A Case Study on Variability in User Interfaces

Andreas Pleuss[1], Benedikt Hauptmann[2], Markus Keunecke[3], Goetz Botterweck[1]

[1]Lero – The Irish Software Engineering Research Centre, University of Limerick, Ireland
[2]Technische Universität München, Germany
[3]Universität Hildesheim, Germany

andreas.pleuss@lero.ie, benedikt.hauptmann@in.tum.de, keunecke@sse.uni-hildesheim.de, goetz.botterweck@lero.ie

ABSTRACT

Software Product Lines (SPL) enable efficient derivation of products. SPL concepts have been applied successfully in many domains including interactive applications. However, the user interface (UI) part of applications has barely been addressed yet. While standard SPL concepts allow derivation of *functionally* correct UIs, there are additional *non-functional* requirements, like usability, which have to be considered. This paper presents a case study investigating UI variability found in variants of the commercial web-based information system *HIS-GX/QIS*. We analyze which aspects of a UI vary and to which degree. The results show that just tweaking the final UI (e.g., using stylesheets) is not sufficient but there is a need for more customization which must be supported by, e.g., UI-specific models.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Software configuration management*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*User interfaces*

General Terms

Design, Algorithms

Keywords

User Interface Engineering; Software Product Lines

1. INTRODUCTION

Software Product Lines (SPL) techniques can be applied to derive a whole range of different *types of assets*, e.g., requirements, code, tests, or documentation. Ideally, product derivation is performed with tool support and automation, e.g., using concepts from model-driven engineering [2].

At a first glance, it seems that a *user interface* (UI) can be handled just like any other type of asset in a SPL. For instance, by applying product derivation techniques, it is possible to automatically derive from a given variability model configuration which UI elements must be present or absent in a product – which leads to a *functionally* correct UI. Also a UI's visual appearance can be easily customized in a systematic manner using, e.g., stylesheets.

However, we argue that there are more aspects of a UI – like navigation, dialog structure, layout, or types of UI elements – which need to vary due to *non-functional* requirements like usability. Such UI aspects cannot just be directly derived from a standard variability model or just be customized in a stylesheet. These UI aspects have not been further considered in SPL research yet.

To provide a first step into this direction, we provide the following contributions: The paper (1) provides empirical evidence on the nature of UI variability and how to measure it, (2) shows that stylesheets alone are *not* sufficient to realize UI variability but must be accomplished by UI-specific techniques, e.g., considering UI variation in the feature model or augmenting the SPL with UI-specific models.

The paper is structured as follows: Section 2 introduces the case study and the rationale for selecting the particular case for analysis. Section 3 explains which data were extracted and how. Section 4 discusses analysis results. In Section 5, we discuss the consequences and threats to validity. Section 6 concludes the paper.

2. CASE STUDY: HIS WEB APPLICATION

In this section we give a brief summary of the selected application and discuss why we selected this particular case.

The *HIS GmbH* is a German company developing university management systems since 1969 [3]. Currently there are two generations of systems: *HIS-GX/QIS* and a newer generation called *HISinOne*. This study focuses on the *online application* part of HIS-GX/QIS, where prospective students can apply online for admission to a course of study. This part is currently used by 145 universities, the majority of German universities.

Both systems have support many functional domains and a wide variety of usage contexts. This is due to several reasons: The regulating authority for universities is distributed to the 16 federal states of Germany with a high degree of autonomy. Furthermore there are several types of universities, from small art schools up to large universities. Therefore the development of both generations follows an ecosystem like approach (see [3]). HIS provides a *reference application*, which covers all functionality required by the majority of universities, in this case for student online application at universities. Customers can adapt or even extend this functionality if required.

SPLC'12 September 02-07, 2012, Salvador, Brazil

Table 1: UI properties and corresponding data extracted in our study.

Concepts		Analysis Techniques for the Presented Case Study	
UI Aspect	**UI Implementation in HTML**	**Extracted Data**	**Extracted Via**
Presentation Units	Screens (*pages*, *frames*)	Screen	HTML page
- Dialogue	Navigation (*links*)	Screen order	HTML page order
UI Elements	HTML elements (*input*, *selection*, *button*, *radio button*, etc.)	UI element	HTML element + attributes *name*, *id* + CSS attribute *visibility*
		UI element type	HTML element + attribute *readonly*
- Properties	(Logical) properties of HTML elements (*multiple selection*, *default selection*, ...)	Selection default value	Attribute *selected*
- Dialogue	Input validation (*max. input length*; mainly server-side validation of input)	Mandatory input	Label with '*' (heuristic)
		Max input length	Attribute *maxlength*
Layout	Spatial properties of HTML elements (*size*, *alignment*, etc.) + structuring (order of elements, *tables*, *divs*, etc.)	UI element order	Order of HTML elements
		UI element size	Attribute *size*
Visual Appearance	Visual properties of HTML elements (*color*, *font*, *border*, etc.), text, images	UI element label	Specific *div* with *id* according to naming convention by HIS (heuristic)

In case of the online application, at least the concrete courses to pick from are specific to the university and thus loaded dynamically from a database. To further customize the application, universities can (1) adapt the database model, (2) adapt textual content (e.g., labels and help texts), (3) adapt the CSS stylesheets, and (4) adapt the provided Velocity templates[1], which gives full control over all aspects of the UI implementation.

The UI of the reference application consists of 14 screens (HTML pages) with a sequential navigation within them. A navigation bar allows to return to any previous screen. The screens altogether contain 111 input elements including text input fields, drop-down lists, and radio buttons.

We selected this particular application for our case study for the following reasons: (1) The UI represents a good sample in terms of size and complexity; residing somewhere in the middle between trivial and very uniform UIs on the one hand and complex highly individual UIs (like in computer games) on the other hand. (2) The application domain is mature and both parties, HIS and the universities, have considerable experience. (3) The application and its usability is important for the universities but there is no excessive need to individualize the UI. (4) There is a well-defined reference application and a large number of products[2]. (5) The reference application covers all commonly required functionality but still all UI details can be customized. (6) The UIs are purely HTML-based (and do not use JavaScript due to accessibility laws) and are publicly accessible, which allows a systematic and mostly automated analysis.

3. DATA EXTRACTION AND ANALYSIS

In this section we will first discuss how to structure a UI conceptually (see the left-hand part of Table 1). Subsequently, we will discuss which of this information can be extracted for the concrete case and how we realize this extraction technically. The corresponding information is summarized in the right-hand part of Table 1.

3.1 Conceptual Structure of User Interfaces

On implementation level, a UI consists of various elements and properties. To gain a conceptual structure for UI aspects, independent of a particular implementation technology, it is worthwhile to consult literature on abstract UI descriptions, e.g., from Model-based User Interface Design (MBUID) [9, 4]. According to this literature a UI can be decomposed into *Presentation Units*, i.e., abstractions of screens, which together provide a part of the *Dialogue*, i.e., the interaction with the user, for instance through navigation between them. Presentation Units contain *UI elements*, which have logical *Properties* that define, e.g., if a selection allows multiple choices and/or has a default value. Through their behavior UI elements provide further functionality for the *Dialogue*. UI elements are arranged according to a *Layout* and presented according to their *Visual appearance* (e.g., colors, fonts, images, text strings for labels).

Table 1 shows these conceptual elements (first column) and the corresponding implementation in HTML (second column). In plain HTML, for instance, the dialogue is implemented through navigation by links and input validation, e.g., by the maximum length of an input field.

3.2 Extraction of UI structure

For the purpose of our study, we extracted data from the reference application[3] and from the corresponding products running at different universities[4]. To achieve this goal we developed a analysis tool based on *Selenium*[5], a framework for browser automation. Our tool traverses the HTML forms at a given URL and extracts the desired data from the HTML and CSS code. To enable the tool to navigate through all the pages on a site it must automatically fill out all mandatory input fields. The values for them were specified once manually; values for custom fields in a product need to be manually added at their first occurrence.

All HTML elements in the HIS-GX/QIS reference application have an id, which we use to identify corresponding elements within the products. In this way we can map the reference application onto the products and the products to each other. We considered only interactive HTML elements; other content, like text, was not considered. Table 1 shows the data our tool currently extracts (third column) and how

[1] http://velocity.apache.org/

[2] By "product" we refer to the concrete instances of the reference application, i.e., the installations customized by the universities.

[3] http://qis-demo.his.de/qisserver/rds?state=wimma&imma=einl

[4] E.g.: https://qisweb.hispro.de/fab/rds?stg=n&state=wimma&imma=einl, https://sbservice.tu-chemnitz.de/qisserver2/rds?state=wimma&stg=f&imma=einl

[5] http://seleniumhq.org/

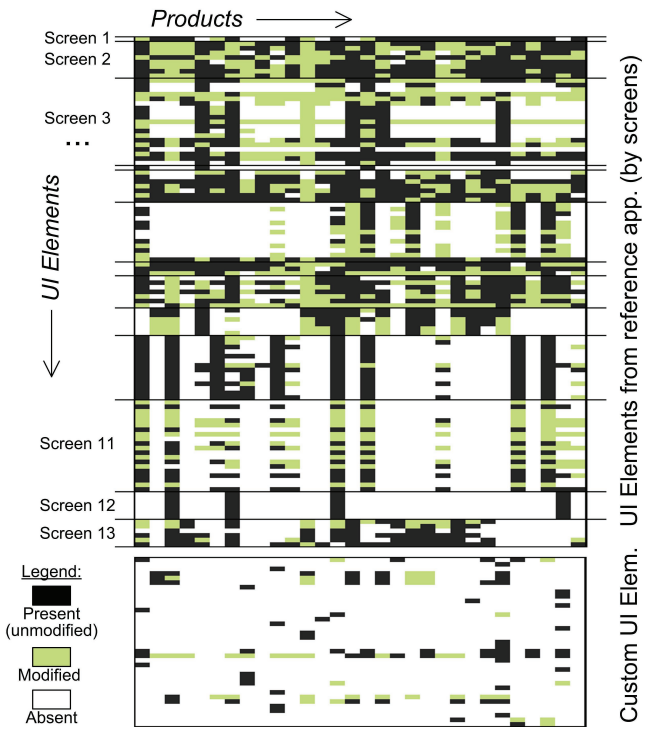


Figure 1: "Product matrix" of UI elements.

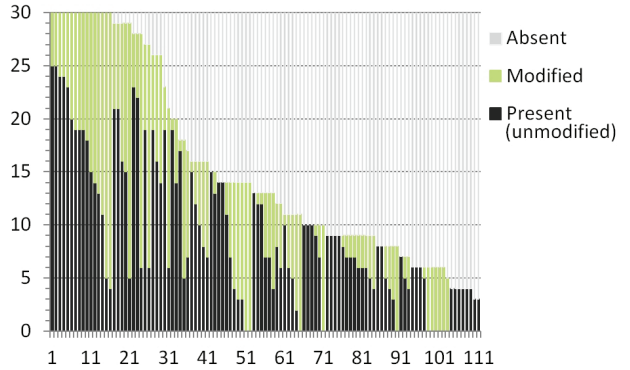


Figure 2: Usage of UI elements in the 30 products.

it is extracted from the HTML/CSS code (fourth column). For some information we use heuristics, like for text labels which are identified based on a code convention introduced by HIS. Mandatory elements can be identified by text labels marked with an asterisk, a convention which is used consistently due to recommendations in German privacy laws.

As shown in Table 1, we extract data about each UI aspect. Some information is incomplete as it cannot be extracted easily, e.g., the server-side input validation (e.g., validation of email addresses and zip codes) and the layout details (nesting of tables and "div" elements, CSS properties). In addition, we refrained from extracting further details of the visual appearance (colors, fonts, etc.) as this part of the adaptation to the universities' corporate identity is straightforward and is easily achieved by adapting CSS stylesheets.

Our tool stores the extracted data as CSV files. We then analyze it by comparison between the reference application and the products. We will describe the results of this analysis in the next section.

4. RESULTS

This section discusses the analysis results according to the UI aspects identified in the previous section.

Figure 1 provides an overview of identified UI elements: The rows represent the UI elements, the columns the products. The products are the first 30 products in an alphabetic list of Universities where the application was active at the time of the study. One product was skipped due to parsing problems caused by too much customized code. We distinguish UI elements from the reference application (111 elements, upper part of Figure 1) and custom UI elements (37 elements, lower part) which are added by products. The UI elements from the reference application are ordered by the screens they reside on (separated by horizontal bars). Several custom UI elements appeared in multiple products, e.g., 20 products added a field for the birth name.

Each cell represents whether a UI element is present without modifications in a product ■, absent ▭, or present but has been modified in the product ▭. Modifications considered here include the modifications to UI elements directly (those shown in Figure 4) but no modifications to screens.

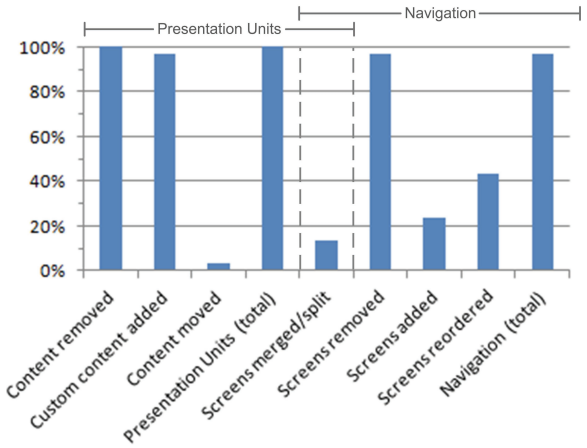
Figure 2 shows how each of the 111 UI elements from the reference application is used over the 30 products (sorted by number of appearance). Several UI elements (16) are used in every product but there is no UI element used in all products without any modification. On the other hand, there is no UI element from the reference application which is never used. For 25 UI elements (22.5%) no modification has been observed if they are used while all others (77.5%) are modified in at least one product. In the following we discuss the detected variabilities in more detail; first on the level of Presentation Units and then for individual UI elements.

4.1 Presentation Units

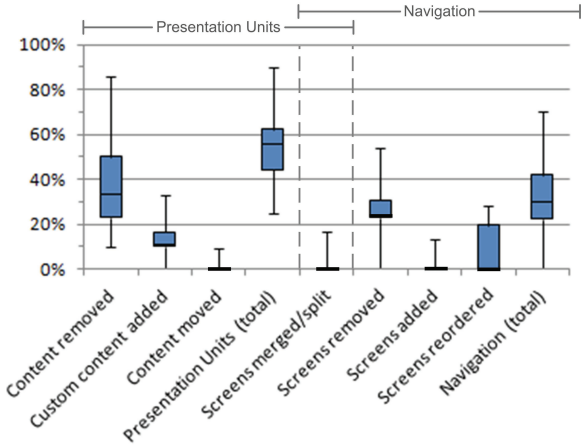
Figure 3 shows the modifications to Presentation Units (screens) observed. Figure 3a shows screen modifications *over* products, i.e., what relative amount of products is affected by a particular type of change. Figure 3b shows screen modifications *within* products, i.e., the amount of screens affected by a particular type of change (relative to the overall number of screens in the product or the reference application for *Screens removed* respectively).

As Figure 3 shows, adding and removing UI elements from or to a screen occurs frequently. More complex changes, like moving UI elements between screens or merging/splitting screens were found as well, but rarely. Please note that merging or splitting screens modifies both, the content of Presentation Units as well as the navigation. Modifications to screens could be found in all products (see *Presentation Units (total)* in Figure 3a and 3b); the median of screens modified in each product is over 50%.

Concerning the navigation, screens from the reference application were frequently removed in products. A few products added custom screens. More frequently, the *order* of screens suggested by the reference application was modified (apart from adding or removing screens). The navigation was modified in almost all products. The median value of screens where the navigation was modified (i.e., redefinition of the link to the next screen) was 30% (of the overall number of screens in the product).



(a) Screen modifications *over* products (% of products where modification occurs).



(b) Screen modifications *within* products as Box Plot (% of screens)

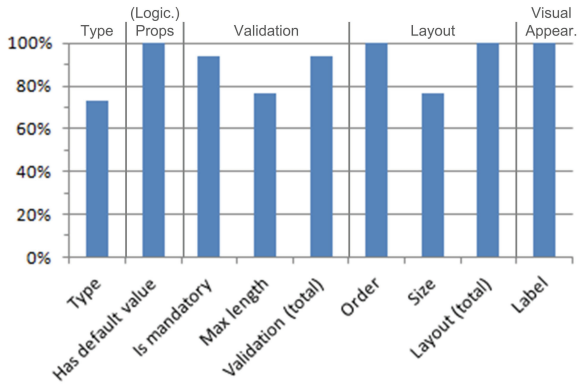
Figure 3: Screen modifications.

4.2 UI Elements

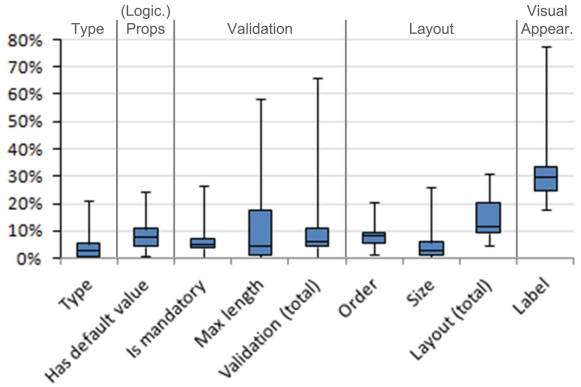
In the following we discuss modifications to UI elements, including layout and visual appearance. Figure 4 shows modifications we observed, again according to Table 1. Analogous to the previous section, Figure 4a shows the amount of products (compared to the overall no. of products) where each modification occurred at least once and Figure 4b shows how often each modification occurred within a product (relative to the no. of reference UI elements in the product).

The modifications in the figures are sorted by the UI aspects like in Table 1. The overall value for validation (*Validation (total)*) refers to all elements where *Is mandatory* and/or *Max length* are modified. *Layout (total)* refers to elements where *Order* and/or *Size* are modified. The values for *Labels* and *Is mandatory* (and *Validation (total)* accordingly) are based on 16 products only as the heuristic we used for parsing labels was not applicable to the remaining 14 ones due to manual code changes in these products.

As the results show, all types of modifications occur in the majority of products (Figure 4a). However, the amount of modifications of a certain type within a product is varying across products (Figure 4b). It also shows that even more complex modifications occur frequently, e.g., changing



(a) UI element modifications *over* products (% of products where modification appears).



(b) UI element modifications *within* products as Box Plot (% of reference UI elem. present in a product).

Figure 4: UI element modifications.

the UI element type or modifying the order of UI elements. Compared to the small number of different UI element types provided by HTML, the number of type modifications is still remarkable. Mostly input fields were changed to selections or vice versa, e.g., for fields that are intended for grades or the number of terms of previous studies. Concerning the order of UI elements, we did not count inserting or removing UI elements as a layout change and considered only those cases where the order is explicitly swapped within a screen. Again this appeared quite frequently and in all of the products at least once.

In summary, all types of modifications could be observed in a modest but considerable amount. The modifications are spread quite evenly over the products and the single UI elements. There are only 25 (out of 111) UI elements from the reference application which were never modified with respect to the analyzed properties.

5. DISCUSSION

Here, we first discuss observations and consequences from the results and then potential threats to validity.

5.1 Consequences from the results

The study has shown that there is a large amount of differences between the products' UIs. In our study we can assume that these differences are intentional (not just caused by different implicit design decisions of different per-

sons) as they result from explicit changes of the given reference application. In the following we discuss how to handle this UI variability based on the study results. Just manual customization of the final UI code is insufficient as it causes maintenance problems, e.g., when *re*-deriving products whenever a new version of the SPL is released [3].

A common solution for UI customization are stylesheets (or similar mechanisms), which allow customizing the visual appearance and layout without modifying the UI code itself. However, our study shows that there is a significant amount of "*structural*" UI modifications which cannot be handled by stylesheets, like (1) modification of Presentation Units (e.g., splitting of screens), (2) moving UI elements between Presentation Units, (3) modification of the navigation (e.g., reordering screens), or (4) variation of UI element types (e.g., replacing a text field by a selection). Hence, there is a need for additional concepts to manage such UI customizations.

Basically, there are two solutions how to specify UI customizations beyond the capabilities of stylesheets in a systematic way: 1) as part of a variability model or 2) as extra information, like specific UI models:

The first option seems promising for at least parts of the UI variability: For instance, a comparison of products as performed here (e.g., as visualized in Figure 1) can help to identify patterns across products which can be abstracted into, e.g., UI-specific features. On the other hand, existing experience reports from industry state that variability models are often not sufficient as the customers require more fine-grained control [1, 3]. This is supported by our study where we observed a high amount of single, fine-grained customizations spread over the products. Moreover, some very specific customizations seem to be tedious to define as part of a (standard) variability model. For instance, some universities want to have input fields in the same order as on their paper forms to increase input speed or to adapt the input validation according to university-specific rules [3].

The second option is to provide additional UI-specific models to support fine-grained UI customizations. First proposals into this direction can be found in the related work: A visual tool to customize Presentation Units is presented in [8]. A general approach to customize UIs within a SPL is introduced in [6, 7] (extended in [5]), based on abstract UI models adopted from the area of MBUID [9, 4]. As our study is (to our knowledge) the first one in this context it might help to refine these approaches in the future.

In summary, the study shows that there is a significant amount of UI customization which cannot be handled with mechanisms like stylesheets. The study indicates that some UI customizations might be captured efficiently by UI-specific features while others might be better specified in UI-specific models. However, a detailed comparison of the two solution alternatives goes beyond the limits of this study and is subject to future research.

5.2 Threats to validity

Our study was based on expert domain knowledge as one of the authors works in a joint project at HIS in the domain of the study. Nevertheless, our semi-automated analysis of UIs has limitations. For some UI aspects, e.g., validation and layout, we measured only a subset of potential variability. We also restricted to the navigation path resulting from our test input and have no evidence whether a specific combination of user input could cause a different navigation path. In addition, we did not consider details of the visual appearance others than labels and any other web site content besides the UI elements for user input. However, taking all those additional aspects into account would result in *equal or more* variability compared to what we found already.

Concerning the selection of the case study we chose an example with a HTML-based UI which is relatively simple compared to desktop or Rich Internet UIs. Again, having a richer UI would most probably result in *more* variability. Nevertheless, the degree of variability is still strongly influenced by the choice of the domain and the concrete application, so there is need for further studies in other domains.

6. CONCLUSION AND OUTLOOK

In this paper we presented a case study focusing on UI variability in web information systems. We have classified the properties of a UI into several aspects and have analyzed 30 products with respect to their UI variability within each of these UI aspects. As the results show, the UIs are customized in all these UI aspects and modifications are scattered over products and UI elements. Hence, mechanisms like stylesheets alone are not sufficient for UI customization as they do not cover all UI aspects (e.g., dialogue structure). There is a need for additional customization techniques like UI-specific models.

7. ACKNOWLEDGMENTS

This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero – the Irish Software Engineering Research Centre, http://www.lero.ie/.

8. REFERENCES

[1] P. Bell. A practical high volume software product line. In *OOPSLA'07*, 2007.

[2] G. Botterweck, K. Lee, and S. Thiel. Automating product derivation in software product line engineering. In *Software Engineering 2009 (SE09)*, 2009.

[3] H. Brummermann, M. Keunecke, and K. Schmid. Variability issues in the evolution of information system ecosystems. In *VaMoS'11*, 2011.

[4] H. Hussmann, G. Meixner, and D. Zuehlke, editors. *Model-Driven Development of Advanced User Interfaces*. Springer, 2011.

[5] J. Müller. Generating graphical user interfaces for software product lines: A constraint-based approach. In *15. Interuniversitäres Doktorandenseminar Wirtschaftsinformatik*, 2011.

[6] A. Pleuss, G. Botterweck, and D. Dhungana. Integrating automated product derivation and individual user interface design. In *VAMOS'10*, 2010.

[7] A. Pleuss, B. Hauptmann, D. Dhungana, and G. Botterweck. User interface engineering for software product lines – the dilemma between automation and usability. In *EICS 2012*, 2012.

[8] A. Schramm, A. Preußner, M. Heinrich, and L. Vogel. Rapid UI development for enterprise applications: combining manual and model-driven techniques. In *MODELS'10*, 2010.

[9] P. A. Szekely. Retrospective and challenges for model-based interface development. In *DSV-IS*, 1996.