Comparison of Implementation Cost and User Effort in Different Tailoring Interfaces

Yu Tang

Department of
Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden
This thesis is submitted to the Department of Software Engineering and Computer Science at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author:
Yu Tang
Address:
Videvagen 7
472 38 Ronneby
E-mail: yutang718@hotmail.com

Main advisor: Lars Lundberg
Secondary advisor: Jeanette Eriksson
Department of Software Engineering and Computer Science

Department of
Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/ipd
Phone : +46 457 38 50 00
Fax : +46 457 271 25
ABSTRACT

Software system usually faces changing and diverse requirements during its use. Traditional solution by redesigning and rebuilding of the system normally costs too much and is hard to maintain. An applicable solution could be employing tailorable system which allows for user customization in run-time.

The contribution of this work is two-fold. On one hand it investigated the state of the art technology of tailorable system. On the other hand it tried to find out how to facilitate user tailoring by providing appropriate tailoring interface. A tailorable prototype with different type of tailoring interfaces was implemented acting as the experiment environment. These interfaces were script file type, wizard type and 2D interactive environment type respectively. In the experiment, three categories of user – customer, tester and programmer were identified. User effort in the experiment was evaluated to identify the best tailoring interface for the user. Results from the experiment show that there is no best interface for all the users. Design of tailoring interface should depend on the skill level of the user.

**Keywords:** Tailoring; Tailorable; Tailorability; User Interface; User Effort.
# CONTENTS

ABSTRACT ..................................................................................................................................... I

CONTENTS ................................................................................................................................... II

1 INTRODUCTION .................................................................................................................. 1  
1.1 GENERAL INFORMATION ABOUT TAILORABLE SYSTEM....................................................... 1  
1.2 MOTIVATION OF THE TOPIC................................................................................................ 1  
1.3 OBJECTIVE AND GOAL ................................................................. 2

2 DESCRIPTION OF METHOD .................................................................................................. 4
  2.1 DOMAIN UNDERSTANDING ................................................................................................ 4
  2.2 LITERATURE SURVEY ........................................................................................................ 5
      2.2.1 Definition ...................................................................................................................... 5
      2.2.2 Different Level of Tailoring ......................................................................................... 6
      2.2.3 Tailorable System Architecture .................................................................................. 7
      2.2.4 Design Principles ....................................................................................................... 8
      2.2.5 Designing Tailoring Interfaces .................................................................................. 9
  2.3 EXPERIMENT ............................................................................................................... 10

3 RESULT .................................................................................................................................... 11
  3.1 INTERVIEW RESULT ........................................................................................................ 11
  3.2 RESULT FROM LITERATURE SURVEY ........................................................................... 11
  3.3 MPpro PROTOTYPE ........................................................................................................ 11
      3.3.1 Introduction .................................................................................................................. 11
      3.3.2 Component Set for MPpro ...................................................................................... 11
      3.3.3 Presentation of Compositions .................................................................................... 12
      3.3.4 Tailoring Interfaces ................................................................................................. 13
  3.4 IMPLEMENTATION COST .................................................................................................. 15
  3.5 EXPERIMENT RESULT .................................................................................................... 15

4 DISCUSSION ....................................................................................................................... 17

5 CONCLUSION AND FUTURE WORK .................................................................................. 19

REFERENCE ............................................................................................................................... 20

APPENDIX A: ASSIGNMENT AND QUESTIONS ...................................................................... 21

APPENDIX B: A SAMPLE OF CDR XML FILE ....................................................................... 22
1 INTRODUCTION

1.1 General Information about Tailorable System

It is common that software system will face new requirements during its use. One simple and direct solution is to let programmer to modify and rebuild it. However there are two main problems with this solution. First of all, software developers have to be involved when new requirement occurs, which is troublesome or sometimes even impossible especially for some commercial software. Secondly, the software developer will also be frustrated when facing constantly modified software, which is hard to maintain and error-prone.

A more desirable solution is to employ tailorable system, which can be tailored by end-user to adapt to diverse and changing requirement in run-time. It is characterized by a unique tailoring process, through which users can change certain features of the system. These features can be everything that the user will experience during manipulation of the system. In some cases they are user interfaces, for example when one changes the background of desktop of Microsoft Windows XP to one of his favorite pictures. It could be functional activities, for example the macro mechanism used in Microsoft Word is used by end-users to facilitate their work, which provides customized functionality rather than only visual appearances. The degree that the system can be tailored is also varying from one system to another. Some system provides only limited tailoring functionality, while some others are built to be almost totally tailorable, like Apple’s HyperCard[15] and EMACS[14].

Tailoring interface bridges the distance between the user and the tailorable system. From the system’s perspective, tailoring interface exposes the tailoring functionality of the system to the user. From user’s point of view, tailoring interface guides the user to explore the possibility to customize the system. Since tailoring is not only an activity to use the system, but also a way to change the behavior of system, so the user should have certain amount of skills when they are carrying out certain tailoring functionalities. Normally experienced user with better domain knowledge and computing skills can employ deeper level of tailoring functionality, while novice user may only be able to use some basic tailoring functionality. Research result shows that many users are even inhibited from using certain tailorable system just because they don’t have enough knowledge to use it[15].

1.2 Motivation of the Topic

Vodafone is a leading mobile service provider. To follow up the rapid growth of telecommunication technology, it refreshes itself by incorporating new mobile devices which provides new services from device suppliers. Those devices can generate certain call data which contain information of its corresponding service. This kind of information is stored in the form of CDR (Charging Data Record), for a multimedia message service, the information could be caller’s telephone number, message size and message type and so on.

In Vodafone, a billing system plays the role of charging all the services by taking in CDR that service node have generated and producing billing results according to certain rules. Figure 1 below shows an overview of the billing configuration.
To verify the correctness of the billing system, what test engineers always do is to generate a number of CDR with specific content, and forward these CDR to the billing system. They do not expect these CDR generated from the real systems since they are only for test use. So in the test environment they use a test tool to generate CDR according to certain service specifications and pass these CDR to billing system to verify the produced results. This test tool takes the place of both Services and Mobile service nodes of the real system (see figure 1). In the past several years, the test tool worked well in the situation with limited services. The potential problem with the test tool is that it is relatively static, which means that its interface can not be changed, unless the whole software system was redesigned and rebuilt which takes too much time and effort. During recent years new services are surging out, service specifications are becoming more diverse and complex. The interface and functionality of the test tool has to be changed constantly to meet the need of new service, Vodafone cannot afford the cost of redesigning and rebuilding any longer.

The topic of this paper is motivated from the problem of the test tool. Detail information about the test tool will be presented in section 2.1.

1.3 Objective and Goal

The goal of this report is to find a solution to the problem of the test tool from Vodafone. Possible solution is to build a system that can be tailored by the user with minimum effort during run-time. This solution raised two technical questions. One is how to develop a tailorable system, the other is how to facilitate user to carry out tailoring functionality.

To find out way to develop a tailorable system, a literature survey was carried out to investigate the state of the art technology of tailorable system. Using the knowledge from literature survey, a tailorable prototype with different type of tailoring interfaces was implemented based on the requirements of the test tool. To find out how to facilitate user tailoring, an experiment was conducted to see which type of tailoring interface facilitate user better.

This report was outlined as follows. In section 2.1, the problem of the test tool was discussed in detail through interviews with test engineers from Vodafone. In section
2.2, a literature study was carried out to investigate existing technology that can be used to develop tailorable systems. The information about the experiment was discussed in section 2.3. In section 3, results from interview, literature study and experiment were presented. Finally this report ends up with a discussion and conclusion.
2 DESCRIPTION OF METHOD

2.1 Domain Understanding

The problem comes from a test tool at Vodafone. The test tool is a web-based application. Since its main function is to generate CDR, the interface only consists of a number of CDR generation pages. Each CDR generation page is corresponding to a CDR specification provided by the device provider. A CDR specification has a detail description of how a CDR looks like, according to which CDR generation pages are created by web designer. The following figure shows a sample of CDR generation page.

![Figure 2 Interface of a CDR Generation Page (in Swedish).](image)

There are lots of information items in a page (see the square area in figure 2). Each item consists of a title which shows the meaning of this item and a text field that let you input information or a CombBox list where you can select from a set of predefined options (see Figure 2). A CDR generated by CDR generation interface is the sequential combination of string value of item fields together with a tag and some other type of information. The following string shows a sample of CDR:

\[(2)cn=DD/(4)sr=web1/(17)tap=internationell(1)/(2)ti=00\ldots\]

This string sample is composed of many small string unit, in this case, these small string unit are

\[(2)cn=DD/ (4)sr=web1/ (17)tap=internationell(1)/ \ldots\]

Take the first string unit for example, the string value “DD” is the value of the first item in this page (see Figure 2). “cn” is the tag or short name for this item, which is also a substitution of the title, since the title is usually too long. “(2)” means that the length of the string value is 2 bytes. Moreover every string unit ends up with a “/”.\]
This kind of CDR generation page offered users limited flexibility to generate different CDR string.

1. Change the content of a Text Field item.
2. Select another predefined value from a ComboBox item.

However newly surging services introduced a large number of new requirements, which cannot be handled by the test tool. These requirements can be classified into the following two categories.

2. Modifying an existing generation page.
   a) Add a new item.
   b) Delete an item.
   c) Add new content into a ComboBox item.
   d) Modify the title.

Basically all these requirements are related to changes to the interface. Some changes seem to be so trivial and occur so frequently that it is not practical to redesign the web page at each time, which cost too much and takes a lot of time. Testers expect to be able to customize the interface by themselves. Actually changing the interface to fit to new requirement is becoming part of testers’ work. Another important un-functional requirement is that they also expect an easy way to carry out these operations, since most of them do not have too much computing skills.

Possible solutions to the problem of test tool could be various. Only the approach of employing tailorable system is discussed in this report. This approach raised two technical questions:

1. How to build a tailorable system that can meet the requirement of the test tool?
2. How to facilitate user to carry out tailoring functionality?

To answer the first question, a literature survey will be conducted in the next section searching for technique and guideline to build a tailorable system. Regarding the second question, the issue of designing appropriate tailoring interface will be discussed.

## 2.2 Literature Survey

In this section, different aspects of tailorable system were discussed. The purpose of this literature survey was trying to find out some clue to develop a tailorable prototype based on the new requirements of the test tool from Vodafone.

### 2.2.1 Definition

In literature, there are varying definitions for tailorable system and tailoring. In report [1], tailoring was defined as “the adaptation of generic applications such as word processors; spreadsheets, e-mail systems, and drawing editors to the specific work practices of a user organization”. Whereas this definition focus tailoring activity carried out in generic applications, report [5] offers a simple and general description that “tailoring is the activity of modifying a computer application within the context of its use.” Based on this definition, the following figure shows the relationship between the tailoring and the system.
Tailorable system is a kind of system that provides tailoring functionality. Basically it has the following five characteristics:

1. “Openness: Accessibility of the technology
2. Efficiency of End User Modification: Modifications implemented deliberately
3. Various Levels of Complexity: Support of novice to advanced users.
4. Control/Restrictiveness: The amount of tailoring end users can do
5. Suite of Functions: Used in tailoring. [8]”

The most important feature of tailorable system is its support of two distinct design phases. First is the design of the initial, primary, or default state. Prior to the use of any technology, whether tailorable or not, an initial state is designed. Second is the act of tailoring, or the user defined design of the technology during its use. [8]

To sum up, tailoring is a unique process which is different from both use and development. It takes place after the original implementation and before the use of the system. Some people even call it “end-user development”, since it is the end-user who carries out the tailoring activity.

2.2.2 Different Level of Tailoring

Anders Morch identified three levels of tailoring functions, which are customization, integration and extension [1] from high to low level. Customization is the basic level of tailoring, which means modifying the appearance or editing the attribute value of the presentation object. The second level tailoring – integration allows users to add new functionality to the system by configure predefined function module or component. The third level of tailoring is extension which can be used to improve the functionality through adding new code. He also acclaimed that it is not a common feature for systems to support tailoring by extension, since it require special support mechanism. A tailorable system may provide one, two or even all levels of tailoring at the same time. Normally the system which provides low level of tailoring might also support the higher level of tailoring, since the higher level of tailoring is relatively easier to implement from technical point of view.

There is another definition for different level of tailoring from GROUP97[5], which are the following four levels:

1. changing parameters/switches,
2. changing the composition of an application,
3. extending the application with high-level computational mechanisms, such as APIs
4. changing the source code of the application

Basically this definition is aligned with Anders Morch’s definition, however it add a new one which seems to be between the second and the third level of Anders Morch’s definition.

Report [7] provides an interesting identification to different type of tailoring, which are (1) content, (2) functionality, (3) presentation/interaction and (4) adaptation based on the quality of Service. It also has the following figure to show relationship between the different types of tailoring. It shows that presentation/interaction type of tailoring is a high level of tailoring which means this type of tailoring is more closer to the user, the quality service type of tailoring is the lowest level of tailoring which means it has a strong tie with the system below it, and the content and functionality type of tailoring is something between them two.

![Figure 4 “Different types of tailoring [7]”](image)

According to the operation that it is supposed to support, the test tool should provide two level of tailoring functions in Anders Morch’s definition. If every item (see Figure 2) in the CDR generation page is viewed as a presentation object, changing the title and content of it is related to change the attributes of it, which is in the first level of tailoring- changing the attributes of presentation object. And if a whole CDR generation page is regarded as a configuration of a set of items, and changes to the configuration of these items is related to the those operations, like adding a new item, deleting an item and even creating a totally new CDR generation page, which are in the second level of tailoring - changing the configuration of components. So for the test tool case, the problem can be handled by two level of tailoring functions.

### 2.2.3 Tailorable System Architecture

In report [9], three design guidelines supporting a stepwise construction of tailororable object-oriented frameworks are proposed, and validated with an open hypermedia system. The guidelines are based on a conceptual model, which is presented as a tailorable relationship within a multi-dimensional design space. Figure 5 shows the design space.

![Figure 5](image)
In the domain of hypermedia, three important varying characteristics are identified in report [9]:

1. “storage (how hypermedia documents are stored)
2. presentation (how they are viewed),
3. And navigation (how they may be linked)”.

The design guidelines are used to cope with three key requirements in open system, which are interoperability, extensibility and distribution. They aim to improve tailorability of this type of system. In essence, these guidelines are some kind of object-oriented analysis method applied on the conceptual model to enable the model to meet the requirements of an open system, therefore may not be applicable to our test tool case. However the idea of three characteristics which are storage, presentation and navigation gave me some clue to develop the architecture of the test tool. The presentation dimension is related to the problem of how the component should be presented. In the storage dimension, the issue is to deal with how these components are to be stored. And to connect the storage and presentation is in the navigation dimension.

Report [2] focused on component-based approach to build a tailorable system. It raised two technical problems:

1. The design of a component model to support the decomposition of a system according to its expected evolution.
2. A runtime and tailoring environment.

It tried to find a component models that support arbitrary decomposition of a system, and studied the existing component models to see if any one meet the need. Unfortunately it turned out that no existing component model can support arbitrary decomposition of a system. Anyhow a FLEXIBEANS component model extended from the JavaBeans component model was implemented, and based on which a runtime tailoring environment called EVOLVE was implemented. EVOLVE aims to provide component-based tailorability for distributed groupware application on the Internet. Tailoring operation supported by EVOLVE platform can be accessed by a set of tailoring API, so different types of tailoring interfaces that are connected to the platform can be employed. In report [2], both 2D and 3D tailoring interface are implemented. I found this report was very useful for our test tool case. Basically I followed the method it described to develop the test tool prototype.

2.2.4 Design Principles

Report [8] proposed some design principles in a general way. As discussed in [2.2.1 Definition], tailoring technology design includes two design phases. One is the design of initial or default state, the other is the user design phases. In this report, Matt Germonprez focuses his research on design principles to build the initial state of tailorable environment through a social-technical approach. He first identified two technical aspects of tailoring technology: Componentization and Use Patterns. With respect to componentization, varying levels of abstraction and flexible couple should be supported especially in a highly tailorable system. Furthermore two kinds of use pattern should be supported: Predefined and emerging use pattern. To support emerging use pattern, “the initial design must then focus on the creation of a design environment within which end users are capable of identifying and implementing modifications”[8]. Matt Germonprez also introduced “aesthetically potent environments and interactive artifacts” to support those design environment based on previous research study of Pask and Gargarian.
The following design principles are pulled out from report [8], since they are related to develop a user design environment and actually creating interactive artifact is usually a built-in process of designing environment.

“Gargarían Design Principles: Design Environment.”

- Environment Design (Design a toolbox)
- Skill Design (Know-How to use Tools/Know-When to use Tools)
- Coherent Design (Explainable know-know/now-when)
- Design Analysis (Method for selecting tools)
- Negotiated Meaning (Neither designer nor interpreter can completely assign meaning)

Gargarían Design Principles: Artifact Design.

- Design by Example (Designing based on personal experience)
- Rationalization (Decisions to promote future actions)
- Exception Principle (Rationalize an inconsistency)
- Self-Referencing (Make a prior solution grounds for a new problem)
- Trial and Error (Acceptable and satisfactory)
- Freedom Through Restrictions (A designer is more free because of restrictions)
- Confidence Building (Reexamination of the design process based on restrictions)
- Judging Fit (How to manage conflicting restrictions)
- Strategic Problem Design (Designing well defined problems)
- Redesign Strategy (Anticipate problems and strategies for dealing with them)
- Working Backwards (First know the solution then see the design)”

Several general recommendations from report [16] suggests that tailoring feature should be “simple, easy and fast to use” and that tailoring feature should become an integral part of the system, switching between tailoring activities and normal task should be as seamless as possible.

2.2.5 Designing Tailoring Interfaces

Designing of tailoring interfaces is an issue in the HCI (Human-Computer-Interaction) field. Earlier research in user interface tailorability has identified different categories of users along with some guidelines for the design of systems [15]. The reason for the categorization of users and tailoring interfaces is that different amount of effort is required to acquire the knowledge and skill to use certain tailoring interfaces. So before starting to design the tailoring interfaces for the prototype, different types of users with different skill levels were identified. In report [15], three different type of user, which are worker, tinkerer and programmer was identified in a tailorable system. Similarly in our case, the users were categorized as customer, tester and programmer. These three kinds of people have different level of skills, act as different roles and therefore prefer different type of interfaces.
• **Programmer**: A person who understands the system inside out. They usually have formal training or experience in programming, and enjoy changing the system in a way that fits current situation or requirement well.

• **Tester**: Customizing the interface is part of his work. He knows how to carry out certain tailoring operation through certain interfaces, even though he does not fully understand it.

• **Customer**: Sometimes he has to make some minor changes to the system interface. He does not know the system well, however he still expect to be able to customize it in a way that he can handle.

### 2.3 Experiment

The goal of the experiment is to find out which type of tailoring interface facilitates users’ work better by means of evaluating user effort in different type of tailoring interfaces. Basically if user effort in certain type of interface is smaller than in other two, we can say this type of interface facilitate user’s work better. Two criteria were used to measure user effort: (1) time and (2) complexity. Time is a common used criterion to measure the efficiency of doing work, while complexity seems to be a quite abstract criterion since it is related to personal opinion. It is used here as a subsidiary criterion to capture users’ overall reflection upon different type of interface.

To make the experiment more practical, the experiment was carried out in Vodafone. Through talking to the manager of test department, I was trying to get more testers to be involved in the experiment. Unfortunately there is only a tester available with the manager himself. To try to more result from the user, several other external people were also asked to join the experiment. Those people have acquired knowledge about the work that a tester usually do and also understand the purpose of the experiment.

Before the experiment, a prototype with three type of tailoring interface was implemented and tested to meet the need of the experiment. I made a presentation to introduce the functionality and feature of the prototype along with the goal and the method of the experiment. I also demonstrated how to use the three types of interfaces. A user manual was also sent out to every participant giving instructions about how to use the prototype. Then the experiment started after making sure everyone understand what they are going to do.

When the experiment began, every participant was asked to finish a number of tailoring assignments using three type of interface, during which time will be tracked down. These assignments are based on the new requirements of the test tool, which makes the experiment more similar to the real work situation (see Appendix A). During and after experiment, interviews were carried out to collect opinions from participants about the complexity of each type of interface and also the overall opinion about which type of interface facilitate their work better. Some results are collected through talking to the testers at spot, while others are gathered through email sent back to me afterwards. Due to the broad perspective nature of some questions, more information was collected through informal interview with the participants.
3 Result

In this section, results from interview, literature survey, implementation of prototype and experiment were summarized.

3.1 Interview Result

Through interview, I got to know some domain knowledge about CDR, CDR generation page and so on, and also figured out the problem of the test tool. A set of new requirements for the new test tool were collected. In general, testers in Vodafone expect to have a new system that can be tailored during run-time and with minimum effort.

3.2 Result from Literature Survey

Lots of related information was found in the literature survey, including varying type of definition for tailoring and tailorable system, different level of tailoring, design principles and guidelines and tailoring interfaces. The knowledge learnt from literature was used to build a prototype – MPpro, which will be discussed in the following section.

3.3 MPpro Prototype

3.3.1 Introduction

MPpro is the name given to the prototype of the test tool. The prototype is written in Java using JavaBeans and XML technology. Basically the implementation is aligned with component-based approach, which is motivated from the research result of report [2] (see Section 2.2.3). The prototype used the JavaBeans component model, which was proved to be suitable for the test tool case during the development process, thanks to some innovative features that JavaBeans provides. First of all, JavaBeans provides properties that can be set and get in order to customize a component instance; second of all, the reflection feature provided by Java technology allows a Java application to introspect itself. Lastly the component interaction supported via an event-listener mechanism is very handy when handling component events, like component selection and value changes and so on. Although JavaBeans does not support arbitrary decomposition of a system, it still works very well in the situation that does not require completely decomposition, like in the test tool case, JavaBeans component model seems to be totally eligible, probably just because there is no complicated relationship within the components in this case.

3.3.2 Component Set for MPpro

The following components were identified and designed to support any possible composition covering all the requirement of the test tool.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComboBox Item</td>
<td>This type of component has a GUI representation that consists of two basic Java components, JLabel and JComboBox. It also exposes four attributes to set or get or both, which are</td>
</tr>
<tr>
<td></td>
<td>1. Title: the text is going to be shown in the left label, and it can be set and get.</td>
</tr>
</tbody>
</table>
2. Index: it is get only and invisible.
3. Name: it is also invisible and only for the purpose of CDR generation. It can be set and get.
4. Content: the text option shown in the right JComboBox. Its type is string, so if you want to have multiple options in the JComboBox, you can connect each option and separate them with a comma. For example “option1, option2, option3” is a sample value for content.

<table>
<thead>
<tr>
<th>Text Field Item</th>
<th>Design Area</th>
</tr>
</thead>
</table>
| This type of component has a GUI representation that also consists of two basic Java components, JLabel and JTextField. It has the same attributes with ComboBox Item. The only difference is that multiple options are not applicable in this item, so you only need to set one complete string to the content attribute. | This component acts as the container for random size of configuration of ComboBox item and Text Field item. It has a blue background, which is similar to a Common Desktop Environment (CDE). It provides the following attributes:
1. ID: it shows the name of this component.
2. Rows: it is the number of rows in the design area.
3. Cols: it is the number of columns in the design area.
4. ItemWidth: it specifies the width for every item in this design area.
5. ItemHeight: it specifies the height for every item in this design area.

In design area component, you can choose how many item in it through setting the value of rows and cols. You also can customize the size of whole page by setting the width and height of each item. |

3.3.3 Presentation of Compositions
To persistently maintain the changes to a CDR generation page, Extensible Makeup Language (XML) technology was used in the prototype. XML is probably the most popular way to format and transfer data in many current systems [10]. In our case, XML file was used to format and store the information of a CDR generation page. In a XML file, you can have your own tag, which are directives for reading this file. A set of customized tags were used by all the XML files of the prototype. Each XML file is corresponding a CDR generation Page, Figure 6 shows a sample of XML file and its corresponding interface.
The content of a XML file is shown on the left side of figure 6, and its corresponding CDR generation page is displayed on the right. A detail complete sample of XML file can be reached in Appendix B. The XML file contains all the required information to construct the interface. For example, the content in the first square of the file describes all the attributes of the first Text Field item in the right interface; the content in the second square is related to the second Text Field item. Besides the information of all the items in a CDR generation page, there is also information of the layout of the interface in a XML file. The content in the third square of the file shows the information how to layout all these items in an interface. In this case, there are one row and two columns of items in this interface, and the width of each item is 200 pt, the height of each item is 25 pt.

3.3.4 Tailoring Interfaces

Three type of tailoring interfaces was implemented in this prototype with respect to different user. These interfaces are the follows:

- Well-formatted Script file
- Interactive Wizards
- Visual Interactive Interface (2D)

As described in section 2.2.4, three different types of user which are programmer, tester and customer were identified. These three type of interface is designed specially for each type of user.

3.3.4.1 Tailoring interface for programmers

Each CDR generation page is essentially represented as a XML file inside the prototype. Thanks to the good support that XML provides for data representation, it was able to separate system kernel implementations, inner representation of user interface and user interface module in the prototype. By using the user interface inner representation, the actual user interface is totally independent from the prototype code kernel.

By modifying the corresponding XML file of a CDR generation page, the user can make changes to the CDR generation page. A text editor is used to edit XML files in the prototype. It offers basic editing functionality, like copy, paste and cut to access...
the file. A programmer with moderate knowledge of XML will have no problem to manipulate these XML files, while it might not be that easy for the other two kinds of users: tester and customer. By using text editor, experienced programmer can carry out different level of tailoring efficiently and effectively.

This type of interface is called script file type in this report. The introduction of this type of interface is actually motivated from the fact that some tailorable system provides this type of interface since they have their own language built in, like Apple’s HyperCard [15]. By using this type of interface, user can program with the built in language which is usually some kind of high level language. In the prototype case, modifying XML file by following certain XML syntax can also give the user an impression of programming.

3.3.4.2 Tailoring interface for testers

Testers usually have a rich knowledge of test environment and CDR generation page, however normally they neither have much programming experience nor formal computing training. For this type of user, the prototype provided a 2D interactive tailoring interface, which consists of a design area and several visual elements. The user can select some of the visual elements and incorporate them into the design area; he can also change the attributes of each element and even the design area. The design concept for this type of interface is “what you see is what you get”, which means that whatever you do in the design area, the changes will completely reflected back to the normal interface.

With 2D interactive interface, testers do not require any programming knowledge. They just need to configure predefined interface elements and customize their attributes in a visual 2D environment. Compared to the interface for programmers, 2D interactive interface doubtlessly lowered skill level for the users; however it might be not as efficient as direct modifying XML files.

The design of this type of interface is motivated from the EVOLVE tailoring environment presented in report [2]. This type of interface provides an innovative approach to let user to access the system through a visually interactive environment. The trickiest part in developing such a 2D environment is how to deal with the layout of interface component. Report [13] also have similar result. This is also the reason why it is relatively harder to implement such type interface than the other two.

3.3.4.3 Tailoring interface for customers

In the test tool case, customers occasionally need to test their own service using the test tool. They normally have neither experience of programming nor rich knowledge of using the test tool. To facilitate this type of user’s work, the prototype offers a wizard type of interface. A wizard usually guide the user through the whole process of a work, for example, the wizard to create a internet connection in Windows system will guide the user to create a internet connection from first step to the last step. It always use the mode of “question and answer” to let the user input required information to finish the whole process.

With this type of interface, users do not even need any experience at all. They just need to follow the wizard, and input some information. Their work is done when the last step is reached. The drawback of this type of interface is that it is the most cumbersome interface among the three. For some experienced user, it is even intolerable to fill in wordy information and switch from one page to another page.
3.4 Implementation Cost

The total time for the implementation of the prototype is about two months of full-time work, including reading technical material, programming and debugging. The fact that the author has two years of Java programming experience may also serve as a reference.

Implementation started with the component-based tailoring environment, including the design and implementation of JavaBeans, tailoring API, XML parsing and so on. After that, I began to design the three type of tailoring interfaces. In the end, some effort went to the interaction between tailoring interfaces. Actually I have spent a lot of time on the component-based environment, which acts as a basis for the three types of interfaces. Table 2 only lists the approximate implementation cost of each type of interface.

Table 2. Implementation time for each type of interface.

<table>
<thead>
<tr>
<th>Tailoring Interfaces</th>
<th>Time(hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Script File Type of Interface</td>
<td>18</td>
</tr>
<tr>
<td>Wizard Type of Interface</td>
<td>42</td>
</tr>
<tr>
<td>2D Interactive Type of Interface</td>
<td>82</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>142</strong></td>
</tr>
</tbody>
</table>

Different type of interface requires different amount of implementation cost. Table 2 shows that 2D interactive type of interface took the longest time to implement, since lot of effort was spent to visualize object and try to offer a user-friendly environment. Script file type required minimum effort because it only provides a text editor allowing user to edit XML file directly. The wizard takes the moderate implementation time.

Different type of interfaces requires different levels of knowledge. When handling the script file type of interface, the user should have knowledge of both the XML itself and how to present a presentation object using XML. While for the wizard, the user does not need to care about XML directly, what he needs to know is just normal computer operation and general knowledge about the CDR generation page. In the 2D interactive type of interface, user will experience a 2 dimensional environment called “what you see is what you get”, they just have to select right component and put it in the right position.

3.5 Experiment Result

During the experiment, most users say that they have encountered many problems when using the prototype to finish those assignments. Most of them are usability problems, for example some user complain that he does not know how to save the changes, which is only one-button click away. This is a quite obvious result since the prototype mainly concerned about the functionality and paid less attention to some usability issue.

Totally six results were collected from participants, and these participants were categorized into customer, tester and programmer according to their work situation and experience. The result consists of two parts. One part is the information about how much time users have spent to finish those assignments. The other part is users’ opinion about complexity of each type of interface. The results were also grouped according to each category of the user. The following table show how much time the customer have spent to finish those assignment using different type of interfaces.

Table 3. Time for customer (Minutes).

<table>
<thead>
<tr>
<th>User</th>
<th>Script file type</th>
<th>Wizard type</th>
<th>2D interactive type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Customer 1

∞  11  ∞  

Notes: ∞ means the time to finish assignment using this type of interface is unpredictable.

The customer is a manager and he does not need to care about too much technical detail in his routine work. He does not have any programming experience either. In this experiment, he was only able to use wizard type of interface to finish all the assignment, which is also the reason why I marked “∞” in the table 3. He told me that after solving some initial problems, he found the wizard type of interface pretty easy to use. However even with the assistance of a user manual and software demonstration, he still did not get the other two type of interface to work. The main reason could be that he found it was difficult to use those two. There is an obvious learning curve for customer to get used to XML file.

Table 4. Time for programmer (Minutes).

<table>
<thead>
<tr>
<th>User</th>
<th>Script file type</th>
<th>Wizard type</th>
<th>2D interactive type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmer 1</td>
<td>3.5</td>
<td>7</td>
<td>4.5</td>
</tr>
<tr>
<td>Programmer 2</td>
<td>4</td>
<td>8</td>
<td>5.5</td>
</tr>
</tbody>
</table>

Two programmers joined in the experiment. One already has good XML knowledge, the other one just know very little about XML. With the help of a sample XML file, both of them felt good about it, and experiment results also proved that. Two programmers finished all the assignments nicely and quickly. They found that directly modifying XML was the fastest way to access a CDR generation page. The second one was the 2D interactive type of interfaces, while it took them the most time on the wizard.

Table 5. Time for tester (Minutes).

<table>
<thead>
<tr>
<th>User</th>
<th>Script file type</th>
<th>Wizard type</th>
<th>2D interactive type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tester 1</td>
<td>5.5</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>Tester 2</td>
<td>4.5</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>Tester 3</td>
<td>6</td>
<td>8.5</td>
<td>7</td>
</tr>
</tbody>
</table>

Three participants were identified to be testers. They are not all real tester from Vodafone. In general, these testers do not have any programming experience, but they know their work well, or more precisely they know the content of a CDR generation page very well. All of them finished the experiment assignment with the help of user manual and XML sample. They all feel that 2D interactive type of interface is interesting and seems to be also handy to finish those assignments. They also found that the wizard is the easiest one to grasp, however it is a little bit cumbersome because they always have to follow the wizard from one step to the last step even for a very simple operation. They have some problem with the XML file, however they managed to use it with the assistance of the sample XML file and user manual.

16


4 DISCUSSION

As described in section 3.4, different type of tailoring interfaces require different amount of implementation cost. Although there is no way to measure how much exactly it cost to implement certain tailoring interface, we could roughly say that it cost much more to implement a 2D interactive type of interface than to implement a script file type of interface, and the implementation of wizard cost moderately. The purpose of tracking down the implementation cost for each type of interface in the prototype is trying to act as a basis for future development of the real system. If evaluation result shows that there are two type of interface that are equally good at facilitating user’s work, probably we should choose the one that requires less implementation cost.

To evaluate user effort, two criteria – time and complexity were used which was discussed in section 2.3.1. The time for different user to finish same tailoring operation was collected from the experiment. Figure 7 below shows the result.

![Average User Operation Time](image)

Figure 7. Average time for different user to finish the same assignment in different interfaces. The arrow in the figure means the value is unpredictable, since customer did not finish the relevant assignment.

As shown in figure 7, for tester and programmer, script file type of interface took them the least time to finish the assignment, wizard type is the most time-consuming way, while 2D interactive type require moderate time. Another fact collected through interview is that wizard type is the most easiest one to use, 2D interactive is the second easiest and script file is relatively the hardest one, which agreed almost by all users. To combine these two finding, we can conclude that there is trade-off between time and complexity in different type of interfaces. The easiest type interface usually takes the user more time to finish their work, while the complicated one will shorten the time. This conclusion is basically aligned with the conclusion about the trade-off between effectiveness and efficiency [13].

Another noticeable fact is that the customer only used wizard type of interface to finish the assignment, which is just because he felt the other two type of interface are too complicated to understand, and thus does not dare to use them. Interview with tester shows that they also encountered some problem to handle with script file type of interfaces, and it takes them a lot of time to learn XML before doing those assignments, while they feel 2D interactive type of interface is relatively easy to understand. Response from programmer shows that there is no problem for them to understand all the interfaces. All these facts show that different user with different knowledge level and expertise response totally different to different type of interfaces.
Different type of interface also requires different level of skill to use, so users require certain amount of effort to acquire the skills necessary to use these tailoring interfaces [15]. Figure 9 characterises the relationship between user skills and tailoring interfaces using a mountain climbing analogy.

![Figure 9. People with different level of skill and tailoring interfaces.](image)

User skills include both domain knowledge and computing skills. A customer normally has the lowest level of skill, so he stands on the lowest part of the mountain. Between customer and tester, there is a steep incline which represents the barrier for customer to gain same level of skills as tester’s to understand how to use 2D interactive type of interface. And another steep cline is between the tester and programmer, which mean that the tester should also improve his skills to be able to use script file type of interface. And interview with the tester shows that they do have to take some time to learn some knowledge before using the script file type of interface. Since programmer has enough skill to handle with all the interfaces, so he stands on the top of the mountain.

To sum up, to facilitate user to carry out tailoring functionality, we can work on from two sides. First from the developer’s side, investing more implementation effort to certain type of tailoring interface could possibly improve the usability of this type of interface, thus to some extent will help to facilitate user’s work. However it is even more important to find an appropriate type of interface for certain type of user with certain level of skills. So before starting to designing tailoring interfaces, different type of user and their skill level should be identified. If there is more than one type of user, designing interface particularly for each of type user seems to be a good practice. It is also important to find the balance between time and complexity in each type of interface. Normally novice user with limited skills more concern about the issue of complexity, while expert user will care more about time since they have enough skill to use the interface, and some other user may care about both time and complexity depending on what kind of operation they will do.

From user’s point of view, to use these tailoring interfaces well, users should be prepared to learn new skills. So it is important to develop a learning atmosphere to encourage users to share their expertise and knowledge with each other inside an organization. It would be also good to develop a tailoring culture which means user should always think about how to improve their working environment by tailoring it.
5 Conclusion and Future Work

This work investigated the technologies to develop a tailorable system based on an industry case from Vodafone. A tailorable prototype was implemented using JavaBeans component model and XML technology, which supports component-based tailorbility introduced in literature [2]. I believe that the prototype presented in this work will solve the problem of the test tool nicely, which actually was proved through interviews with testers, and the techniques and guidelines applied in this prototype probably will also be used in the implementation of other tailorable system. Through an experiment, this work tried to find out how to facilitate user’s work by introducing different type of interface. Result from the experiment suggested that there is no best tailoring interface that can fit into all the situations. It seems that each category of user has his preferred type of interface. For a programmer, his preferred interface is script file type of interface since this type of interface helps him to finish his work most efficiently. For a tester, he also has his preferred type – 2D interactive type. Wizard takes him the most time, while script file type is little bit hard to use. For a customer, the situation is quite easy. Since he can only use one type of interface – wizard, his preferred choice can only be wizard.

For the consideration of future development of the test tool, if the users are mostly programmer, we may consider using script type of interface; if most users are customer, wizard probably will be the best choice; if most users are tester, 2D interactive type could be selected; if the user consists of programmer, tester and customer, more than one type of interface could be used.

Due to limited time allowance for the participant from Vodafone, the experiment carried out in this work was limited to in a small environment with only a few participants, and it definitely can be improved in some way.

In general, this work provided some general suggestions to the development of tailorable system and tailoring interfaces from technical point of view. I would like to follow up this study with research aimed at what type of interface is best suitable for what type of tailoring functionality. This prototype take into account three type of user when designing the tailoring interfaces, while limited information about user working environment was collected before the experiment. Additional studies should be done to investigate how to incorporate the software into the users’ working environment. And only individual user was considered in this prototype, future studies should also consider group-work situation with a number of user involved in.
REFERENCE


APPENDIX A: ASSIGNMENT AND QUESTIONS

Assignments:
1. Create a new interface according to part of a CDR specification (mmsc). For example:

<table>
<thead>
<tr>
<th>ASCII TAG</th>
<th>Field</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mi</td>
<td>MMS Message Id</td>
<td>(4)mi=1140/</td>
</tr>
<tr>
<td>po</td>
<td>SAS port</td>
<td>po=23/</td>
</tr>
<tr>
<td>cg</td>
<td>Originator</td>
<td>(9)cg=178914821/</td>
</tr>
<tr>
<td>ni</td>
<td>NASID</td>
<td>(5)ni=relay/</td>
</tr>
<tr>
<td>ty</td>
<td>Account Status Type</td>
<td>ty=12/</td>
</tr>
<tr>
<td>Id</td>
<td>CDR Transaction ID</td>
<td>(20)=10246077105261017043/</td>
</tr>
</tbody>
</table>

2. Add a CDR item to the interface that you have just created.

<table>
<thead>
<tr>
<th>ASCII TAG</th>
<th>Field</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vs</td>
<td>Vendor Specific</td>
<td>(4)vs= /</td>
</tr>
</tbody>
</table>

3. Delete a CDR item from the interface that you have just created.

<table>
<thead>
<tr>
<th>ASCII TAG</th>
<th>Field</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vs</td>
<td>Vendor Specific</td>
<td>(4)vs= /</td>
</tr>
</tbody>
</table>

4. Add new content into a Combo Box item field.

<table>
<thead>
<tr>
<th>ASCII TAG</th>
<th>Field</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cg</td>
<td>Originator</td>
<td>(9)cg=123456789/</td>
</tr>
</tbody>
</table>

5. Modify the title of one of the item.

<table>
<thead>
<tr>
<th>ASCII TAG</th>
<th>Field</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mi</td>
<td>MMS Message new Id</td>
<td>(4)mi=1140/</td>
</tr>
</tbody>
</table>

6. Modify the content of one of Text Box item field.

<table>
<thead>
<tr>
<th>ASCII TAG</th>
<th>Field</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cg</td>
<td>Originator</td>
<td>(9)cg=789654321/</td>
</tr>
</tbody>
</table>

Question:
1. How much time have you spent to finish these assignments using the following tailoring interfaces respectively? 
   a) Wizard style.  
   b) Scripts file style.  
   c) Visual interactive style.

2. Which style of the following interface is the easiest to learn? And which one is the hardest?  
   a) Wizard style.  
   b) Scripts file style.  
   c) Visual interactive style.  
   And why?:

2. Which style of the following interface do you prefer?  
   a) Wizard style.  
   b) Scripts file style.  
   c) Visual interactive style.  
   And why?:
APPENDIX B: A SAMPLE OF CDR XML FILE.

<?xml version="1.0" encoding="ISO8859_1"?>
<CDRSpecification>
    <SpecName />
    <DataItems>
        <Item INDEX="0" TYPE="TEXT_BOX">
            <ItemName>cn</ItemName>
            <ItemTitle>Kopplas till (CONNECT_NUMBER)</ItemTitle>
            <ItemValue>DD</ItemValue>
        </Item>
        <Item INDEX="1" TYPE="TEXT_BOX">
            <ItemName>so</ItemName>
            <ItemTitle>Kalls (Source)</ItemTitle>
            <ItemValue>web1</ItemValue>
        </Item>
        <Item INDEX="2" TYPE="COMB_BOX">
            <ItemName>ta</ItemName>
            <ItemTitle>TAP Typ av number(TAP_TON)</ItemTitle>
            <ItemValue>Internationell(1),gudan</ItemValue>
        </Item>
        <Item INDEX="3" TYPE="TEXT_BOX">
            <ItemName>tap</ItemName>
            <ItemTitle>Int Acc Code(TAP_IAC)</ItemTitle>
            <ItemValue>00</ItemValue>
        </Item>
    </DataItems>
    <SpecLayout>
        <ItemRows>2</ItemRows>
        <ItemColumns>2</ItemColumns>
        <ItemWidth>300</ItemWidth>
        <ItemHeight>25</ItemHeight>
    </SpecLayout>
</CDRSpecification>