic prototyping on a solution to this problem (see figures 9.1 to 9.3), but unfortunately never got around to implement it.

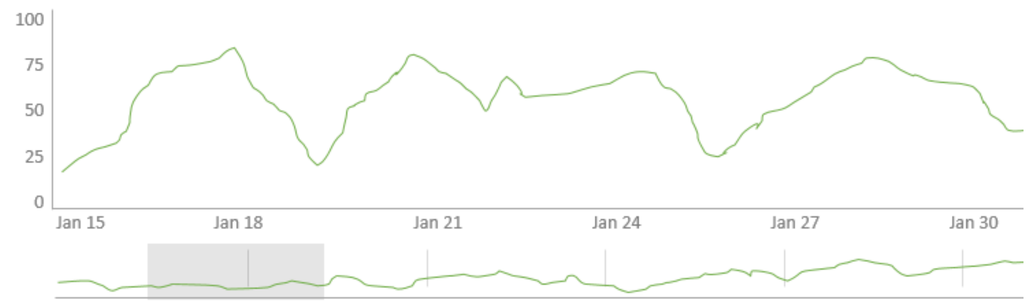


Figure 9.1: Micro/macro analytics graph, prototype 1.

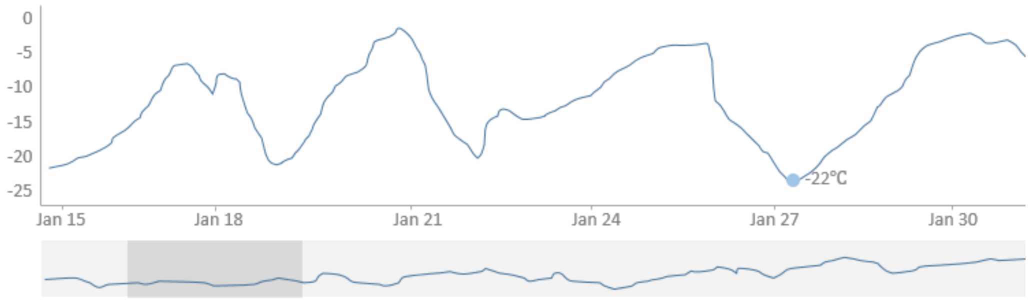


Figure 9.2: Micro/macro analytics graph, prototype 2.

9.2.2 The analytics view

We are not entirely pleased with how the analytics view turned out. Our initial idea was to allow the user to plot data in many different kinds of graphs. This

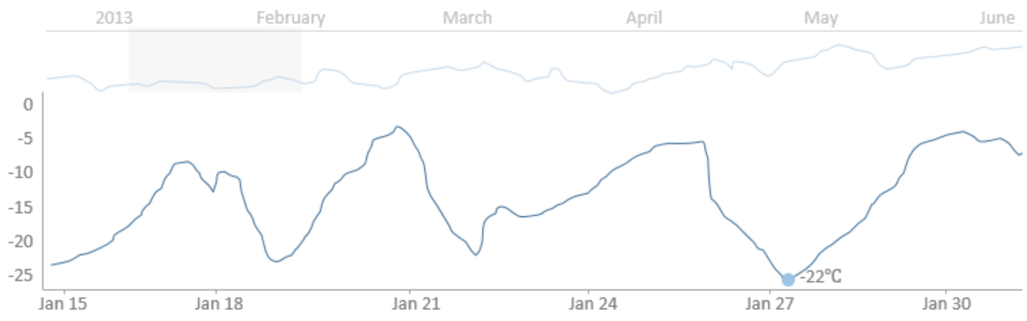


Figure 9.3: Micro/macro analytics graph, prototype 3.

would give the user more freedom and control over the data analysis, which we think would help the user to reason about the data. Implementing this turned out to be much more difficult than we expected. This meant we had to adapt our solution and remove many of our planned features. In the final version the analytics view only supports time series line graphs. We deemed this type of graph to be the most important in the task of analyzing the cameras' performance over time. Bar charts could be useful when comparing different groups of cameras to detect similar behaviors, e.g. by comparing the number of reported critical failures when temperatures goes below -30 degrees Celsius.

Another issue in the analytics view is the way graphs are added and edited. The add graph button opens up a dialogue window where the user have to fill in which data to plot. There is no preview of the plot. This dialogue completely disconnects the user from the analytics view. A better way would be to simply add an empty graph when the user clicks the add graph button. The user could then be able to choose which data to plot directly from a drop down list in the side menu of the graph. This solution would make it much easier to add graphs and encourage the user to explore different data.

Another issue with the add graph module is the fact that it takes a lot of time and effort to add many graphs. Our initial idea was to implement save functionality which would let the user save the current graph setup. This somewhat solves the problem since the user only have to add the graphs once, but it would still be painful the first time. Additional potential pain relievers could be to allow the user to copy graphs, one click graph adding for several sites/cameras

9.3 Rendering performance and its impact on user experience

We spent most of our time developing a working solution and hence did very little code optimization. This has unfortunately led our interface to become slow and sluggish at times. The main reason for this is the amount of data that the interface has to handle. Every day a Live Site produces 100-200 MB of data. In a year this accumulates to 35-70 GB. Adding several Live Sites and years the amount of data reaches quite a high amount. This high amount of data can lead to slow loading and jaggy interaction. An example is the camera dashboard for the Lund site. This site has sixteen cameras, and for each camera there are two images displayed and a graph with multiple lines. When the user interacts with the time bar it takes considerable time for the interface to filter the data and update the displayed images and graphs. This leads to an experience where the interface feels unresponsive and slow.

Another example is the analytics view. This view allows the plotting of whichever data the user chooses. For the user to be able to choose, and not have to wait for the data to load, all the data must be available at when the page is initially displayed. This can make the load time of the analytics view unbearably slow, which makes the user lose focus and interest. Our solution was to limit the data transferred by the initial fetch of the analytics view by sending the daily average data instead of the detailed data. This meant a reduction from 144 entries per day to 1 entry per day. This significantly reduced the load time, but also harmed the data analysis since not all data is available. The detailed data is not requested by the client until the average data has been loaded. This lets the user start analyzing the daily average data, while the rest of the data is being transferred in the background without the user knowing. When the user zooms in to view the detailed data, the data has already been transferred in the background. This leads to a better experience since the user is tricked into believing that the data is already there. In reality it actually takes a longer time to transfer all the data in this way, but since the view appears to have loaded and the user can start interacting with it the negative effect of the data loading is countered and hidden. Another improvement would be to only send detailed data for the currently filtered time period. This solution might be even smoother, and would further reduce the data transfer load, but is more difficult to implement.

9.4 Comparison

We did not do a comparison of our final implementation and the process as it was before the project started. We do however believe our solution to be more efficient based on the fact that we have eliminated some of the time consuming steps in the previous process. We have also recieved a lot of positive feedback on our solution which suggests this to be the case.

Chapter 10

Conclusion

During this project we outlined a practical approach to design in an agile software development process. This process uses Google Venture's design sprint model and Jacob Nielsen's ideas on discount usability. We used this approach in a real project to design and develop a web interface for Axis global testing Live Sites. The project also required us to study the design field of data visualization since this would become a large part of the interface.

Our development process worked well. We were able to run four development iterations during the course of the project. During every iteration new design problems were addressed and solved through a design sprint and these solutions were implemented in the coding sprint. Continuous evaluation of prototypes and the product ensured us that we were working on the right thing. Heuristic evaluation and simplified user testing was employed to make usability evaluation effective and cheap. By following this process we could spend just enough time and effort on design. This allowed us to spend more time on actually building the solution while at the same time feeling confident that the design would work.

We slightly misjudged the complexity of our design and hence did not manage to implement all the features we would have liked. Our opinion is that our process is not to be blamed for this, but that the project simply turned out to be too great in scope and too complicated to be completely finished within the set time by a team of two developers. The product can still be used in its current state, but we believe that we did a good enough job for our ideas to be finalized by someone else in the future.

10.1 Future work

Even though we were able to spend a lot of time on implementing our product, it is still in its infancy. To be able to fully use the product as intended there is still some work that needs to be done. Since we have focused on delivering a working design, less effort has been spent on performance and security. The product only implements very limited access control and there is no data encryption, except the cloud synchronization. These are important issues that should be looked at in the future. The performance of our interface is also a concern. Currently the interface can get unresponsive and slow because of the amount of data and rendering updates in the DOM. We chose to focus on the interaction design of working with the graphs and hence could not spend the time needed to optimize the code for performance. Performance is however a major user experience issue and should, in our opinion, be prioritized in future work.

When the product goes live, user feedback and further user testing can be used to find which areas of the design need further improvement.

References

[1] T.S. da Silva, M. Selbach Silveira, and T. Maurer F. and Hellmann. “User Experience Design and Agile Development: From Theory to Practice”. In: *Journal of Software Engineering and Applications* 5 (2012), pp. 743–751.

[2] D. Fox, J. Sillito, and F. Maurer. “Agile Methods and User-Centered Design: How These Two Methodologies are Being Successfully Integrated in Industry”. In: *Agile, 2008. AGILE ’08. Conference*. Aug. 2008, pp. 63–72. DOI: `10.1109/Agile.2008.78`.

[3] S. Blomkvist. “Towards a model for bridging agile development and user centered design”. In: *Human-Centered Software Engineering - Integrating usabillity in the software development lifecycle* (2005), pp. 219–244.

[4] J. Nielsen. “Agile Development Projects and Usability”. In: *Nielsen Norman Group* (Nov. 17, 2008). URL: `http://www.nngroup.com/articles/agile-development-and-usability/`.

[5] K. Beck et al. *Manifesto for Agile Software Development.* 2001. URL: `http://www.agilemanifesto.org/`.

[6] R.C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Alan Apt Series. Pearson Prentice Hall, 2012. ISBN: 9780132760584. URL: `https://books.google.se/books?id=Ak8OtwAACAAJ`.

[7] M. McNeill. “7 Benefits of Agile and User Centered Design”. In: *www.thoughtworks.com* (Oct. 2013). URL: `http://www.thoughtworks.com/insights/blog/agile-and-user-centered-design`.

[8] H. Loranger. “Doing UX in an Agile World: Case Study Findings”. In: *Nielsen Norman Group* (May 26, 2014). URL: `http://www.nngroup.com/articles/doing-ux-agile-world/`.

[9] John D Gould and Clayton Lewis. “Designing for usability: key principles and what designers think”. In: *Communications of the ACM* 28.3 (1985), pp. 300–311.

[10] Donald A Norman and Stephen W Draper. “User centered system design”. In: *Hillsdale, NJ* (1986).

[11] J. Gulliksen et al. “Key principles for user-centred systems design”. In: *Human-Centered Software Engineering—Integrating Usability in the Software Development Lifecycle*. Springer, 2005, pp. 17–36.

[12] D.A. Norman. *The Design of Everyday Things*. Mit Press, 2013. ISBN: 9780262525671. URL: `https://books.google.se/books?id=heCtnQEACAAJ`.

[13] S. Krug. *Don't Make Me Think: A Common Sense Approach to Web Usability*. Computers & Internet, Business. Que, 2000. ISBN: 9780789723109. URL: `https://books.google.se/books?id=DI1tBgAAQBAJ`.

[14] J. Nielsen. “Turn User Goals into Task Scenarios for Usability Testing”. In: *Nielsen Norman Group* (Jan. 12, 2014). URL: `http://www.nngroup.com/articles/task-scenarios-usability-testing/`.

[15] M. Margolis. *Finalize schedule and complete interview guide (day 3)*. URL: `http://www.gv.com/lib/the-gv-research-sprint-day-3` (visited on 03/12/2015).

[16] J. Nielsen. “Discount Usability: 20 Years”. In: (Sept. 14, 2009). URL: `http://www.nngroup.com/articles/discount-usability-20-years/`.

[17] J. Nielsen. “Why You Only Need to Test with 5 Users”. In: (Mar. 19, 2000). URL: `http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/`.

[18] J. Knapp. “Paper prototyping is a waste of time”. In: *Medium* (Mar. 25, 2014). URL: `https://medium.com/@jakek/paper-prototyping-is-a-waste-of-time-353076395187`.

[19] J. Nielsen. “How to Conduct a Heuristic Evaluation”. In: *Nielsen Norman Group* (Jan. 1, 1995). URL: `http://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/`.

[20] *THE DESIGN SPRINT*. URL: `http://www.gv.com/sprint/` (visited on 03/12/2015).

[21] M. Margolis. *The GV research sprint: a 4-day process for answering important startup questions*. URL: `http://www.gv.com/lib/the-gv-research-sprint-a-4-day-process-for-answering-important-startup-questions` (visited on 03/12/2015).

[22] J. Knapp. “From Google Ventures, The 6 Ingredients You Need To Run A Design Sprint”. In: *Fast Company*. How To Conduct Your Own Google Ventures Design Sprint (June 25, 2013). URL: `http://www.fastcodesign.com/1672889/from-google-ventures-the-6-ingredients-you-need-to-run-a-design-sprint`.

[23] J. Knapp. "The First Step In A Design Challenge: Build Team Understanding". In: *Fast Company.* How To Conduct Your Own Google Ventures Design Sprint (June 26, 2013). URL: `http://www.fastcodesign.com/1672905/the-first-step-in-a-design-challenge-build-team-understanding`.

[24] J. Knapp. "The 8 Steps To Creating A Great Storyboard". In: *Fast Company.* How To Conduct Your Own Google Ventures Design Sprint (Dec. 21, 2013). URL: `http://www.fastcodesign.com/1672917/the-8-steps-to-creating-a-great-storyboard`.

[25] M. Margolis. *Start recruiting participants (day 1).* URL: `http://www.gv.com/lib/the-gv-research-sprint-day-1` (visited on 03/12/2015).

[26] J. Knapp. "How To Decide What Ideas To Prototype". In: *Fast Company.* How To Conduct Your Own Google Ventures Design Sprint (June 28, 2013). URL: `http://www.fastcodesign.com/1672929/how-to-decide-what-ideas-to-prototype`.

[27] M. Margolis. *Schedule participants and draft interview guide (day 2).* URL: `http://www.gv.com/lib/the-gv-research-sprint-day-2` (visited on 03/12/2015).

[28] J. Knapp. "A Lightning-Fast Way To Make A Digital Prototype". In: *Fast Company.* How To Conduct Your Own Google Ventures Design Sprint (July 1, 2013). URL: `http://www.fastcodesign.com/1672940/a-lightning-fast-way-to-make-a-digital-prototype`.

[29] J. Knapp. "Got A Bright Idea? Test It With A Rapid-Fire User Study". In: *Fast Company.* How To Conduct Your Own Google Ventures Design Sprint (July 2, 2013). URL: `http://www.fastcodesign.com/1672947/got-a-bright-idea-test-it-with-a-rapid-fire-user-study`.

[30] E.R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, 2001. ISBN: 9780961392147. URL: `https://books.google.se/books?id=GTd5oQEACAAJ`.

[31] E.R. Tufte. *Beautiful Evidence.* Graphics Press, 2006. ISBN: 9780961392178. URL: `https://books.google.se/books?id=v302PAAACAAJ`.

[32] C. Ware. *Information Visualization: Perception for Design.* Information Visualization: Perception for Design. Morgan Kaufmann, 2013. ISBN: 9780123814647.

[33] S. Few. "Dashboard Confusion". In: *Intelligent Enterprise* (Mar. 20, 2004).

[34] P.M. Healy and Palepu K.G. "The Fall of Enron". In: *Journal of Economic Perspectives* 17.2 (2003), pp. 3–26. URL: `http://www-personal.umich.edu/~kathrynd/JEP.FallofEnron.pdf`.

[35] S. Few. *Information Dashboard Design: The Effective Visual Communication of Data*. O'Reilly Series. O'Reilly Media, Incorporated, 2006. ISBN: 9780596100162.

[36] ISO. *Ergonomic requirements for office work with visual display terminals (VDTs)—Part 11: Guidance on usability*. ISO 9241-11:1998. Geneva, Switzerland: International Organization for Standardization, 1998.

[37] J. Nielsen and Norman D. *The Definition of User Experience*. URL: `http://www.nngroup.com/articles/definition-user-experience/`.

[38] ISO. *Ergonomics of human-system interaction—Part 210: Human-centred design for interactive systems*. ISO 9241-210:2010. Geneva, Switzerland: International Organization for Standardization, 2010.

[39] B. Shneiderman. *Designing the User Interface - Strategies for Effective Human-Computer Interaction*. Addison Wesley, 1986. ISBN: 9788131732557. URL: `https://books.google.se/books?id=7NeB6KAIbUkC`.

[40] B. Shneiderman and C. Plaisant. *Designing the User Interface: Strategies for Effective Human-computer Interaction*. Addison-Wesley, 2010. ISBN: 9780321601483. URL: `https://books.google.se/books?id=pddxRQAACAAJ`.

[41] J. Nielsen. "10 Usability Heuristics for User Interface Design". In: (Jan. 1, 1995). URL: `http://www.nngroup.com/articles/ten-usability-heuristics/`.

[42] *Usage of server-side programming languages for websites*. Mar. 9, 2015. URL: `http://w3techs.com/technologies/overview/programming_language/all` (visited on 03/09/2015).

[43] *Usage of web servers for websites*. Mar. 9, 2015. URL: `http://w3techs.com/technologies/overview/web_server/all` (visited on 03/09/2015).

[44] *February 2015 Web Server Survey*. Feb. 2015. URL: `http://news.netcraft.com/archives/2015/02/24/february-2015-web-server-survey.html` (visited on 03/09/2015).

[45] *Web Framework Benchmarks: Round 9 Results*. May 1, 2014. URL: `https://www.techempower.com/benchmarks/#section=data-r9&hw=peak&test=plaintext` (visited on 03/09/2015).

[46] J. Nielsen. "F-Shaped Pattern For Reading Web Content". In: *Nielsen Norman Group* (Apr. 17, 2006). URL: `http://www.nngroup.com/articles/f-shaped-pattern-reading-web-content/`.

Appendix A
User test guide

This guide was used as reference when we conducted our task based user tests. All the tests were conducted in Swedish, hence the guide is also written in Swedish.

A.1 Användbarhetstest

Intro

Tack för att du ställer upp på detta test. Innan vi börjar tänkte jag dra igenom en hur detta test kommer gå till. Först och främst vill jag säga att det är vår produkt vi testar och inte dig. Det finns inga rätta eller fel svar på de frågor jag ställer.

Testet kommer gå till på följande vis. Först kommer jag ställa lite frågor om din bakgrund. Därefter ger jag en kort introduktion till produkten och efter det sätter vi igång själva testet, under vilket jag kommer jag be dig utföra ett antal uppgifter. Medan du utför uppgifterna vill jag att du "tänker högt". Försök förklara vad du ser och hur du tänker när du gör saker. Säg till om något är förvirrande eller du inte förstår något, men säg också gärna till om du ser saker du gillar.

Tänk på att din feedback inte kommer påverka oss känslomässigt. Vi vet att det finns problem i vår produkt, vi vet bara inte exakt var de finns och det är här vi behöver din hjälp. När du stöter på dessa problem och fastnar kommer jag i första hand inte hjälpa dig. Det ger oss värdefull information att se hur du löser dessa problem. Men skulle det hända att du fastnar totalt så kommer jag förstås hjälpa dig.

Eftersom produkten vi testar idag inte är färdig så kommer det finnas knappar och saker som inte riktigt fungerar som de ska. Du kan fortfarande klicka var du vill, inget kan gå sönder. Jag kommer tala om ifall du försöker använda något som inte fungerar så du inte blir mer förvirrad än nödvändigt.

Några frågor innan vi börjar?

Kontextfrågor

1. Vad jobbar du med?
2. Är du intresserad av att veta hur klimat påverkar Axis produkter?
3. På vilket sätt?
4. Vilka klimat?
5. Hur skulle du göra idag för att undersöka detta?
6. Vad vet du om Live Sites projektet?

Intro Live Sites

Förklara vad Live Sites är för något. Anpassa efter hur mycket testpersonen vet sedan innan.

Uppgifter

Tänk på att anpassa alla förklaringar/scenarios beroende på vem som testar!

Fråga hela tiden vad testaren förväntar sig ska hända när hen tittar och letar i gränssnittet.

Live Site manager användare:

1. Du är QA's livesite manager. I ditt jobb ingår att överse livesiterna och se till att allt kör på som det ska. Du har just kommit in till jobbet på onsdags morgon och det är dags att ta en titt på hur livesiterna mår. Undersök om det hänt något viktigt den senaste tiden.
2. Nu undrar du ju såklart hur och varför detta skett så du kan göra något åt det och förhindra att det händer igen. Försök dra någon slutsats utifrån den tillgängliga datan.
3. I ditt arbete ingår det även att hitta nya defekter i kameror som uppkommit på grund av klimat. Hitta och notera en defekt i en kamera i Novosibirsk.
4. I vissa klimatkombinationer är det hög risk att det uppstår problem i kamerorna. Undersök.

Allmänt Live Site intresserad användare:

1. Du är produktansvarig(eller något annat kul) för en viss kamera. Du vet att det genomförts klimattester på din kamera, men skulle själv vilja veta lite mer om hur kameran faktiskt fungerar i verkligt klimat. Din kamera är designad för att vara placerad utomhus, och det är mycket troligt att den

kommer utsättas för snöfall, undersök hur kameran ser ut efter snöfall och hur dess vy påverkas av detta.

2. Du har också en liten aning om att kameran skulle kunna påverkas av {klimatkombination, ex hög luftfuktighet + hög temperatur}. Undersök detta.
3. En kund har rapporterat att en kamera har problem med heatern. Du vet att kundens kamera är placerad i liknande klimatmiljö som den i Novosibirsk. Undersök om någon av våra kameror i Novosibirsk rapporterar liknande fel.
4. Du vet att en din kamera ska klara av att operera normalt upp till en viss innertemperatur. Undersök hur ofta denna temperatur överstigs i Lunds klimat.
5. Du har nu genomfört en massa undersökningar och dragit en del slutsatser. Rapportera detta till {person, ex produktspecialist}.

Debrief

1. Vad gillade/ogillade du?
2. Vad skulle du ändra på om du kunde välja?
3. Tror du att du skulle ha användning av detta?
4. Varför?/Varför inte?
5. Har du några tankar om {feature}?
6. Tyckte du att det fattades något?