

Utilizing principles from the theory of tailoring to meet the needs of end users in generic software

Ullvar Brekke



Thesis submitted for the degree of
Master in programming and networks
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2020

Utilizing principles from the theory of tailoring to meet the needs of end users in generic software

Ullvar Brekke

2020

© Ullvar Brekke 2020

Utilizing principles from the theory of tailoring to meet the needs of end users in generic software

<http://www.duo.uio.no/>

Printed: Representralen, University in Oslo

Abstract

One of the main reasons to buy or implement generic software is to save time and money on development. On the other hand, it is “impossible to design systems which are appropriate for all users and all situations”. Building on this statement, this thesis takes on the challenges met when trying to adapt generic software to meet the needs of end users.

Following the Design Science Research methodology and the theory of tailoring. The literature mentions five main areas of challenge, being: time, money, knowledge, local needs and end user usability. These challenges are in this context related to the DHIS2 platform and the countries and organizations who use it. Generic software aims to provide solutions to the challenges mentioned but by nature is unable to do so.

This thesis aims to create artefacts to meet these challenges. As the challenges have influence on each other, focusing on one will help improve the others. One of the artefacts tested in this thesis is a “drag and drop” MVP for web app development.

This thesis provides principles on how to meet the challenges of end user needs in generic software, with focus on time, money, knowledge, local needs and end user usability.

Key words: theory of tailoring, meta design, design for design, generic software, design science research

Acknowledgements

I would like to thank my supervisors Petter Nielsen and Magnus Li for valuable insights and guidance throughout the thesis work.

I would like to thank the HISP UiO UX design lab and all students contributing to it, for its regular meetings and valuable discussions. This is a great initiative by Magnus Li and something I hope will continue on in the future.

I would like to thank my mother Siri and my father Trond and my sisters Elida and Ellinor for all the support throughout my studies.

Lastly, I would like to thank my girlfriend Agnete for all the support and good memories, looking forward to many more in the future.

Table of contents

| | | |
|-------|--|----|
| 1 | Introduction | 1 |
| 1.1 | Health Information Systems Programme (HISP)..... | 1 |
| 1.2 | District Health Information Software 2 (DHIS2)..... | 1 |
| 1.3 | Problem | 3 |
| 1.4 | Goal..... | 5 |
| 1.5 | Research Question..... | 6 |
| 1.6 | Research contribution | 6 |
| 1.7 | Prerequisites and limitations | 6 |
| 1.8 | Reader's guide | 7 |
| 2 | Methodology..... | 8 |
| 2.1 | Design Science Research | 8 |
| 2.1.1 | Identifying problem and motivate | 8 |
| 2.1.2 | Define objectives of a solution | 9 |
| 2.1.3 | Design and development | 9 |
| 2.1.4 | Demonstration | 9 |
| 2.1.5 | Evaluation | 10 |
| 2.1.6 | Communication | 10 |
| 2.2 | Methods..... | 11 |
| 2.2.1 | User testing..... | 11 |
| 2.2.2 | Interview | 12 |
| 2.2.3 | HISP UX Design lab at UiO..... | 13 |
| 2.3 | Ethics..... | 13 |
| 3 | Theoretical framework..... | 14 |
| 3.1 | A theory of tailorable technology design | 14 |
| 3.1.1 | Nine Principles in the Design of Tailorable Technologies | 16 |
| 3.2 | Design for design | 19 |
| 3.3 | Levels of design | 20 |
| 3.4 | Generic software | 21 |
| 3.5 | Current situation | 22 |
| 3.5.1 | Description of existing editor solution..... | 22 |
| 3.6 | Technologies | 25 |
| 3.7 | User interface design | 26 |
| 3.8 | Minimum Viable Product (MVP) | 27 |

| | | |
|-------|---|----|
| 4 | Process | 28 |
| 4.1 | Preparations and planning | 28 |
| 4.1.1 | Creating a timetable | 28 |
| 4.1.2 | Task management | 29 |
| 4.1.3 | Development tools | 29 |
| 4.2 | Process iteration introduction | 31 |
| 4.3 | Initial review and ideal artefact | 31 |
| 4.3.1 | Design and development | 32 |
| 4.3.2 | Demonstration | 36 |
| 4.4 | Development of MVP | 37 |
| 4.4.1 | Design and development | 37 |
| 4.4.2 | Evaluation | 42 |
| 4.4.3 | MVP overview | 46 |
| 4.5 | Applying learning outcome to ideal artefact | 50 |
| 4.5.1 | Design and development | 50 |
| 4.5.2 | Demonstration | 52 |
| 5 | Discussion | 53 |
| 5.1 | The use of tailoring principles..... | 53 |
| 5.2 | Technical limitations of the thesis..... | 53 |
| 5.3 | Future development of tailoring tools | 53 |
| 5.3.1 | Pre configuration of design environment..... | 53 |
| 5.3.2 | Handling more complex tasks | 54 |
| 6 | Conclusion | 55 |
| 7 | Bibliography | 56 |
| 8 | Appendix | 59 |
| 8.1 | Interview and test guide | 59 |
| 8.2 | Interview summary | 68 |

List of figures

- Figure 1-1.1 Overall Architecture (DHIS2 software, 2019) 2
- Figure 2-1 DSRM Process Model (Ken Peffers, 2014)..... 8
- Figure 3-1 Example of minified JavaScript code 23
- Figure 3-2 Data entry editor with database properties..... 24
- Figure 3-3 Resulting data entry form 24
- Figure 4-1 DHIS2 data entry form in CKEditor 38
- Figure 4-2 DHIS2 data entry form in data entry application..... 38
- Figure 4-3 MVP configuration prompt..... 46
- Figure 4-4 MVP design space 46
- Figure 4-5 MVP Component and element menu 47
- Figure 4-6 MVP input element with database field select..... 48
- Figure 4-7 MVP device view selector 48
- Figure 4-8 MVP top menu..... 49
- Figure 4-9 MVP design space with active form 49

List of tables

Table 3-1 Nine principles of tailorable technology (Matt Germonprez, 2007)..... 16

Table 4-1 Timetable week 24 to 44 28

Table 4-2 Theory of tailoring coverage matrix 36

1 Introduction

1.1 Health Information Systems Programme (HISP)

HISP is a global network created to strengthen information systems related to health, especially in developing countries. HISP was started in South Africa in the 1990s. HISP's branch at the University of Oslo is one of the leading organizations in the HISP network. HISP UiO contributes with implementation development and research, as well as the operation and development of DHIS2's core architecture (UiO, 2019).

1.2 District Health Information Software 2 (DHIS2)

DHIS2 stands for "District Health Information System 2". DHIS2 is a free software platform for managing health data. DHIS2 is mostly used in between low-income countries and organizations (HISP, 2019).

DHIS2 as a platform consists of a core architecture developed in Java by HISP's UiO team. This platform aims to standardize core functionality such as database transactions and authentication. Such database transactions occur through a standardized Application Programming Interface (API). Using the authentication feature, it is possible for third parties to develop applications that are tailored to their needs. This makes DHIS2 a flexible system. In addition to allowing development of custom applications, DHIS2's "core team" also develops some web applications that allows for easy collecting and visualizing of data. (HISP, 2019)

DHIS2 has now also come up with a design library that aims to help and standardize the development of standard and third-party applications.

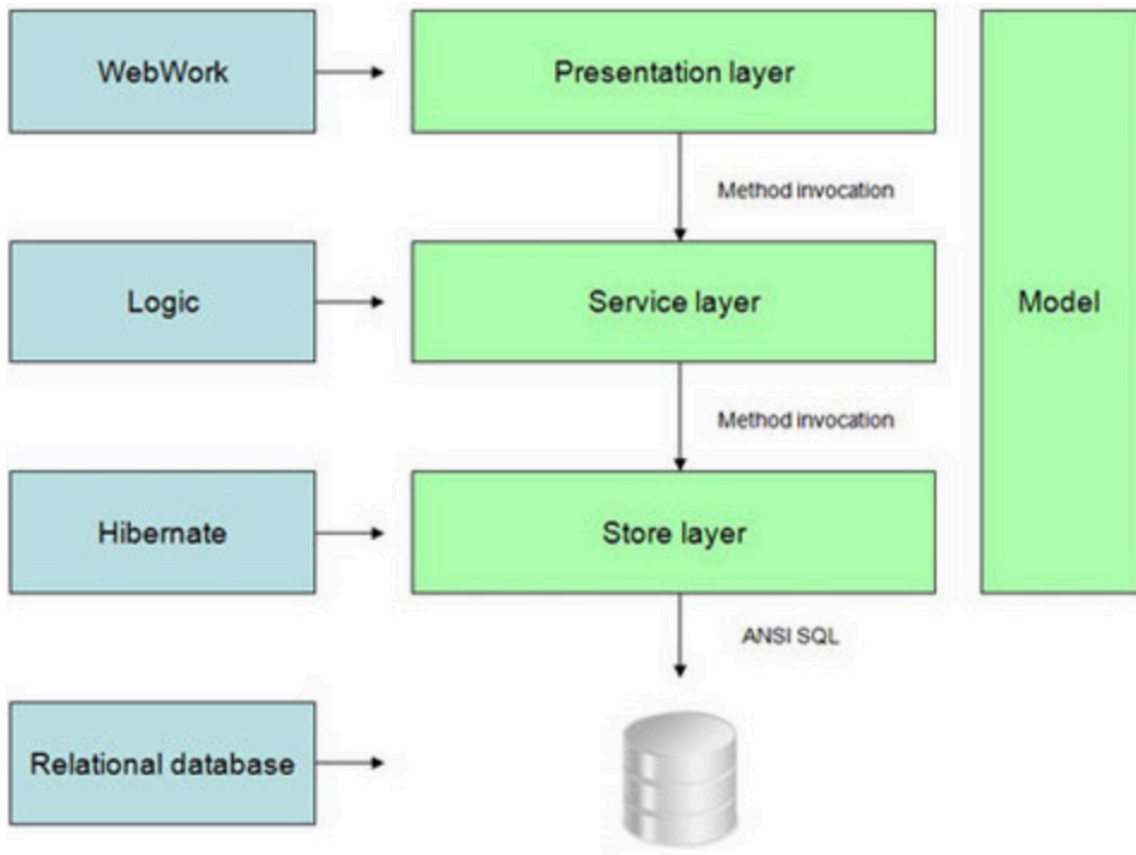


Figure 1-1.1 Overall Architecture (DHIS2 software, 2019)

1.3 Problem

Five main areas of challenge are identified in the thesis proposal given by Magnus Li. These areas are mainly in regard to developing countries, but some would apply to the general software development setting.

Local needs

The first area of challenge is the fact that generic software does not always meet local needs (Li M. , 2019). Local needs in this context refers to the individual needs of the organization or country implementing the software. Generic software is designed to be used in a wide variety of areas, by handling standard operations. The problem comes when the end user has a very specific problem. An interview in relation to this thesis showed that members of an organization are forced to use the generic software with no way to customize. This forces the implementors/developers of the organization to "hack" the software, by using it in ways that it was not designed for. This again can lead to problems in functionality and worst case the software crashes and data is lost.

Funds

One reason to choose generic software is because organizational funds may be limited (Li M. , 2019). This is also the case for many developing countries and the reason for why they would want to go for a generic software solution. Even though when the generic software is designed as a platform like DHIS2 and allows for extensions by applications, this still requires the organization to hire and pay developers to create extension software.

Skills and experience

Based on the results in the article “*Which country has the best programmers?*” (Trikha, 2016) and the list of countries considered to be the least developed (Leraand, 2019), it is fair to argue that countries considered to be in the developmental stage tends to have less people with skills and experience in advanced software development. It is not always enough to have just one person knowing how to program in order to create a working piece of software. To create a user-friendly interface, one would usually hire a designer. To manage the project, one would usually need someone with project management skills.

Time

It takes time to develop software, but many developing countries may be in need of better information handling than the time it takes to develop a custom piece of software. This is also one of the reasons for choosing generic software, namely the time effectiveness of implementation. There’s a saying that goes: “*It is expensive to be poor*”. There are many ways it can be expensive to be poor, but in this context, one of them would be the fact that you can’t give yourself the time to improve the software you have, as there is too much time spent on just maintaining it. The end result is that organizations and countries would fall behind, by maintaining old software and never get the time to develop new and better software.

Usability

While talking about how not user friendly many of the generic software packages can be (Li M. , 2019), the same can be said for custom software. As mention in the section about skills and experience, if there is lacking experience or skills in the developer or team, then the outcome may be worse than the generic option. It is also important to mention that the user interface should not only show complementary colours, with a good contrast, but also be intuitive to the user. This is also known as User Experience (UX).

Summary

To generalize the problem, in my own words. When there is a lack of *resources*, where resources can be seen as one or many of the areas of challenge mentioned above, the end result is software that is not user friendly, because it was hastily put together with less experienced developers. The software will not fit the local needs and the developers would be required to spend more time on maintenance, because the software was poorly designed and put together.

1.4 Goal

The goal of this thesis is to research the theory of tailoring in practice, in order to better fit generic software to the end user's needs. The theory of tailoring is mostly theoretical but may have application in real world software. One intermediate objective is then to research the application of theory of tailoring in generic software. As explained in the previous chapter, there is especially in developing countries, a need for cheap, fast and simple implementation of generic software.

Such an implementation of generic software does often come with either configuration and/or development of third-party applications. It is in this use of third-party applications and lack of functionality to fit local need, the potential to solve a problem lies.

Another intermediate goal of this thesis is to research the possibility to create a piece of software that lets implementors and developers create custom software, following the theory of tailoring's nine design principles. One could call such a solution meta-design, or "design for design".

In order to gauge the usefulness of such a solution it is important to test the user interface. The interface needs to be easy enough to understand such that users without a lot of experience in software development can have the ability to use it. On the other side, the interface cannot be too limiting to the experienced developer. One solution to this problem is not allow the user to download the source code generated by the application for further customization.

The solution is going to be created as a web application. By using the web platform, a lot of problems with cross functionality is eliminated. This means that the

application only needs to meet the requirements of known internet browsers and not widely different operating systems.

The final goal is then to use the insights gathered from the testing and apply them to the tailoring principles, in order to create some more practical principles in relation to generic software.

1.5 Research Question

The research question goes as follows:

How can the theory of tailoring help to better meet the needs of end users in generic software?

1.6 Research contribution

This thesis aims to contribute findings back to HISP, HISP UiO and the HISP UX design lab at UiO, in the form of principles and findings that would allow for adoption of generic software that meets the local needs of the end users. This thesis contributes back to the literature a more practical approach to the theory of tailoring.

1.7 Prerequisites and limitations

This thesis builds on Magnus Li's master thesis proposal "Meta-design: tools for user interface customization". The proposal mentions several areas of focus, with some idea for solution, other than that the thesis and research work have few limitations.

One limiting factor of this thesis was a planned trip to India which had to be cancelled. This trip was meant to serve as an opportunity to test and learn from the users this thesis work is aimed at.

1.8 Reader's guide

Artefact in this context is used the same way as the word solution. An artefact is simply anything that can serve as a solution to the problem, that being anything from principles to implementation of actual working software.

Developer is anyone creating applications used in DHIS2. Anyone working in the DHIS2 core team is not considered as developers in this context.

Implementor is anyone who configures DHIS2 and/or DHIS2 complementary applications.

User is, unless otherwise stated, referring to the end users of DHIS2. The end users of DHIS2 is the users that use what the implementors and developers create.

2 Methodology

This chapter will cover the chosen research methodology for this thesis and some general rules of ethics when researching.

2.1 Design Science Research

Design Science Research Methodology (DSRM) aims to give a methodology for conducting Design Science research in information systems (Ken Peffers, 2014). The DSRM consists of principles and practices covered in six steps. The DSRM is structured as six consequential activities, but it is possible to start on any of the four first activities, depending on the problem. The most important of the DSRM is the problem centred focus. There is also room for iterations over multiple activities to refine the final artefact.

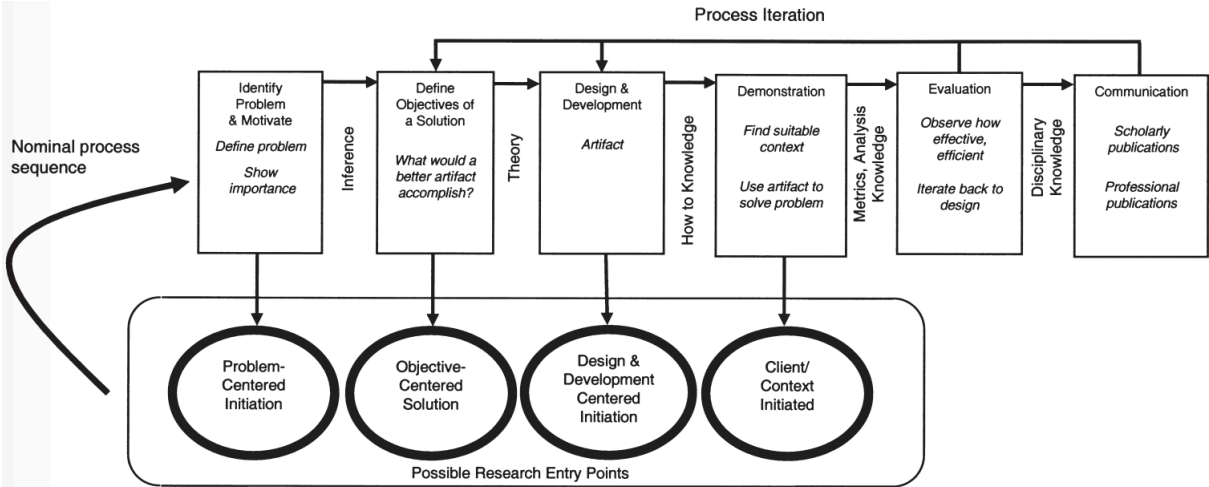


Figure 2-1 DSRM Process Model (Ken Peffers, 2014)

2.1.1 Identifying problem and motivate

Following the model above, the DSRM methodology starts by identifying a problem and a motivation to have this problem solved. There needs to be a real-world problem that is in need of a solution, as the DSRM methodology needs to provide some solution to be valid. The importance of solutions to the problems identified in this thesis is mentioned briefly in chapter 1 and more in depth in chapter 3.

2.1.2 Define objectives of a solution

After identifying the problem, objectives of a solution need to be defined. It is useful to look at the current situation and how things currently are being done. What would a new solution accomplish? It is of great value to do some abstraction from the actual implementation of a solution, so as to not get too one-sided. When not giving enough focus to identifying the problem, one could easily end up with a solution that does not solve the problem.

As the DSRM shows, it is possible to iterate back to this step and define new objectives if the evaluation discovers new problems.

2.1.3 Design and development

The design and development can begin when the objectives of a solution are clear. This activity in the DSRM framework may be the most practical as it this step often includes development of a prototype or a working software system. However, it does not need to be any kind of working software. The goal of this activity is to come up with some kind of artefact, that is intended to solve the initial problem. It is important that this artefact can be used to demonstrate that the artefact solves the problem.

2.1.4 Demonstration

After designing and developing the artefact, DSRM requires a demonstration of the artefact to be done. The demonstration is intended to show that the artefact solves the problem. The DSRM does not mention a specific way to demonstrate the artefact, as the artefacts may vary during the process iteration, but may be in the form of a simulation or appropriate activity.

2.1.5 Evaluation

In close relation to the demonstration comes evaluation. The evaluation is a more formal activity usually consisting of customer feedback or simulations. An evaluation may in theory consist of any empirical or logical proof. The evaluation is meant to be a comparison between the objectives of a solution and the observed effects of the demonstration.

The DSRM does not state that an evaluation needs to take place if there has been a demonstration. Usually the demonstration is of a more theoretical nature where a concept or idea may be tested, then designed and developed (process iterations), followed by an evaluation.

2.1.6 Communication

The last step in the DSR is communication. This step aims to contribute the findings back to the literature. This includes the identified problem, the artefact and its usability.

2.2 Methods

2.2.1 User testing

User testing was chosen when evaluating the effectiveness and qualitative properties, as well as technical challenges related to the user interface, of the proposed solution. The users, in this context, refers to the implementors and developers working with DHIS2. Even though the proposed solution is tested on the users working with DHIS2, it is not exclusively a DHIS2 solution. The users are more of a representation of competency profiles. The competency profiles will be explained more in-depth in chapter 4's evaluation.

When selecting the users to test the proposed solution Nielsen states that there is only need for a small amount of users, usually somewhere under 5 (Nielsen, Nielsen Norman Group logoNielsen Norman Group, 2000). The reasoning for this is that the first time someone test something, everything is going to be new, but the design coverage that has been tested can be as much as 33%. The second time another user tests something, a lot of the findings are going to be the same. It then makes more sense to iterate on the findings to develop a new solution to be tested.

Some important factors to consider when doing user testing (Reindal, 2018):

1. Remain neutral

In many of test scenarios, it is the developer or designer who are in charge of the testing. It is therefore important not to take criticism of the system being tested, as personal insults. Just as the testing is not about the skills of the user, the same can be said for the developers or designers.

2. Set clear expectations

It is important to set clear expectations at the start of the testing. Let the users know that it is not their skill level that is being tested, but rather the functionality of the system. When having clear expectations, the users normally feel more relaxed and more quality data can be collected.

3. Balanced guiding

It can be tempting to guide the user during the testing when the user feels stuck. It is important to ignore this impulse and rather than helping the user straight away, let the user figure how what to do in these kinds of situations.

This can lead to valuable insights. It is however important to find a balance and help the user if no progress is being made.

4. Give room to think

Usually the testing comes to a point where the user needs some time to think. It is important to listen to the user when this happens, and to let them continue thinking. By allowing the user time to think, some useful insights might appear.

5. Ask questions

To better understand the processes going on inside the users head it can be useful to ask questions, as some users are not used to think out loud. It is however important not to ask leading questions. A good rule of thumb is to be genuinely interested about what the users are doing.

2.2.2 Interview

Interviews can be an important part of exploratory research (Nielsen, Nielsen Norman Group, 2010) and as this thesis is more of an explorative nature and seeks to identifying problems and solutions to them, it makes sense to conduct interviews.

The interviews in this thesis are not standalone, but rather used as an addition to the user testing. There is little value in asking users what they think about a system a long time after they have used it (Nielsen, Nielsen Norman Group, 2010). The users will do their best to fabricate answers as they most of the time does not remember details about the system.

When choosing questions, it is important not to ask to specific questions about design. A question about whether the menu's background colour should be green or white won't give as much insights as observing how the user experience the colours when using the system.

The best insights from interviews comes from asking the user what they think about a system. It is best to do this not to long after the testing is done.

It is also important to note that users are great at giving their opinions on anything they are asked. This can lead to the users giving comments on things that does not matter (Nielsen, Nielsen Norman Group, 2010). Also be careful not to give leading

questions as this might force the users to give comments that don't reflect their opinions.

2.2.3 HISP UX Design lab at UiO

Health Information Systems Programme User Experience Design lab at UiO is an initiative started by Magnus Li, to share and discuss research with fellow students. As many of the master students working with the Information Systems (IS) research group have many of the same experiences and challenges, it is productive to have a place to meet and share knowledge. The process of writing a master's thesis can be a lonely one and it can be challenging to make progress on your own if you are stuck. It can also be intimidating to navigate a whole field of research on your own. Both of these are challenges I have faced myself and probably my fellow master students as well and are something the design lab solves really well.

2.3 Ethics

This thesis does have work related to other people, mainly the testing. This thesis is however not in need of personal or identifying information such as names. No personal identifying material have been saved and or included in this thesis. All individuals involved in this research project have been informed what kind of data will be collected and are aware of how it has been used. There have not been collected any signed agreement from the individuals, as this could be used to identify the individuals.

3 Theoretical framework

This chapter serves as a theoretical framework for this thesis.

3.1 A theory of tailorable technology design

(Allan MacLean, 1990) noted that it is *“impossible to design systems which are appropriate for all users and all situations.”*

Furthermore:

“Users must be provided with a rich design environment or design space in which technology can be tailored based on user-constructed parameters.” (Matt Germonprez, 2007)

The research article *A Theory of Tailorable Technology Design* identifies four definitional characteristics of tailorable technology. These four characteristics are then used to propose nine design principles to support design of tailorable technologies.

The four characteristics of tailorable technology are:

- 1. A dual design perspective that includes the initial design and user-defined tailoring*
- 2. User engagement that supports user tailoring through functional components*
- 3. Recognizable environment that supports user tailoring within the tailoring environment*
- 4. Component architectures that support functional and applied characteristics necessary in tailoring technology*

The *dual design perspective* mentions two design “phases”. The first phase is the initial design of the *default state* that provides functional components and an environment that lets the user tailor the technology. The second phase is the ongoing act of tailoring the technology based on the user-defined design.

User engagement in tailoring happens when a user can modify the use of the system outside of the intended task domain. A system will often be used in ways that were not anticipated, if the designers fail to create a simple set of task goals and operators. This is not a failure of user engagement toward tailoring but can lead to work around and improvisation. To succeed engaging in user tailoring, the system has to follow the structure of the task domain as well as allowing for modification of the system.

The designers have to create a familiar and *recognizable environment* that ideally provides the users with metaphors. This is done to promote better learning and innovative actions. Metaphorical technology and components can help the user with understanding the intent. Metaphors usually relate to something in the physical world, like the shopping cart of an online store. Using the metaphor of the shopping cart, it is easier for the user to understand the intent of the component. When the users understand the intent, it is easier for them to use that component in ways it was designed to be used, as well as using it in ways that not yet have been discovered.

Tailorable technologies have in part some relation to *component architectures*. Components architectures are defined as technologies consisting of mostly independent components working together, ideally creating one complex system. Components architectures can lead to easier development and maintenance, as the users/developers only have to focus on a small part of the system at one time.

3.1.1 Nine Principles in the Design of Tailorable Technologies

In relation to the four definitional characteristics of tailorable technology, the theory of tailoring proposes nine design principles to follow when developing tailorable technologies.

| <i>Environment</i> | <i>Principle</i> | <i>The Technology Supports...</i> | <i>Evidenced By...</i> |
|--------------------|----------------------------|--|------------------------|
| Reflective | Task setting | Variable tasks and problems. | Gargarian Madsen |
| | Recognizable Components | Components from existing technologies. | Alexander Pask |
| | Recognizable Conventions | Use patterns from existing technologies. | Alexander Pask |
| | Outward Representation | The context that it will likely be used in. This includes individual, group, and organizational. | Alexander Madsen |
| | Metaphor | Symbolic representation. | Madsen Pask |
| Active | Tools | Existing design tools. | Gargarian Pask |
| | Methods | Existing design methods. | Gargarian Pask |
| | Functional Characteristics | Functional requirements. | Alexander Pask |
| | User Representation | The representation of users. | Alexander Pask |

Table 3-1 Nine principles of tailorable technology (Matt Germonprez, 2007)

1. Task setting

Specify how a technology could be used and what tasks could be performed. However, do not prescribe when or how to use the technology, instead provide flexibility. Design focus should be on two main points, being;

1. *Supporting unknown user tasks through functional characteristics*
2. *Providing outward representations of the technology for users to tailor the technology*

2. Recognizable components

The technology as a whole should be designed with recognizable components in mind. This includes everything from the menu to page layouts, even the individual elements of the technology. Recognizable components would be components the users generally have seen and used before.

3. Recognizable conventions

The technology should use design patterns from existing technology. This includes recognizable ways of interacting with the technology, e.g. “point and click” is a way of interaction the users most likely will find familiar. This again could be tied to familiar ways of addition, rearrangement or removal.

4. Outward representation

Specify how the technology represent the individual, group or organizational context within which it is used. Technology must be inherently flexible in order to support the change of existing practices into ideal desired one, even when these ideals were imprecise. Promote cost saving.

5. Metaphor

Gives a definition of how the technology is described and communicated. Create metaphors that represent something real and relatable to the physical world, in order to better understand the purpose of the technology.

6. Tools

The technology should present the users with tools that support guidance. It should present means for selecting the different elements that comprises the technology. There should be mechanisms for changing the aesthetics.

7. Methods

The tailorable technology should support methods e.g. for learning. An example of this is the “design-by-learning” method, that encourages the user to tailor based on feedback from the technology.

8. Functional characteristics

The functional characteristics comes from component-based technologies, whereas the component-based technologies include five functional requirements being:

Splitting, Substituting, Augmenting, Excluding and Porting.

Splitting refers to the act of splitting one components into smaller pieces that can individually be used.

Substituting refers to the act of replacing on component with another, possibly better or different type.

Augmenting is the act of including new components to the technology.

Excluding refers to the act of removing a component from the technology.

Porting refers to the act of transferring a component from on technology to another.

9. User representation

Users should be represented in the technology by allowing them to provide feedback on the technology. Users should be provided with training on how to use the technology.

3.2 Design for design

Meta design or design for design as a concept aims to create technical infrastructures that encourages end users to extend the functionality of a piece of technology or software. This is done by letting the end users become co-developers or co-designers (Giaccardi, 2004).

Referring to what previously was stated by MacLean, one could deduct that there is no one piece of software that can support any and all of the individual needs (local needs) of the end users. A solution to this would be to bring the end users into the design process, as this is more likely to have the end users work with a software solution that fits their needs.

This can however be a challenging task for the creators of meta design technology, as there usually needs to be a balance between usability and flexibility in functionality (Dvořák O., 2017). One could argue that the more complex the technology is, the more use cases can it handle, which again would require more knowledge and experience from the end users. On the other side, if the technology is designed to solve one specific task, the complexity may be lower, resulting in less experience and training is required from the end user.

3.3 Levels of design

Magnus Li and Petter Nielsen presents two new terms in regard to design development in generic software, generic level design and implementation level design (Li M. &, 2019).

Generic level design refers to the design and development taking place during the development of generic software, whereas the implementation level design refers to the design, development and/or configuration of the generic software. It is important to mention these levels, as the method of design are different.

The generic level design is more focused on the traditional process of software development, where the end result is a finished product, as an opposite to implementation level design. Implementation level design focuses on the design happening after the generic level design process. The generic level design is normally executed by experienced developers, whereas the implementation level design is done by the end users.

3.4 Generic software

Generic software is software that is developed to be used “right out of the box”. If it was possible to mass produce software, then generic software would fall under that category. Instead of mass producing the software itself, it is rather the use and application of it that is produced to fit the masses. A generic software will select a large consumer segment, like health organizations, and try to create a product that covers most of the general needs. It is however difficult to cover all the needs of all the users (Revision World Networks , 2019).

A generic software is also a time and money saver because the need for ground up development is taken out of the equation. The configuration of generic software is usually easy to take care of for users with little to none experience in software development. The configuration should also be quick, so that the software is up and running fast.

The challenges of generic software come in to play when the user needs some specialized feature. It is possible to, if the source code is open, implement some new feature to the generic software, but this again would require someone with enough experience in coding and software development. If the user does not have this software development experience, there would be needing to hire someone, which would raise the expenses related to the software.

3.5 Current situation

In addition to the challenges mentioned it is useful to look at the current functionalities of DHIS2. The DHIS2 platform is designed as a piece of generic software. There is a standard set of functionalities such as data gathering and data visualization. DHIS2 offer configurability in the form of a maintenance application and customizable dashboard (DHIS 2 Documentation team, 2019).

As a supplement to offering configurability, the DHIS2 has an application programming interface (API) that can be interacted with by applications.

In short, the applications of DHIS2 can be divided into applications developed by DHIS2's core team, and applications developed by third parties. The applications can either be created as dashboard applications. This is a smaller application, usually created for data visualization on a dashboard. A standard DHIS2 application that are run inside DHIS2 but works and looks like a normal web application. Lastly, a standalone application ran on a different server.

3.5.1 Description of existing editor solution

It is important to note that the findings and goals of this thesis is not to aimed at replacing the current form creator/editor, but there should be a mention of it, as there can be gathered insights from it.

The current editor is a CKEditor (CKSource, 2019) that is mainly used to create data entry forms, but as shown in the interviews, the editor has been used to create app-like results. Because the editor exists of a WYSIWYG (What You See Is What You Get) interface, with the possibility of direct text/HTML/JS/CSS entry, it is possible to enter some additional JavaScript code to enhance the capabilities of the resulting data entry forms. This however comes with a great drawback in maintainability. In the worst cases the developers may end up maintaining code as shown below:

```

(editor,opt={}){const c=opt;const domc=editor.DomComponents;const defaultType=domc.getType('default');const defaultView=defaultType.view;const textModel=textType.model;const textView=textType.view;const idTrait={name:'id',label:c.labelTraitId,};const forTrait={name:'for',label:c.labelTraitFor,};const placeholderTrait={name:'placeholder',label:c.labelTraitPlaceholder,};const valueTrait={name:'value',label:c.labelTraitRequired,};const checkedTrait={label:c.labelTraitChecked,type:'checkbox',name:'checked',;control:'handleClick',,handleClick(e){e.preventDefault(),});domc.addType('form',{model:defaultModel,draggable:'not(form)',traits:[{type:'select',label:c.labelTraitMethod,name:'method',options:[{value:'on',name:'action',}],},],init(){this.listenTo(this,'change:formState',this.updateFormState)},updateFormState('success');break;case 'error':this.showState('error');break;default:this.showState('normal');failVis='none';successVis='block'}else if(st=='error'){failVis='block';successVis='none'}else{failVis='none';successVis='block'}else if(st=='error'){failVis='block';successVis='none'}else{failVis='none';successVis='block'}},getStateModel('error');var successStyle=successModel.getStyle();var failStyle=successModel.getStyle();failModel.setStyle(failStyle)},getStateModel(state){var stateModel=comps.models[i];if(model.get('form-state-type')=='Success;if(st=='error'){contentStr=formMsgError}stateModel=comps.add({'form-state-type':st,type:'form-state-type':st,content:contentStr,})}return stateModel},},{isComponent(el){if(el.tagName=='FORM'){return{type:'form',model:defaultModel.extend({defaults:{...defaultModel.prototype.defaults},droppable:!1,traits:[nameTrait,placeholderTrait,{label:c.labelTraitType,type:'select',name:c.labelTypeEmail},{value:'password',name:c.labelTypePassword},{value:'number',name:c.labelTypeNumber}],view:defaultView,}),view:defaultView,});var inputType=domc.getType('input');var inputModel=inputType.model.extend({defaults:{...inputModel.prototype.defaults,name:c.labelTextAreaName,tagName:'textarea',title:c.labelTextAreaTitle},view:defaultView,});domc.addType('select',{model:defaultModel.extend({defaults:{...defaultModel.prototype.defaults},name:c.labelSelectName,tagName:'select',traits:[nameTrait,{label:c.labelTraitOptions,type:'select-opt

```

Figure 3-1 Example of minified JavaScript code

There are also no restrictions on the colours, layout or images that can be added to the data entry forms. In short, there are no guarantees that the forms created in this editor will be user friendly to the end users.

Below is a picture of the existing editor used in DHIS2 to create HTML forms.

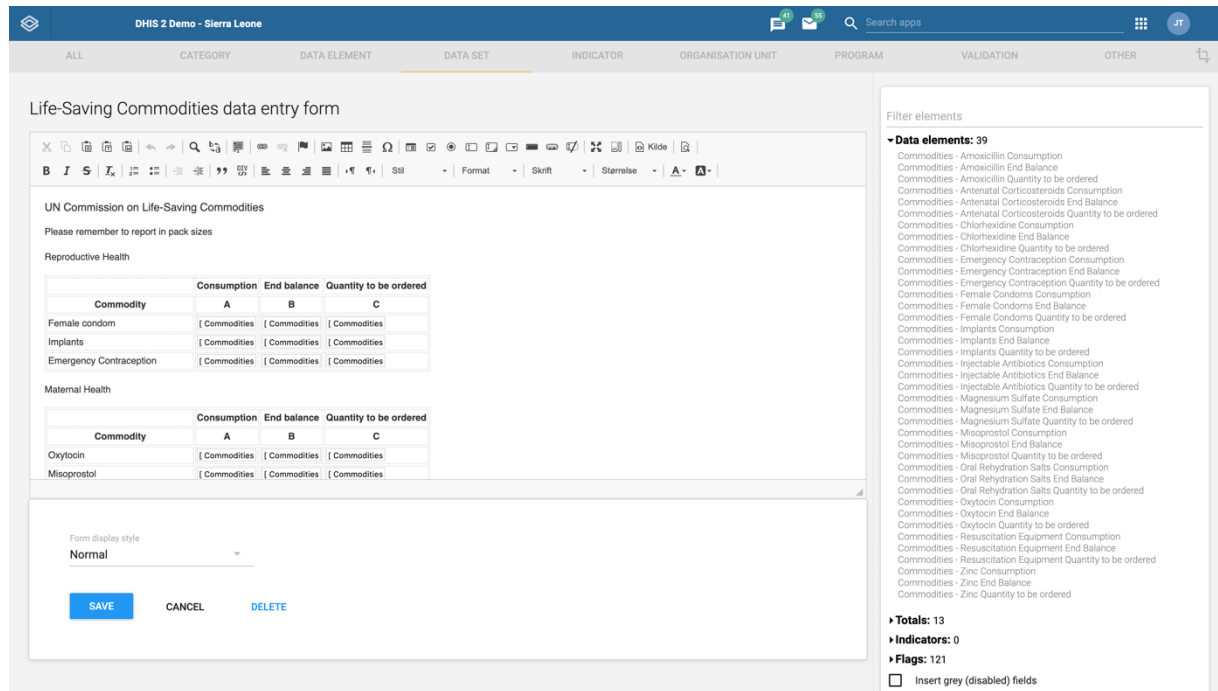


Figure 3-2 Data entry editor with database properties

And below is a picture of the live form created in the editor above.

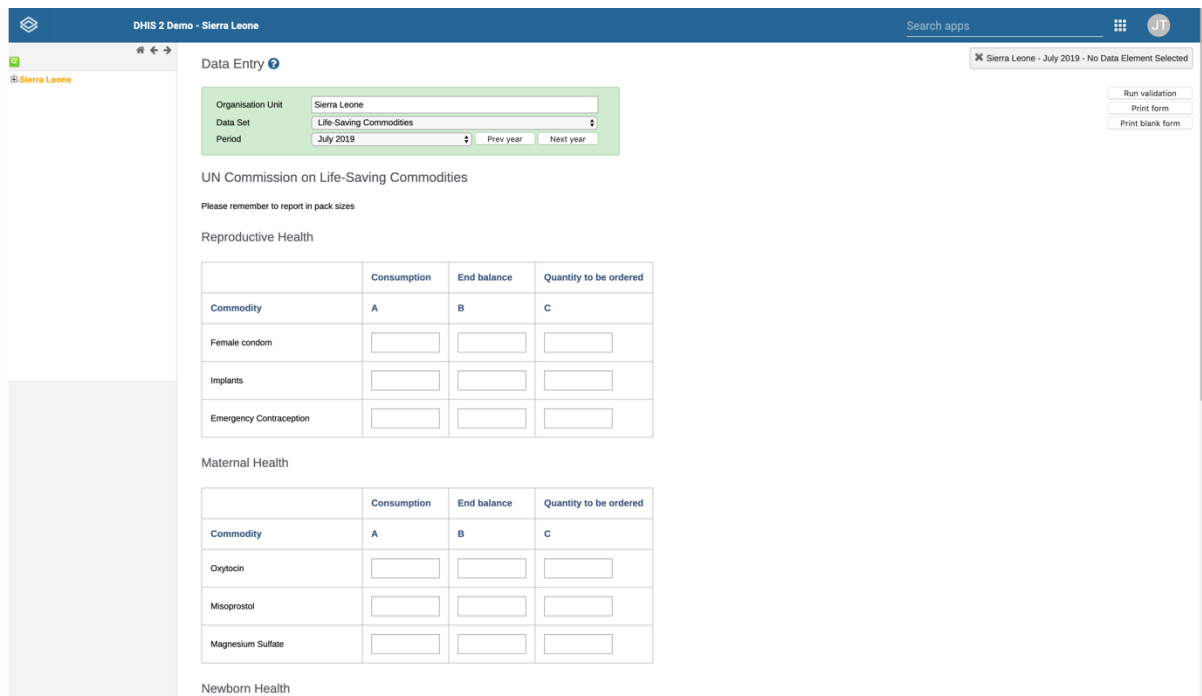


Figure 3-3 Resulting data entry form

The existing editor works by first creating a mark-up e.g. a HTML table structure. Then create a header text in the table header. Then create an input box. The input box can be linked with the database by selecting database elements from the overview on the right. To do this the user has to click inside the input element and then click on the database element in the overview to the right.

3.6 Technologies

This project aims to develop a web application using JavaScript as the main programming language. According to the 2019 Stack Overflow developer survey JavaScript is the most frequently used programming language for the web (Stack Overflow, 2019) and can run on every browser with little to none custom configuration (Mozilla, 2019). This makes JavaScript an ideal language for open source projects as there can be many contributors, with a wide set of use cases.

3.7 User interface design

Jakob Nielsen defines 10 “rules of thumb” called heuristics related to user interface design (Nielsen, 1994). These are not definitional user interface design guidelines, but important pointers when it comes to user experience. Nielsen speaks about the timeless nature of the heuristics and how they have proven effective in the past, present and most likely the future. Therefore, it is a good practice to make sure the heuristics are well considered when designing a user interface.

Following the 10 heuristics, Nielsen initially mentions the importance of clearly showing the status of the system. Showing system status can cover many things, but the main point is that the user is getting feedback on what the system is doing. Furthermore, the system should show resemblance to the real world, e.g. by using normal words instead of words and terms used by developers. The users should have full control of what’s happening, and they should be provided with options to return to system default state. There should be a consistency in naming conventions for the systems informative components. Errors should be handled in a way that is useful to the user, e.g. showing a custom error message instead of a system generated message. The user should not be forced to remember every part of the system; therefore, it is important that everything is clearly visible. The system should have an overall simple design and should not contain any excess text or components. Finally, the system should offer help to the user, normally in the form of documentation.

3.8 Minimum Viable Product (MVP)

To better evaluate the contribution of the theory of tailoring in generic software, some kind of MVP is going to be created.

A Minimum Viable Product (MVP) aims to provide just enough functionality to test a certain hypothesis. This is not meant to be a fully working piece of software and is most not meant to work as a framework or base layer for future development. The most important quality of an MVP is to give insights and information so that future development will be more complete (Reis, 2011).

Generally, in software development there has been an idea that to test some piece of software, you need to have the whole thing done. This is risky as sometimes the hypothesis is wrong, and a lot of valuable time and money has been spent developing something the end user does not want or need (Mavuru, 2018).

4 Process

This chapter describes the process of developing artefacts according to the DSR framework, as well as planning the project as a whole.

4.1 Preparations and planning

4.1.1 Creating a timetable

Below is an overview of the rough schedule set from week 24 to week 44. It is useful to have a timetable even when working alone, as this can help ensure a stable and continuous progression.

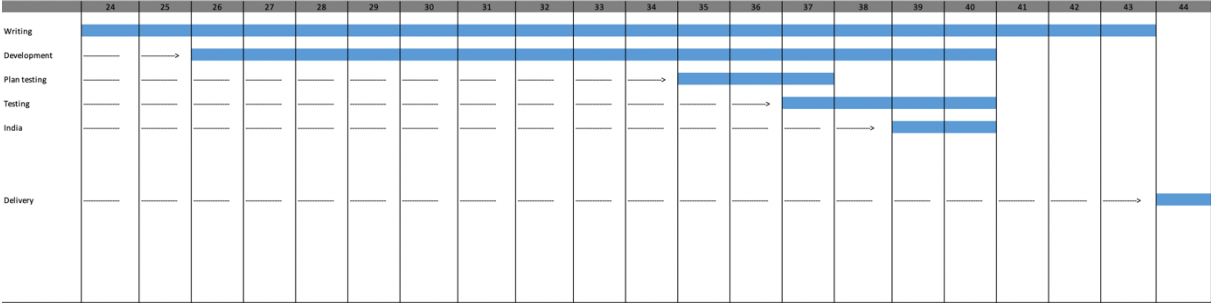


Table 4-1 Timetable week 24 to 44

This thesis ended up with an extended schedule as the delivery date of the thesis was moved from 1st of November to 3rd of February. The main task during the period from week 44 to week 5/6 was refinement of the contribution and writing of the thesis.

The extension of the thesis work was partly due to a trip to India that had to be cancelled. The plan for this trip was to test out an MVP and learn more about the current situation. This introduced new limitations to the thesis work.

4.1.2 Task management

To manage tasks and other “to do” items, a mix of Trello and Todoist have been used.

Trello works as a customizable Kanban board, with lists, tasks and labels. This is a useful tool to remember tasks that needs to be done, as well as prioritizing the tasks. There are different ways of prioritizing tasks. One method of prioritizing is based on the importance of the tasks. I have done a similar way of prioritizing; in that I place the most important tasks at the top of each list. In addition to sorting the lists, I've created separate lists based on the level of focus needed to complete the task. As I am naturally more focused in the mornings, more challenging tasks should be completed then. Tasks related to development and coding are naturally more demanding than tasks related taking screenshots of related material.

Todoist is a tool that lets the user create custom checklists. This can be used to create lists of literature and source material.

4.1.3 Development tools

Visual Studio Code

There will be need for tools in relation to software development. From my experience working with web development and JavaScript, I feel confident using a simple tool like Visual Studio Code.

Visual Studio code is a free and open source editor that provides simple yet useful tools like IntelliSense, debugging and git commands. In addition, there is the possibility to extend the editor further by downloading plugins (Microsoft, 2019).

Git/GitHub

Git and GitHub will be used to keep track of the development process and changes. Git is a version control tool used to track changes in any type of file but is mostly used in software development to track changes in codebases (Git, 2019). GitHub is providing remote hosting of Git code bases, in order to facilitate collaboration, both in open and closed source codebases.

Even though I am the only one working on this codebase, it is useful to have some form of version control. It is also a good practice to use version control even on solo projects as this is frequently used in the business world. GitHub is also used in this thesis to automate builds on Netlify.

Netlify

Netlify provides serverless application hosting (Netlify, 2019). By using a hosting service testing can be more efficient as multiple users can test applications simultaneously on their own computers. Netlify is a good option as the serverless architecture mitigates a lot of the standard setup needed to run a server. This frees up more time to development.

4.2 Process iteration introduction

The following chapters describes the process of developing artefacts to serve as solutions to the mentioned challenges, by utilizing the principles mentioned in the theory of tailoring.

Each process iteration shown below will produce an artefact intended to solve the initial problem and related challenges, which then again is used to serve as starting point for the next iteration or as a final solution. It is important to note that the artefact produced each iteration may change.

The design and development followed three main process iterations, as a part of the DSR framework. These iterations being;

1. Initial review of the problem, literature and reflection, resulting in an “ideal artefact”
2. Development of an MVP by implementing some of the functionality of the ideal artefact
3. Applying the learning outcome of the MVP to the ideal artefact

As shown in the layout, the end result is an ideal artefact, which consists of four design principles.

4.3 Initial review and ideal artefact

An initial *ideal* artefact was created in order to get a better understanding of what needs to be done, in order to meet the problem mentioned. The ideal artefact was created by reviewing the problem, the relevant literature and personal reflection. The term ideal artefact is used as this artefact is meant to serve as the “ideal” solution to the challenges mentioned. Its purpose is to serve as a starting point when trying to develop a more tangible artefact.

4.3.1 Design and development

This chapter will start by initially reviewing each of the areas of challenge against the existing literature, with some of my own reflections. I will in the end summarize how each of the principles in the theory of tailoring is covered. This will result in an ideal artefact. The ideal artefact does not explore the technical difficulties of implementation, mainly because it's only serving as a theoretical starting point for the next process iteration.

It is important to note here that the theory of tailoring's principles is not necessarily aimed at solving all of the challenges mentioned, but by generalizing the principles they may provide as useful insight. The main difference between the theory of tailoring and this thesis is the fact that this thesis's goal is creating principles that lets users create software, whereas the theory of tailoring is more focused on configuring already made components to fit the user's needs. I am however using the theory of tailoring as the users are not likely meant to write code, but rather do visual programming. You can say that this thesis will be focused on *splitting* the components mentioned in the theory of tailoring into even smaller components, that then can be put together to create a wider range of components.

Magnus Li does give mention of a drag and drop solution. I keep that in mind but try to focus mainly on the challenges in this chapter. Drag and drop might prove to be the best option, but I need to first clarify what the ideal artefact is trying to solve.

The process started with reviewing the problem mentioned in chapter 1.4. Here, five main areas of challenges were identified. In short, there is:

- 1. Local needs**
- 2. Funds**
- 3. Skills and experience**
- 4. Time**
- 5. Usability**

As mentioned, all of these challenges can be summarized into a lack of resources. This is important to notice, because they have a shared effect on each other, e.g.

where there is a lack of funds, there is not possible to hire good developers and that again will result in a poor coverage of local needs.

So, let's say now then, that the ideal artefact we are trying a create here is going to solve all of these problems. By no specific selection process, let's start at the top, with the challenge of local needs.

Local needs

It is established that it's not possible to create a single piece software that covers all the needs of the users. There is however possible to create software that by design is meant to be changed to fit the needs to the users. This does fit in with the general purpose of tailorable technologies. The ideal artefact needs to function in a way that allows for change, that is not too limited based on context.

Funds

There is an argument to be made for cost savings by having the end users design their own software. This does again tie in with the idea of end user development in tailorable technologies. It would be valuable if the end users can do many of the tasks previously only taken on by a developer.

Skills and experience

The need for skills and experience can be lowered by having a simple tailorable environment. Simple in this context refers to the limitation of options related to the context. The more menu option the user is presented with, the more training the user needs. The ideal artefact should then support a user to user-based configuration that allows for the artefact to limit options based on context.

Time

The ideal artefact needs to improve on the time spent developing or doing configuration. Ready-made components and responsive design environment can help save time. The time part of the problem does mostly have to do with developers, as inexperienced users don't have the ability to create software in the first place. Time can be saved by the developer e.g. when using the ideal artefact to create templates that can further be more customized with traditional software development. Another time-saver would be the ability to let more users contribute to the software design process, e.g. by letting in-experienced users tackle simpler tasks that they previously did not have the ability to do. This again would free up time with the developers so that they can work on more complex tasks.

Usability

The ideal artefact needs to support a user-friendly interface. The 10 design heuristics mentioned in chapter 3.8 would be a good place to start when designing a good user interface. Another important factor in the usability of the artefact is the end result. The end user interface experience needs to be satisfying in order for the ideal artefact to work. In addition to smaller and less complex building blocks, there should be an option to implement ready-made components. Many design decisions can be avoided by having complete components, which again would open up for a wider range of contributors.

General requirements

It is important that in addition to the ideal artefact having good user experience, it offers maintainability in the tailored items of the users. Most modern web technologies are in need of continuous maintenance and development. This can come as a result in browser updates or change in user requirements.

The ideal artifact would be something that can serve both experienced developers and implementors. The ideal artefact can allow experienced developers to create templates that may save them development time.

A configuration file should be implemented in order to simplify the configuration section. This configuration file may include colors, fonts and specific components/elements that the user can use in the design process. A configuration file does not need to be altered a lot, let's say every organization may have its own standard file. This is something a single developer won't have to use a lot of time to handle.

There should also be possible to use components and configuration files that other organizations or users have created.

The main challenge of creating an ideal artefact is to maintain the balance between complexity and usability. There is not really any general agreement on the relation between complexity, where complexity refers to the ability of the software to solve complex problems and tasks, and usability, where usability refers to the levels of skill and experience required of the user in order to fully utilize the software. But generally, if someone tries to create software that can handle any use case, it's going to have a lot of features, which in order may demand more of the users.

4.3.2 Demonstration

The coverage matrix below shows how the challenges relate to the principles of the theory of tailoring.

| | Local needs | Funds | Skills and experience | Time | Usability |
|----------------------------|-------------|-------|-----------------------|------|-----------|
| Task setting | x | x | | | |
| Recognizable Components | | | | x | x |
| Recognizable Conventions | | | | x | x |
| Outward Representation | x | x | | | x |
| Metaphor | | | x | x | x |
| Tools | | | x | x | x |
| Methods | x | | x | | |
| Functional Characteristics | x | x | | | |
| User Representation | x | | x | | x |

Table 4-2 Theory of tailoring coverage matrix

The matrix shows that all of the challenges are to some degree covered by the theory of tailoring. Most notably is the *local needs* and *usability*. This makes sense as the theory of tailoring emphasizes the inclusion of end users in the design of the technology. Second is the *skills and experience* and *time*, which again is something the theory of tailoring covers well with the principles on familiarity conventions. Lastly, one factor not directly mentioned in the principles is *funds*, except in the case of *outward representation*. There is however possible to infer coverage in several of the principles.

By looking at the requirements and the coverage matrix, it is in theory demonstrated that an ideal artefact that follows the requirements mentioned above will provide solutions to the mentioned problems.

4.4 Development of MVP

To further understand the contribution the theory of tailoring can make to the challenges related to local needs in generic software, the next process iteration will cover the development and evaluation of an MVP. The MVP was based on the findings in the previous chapter, in order to test these findings on actual users and the tailoring principles in practice. As previously stated, the MVP was not designed to be a fully working piece of software and should not be directly copied in further research.

4.4.1 Design and development

The main focus on the development of the MVP is functionality related to the tailoring principles, i.e. the interaction of the users with the design environment. This is also the reason for why no time has been used to test the technical aspects, like uploading the MVP to DHIS2, as well as uploading the applications created in the MVP to DHIS2. Further discussion on potential technical areas not tested can be found in this chapter, as well as chapter five.

The MVP is going to be designed as a standalone addition to DHSI2. The MVP needs to simulate some form of activity in relation to user needs in DHSI2. One big part of DHIS2 is data collection. This data collection is done mostly through web forms created by the end users. The web forms are mainly created in a CKEditor, mentioned in chapter 3.5.2. As mentioned, there is a big difference in the design environment and the end result, see below pictures for another comparison:

| Management and supervision | |
|---|------------------------|
| Question | Response options |
| Are national guidelines and standards for FP and CAC service provision available in each unit? | [CMC National guide |
| Have staff in the FP and CAC units received a facilitative supervision visit from on-site / off-site supervisor or ABRI project staff during the past six (6) months (Verify through review of dated supervision report at facility that specifies problems discussed action steps recommended) | [CMC Staff in FP/CAC |
| Is this facility part of health care finance (HCF) board scheme? if so; does the facility use part of the revenue generated for FP/CAC related services? | [CMC Facility part of |
| Do FP and CAC service units have dedicated hours or space for youth clients (Verify through observation) | [CMC FP and CAC se |

Figure 4-1 DHIS2 data entry form in CKEditor





| Management and supervision | |
|---|--|
| Question | Response options |
| Are national guidelines and standards for FP and CAC service provision available in each unit? | <input type="radio"/> Yes <input type="radio"/> No  |
| Have staff in the FP and CAC units received a facilitative supervision visit from on-site / off-site supervisor or ABRI project staff during the past six (6) months (Verify through review of dated supervision report at facility that specifies problems discussed action steps recommended) | <input type="radio"/> Yes <input type="radio"/> No  |
| Is this facility part of health care finance (HCF) board scheme? if so; does the facility use part of the revenue generated for FP/CAC related services? | <input type="radio"/> Yes <input type="radio"/> No  |
| Do FP and CAC service units have dedicated hours or space for youth clients (Verify through observation) | <input type="radio"/> Yes <input type="radio"/> No  |

Figure 4-2 DHIS2 data entry form in data entry application

The pictures show an outtake of a form. The first picture shows the web form in the CKEditor, and the second picture shows the web form in the data entry application. This is a simple form, but there are already some clear design differences in the two. It is hard to design anything without seeing the end result.

Now, this thesis does not aim to replace the CKEditor, but rather using the basis of web form creation in the MVP. This way it is possible to test some real-world use cases of the local needs, and maybe give some pointers on how to improve the current editor.

The goal of this process iteration was then to test out the tailoring principles in practise on a real-world scenario (web form creation), through the process of meta design. The MVP should then be created as a *tailoring environment*, somewhat similar to the current editor, but with the premonition of more extensive functionality in the future.

Choosing technologies

Since the goal of this iteration was to research the use of a tailoring environment in generic software, and not the direct implementation of one, a sort of framework was to be considered. By choosing such a framework, or engine, it is possible to save valuable time not having to do a lot of standard coding and setup. It is also useful to test out existing frameworks and engines, so that in the future, time could be saved when creating a production application.

The theory of tailoring mentions recognizable conventions as one of its principles, such as “point and click” and “drag and drop”. It then makes sense to try to implement such features in the MVP.

There are a lot of “drag-n-drop” type website builders. Some of them like Wix (wix.com, 2019) and Squarespace (Squarespace, 2019), offer drag and drop building functionality. Unfortunately, the source code on those kinds of websites are not open source. To be able to do the testing needed for this project, the source code needed be open source. This is because the logic behind the system may be useful, and because there needed to be possible to do customization.

Another thing to consider when talking about customization is the structure of the source code. Preferably, the source code should be created in one of the leading JavaScript frameworks or libraries such as Angular, Vue or React.

Grapesjs

One editor to be considered is the Grapesjs editor. This is open source. It does not contain any excess components and is simple to set up (Arseniev, GrapesJs, 2019). It is created using the Backbone JavaScript library (Arseniev, Github, 2019). The Backbone library is unfortunately not advised to use any longer due to old principles of application architecture (Bekk Consulting AS, 2017). This however should not be a limiting factor when using Grapesjs as an MVP in this project, but this also means that Grapesjs should not be used for future development.

Other frameworks that were considered were:

VvwebJs (Givan, 2019)

Microweber (microweber, 2019)

After much consideration it was the Grapesjs editor that would fit this project the best. Although it is not recommended to use, due to outdated architecture, it has a lot of the functionality that is useful to test in this project. The other options were either too extensive and not worth trying to customize, like Microweber, or was too limited in functionality like VvwebJs.

Adjusting the editor

Although the Grapesjs editor is meant to be an all-purpose HTML/CSS editor, some adjustments had to be made. Fortunately, the Grapesjs editor comes with good documentation (Arseniev, Grapesjs, 2019) on how to customize the editor. The main part of the customization was done using the ready-made plugin extensions provided with the editor. The plugins that were used were:

- grapesjs-plugin-forms

The forms plugin gives basic form elements such as input, textarea, radio button and checkbox, in addition to the form element itself.

- grapesjs-blocks-basic

The basic blocks plugin gives HTML elements like image and text. Some of the forms created today uses an image as header banner.

- grapesjs-plugin-export

The export plugin lets the user export (download) the HTML and CSS from the editor. This is useful as this can simulate the process of template building with templates that's going to be further customized outside the editor, by a developer. A typical use case would be to download the code, and then upload it to the DHIS2 platform as a standalone application.

Making sure the editor implements the tailoring principles

- Task setting

The inherent design flexibility of the editor does not set limits to what the user can create.

- Recognizable components

The components in the MVP consists of single HTML elements and a more complete form component. The form component is possible to configure in the popup menu at the start.

- Recognizable conventions

Much of the conventions and patterns used in the MVP can be found elsewhere, like "point and click" and "drag and drop".

- Outward representation

Colors matching the DHIS2 where implemented in order to give a more organizational context.

- Metaphor

The metaphor of a digital makerspace where used in describing the MVP to the users.

- Tools

The MVP gives menus for selecting pieces. The menu consists of text and icons that where intended to guide the user.

- Methods

The MVP supports “design-by-learning”.

- Functional characteristics

The MVP is somewhat limited in the relation to supporting heavy technical flexibility right out of the box. It does however as mentioned serve as a template builder, that indirectly can support more technical solutions.

- User representation

The MVP does not directly support feedback from the users, but HISP does have a frequently used online forum that would serve better as a way for the users to give feedback. A short intro guide where provided the users testing out the MVP.

A more detailed look at the source code can be found at:

<https://github.com/Ullvar/dhis2-dragdrop>

Chapter 4.4.3 gives a visual representation of the final MVP.

4.4.2 Evaluation

The plan was to test the MVP on people with similar levels of experience and competence as the implementors and developers working with DHIS2. The reason for this is that the MVP is somewhat detached from the DHIS2 system, meaning that it gives greater value to test the implemented tailoring principles, than the actual integration in DHIS2. The test case however was something that could be a real-life scenario working with DHIS2. As the access to real life implementors and developers was limited due to the limitations of this project, people with similar skills was selected for testing.

Testing

The testing of the prototype was done in three parts. First some questions to establish who the test user was and their level of experience. Then, the user was asked to re-create a “paper” form in the MVP. If the user had experience with coding, the user where also asked to create the form using standard methods of coding, in order to compare the two. Lastly, the user was asked questions about the MVP, and

to give a comparison between the standard way of coding and using the MVP. The last part was just for the persons with experience similar to that of a developer. Some of the questions did change between the implementor and developers.

Main Goals

The main goal of the user testing was to demonstrate and evaluate the MVP created, following the design principles of tailorable technologies.

The following points were taken from the nine design principles. Some of the nine principles can be demonstrated without user testing. The selection of principles below was the ones that would benefit from user testing.

The numbers in front relate to the questions asked at the end of the user testing.

How does the MVP support:

- (1) Technical flexibility? (**Functional Characteristics**)
- (2) Design by learning? (**Methods**)
- (3) Point and click (drag and drop)? (**Recognizable Conventions**)
- (4) Existing components that are approachable and usable? (**Recognizable Components**)
- (5) Flexibility in what tasks could be performed? (**Task setting**)
- (6) Description of the technology as something real? (**Metaphor**)

Competency profiles

The main job of the implementor, is to handle customer contacts, gathering of requirements and configuration of applications. The implementors usually have lower degree of coding skills than the developers. Some test users with similar background were selected to act as implementors.

A good selection would be someone who have little to no experience in coding, but a general understanding of IT solutions, e.g. Microsoft Excel.

The main job of the developer is to customize the DHIS2 core and applications, as well as creating new applications. There is no clear requirement of any formal training, but it is usually someone with experience in coding.

A good selection would be someone with some, to high experience and knowledge of coding.

Findings

Two persons that fit the developer profile and one person fitting the implementor profile did contribute in the testing. There was a plan to involve one more implementor, but the findings from the first three tests did not warrant for another round of testing.

The general feedback from the testing where positive and the users found the MVP useful. When the developers where asked to create a web form from scratch a few interesting insights where discovered. The first ting where that both the developers started out with copying HTML and CSS, in order to have some form of template. The most interesting thing where that they chose two different web form layouts. Whereas on developer chose the layout to consist of div elements and the other developer chose to use a traditional table structure. This shows the need for a more standardized way of creating templates when it comes to developers.

There were not enough tests to really test the effectiveness of the MVP compared to standard development methods. The developers were however timed when creating the form both times, and in both cases the developers took longer to create the form using standard development methods. One developer also noted that the faster design process of the MVP was preferred over standard methods of design.

In all test cases the users where given a short introduction and some time to play around with the MVP. This seemed to help the users get familiar with the MVP. The users were also not afraid to play around with the MVP design environment, as they felt in control of the space. On developer mentioned that the drag and drop feature gave more control over the design process.

All the users found the MVP design space familiar and usable. One developer noted that the configuration prompt at the start where really useful and could see that being expanded more in the future. Both the developers found the menu useful as it displayed all the elements and components with both icons and text.

The implementor did actually mention that the possibility of having some kind of scripting would be useful. This is backed by the developers who said that the MVP would be harder to use for more complex tasks.

4.4.3 MVP overview

This chapter aims to give a brief overview of the MVP created.

The first thing the users see when entering the MVP is the “configuration” prompt. Currently the users can only select rows and columns in the form component. This configuration prompt may be extended in the future in order to do more pre configuration, possibly by uploading a configuration file as well.

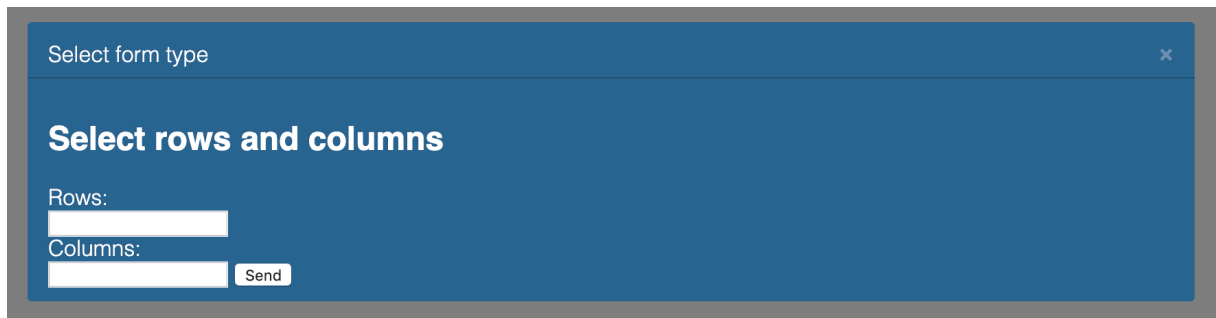


Figure 4-3 MVP configuration prompt

After entering the preferred configuration, the users are met with the design environment, with the clearly visible element/component menu on the right. On top is the more general menu related to the MVP configuration.

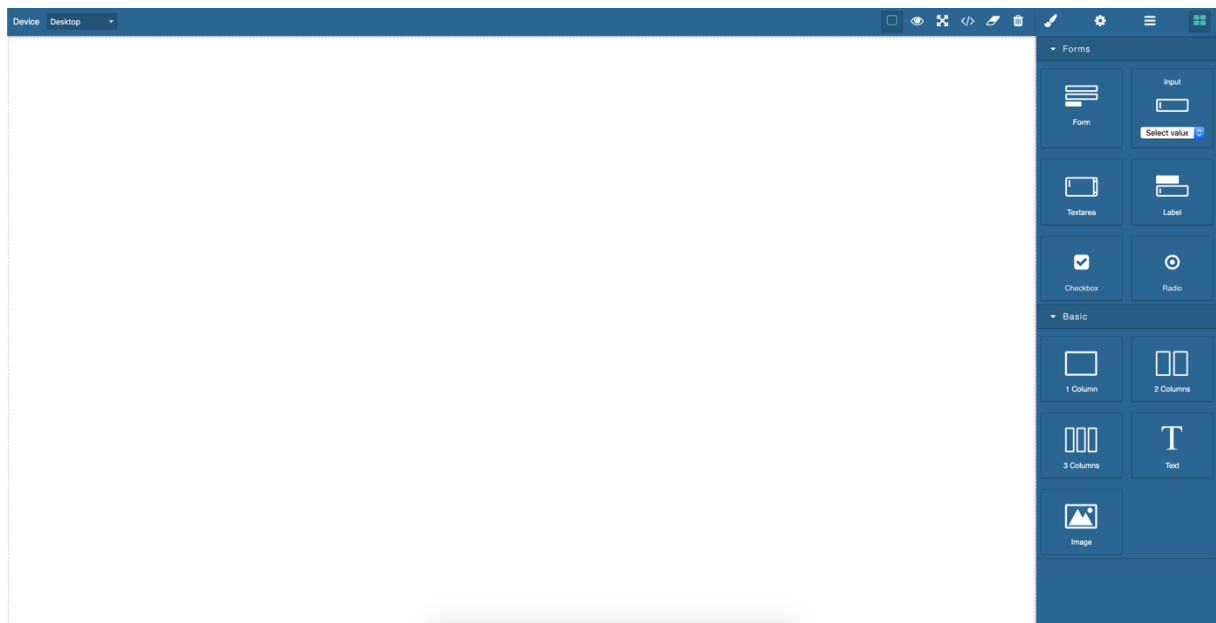


Figure 4-4 MVP design space

The right-hand menu aims to give a good description of the elements and components with a mix of labels and icons.

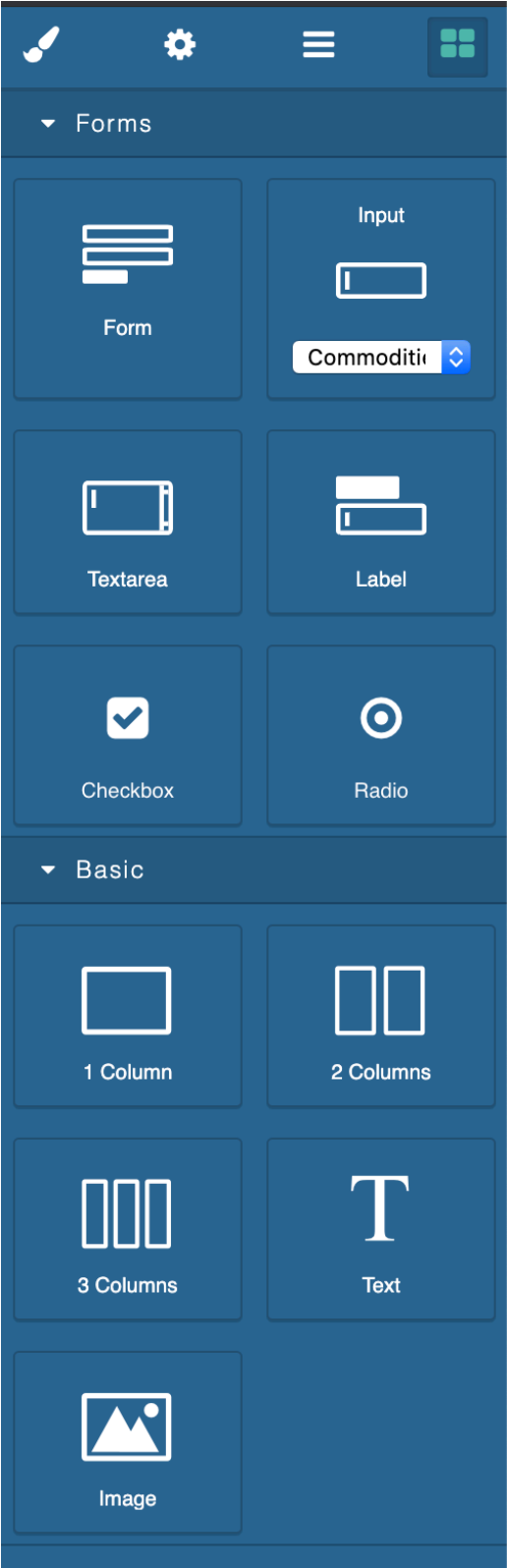


Figure 4-5 MVP Component and element menu

The input element lets the users select what database field the input is going to be related to. The MVP did only use mock data for this.

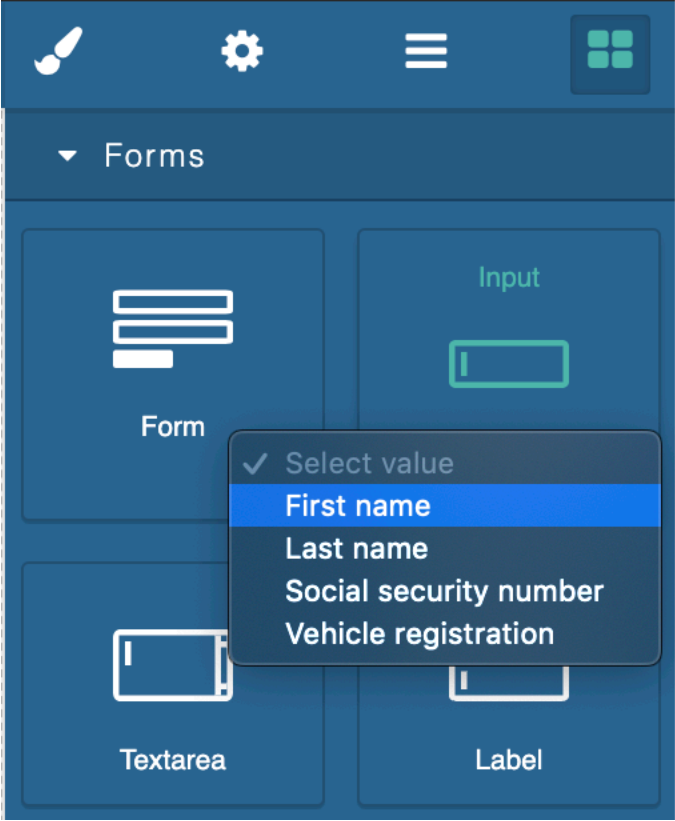


Figure 4-6 MVP input element with database field select

The top menu left lets the users select different views in order to test the responsiveness of the web forms created.



Figure 4-7 MVP device view selector

The right side of the top menu lets the users view the end result (eye icon), download the code (</> icon), clear canvas (eraser icon) and delete configuration settings (trashcan icon).



Figure 4-8 MVP top menu

The last picture shows the full MVP with the template form already dragged in the canvas. The input fields have also been dragged in. It is possible for the user to edit all the rows and columns in order to customize the web form.

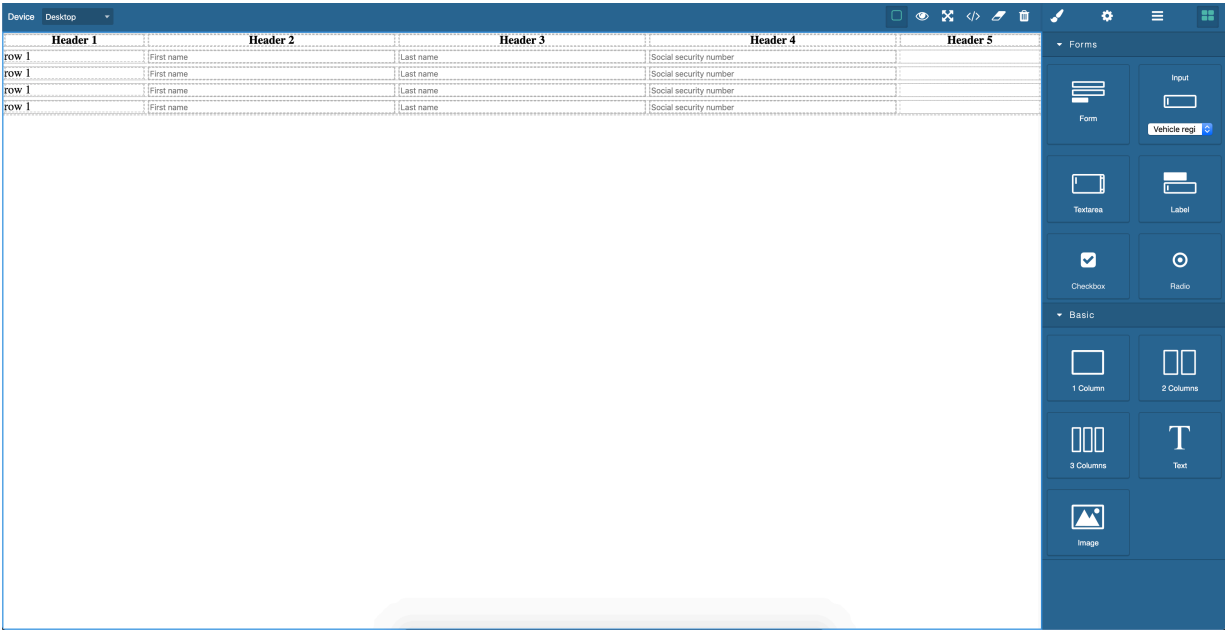


Figure 4-9 MVP design space with active form

4.5 Applying learning outcome to ideal artefact

The last process iteration takes in the ideal artefact and combines it with the findings of the MVP testing.

4.5.1 Design and development

The theory of tailoring principles where to various degrees covered by the MVP and shown to provide some solution to the problem. As mentioned, the theory of tailoring is not meant to be directly used to support end user development. Therefore, this last process iteration takes on the task of creating a set of principles built on the theory of tailoring and user testing. These principles have end user development in generic software application in mind but is not limited to that context.

This led to the resulting four general design principles:

The first principle comes from the test results and the theory of tailoring. As there is not possible to accommodate all users and all use cases, there needs to be some kind of flexibility in the way the end users' needs are met. Therefore, I propose to define a principle that takes in to account that an "unspecified" amount of needs should be met. These needs should then be limited by letting the users define their own levels of skill and the current context. Then this again limits the options of the design environment. The users testing the MVP where positive in regard to having the ability to do configuration.

Flexibility

Limit design flexibility to the applied context. When limiting the options that is not needed, the user can perform tasks quicker and with greater ease. It can be hard to always know when to set the limit but letting the users do it is generally a good idea.

The second principle comes from the tailoring principle that covers methods. One thing that the testing of the MVP showed is how important it is to have a “friendly” design environment, that encouraged the users to try out the features and learn how the MVP worked.

Environment

Friendly environment that encourages users to try out features without the possibility of “destroying” something. Users generally learn a lot about what the tool has to offer when just using it.

The third principle shows the importance of encouraging the users to use the design environment. One way to do this is to look at the way the users interact with the design environment. During the user testing, one of the developers noted that the drag and drop functionality gave a feel of better control over the design process.

Interactivity

An interactive environment that encourages the users to tailor. Demonstration in this thesis showed that “drag and drop” functionality where something the users liked and felt that gave them control over the design process.

The fourth and last principle emphasizes the importance of giving feedback to the users. This again will let the users feel like they are in control. One of the developers testing the MVP noted that the ability to have a preview of how the final design would look like, where useful to have during the design process.

Feedback

Provide feedback and guide the user, anything from previews to tips. By giving instant feedback, development time can be reduced.

4.5.2 Demonstration

It has been shown that the tailoring principles can be used to develop tools that enable end user development in generic software. The principles created is a direct result of the user testing and have shown to facilitate end user development. The most interesting result of the MVP testing was how well someone with limited experience and knowledge in coding, could be able to develop simple web forms.

5 Discussion

5.1 The use of tailoring principles

As stated previously in the thesis, the tailoring principles are not meant to cover the process of end user development. This is something I was aware of before trying to implement them in the ideal artefact or MVP. I did however discover that by abstracting the principles from the original context, they could be used to create tools that allowed end user development. Now there may be other principles or methods that could work as well, or in collaboration with the tailoring principles. This is something that should be researched further in the future.

5.2 Technical limitations of the thesis

What should be clear by now is that this thesis does not research the technical aspects of creating tailoring tools for generic software. Where technical meaning everything that has to do with what the users does not see, like code. This is because it is currently more important to research what should be done in the future. There needs to be a clear understanding of what needs to be done, and what should be done. By creating these principles, I feel that I don't limit the options for anyone potentially trying to implement such a tailoring tool in the future.

5.3 Future development of tailoring tools

5.3.1 Pre configuration of design environment

One element of the design environment not being implemented in this MVP, that should be given focus in future development, is the option to automatically or manually configure the design environment's menu options based on the user's needs. This would be helpful for two main reasons.

Firstly, if the user only requires simple HTML forms with only input boxes and text labels, an environment with a lot more options would quickly become bewildering. This again would lead to more time spent finding the right elements, and then leading to longer development time.

Second, the possibility of configuration/customization would mean that it is possible to limit the design options to a set standard. This could e.g. be a style guide with pre-

defined colors and styles, that would eliminate the possibility of having colors with poor contrast.

5.3.2 Handling more complex tasks

The MVP showed a clear limitation in that it was not able to support any form of JavaScript out of the box. This limits the MVP to less complex tasks. In the future, having the ability to do JavaScript in the design environment would greatly increase the use cases it can handle. There is of course important to keep in mind that the design environment can't just become another text editor for JavaScript as this would exclude many users whom does not have experience with coding. Therefore, one solution to this could be to go in the direction of more traditional tailoring mentioned in the theory of tailoring, namely the use of ready-made components. One such solution could be the low-level JavaScript components that can be used to do visual programming.

6 Conclusion

I have in this thesis researched the theory of tailoring in practice, and how the theory of tailoring can be used to meet the needs of end users in generic software. I have done this through review of the original problem, the related theory and my own reflections. I have used these findings to create an MVP. The MVP where tested on relevant users who gave valuable feedback through interviews. The result of these interviews where then used, combined with the tailoring principles, to create four new principles related to end user development in generic software. The ability to do end user development will help end users make use of generic software that fits their needs.

7 Bibliography

- Allan MacLean, K. C. (1990). User-Tailorable Systems: Pressing the Issues with Buttons . *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (p. 175). ACM Digital Library: ACM.
- Arseniev, A. (2019). *Github*. Retrieved from GrapesJs: <https://github.com/artf/grapesjs/blob/dev/package.json>
- Arseniev, A. (2019). *Grapesjs*. Retrieved from Docs: <https://grapesjs.com/docs/>
- Arseniev, A. (2019). *GrapesJs*. Retrieved from <https://grapesjs.com/>
- Bekk Consulting AS. (2017). *Bekk teknologiradar*. Retrieved from Bekk teknologiradar 2017: <https://radar.bekk.no/tech2017/frontend-og-mobil/backbone>
- CKSource. (2019). *CKEditor Ecosystem*. Retrieved from <https://ckeditor.com/>
- DHIS 2 Documentation team. (2019). *DHIS2 documentation*. Retrieved from https://docs.dhis2.org/master/en/user/html/dhis2_user_manual_en_full.html
- DHIS2 software. (2019). *Overall architecture*. Retrieved from [dhis2.org: https://docs.dhis2.org/2.28/en/developer/html/technical_architecture.html](https://docs.dhis2.org/2.28/en/developer/html/technical_architecture.html)
- Dvořák O., P. R. (2017). Tackling the Flexibility-Usability Trade-off in Component-Based Software Development. In C. A. Rocha Á., *Recent Advances in Information Systems and Technologies* (pp. 861-871).
- Giaccardi, G. F. (2004). Meta-Design: A Framework for the Future of End-User Development. In H. P. Lieberman, *End User Development - Empowering People to Flexibly Employ Advanced Information and Communication Technology*. The Netherlands: Kluwer Academic Publicshers.
- Git. (2019). *Git*. Retrieved from <https://git-scm.com/>
- Givan. (2019). *Github*. Retrieved from <https://github.com/givanz/VvwebJs>
- HISP. (2019). *DHIS2*. Retrieved from <https://www.dhis2.org/>
- HISP. (2019). *DHIS2 Github*. Retrieved from <https://github.com/dhis2/dhis2-core>
- Ken Peffers, T. T. (2014, 12 08). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*.
- Leraand, D. (2019, 08 29). *Store Norske Leksikon*. Retrieved from Utviklingsland: <https://snl.no/utviklingsland>
- Li, M. &. (2019). *Making Usable Generic Software. A Matter of Global or Local Design?* 10.13140/RG.2.2.31514.49603.
- Li, M. (2019, May). *researchgate.net*. Retrieved from Making Usable Generic Software - The Plattform Appliances Approach: www.researchgate.net/publication/333221116

- Malt, U. (2015, September 4). *Store Norske Leksikon*. Retrieved from Kvalitativ: <https://snl.no/kvalitativ>
- Matt Germonprez, D. H. (2007, 06). A Theory of Tailorable Technology Design. *Journal of the Association for Information Systems*, pp. 352-367.
- Mavuru, I. (2018, June 22). *KPI Partners Blog*. Retrieved from Traditional vs. Agile Software Development Methodologies: <http://www.kpipartners.com/blog/traditional-vs-agile-software-development-methodologies>
- Microsoft. (2019). *Visual Studio Code*. Retrieved from <https://code.visualstudio.com/>
- microweber. (2019). *Github*. Retrieved from microweber: <https://github.com/microweber/microweber>
- Mozilla. (2019, 10 10). *JavaScript*. Retrieved from <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- Netlify. (2019). *Netlify*. Retrieved from www.netlify.com
- Nielsen, J. (1994, 04 24). 10 Usability Heuristics for User Interface Design.
- Nielsen, J. (2000, 03 18). *Nielsen Norman Group logo Nielsen Norman Group*. Retrieved from Why You Only Need to Test with 5 Users: <https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/>
- Nielsen, J. (2010, 07 25). *Nielsen Norman Group*. Retrieved from Interviewing Users: <https://www.nngroup.com/articles/interviewing-users/>
- Reindal, H. O. (2018, 02 19). *Bouvet*. Retrieved from 5 tips for å lykkes med brukertesting: <https://www.bouvet.no/bouvet-deler/5-tips-for-a-lykkes-med-brukertesting>
- Reis, E. (2011). Test. In E. Reis, *The Lean Startup* (pp. 92-97). United States: Currency.
- Revision World Networks . (2019). *Revisionworld*. Retrieved from Application Software: <https://revisionworld.com/gcse-revision/ict/software/application-software>
- Ries, E. (2011). *The Lean Startup*. United States: Currency.
- Squarespace. (2019). *Squarespace*. Retrieved from <https://www.squarespace.com/>
- Stack Overflow. (2019). *Stack Overflow Developer Survey 2019*. <https://insights.stackoverflow.com/survey/2019#technology>: Stack Overflow.
- Tiwana, A. (2014). Platform Ecosystems. In A. Tiwana, *Platform Ecosystems* (p. 84). Morgan Kaufmann; 1 edition (1709).
- Trikha, R. (2016, 08 25). <https://blog.hackerrank.com/>. Retrieved from <https://blog.hackerrank.com/which-country-would-win-in-the-programming->

olympics/: <https://blog.hackerrank.com/which-country-would-win-in-the-programming-olympics/>

UiO. (2019). *Health Information Systems Programme (HISP) - Department of Informatics*. Retrieved from <https://www.mn.uio.no/ifi/english/research/networks/hisp/>

wix.com. (2019). *Wix*. Retrieved from <https://no.wix.com/>

8 Appendix

8.1 Interview and test guide

The testing of the prototype will be done in three parts. First some questions to establish who the test user is and their level of experience. Then, the user will be asked to re-create a “paper” form in the drag and drop prototype. If the user has experience with coding, the user will also be asked to create the form using standard methods of “coding”. Lastly, the user will be asked questions about the prototype, and give a comparison between the standard way of coding and using the prototype. The last part is just for the persons with experience. Some of the questions will change depending upon who is asked. Marked (Imp) is the persons with experience similar to that of an implementor. Same goes for developers (Dev). Questions for everyone will be marked with (All).

Questions before testing

How many years of coding experience do you have? School, work e.g. (All)

Establishes the level of skill and experience.

Do you feel competent when it comes to IT solutions, e.g. Microsoft Excel?

(Imp)

Depending on last question, this question may be asked. Someone with no experience with IT solutions will not have what it takes to make use of the prototype.

How would you go about creating a simple HTML form from scratch? (Dev)

Establishes the current situation and gives ground for testing.

Are you familiar with the DHIS2 system? (All)

Someone with knowledge of the DHIS2 system may have different insights than someone who have never used the system.

How have you used the DHIS2 system? (All)

Depending on last question.

Testing

Give a short introduction to the drag and drop prototype before testing it.

If the user has experience and/or skills with coding, ask them to create the form using a text editor and “typing”.

Then ask both the implementor and developer to create the form in the drag and drop prototype.

Not sure if relevant to the project, but the developers could be timed when creating the forms. Note that additional time can be saved as the ideal artefact would be able to add things like JS code for functionality and a manifest file.

Questions after testing

What method of form creating worked the best for you? (Dev)

Trying to do a comparison between the current situation and the new prototype.

**(1) How do you feel about using the prototype system to “program” a web app?
(Imp)**

Depending on the level of skill and experience. To what degree does this prototype offer something helpful.

(1) Do you feel the prototype is flexible enough to support application development? (Dev)

(5) Are there any specific tasks that could be solved when using this prototype? (All)

(3) Did you find the drag and drop familiar and usable? (All)

(6) How would you describe the prototype related to a “real world” tool? (All)

(4) Did you find the components usable? (All)

(2) Do you feel that the prototype provided enough feedback to support the development? (All)

What worked for you when trying to create an application? (All)

Things that may directly provide a solution to the problem(s).

Is there anything that could be done better, why? (All)

Constructive criticism. What can be improved in later iterations or ideal artefact.

Description of case

The police in city X is using DHIS2 to gather and manage all their information. A large portion of the gathered information comes from traffic stops.

“A traffic stop, commonly called being pulled over, is a temporary detention of a driver of a vehicle by police to investigate a possible crime or minor violation of law.” (https://en.wikipedia.org/wiki/Traffic_stop, 2019)

Even though there is not found anything unusual with the vehicle, the police still have to register that the vehicle has been stopped and checked.

For a long time, this was done using a mix of paper forms and word documents. After DHIS2 was introduced the police department tried to recreate the paper forms they used, in DHIS2's data entry app. This solution works, but it has some issues. The main problem being that it takes a lot of time to open the app before data can be entered. This cause some police officers to use the old paper forms and after the days is over, transfer this data to the data entry. Not and ideal solution.

The task for you, developer/implementor in city X, is to create a new application that is better suited for this purpose. The requirements are listed below.

The new application needs to have support for the following entry fields:

- First name of driver
- Last name of driver
- Social security number of the driver
- Vehicle registration number
- Indication of whether anything is irregular
- Comment on possible irregularities

It is also important to not create the form to different from the original paper form, as the police department needs to take this in use right away, with little to no need for extra training.

To better help you understand what is required of the form, the old paper form is attached below:

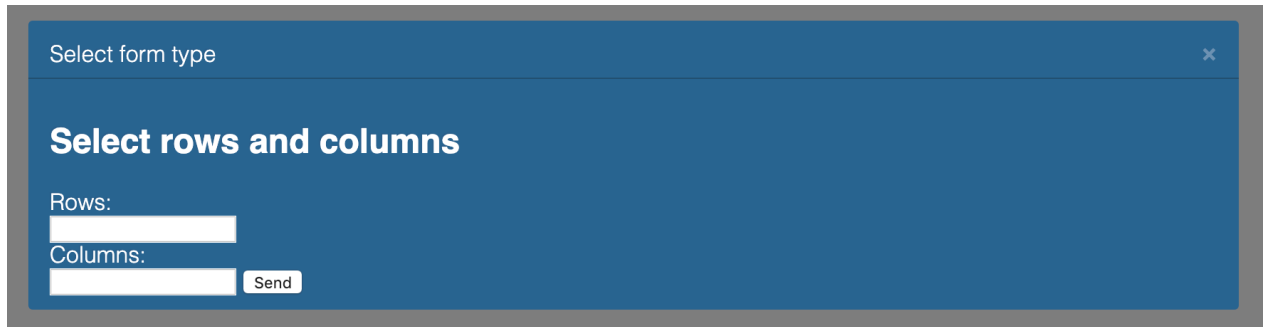
| | |
|---------------------------------------|--|
| First name: | |
| Last name: | |
| Social security number: | |
| Vehicle registration number: | |
| Irregularities found during the stop: | <input type="checkbox"/> Yes <input type="checkbox"/> No |

| | |
|-------------------------------------|--|
| Comment on possible irregularities: | |
|-------------------------------------|--|

DHIS2 drag and drop prototype short form guide

Start by going to <https://dhis2-dragdrop.netlify.com/>

To create a form first enter the number of rows and columns. This will set the standard for the form block.



Select form type

Select rows and columns

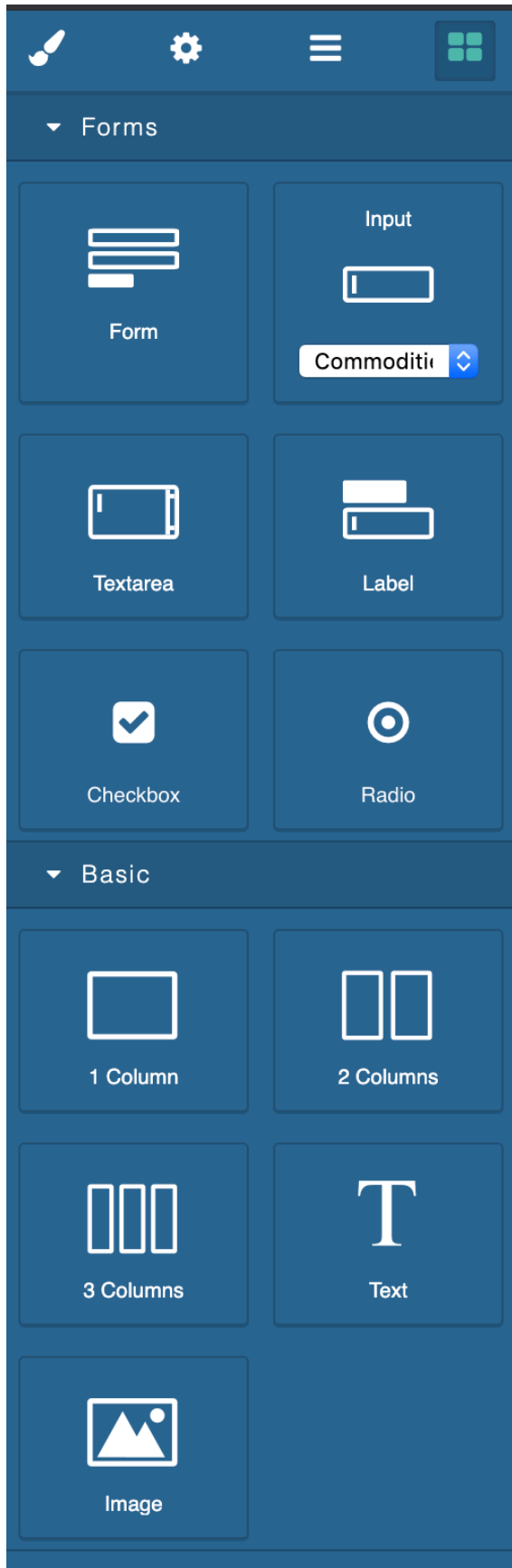
Rows:

Columns:
 Send

Click “send” when done.

Next navigate to the block’s menu found in the top right.





The form in the block's menu will have the dimensions previously entered. To create a form, simply drag and drop the form block onto the white canvas.

The same method is used for adding the other block's in the block menu, e.g. an input field to the form.

Before adding an input field, the input type must be selected. This can be done by selecting the type in the dropdown menu in the input block.



The input block can be added to the form after the input value is selected. A placeholder should be visible with the selected input value after adding the input to the form.

Click the “eye” button when the form is done to show how it looks when opened in a browser.



8.2 Interview summary

Findings from MVP testing

Did the same test on four individuals. One individual who fit the implementor profile and two individuals who fit the developer profile.

The “developers” are identified as DevA and DevB.

The “implementor” are identified as ImpA.

Questions asked before the testing

How many years of coding experience do you have? School, work e.g.

Both of the developers had around 3 years of coding experience, that mainly consist of schoolwork, with some relevant part time work.

The implementor had about 5 years of schooling related to informatics, but not as much practical coding involved. Did not have any professional experience with coding.

Do you feel competent when it comes to IT solutions, e.g. Microsoft Excel?

This question where not needed in this testing.

How would you go about creating a simple HTML form from scratch?

Both the developers said that it depends upon what the requirement was. Both of them would use a standard library like bootstrap and a simple text editor like visual studio code. The process of creating the form would also usually involve some kind of “googling”.

Are you familiar with the DHIS2 system?

All of the participants where familiar with the DHIS2 system. All of the participants have worked with implementation and or development of features for the system.

How have you used the DHIS2 system?

One of the developers had mainly just worked with tracker. The other developer had also worked with tracker, but also create some custom apps and configured meta data.

The implementor has worked with both aggregate data and tracker.

Observational notes on the testing

Both the developers where asked to complete the task using traditional tools. They both chose to use visual studio code for this.

DevA started with searching up how a HTML table structure looked like but chose not to do a “copy-paste”, rather just write everything “by hand”.

DevB used some old code from a previous project and copied that instead. DevB did also choose to go for a “div” structure instead of a table structure.

Both of the developers chose to use radio buttons on the Yes/No input.

DevB did also more time on styling than DevA, even though DevB used a bootstrap library. DevB did also choose to use some time on setting the right “types” in the input fields. DevB did also search the internet on how to use radio buttons. DevB did also use a “
” tag to create a line shift, this is not recommended as it can affect the responsiveness of the form.

DevA used 06:12 to create a form.

DevB used 11:24 to create a form.

Both the developers and the implementor where given a short introduction to the prototype before doing the case. When doing this they were both allowed to play around with tool, this gave them a chance to learn the tool. This seemed to be working. Both the developers naturally used less time to “write” code, and more time on “creating”. None of the developers needed to look up how to do thing on the internet. DevB did not need to search the internet on how to use radio buttons.

DevA used 05:27 to create a form.

DevB used 05:22 to create a form.

The implementor did not have any troubles with using the prototype after some introduction and doing some play around with it. The implementor was not timed as there was not anything to compare to. The main point was to see if it was possible for an implementor to create a form.

What method of form creating worked the best for you?

DevA did like the traditional way better, but this was mainly because DevA did not have as much time as DevB playing around with the prototype. DevB liked better the prototype for this kind of task as it was much faster.

How do you feel about using the prototype system to “program” a web app?

The implementor was asked this and said that this is something that could be useful, even for implementors who does not normally create forms. It was also a great way to link the data elements to the form. It was a good option when it comes to designing forms.

Do you feel the prototype is flexible enough to support application development?

The developers where asked this. They feel that when it comes to form creation it would be flexible enough, but more complex application development would be harder. DevA noted that a more specialized design could be hard. DevB also noted that it gave a feeling of more control when it came to development.

Are there any specific tasks that could be solved when using this prototype?

Both of the developers said that it would be most suitable to create forms.

DevB noted that it would be useful as a template building tool. Then downloading the template and adding more functionality.

The implementor also said that form building is a main thing with the prototype, and that it was important to add logos.

Did you find the drag and drop familiar and usable?

DevA did find the drag and drop usable as DevA has used similar services like WIX etc. DevA did also relate the drag and drop to photoshop

DevB could relate the form creation to Word. As a developer the icons where familiar. Some of the menu items could be more advanced for e.g. an implementor.

The implementor did also recognize the icons and did not that the text on the components could help to understand the components.

How would you describe the prototype related to a “real world” tool?

This was a harder question to answer, maybe there is a misinterpretation from my side.

One thing DevB noted was that the menu icons did resemble real items. Like the eye showed a preview. The implementor did also note that the menu items were familiar in relation to a real-world tool.

DevB did also somewhat explain the drag and drop as creating something with your hands in real world.

Did you find the components usable?

All the test users feel that the components were usable. DevA noted that a “button” component was missing and DevB noted again that it would be harder to create something more complex.

Do you feel that the prototype provided enough feedback to support the development?

DevA did feel like it, as it did show a preview and the ability to view code.

DevB did like the icons and tooltips. The icons were good as they were familiar, and the developer did not need to read the text. DevB liked that the drag and drop showed a “ghost” icon when being dragged.

The implementor noted that the prototype looked friendly and not intimidating to play around with.

What worked for you when trying to create an application?

Did not ask.

Is there anything that could be done better, why?

DevA noted that there should be an option to edit code directly in the drag and drop system. DevA also noted that there should be a tutorial on how to use the tool.

DevB wanted to resize the form elements using the mouse, this is not possible at the moment. This was also noted by the implementor. The implementor would also like to have a guide.

Other findings.

The implementor wanted to have some possibility to create simple scripts.

The implementor did also note that some of the forms created today in DHIS2 data entry are being “hacked” together. This means that users are adding their own JavaScript, html and CSS directly into the editor. Custom functionality is added this way like dynamic adding of input fields and the freezing of the top header. This creates a lot of problems when It comes to the maintenance of these forms.

DevA did like the configuration at the start.

DevB noted that the ability to choose different views like mobile and desktop where good.