An Evaluation of Advanced User Interface Customization

Clemens Zeidler
University of Auckland
Auckland 1010, New Zealand
clemens@cs.auckland.ac.nz

Christof Lutteroth
University of Auckland
Auckland 1010, New Zealand
christof@cs.auckland.ac.nz

Gerald Weber
University of Auckland
Auckland 1010, New Zealand
gerald@cs.auckland.ac.nz

ABSTRACT

Many graphical user interfaces (GUIs) are customizable. While there are many approaches to user interface customization, most of them are fairly simplistic, e.g., they only allow users to customize menus and toolbars. However, one can think of more advanced customization approaches that allow more complex GUI layout customizations and even functional customization. Functional customization goes deeper into the application logic and makes it possible to change the behavior of an application. In this paper we target two open questions: (1) Are technical users able to use such advanced customization approaches? (2) Would technical users apply such approaches in practice?

We introduce prototypical systems for layout and functional customization of GUIs. In a user study, these systems were evaluated to address the research questions mentioned above. 18 technical users were given customization tasks for three layout and two functional customization scenarios. The participants were observed during the tasks and were asked to complete questionnaires. The results indicate that users are able to use the proposed customization systems, and would also employ them in practice. This suggests that it would be beneficial to include such customization facilities into current and future applications.

Author Keywords

layout customization; functional customization; user study; adaptable user interfaces

ACM Classification Keywords

H.5.2 User Interfaces: Evaluation/methodology

INTRODUCTION

It is an old and well-known challenge that applications may have to be customized to fit different needs [1, 11, 19]. Different users naturally have different requirements, e.g., depending on their role, preferred workflow, expertise, visual acuity and motor skills, and the devices they use. For example, professional users may want to streamline a GUI in order to make work more efficient. Developers cannot anticipate all the users' needs, and changing a GUI after release may not be possible or too expensive. Giving users the ability to do their own customization may solve this problem and lead to a better user experience. An obvious potential issue with customization is that it might disrupt vital functionality of an application. However, this may be addressed by introducing options for explicit limitations of customization, and is not part of the scope of this study.

Previous work mostly addressed customization of menus and toolbars (see related work). However, more advanced customization, covering all aspects of layout and also addressing functionality, are largely unexplored. In this paper, we focus on two advanced customization systems that allow users to change applications at runtime. The first system allows users to do complex layout editing, by rearranging, adding and removing arbitrary widgets. The second system enables users to change the behavior of an application, by adding, removing and rewiring functional components of the application. For both systems, the intended target population is that of technical users, i.e., technically skilled users that spend a considerable time using certain applications. The customization systems were implemented for the Haiku open-source operating system[1].

In particular, we are addressing the following two research questions:

R1 Are technical users able to use customization approaches for layout and functionality?
R2 Would technical users apply customization approaches for layout and functionality in practice?

Answering these questions helps us to understand whether such approaches would be useful.

To target these questions, a user evaluation was conducted, based on the two customization systems for layout and functionality. Both customization systems were demonstrated to 18 technical users, who were asked to perform customization tasks for three layout and two functional customization scenarios. The scenarios were designed to represent real-world customization use cases. The participants were observed during the tasks and given questionnaires at the end.

Most participants stated user interface customization is useful and they would use it if available. In general it was easy for them to customize an application, and they

[1] `haiku-os.org`

OZCHI '13, November 25–29 2013, Adelaide, Australia

easily understood how a problem could be solved using the customization systems. Participants stated that they encountered non-optimal GUI layouts during their daily work, and that they would like to be able to remove widgets, add non-standard functionality or optimize a layout using layout customization. Similar for functional customization, participants encountered applications where they would have liked to alter the behavior. Some participants feared that a customized application would not work properly, which illustrates the importance of a customization system to keep an application in a sound state. Furthermore, participants came up with interesting suggestions, e.g., that the system could be used to create GUI mockups or to enable non-programmers to create new applications.

The next two sections briefly introduce the prototypes used for layout and functional customization. This is followed by a discussion of related work. Finally, the layout and functional customization prototypes are evaluated and the results are discussed.

LAYOUT CUSTOMIZATION

Layout customization gives the user the ability to rearrange, add and remove widgets in a layout. The underlying layout model used for our customization prototype is the constraint-based layout model. This layout model is very powerful and can describe layouts that cannot be described with most other layout models [17]. The customization prototype uses the edit operations of the Auckland Layout Editor (ALE) [27] to edit constraint-based layouts. These edit operations leave a constraint-based layout in a sound state, i.e., the layout stays solvable and non-overlapping. Furthermore, all edit operations keep widgets automatically aligned to each other, which increases the productivity when specifying a new layout or editing an existing layout [28].

To customize a layout, the following drag and drop edit operations can be used (see also [27]). Existing widgets can be moved to an empty area in the layout. Here, a moved widget is aligned to existing widgets. A widget can also be moved between two other widgets or between a widget and a layout border. A widget can be resized by dragging a border of that widget to the border of another widget. Dragging a widget onto another widget swaps the positions of the two widgets. Furthermore, widgets can be removed from the layout by dragging them out of the layout, and added back in again if desired. Similar to the move operation, new widgets can be inserted into an empty area, between two widgets or between a widget and a layout border. All operations can be undone if necessary.

The prototype lets users switch an application into an editing mode at runtime, using a special keyboard shortcut. This is shown in Figure 1 for a media player application. In the editing mode, the GUI becomes editable, using the operations outlined above. A properties window assists in the editing process, letting the user change the layout properties of the widgets. This window also

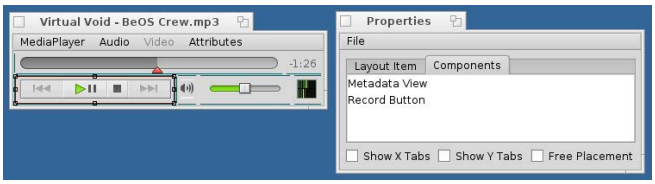

Figure 1. Media Player: Users are able to rearrange widgets, remove widgets and add hidden widgets.

contains a list of widgets that have been removed from the layout, or were not part of the initial layout.

FUNCTIONAL CUSTOMIZATION

Our customization prototype is built on top of a component framework, i.e., the application to be customized is composed of components that are connected to each other [12, 15]. Functional customization lets users change the functionality of an application by adding, removing and rewiring components.

In a component framework, e.g., COM[2] or CORBA[3], components usually have one or more interfaces that define their properties, methods and events. We implemented a simple component framework that is inspired by OpenBinder[4]. The component framework provides a unified way to access properties and to call methods. Events and methods both have parameter signatures, and an event can be connected to a method with the same signature. Whenever an event is emitted from within a component, all connected methods are called. This can be done in a synchronous or an asynchronous manner; by default we use asynchronous events so that concurrent events can be handled in parallel. This keeps the user interface responsive and the user cannot create GUI deadlocks by creating circular connections.

Sometimes the connections between two components can be complex, i.e., if the communication between the components involves a complex protocol. The simple mechanism of event-method connections would be too low-level and tedious in this case. For that reason we introduce the concept of *socket-interface connections* that allows to connect a whole set of functionality to a component. A component A can have multiple sockets and each socket is associated with a certain interface. If a component B implements the interface of a socket, then B can be wired to that socket. The component A can then use functionality of the connected component B through this interface. However, how A uses B's functionality cannot be directly controlled by the user. It is similar to a plug-in that adds functionality to an application.

Similar to layout customization, the user can switch an application into an editing mode using a special keyboard shortcut. In this mode, there is an additional *component layer window* that shows all customizable components of the running application (see Figure 2 from the

[2] `microsoft.com/com/default.mspx`
[3] `omg.org/spec/`
[4] `angryredplanet.com/~hackbod/openbinder`

Message Composer example explained later on). To add a new component, the user can select from a list of available components that can be instantiated. The available components are usually part of the application but could also be provided by third-party libraries. A new component can be integrated into an application by dragging it into the component layer window and then wiring it to existing components. Components can be wired using simple drag and drop operations. By dragging at an event slot and dropping at a compatible method slot (or vice versa), a connection between compatible events and methods can be established. Analogously, a socket can be connected to an interface.

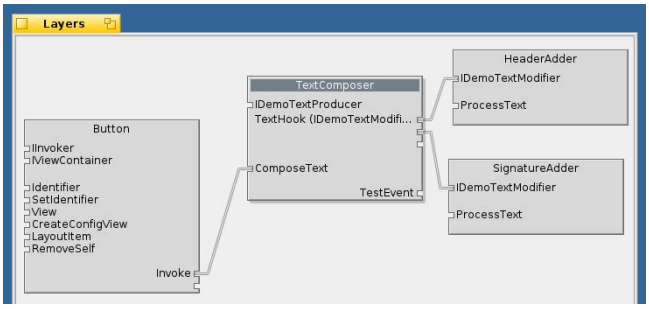

Figure 2. Functional customization: Methods are listed on the bottom-left and events on the bottom-right of a component. Interfaces are shown on the top-left and sockets on the top-right.

RELATED WORK

Various reasons have been identified that can trigger customization [18]. For example, external events may require users to customize their system to adapt to a new workflow. Furthermore, social factors can trigger customization, e.g., a customization may be suggested by friends or colleagues. Other triggers are software updates, the need to fix issues, internal factors such as spare time to tinker with software, and "bloated" applications that have more functionality than needed by the user [22]. Users are more likely to customize if they are made aware of the customization features [1]. However, customizing a system by removing functionality can also lead to a loss of awareness of that functionality, potentially decreasing the performance for new tasks [8].

A previous evaluation explored how users customized a text processor (Word Perfect) [23]. They found that users actually use macros, customize toolbars and to some extent change the appearance of a GUI. It was also shown that the performance of an application can be improved by customizing its GUI [7, 21]. Customization can also lead to a higher sense of control and identity with an application [20].

Early work already enabled users of different skill levels to customize a software system to their needs. The system Buttons tries to make it easier for users to learn customization skills by allowing users of different skill levels to customize a system [19]. Here, application functions can be assigned to buttons, and normal users can rearrange the buttons on the desktop to simplify a certain task. Advanced users ("tinkerers") can edit parameters and attributes of the functions, while experts can use a programming language to further customize or create completely new buttons.

Adaptive approaches automatically optimize GUIs [4, 9, 10, 11]. For example, an adaptive system can add or remove buttons depending on the previous usage of the buttons in a toolbar [6]. However, it is not clear if adaptive GUIs are superior to customizable (adaptable) GUIs. For this reason, mixed-initiative approaches have been proposed that combine the two [2, 3, 7]. Here, the system merely proposes a GUI customization; the user is still in control and may apply or modify it. While previous work mostly looked at how menus or toolbars can be customized, our work enables the user to do more complex layout customization. An example of such customization is the ISATINE system [16], which is able to propose changes to more complex layouts, e.g., users can switch to a minimal representation of a dialog. More comparable to our approach are User Interface Façades, which allow users to compose a new layout by cloning and integrating visual areas from arbitrary windows in a single layout [25].

Mashup tools allow users to combine existing web resources and widgets in a new application [13], using a web component framework such as Google Gadgets[5]. The web resources or widgets do not need to make provisions for interoperability, i.e., the approach is very lightweight. For example, mashups can display information from different web pages in a single view. To make heterogeneous resources work together, adapters are used that achieve compatibility with the mashup tool. In a mashup editor, end-users can manipulate and wire different mashup components. Notable here is that there is often no difference between the design and the runtime phase [13]. This approach is quite similar to our functional customization prototype where users are allowed to modify applications at runtime. Setting up the communication between components is sometimes considered too hard for end-users. However, this can be addressed with approaches for auto-wiring of components [26].

The idea to wire predefined components at runtime is already quite old [14]. Node-based programming tools such as Quartz Composer[6] and LabVIEW[7] are frequently used for domain-specific applications, e.g., in multimedia and engineering. Block-based visual programming tools such as Scratch [24] and StarLogo [5] have been used for teaching programming to children. They allow users to assemble procedural programs from visual instruction blocks that behave similarly to puzzle

[5] developers.google.com/gadgets/docs/spec
[6] developer.apple.com/technologies/mac/graphics-and-animation.html
[7] ni.com/labview

pieces. While easier for novices than textual programming languages, these approaches are still too hard to use for average end-users.

EVALUATION

We investigated the two research questions in a user study. Each participant went through two evaluation parts, one for layout customization and one for functional customization. The participants had never used the customization systems before. Each part consisted of a guided walkthrough as a structured training and tasks participants had to complete, followed by a questionnaire.

The detailed schedule of each part was as follows. In a short introduction, participants were made familiar with layout and functional customization in a number of customization scenarios. These scenarios covered the removal and insertion of widgets and functionality, as well as the optimization and simplification of UI layouts. For each scenario a customizable example application was given. The applications were chosen from a wide range of different domains to convey a representative impression of the many possibilities of UI customization. First, participants were given time to explore the customization features of the application themselves, until they were comfortable using them. Then, they were asked to perform some customization tasks. If a participant got stuck during a task, the participant was helped by the experimenter. Participants were encouraged to ask questions anytime. After finishing all tasks, they were asked to fill in a questionnaire with Likert-scale and open-ended questions.

Methodology: Layout Customization

The first part of the evaluation covered layout customization. In a short introduction, participants got an overview of the use-cases for layout customization covered by the evaluation. These use-cases are that layout customization can be applied to:

a) remove unneeded widgets,
b) add desired widgets that are hidden,
c) optimize a layout, e.g., by simplifying it.

The first example application was a media player with a layout that was completely editable (see Figure 1). Furthermore, there were two widgets available, a record button and an MP3 tag view, which were not part of the initial layout. After letting the participants play with the editing functionality, they were asked to adjust the interface to their personal needs. In a second step, they had to add the hidden widgets, the record button and the MP3 tag view, to a position in the layout that they thought appropriate. Figure 3 shows examples of customized media players. These tasks covered all the use-cases a), b) and c).

The second application was a mail application with an editable mail header interface (Figure 4). The initial layout of this application was not optimal, i.e., it used a lot

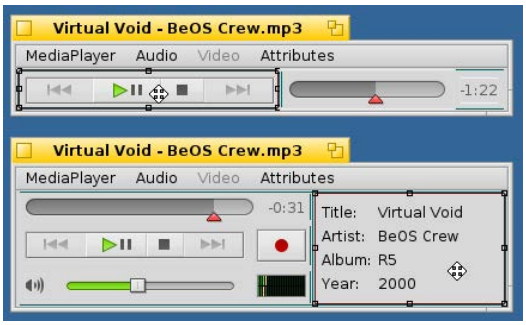

Figure 3. Examples for layout customization of the Media Player. Top: the interface was reduced to only a few widgets. Bottom: a record button and an MP3 tag view were added and the layout was rearranged.

of space and there was a very uncommon text encoding field. The participants were asked to simplify and improve this interface, which covered the use-cases a) and c).

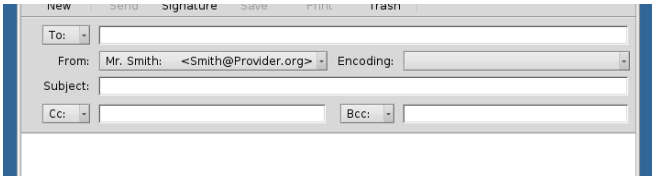

Figure 4. Mail application with customizable mail header interface.

While the first two scenarios used customizable versions of standard applications of the Haiku operating system, the last scenario used a mockup for a typical enterprise application, Employee Manager. This application shows a list of employees and an editable view of their attributes (left of Figure 5). There were two ways to select an employee: by directly selecting an entry of the employee list, or by using the up and down buttons below the list.

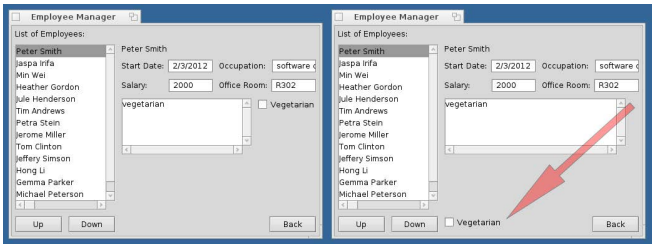

Figure 5. Employee Manager: The left side shows the original layout of the application. On the right side, the vegetarian checkbox has been moved beside the down button. This shortens the mouse path between both widgets and the task can be performed much quicker.

The following user story was presented to the participants: The chef of a company wants to know the exact number of employees who are vegetarian. So far this information was only stored in a general text comment field, and thus it cannot be easily queried from the database. In an upgrade of the Employee Manager,

a new checkbox for vegetarian choice was added, which should now be used for the existing data records. In order to do so, somebody has to go through the whole database manually, read the comment field and tick the vegetarian checkbox if applicable. This task only has to be done once and the GUI is not optimized for this use-case, i.e., the mouse path between the vegetarian checkbox and the employee list is quite long, which has a significant impact on performance. However, the GUI layout can be optimized for the task, e.g., by moving the checkbox next to the employee list or the up and down buttons, making the task much quicker and easier.

The layout was designed so that a clear performance improvement can be achieved by optimizing it. The question is not how much faster the participants perform the task after the customization, but if the participants are able to see the usability problem and customize the layout accordingly, targeting use-case c). The participants were first asked to do the checkbox-ticking task once without customization, and then customize the layout and do the task a second time. The participants were not told how an optimized layout could look like.

Finally, the participants were asked to fill in a questionnaire with the following 5-point Likert-scale questions.

General questions:

Q1 I often use computers in my everyday life.
Q2 I would like to theme a GUI or change its look and feel.
Q3 If there are alternative widgets to control an application, I would like to choose between them.

Questions about the layout customization tasks:

Q4 I understood the Media Player example.
Q5 I understood the Mail example.
Q6 I understood the Employee Manager example.
Q7 It was easy to customize the GUI layouts.
Q8 I think I would be able to do my own GUI layout customizations using the customization system.
Q9 I understood what layout customization is.

Opinions about layout customization:

Q10 I would use layout customization.
Q11 Layout customization is useful.
Q12 I would use layout customization for applications I am using frequently.
Q13 I would use layout customization for applications I am using rarely.
Q14 I see no need for layout customization.

Use-cases for layout customization:

Q15 I have encountered GUI layouts which are not optimal for my purposes.
Q16 I have encountered layouts that did not contain all the functionality I needed.
Q17 I would use layout customization to add non-standard functionality to a layout, i.e., add "hidden" widgets (similar to adding a record button to Media Player).
Q18 Applications have more functionality than I normally use.
Q19 I would use layout customization to remove widgets from a layout that I do not use (similar to the peak view in Media Player or the encoding field in Mail).
Q20 I would use layout customization to optimize certain tasks (similar to the Employee Manager example).

Reusing layouts and expected customization problems:

Q21 I would like to share my customized layouts with other users.
Q22 I see the problem that my customized layouts could not work properly.
Q23 Layout customization is complicated.
Q24 Depending on the task I would like to have different layouts for the same application.

There were also some general open-ended questions:

- Can you give examples where you wanted to change the layout of a GUI?
- Can you give examples where you wanted to add additional widgets to an application?
- Can you give examples where you wanted to change the appearance of a layout, e.g., by rearranging widgets or removing unused widgets?
- Can you give examples where you wanted to change a layout to improve productivity for your purposes?

Methodology: Functional Customization

One of the goals of the functional customization evaluation was to find out whether participants are able to understand how to manipulate components to achieve a certain functionality (see R1). There were two example applications: a simple "Bitmap Viewer" and a more complex "Message Composer".

Bitmap Viewer was a very simple image viewer, comprising an image with a reload button situated below that image. When switching to the editing mode, the component layer window already contained a "bitmap input" component that loads a bitmap image from disk, and a "bitmap view" component to display a bitmap image. The widget palette contained another bitmap view and a "gray-scale filter" component, which could be connected to a bitmap view.

First, the participants were asked to connect the bitmap input with the bitmap view component in order to display the bitmap. Then they had to insert a new bitmap view component into the layout and connect it to the existing bitmap input component, so that the bitmap was shown twice. In the next step, a gray-scale filter had to be added and connected to one of the bitmap views. The resulting application displayed the same bitmap image twice: once as a color image and once as a gray-scale image (Figure 6).

Message Composer was a simple chat application: it had a text view for entering a chat message, a "New Text"

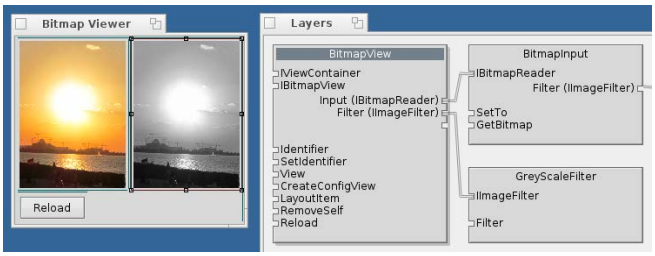

Figure 6. Bitmap Viewer: A second bitmap view was added to the layout in the left window. By connecting it to a gray-scale filter in the component layer window on the right, a gray-scale image was displayed next to the color image.

button for clearing the text view, and a "Send" button for sending the message entered into the text view. When switching into editing mode, the component layer window contained components for the aforementioned widgets, as well as a "text composer" component, which was used to format and embellish the text entered in the text view. After pressing the "Send" button, the entered text got modified by the text composer and was redisplayed in the text view.

The widget palette contained a new "timer view" widget, which implemented timer functionality. This widget was to be used to generate message timestamps, and show the timestamp of the last message. Furthermore, there was a set of "adder" components that could be connected to the text composer component to add text to messages: a header adder to automatically add a message header, a signature adder to add a signature at the end, and a timer adder to add a message timestamp obtained by connecting the adder to the timer view (Figures 7 and 2).

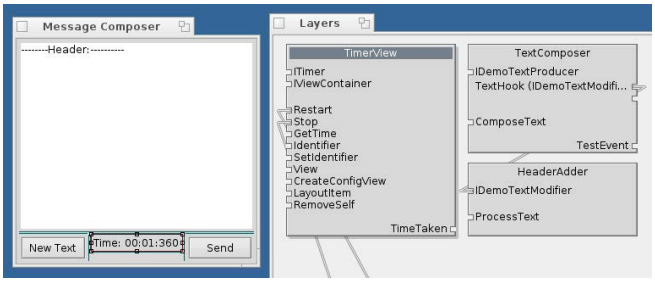

Figure 7. Message Composer: On the left side of the component layer window the timer view can be seen. This component is connected to the "New Text" and the "Send" button to start and stop the timer (the button components are not shown here). On the right side a header adder is connected to the text composer.

The participants were asked to add and connect the header and signature adders to the text composer. Afterwards, they had to add the timer view to the layout, and connect the "New Text" button to the restart method and the "Send" button to the stop method of the timer view. In order to add the composing time to the message text, they had to add the timer adder component and connect it to the text composer and the timer view.

After each step, they were able to test if their customization worked as intended.

Finally, the participants were asked the following 5-point Likert-scale questions:

Q25 I understood what functional customization is.
Q26 I understood the difference between functional and layout customization.
Q27 I understood the functional customization examples.
Q28 There is need for functional customization.
Q29 I have used applications that did not have the behavior that I expected.
Q30 I would use functional customization.
Q31 I would have liked to change the behavior of some applications I have encountered (similar to adding a gray-scale filter to the Bitmap Viewer).
Q32 I would have liked to add more functionality to some of the applications I have encountered (similar to adding a timer view to the Message Composer).
Q33 Functional customization is not necessary.

There were also some general open-ended questions:

- Can you give examples where you wanted to change the behavior of an application, similar to the example of adding a gray-scale filter?
- Can you give examples where you wanted to add functionality to an application, e.g., by adding new components?
- Comments? Suggestions?

RESULTS AND DISCUSSION

The evaluation was conducted during a user meeting of the Haiku community (BeGeistert 026, November 2012). During this meeting, 18 participants were recruited. They were all male, between 19 and 43 years old (average age 34). All of them were technical users, and 11 of them had programming experience. The media player and the mail application used in the study are standard applications of Haiku. Hence, we could assume that participants had seen or even used these applications before.

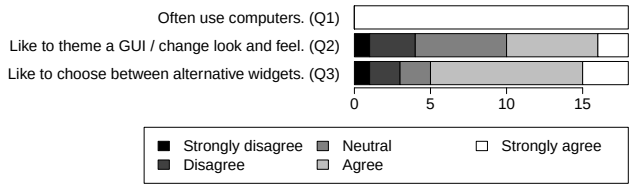

Figure 8. General questions (Q1 - Q3).

While only roughly half of the participants were interested in changing the look and feel of an application, over two-thirds stated they would like to choose between different alternative widgets if available (Figure 8). One interpretation for that is that people are less interested in purely visual changes, but see the possible benefits of alternative widgets, which could potentially offer new functionality.

Observations: Layout Customization

It seemed to be easy for the participants to change the layout of the media player. They rearranged, removed and added new widgets to the layout without problems. This is consistent with the findings of an earlier evaluation of the layout edit operations used [28].

For the mail application, participants came up with a variety of interesting layouts. For example, two participants recreated a layout they knew from other mail applications, e.g., Outlook. Other participants removed widgets they were not using frequently, e.g., the CC, BCC and the encoding widget. One user made the layout very space efficient by using only two rows.

For the Employee Manager application there were two main customizations that participants performed to optimize the GUI. The most common customization was to move the vegetarian checkbox directly beside the list box (9 participants). The second-most common customization was to move the vegetarian checkbox beside the up and down buttons (4 participants). One participant updated his initial customization by moving the vegetarian checkbox from beside the list view next to the up and down buttons after he realized that this might be an even better customization. The customizations show that participants understood the problem and were able to solve it using layout customization.

Questionnaire: Layout Customization

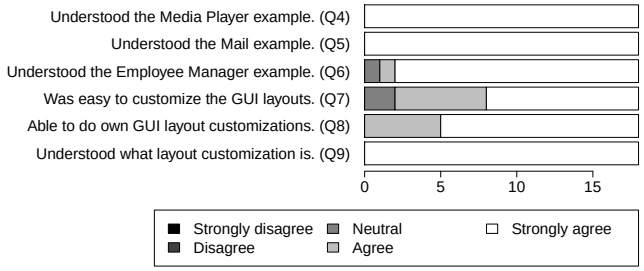

Figure 9. Questions about the layout customization tasks (Q4 - Q9).

All participants understood the layout customization tasks (Figure 9, Q4 -Q6). Almost all of them stated that it was easy to customize the layouts (Q7). All participants agreed they would be able to do their own layout customizations (Q8) and understood what layout customization is (Q9).

These results indicate that the tasks were easy to understand and that the participants got a good understanding of layout customization. This is in agreement with the observations made. The participants had no problems using the layout customization system, and would likely also be able to use the system without the help of the experimenter.

There was a wide agreement (89%) that layout customization is useful (Figure 10, Q11), and conversely a wide disagreement (89%) that there is no need for

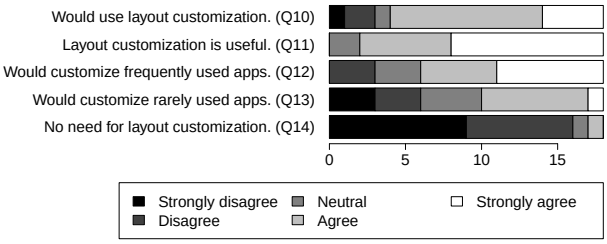

Figure 10. Opinions about layout customization (Q10 - Q14).

layout customization (Q14). Over 75% of the participants stated they would use layout customization (Q10). While 66% of the participants stated they would use it for frequently-used applications (Q12), 44% stated they would also use it for rarely-used applications (Q13).

While it was not surprising that people found layout customization useful, it is interesting that most users also found they would use layout customization in practice. Many participants stated they would not only use it for frequently used applications, but also for rarely used applications. This indicates that there is an actual demand for layout customization.

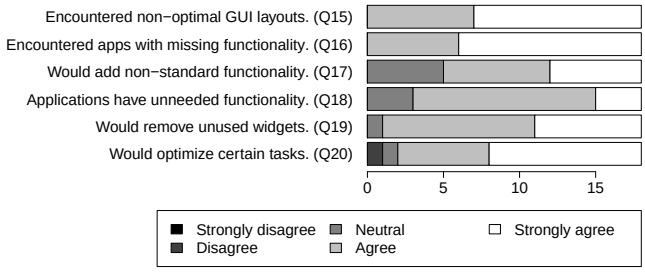

Figure 11. Questions about use-cases for layout customization (Q15 - Q20).

There was wide agreement (100%) that the participants had encountered applications with non-optimal layouts (Figure 11, Q15). All participants had encountered layouts with missing functionality (Q16), and 72% would use layout customization to add non-standard functionality to a layout (Q17). Over 80% of the participants said they used applications which had more functionality than necessary for them (Q18), and almost all agreed that they would remove such functionality using layout customization (Q19). Furthermore, over 85% stated they would use layout customization to optimize a layout for a certain task (Q20).

These results indicate that layout customization is considered valuable. Almost all participants said they had encountered applications which would benefit from layout customization, and agreed that they would use it. All three suggested use-cases, i.e., adding widgets, removing unused widgets and optimizing a layout, were considered relevant in practice.

There was no clear agreement whether users would like to share their customized layouts with other users (Fig-

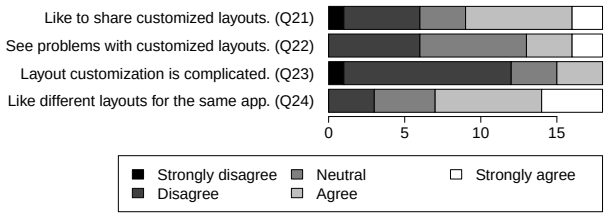

Figure 12. Answers for questions (Q21 - Q24).

ure 12, Q21). According to Q21, many participants seem to think that their own customized layouts can also be useful for others. However, it is not clear if users would also like to use customized layouts from others. The results from (Q24) suggest that many participants can imagine managing multiple layouts for an application, e.g., to optimize it for different tasks. Only three participants disagreed that they would like to use different layouts for the same application (Q24).

Most participants disagreed that layout customization is complicated (Q23), which is compatible with the converse question (Q7). While 33% of the participants did not see dysfunctional layouts as a result of customization as a problem, 28% had concerns about them (Q22). For example, users fear that modified layouts may break or hide an application's functionality, or reduce their productivity. The participants of this study were technical users, so one might expect stronger concerns from less experienced users. This indicates the importance for a customization system to leave the layout sound and the application in a functional state.

Observations: Functional Customization

The observations for the functional customization scenarios were similar to those of layout customization. Most participants had a clear idea of how to approach the customization tasks. A common problem was that participants forgot to connect some of the components. However, the participants had no big difficulties finding the problem when trying out the application, at most requiring a short comment from the experimenter to point them into the right direction. Some participants did not even need the full task explanation from the experimenter, e.g., one participant asked if certain components need to be connected even before the task was explained.

Questionnaire: Functional Customization

Almost all participants understood what functional customization is; only one participant was neutral on this point (Figure 13, Q25). Only one participant did not understand the difference between layout and functional customization (Q26). All participants understood the two functional customization examples (Q27). Only one participant disagreed and 72% agreed that there is need for functional customization (Q28). This is consistent with question Q33, were 66% disagreed that functional customization is not necessary. 78% of the participants had used an application that did not have the behavior they expected (Q29). Roughly 60% would use functional

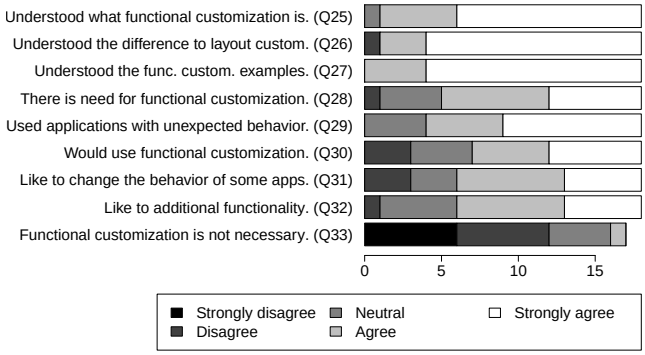

Figure 13. Questions about functional customization (Q25 - Q33). Note: one participant did not answer Q33.

customization (Q30). Questions Q31 and Q32 had similar responses: only 1-2 participants disagreed, while 66% agreed that they would have liked to change or add functionality to an application.

From the responses we can say that the introduction to functional customization was generally sufficient to familiarize participants with it and make them understand what the difference to layout customization is. Furthermore, participants seemed inclined to use functional customization in practice and would have liked to change or extend the functionality of some applications. Although functional customization is typically more complex than layout customization, most participants still wanted to use it in practice. However, the participants saw a lesser need for functional customization than for layout customization.

Feedback

There were many comments given in the open-ended questions of the questionnaire. Participants listed many applications where they would have liked to change the layout or the functionality. In the following, only the most frequent and interesting comments are summarized.

About layout customization, one participant said he "would like to use the layout customization system to create mockups for applications." Another participant mentioned he already changed the GUI of an application by modifying the source code, since no customization options were available. There were many comments stating that participants would like to add widgets (e.g., menu items) for certain functionality to applications. As examples of where unused widgets should be removed, two participants mentioned complex application such as Photoshop, Gimp and Blender[8].

One participant suggested that functional customization could work well as a simple way for non-programmers to build basic applications, given that a sufficiently large component library is provided. Moreover, there were many examples where participants wanted to enhance

[8] `photoshop.com`, `blender.org`, `gimp.org`

the functionality of an application. Two participants wanted to exchange data between different applications. Another interesting example was to add consistency checks before sending a mail to specific recipients. One participant wrote that he would like to be able to execute a macro after a certain event occurred.

Threats to Validity

There are some internal threats to validity. At the beginning of the user meeting where the evaluation was conducted, the experimenter gave a talk about layout and functional customization, and also showed some customization examples. Some participants did not attend this talk, so they may have had a disadvantage compared to the participants who attended. However, during the experiment no difference between the two groups was observed and all participants understood the tasks. Hence, we can assume that the talk had no significant impact on the results of the evaluation.

The functional customization prototype was in a very basic state, e.g., the component layer window showed a lot of irrelevant information and connecting components was implemented in an unintuitive way. Despite these limitations, participants had no major difficulties in performing the functional customization tasks. One would expect even more positive results for a more mature functional customization prototype.

Another potential issue is that most participants were part of the Haiku community. The experimenter, who had implemented the prototypes for Haiku, is also part of that community, which could lead to a social desirability bias. This means the participants could have been influenced in favor of the customization prototypes. Although some questions were asked twice with different formulations to counterbalance this effect (Q11 and Q14, and Q28 and Q33), one must be aware of this fact when interpreting the results.

An external threat to validity is that most participants were technical users or even programmers. It is difficult to say if the results would be the same for less experienced users. However, layout and functional customization targets mainly experienced, technically-skilled users who are interested in adapting their applications.

In the evaluation, there were in total five examples of layout and functional customization. This limited set of examples may have conveyed an incomplete picture to the participants of what layout and functional customization is. To avoid this problem, the examples had been designed to cover a broad spectrum of customization use-cases. From our own experience with user interface customization, we believe the use-cases in this study were representative enough to allow generalization of the results to other common use-cases of layout and functional customization.

Finally, there could be a difference between the self-reported motivation for customization and the actual motivation in a practical situation. Some of the Likert-scale questions are hard to disagree with (e.g., Q15 and Q16) and may not be a good indicator of customization in practice. To overcome such limitations, a field study of customization in practice would have to be conducted.

CONCLUSION

Previous work about customization by users focused on simple customization such as for toolbars and menus, leaving more advanced customization approaches largely unexplored. We presented prototypical systems for layout and functional customization and a user study with theses systems, exploring whether technical users are able to use such advanced customization approaches (R1), and whether they would apply them in practice (R2). The study was performed with 18 participants and customization tasks for five different plausible scenarios.

The results suggest that technical users are able to use advanced customization approaches for layout and functionality. According to our observations, it was easy for the participants to perform the given layout and functional customization tasks. The results also indicate that technical users would customize their applications in practice. Most participants stated that they do encounter use-cases for both layout and functional customization in the applications they are using, that they would customize layout and functionality, and that they consider such customization useful. More specifically, they agreed that they would like to add functionality to UIs that did not contain all the functionality they needed, remove unused widgets, and optimize layouts for their tasks.

Several participants had concerns that customized layouts may not work properly. Customization can indeed potentially render an application unusable, so it would be important to develop safeguards against this. This could be done by making sure that important widgets cannot be removed, or that a sound application state can always be restored. Customization may also lead to user documentation getting out of sync with the current application UI, and it may interfere with keyboard-based interaction. These problems constitute interesting questions for future research.

While this study provides some initial insight into the research questions, a long-term field study including average users could provide more general insights into customization, its application in practice and its long-term benefits. In many comments participants expressed that they would like to use layout and functional customization in many of the applications they were using. Some of the suggested use-cases would lead to entirely new applications, such as using layout customization to create UI mockups, or using functional customization as a tool for non-programmers to create basic applications. Exploring the long-term uses and benefits of customization is future work.

REFERENCES

1. Banovic, N., Chevalier, F., Grossman, T., and Fitzmaurice, G. Triggering triggers and burying barriers to customizing software. CHI (2012), 2717–2726.
2. Bunt, A., Conati, C., and McGrenere, J. What role can adaptive support play in an adaptable system? In *IUI* (2004), 117–124.
3. Bunt, A., Conati, C., and McGrenere, J. Supporting interface customization using a mixed-initiative approach. IUI (2007), 92–101.
4. Cockburn, A., Gutwin, C., and Greenberg, S. A predictive model of menu performance. CHI (2007), 627–636.
5. Colella, V., Klopfer, E., and Resnick, M. *Adventures in Modeling: Exploring Complex, Dynamic Systems with StarLogo*. Teachers College Press, Columbia University, 2001.
6. Debevc, M., Meyer, B., Donlagic, D., and Svecko, R. Design and evaluation of an adaptive icon toolbar. *User Modeling and User-Adapted Interaction 6*, 1 (1996), 1–21.
7. Findlater, L., and McGrenere, J. A comparison of static, adaptive, and adaptable menus. CHI (2004), 89–96.
8. Findlater, L., and McGrenere, J. Beyond performance: feature awareness in personalized interfaces. *International Journal of Human-Computer Studies 68*, 3 (2010), 121 – 137.
9. Findlater, L., Moffatt, K., McGrenere, J., and Dawson, J. Ephemeral adaptation: the use of gradual onset to improve menu selection performance. CHI (2009), 1655–1664.
10. Gajos, K. Z., Czerwinski, M., Tan, D. S., and Weld, D. S. Exploring the design space for adaptive graphical user interfaces. AVI (2006), 201–208.
11. Greenberg, S., and Witten, I. H. Adaptive personalized interfaces – a question of viability. *Behaviour & Information Technology 4*, 1 (1985), 31–45.
12. Heineman, G. T., and Councill, W. T., Eds. *Component-based software engineering: putting the pieces together*. Addison-Wesley, 2001.
13. Hoyer, V., and Fischer, M. Market overview of enterprise mashup tools. In *Service-Oriented Computing – ICSOC*, vol. 5364 of *Lecture Notes in Computer Science*. 2008, 708–721.
14. Kantorowitz, E., and Sudarsky, O. The adaptable user interface. *Commun. ACM 32*, 11 (1989), 1352–1358.
15. Lau, K.-K., and Wang, Z. Software component models. *IEEE Transactions on Software Engineering 33*, 10 (2007), 709–724.
16. López-Jaquero, V., Vanderdonckt, J., Montero, F., and González, P. Towards an extended model of user interface adaptation: The ISATINE framework. In *Engineering Interactive Systems*. 2008, 374–392.
17. Lutteroth, C., Strandh, R., and Weber, G. Domain Specific High-Level Constraints for User Interface Layout. *Constraints 13*, 3 (2008).
18. Mackay, W. E. Triggers and barriers to customizing software. CHI (1991), 153–160.
19. MacLean, A., Carter, K., Lövstrand, L., and Moran, T. User-tailorable systems: pressing the issues with buttons. CHI (1990), 175–182.
20. Marathe, S., and Sundar, S. S. What drives customization? Control or identity? CHI (2011), 781–790.
21. McGrenere, J., Baecker, R. M., and Booth, K. S. An evaluation of a multiple interface design solution for bloated software. CHI (2002), 164–170.
22. McGrenere, J., and Moore, G. Are we all in the same "bloat"? In *Proc. Graphics Interface 2000 Conference* (2000), 187–196.
23. Page, S. R., Johnsgard, T. J., Albert, U., and Allen, C. D. User customization of a word processor. CHI (1996), 340–346.
24. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. Scratch: programming for all. *Commun. ACM 52*, 11 (2009), 60–67.
25. Stuerzlinger, W., Chapuis, O., Phillips, D., and Roussel, N. User interface façades: towards fully adaptable user interfaces. UIST (2006), 309–318.
26. Tian, S., Weber, G., and Lutteroth, C. A tuplespace event model for mashups. OzCHI (2011), 281–290.
27. Zeidler, C., Stuerzlinger, W., Lutteroth, C., and Weber, G. The Auckland Layout Editor: An improved GUI layout specification process. UIST (2013).
28. Zeidler, C., Stuerzlinger, W., Lutteroth, C., and Weber, G. Evaluating direct manipulation operations for constraint-based layout. INTERACT (2013).