User Experience Design in Software Product Lines

Nikolay Harutyunyan
Computer Science Department
Friedrich-Alexander University Erlangen Nürnberg
nikolay.harutyunyan@fau.de

Dirk Riehle
Computer Science Department
Friedrich-Alexander University Erlangen Nürnberg
dirk@riehle.org

Abstract

User experience design is an important part of software product development, and yet software product line engineering has largely ignored this topic. This paper presents a set of industry best practices for user experience design in software product lines. We conducted multiple-case case study research using two different product lines within the multinational company Siemens AG: in a healthcare software division and in an industrial automation software division. We performed a preliminary exploratory study that will serve as a baseline for future research in the design, implementation, and management of user experience design in the context of software product lines. Practitioners can use our findings and the resulting best practices to improve their user experience design, particularly within healthcare and industrial automation software product lines.

Keywords

User experience design, user interface design, software product lines, UXD, SPL, engineering best practices, handbook method, case study research

1. Introduction

In the last decade, research into user experience design (UXD) has become an important part of the field of human-computer interaction (HCI) and interaction design [10]. An increasing number of companies realize the importance of UXD in their products, as well as the importance of UXD as an essential part of software engineering. While the functional aspects of software products have traditionally been the center of attention, user experience design is now also coming into focus of the industry. UXD includes studying how users interact with and utilize a product's features, which can arguably make the difference between successful and unsuccessful products [6].

User experience is a multidimensional concept and a commonly accepted definition is still lacking [2]. Hassenzahl and Tractinsky [10] suggest that the concept of user experience attempts to go beyond the task-oriented approach of traditional HCI by bringing out aspects such as beauty, fun, pleasure, and personal growth that satisfy general human needs, but have little instrumental value. User experience is about the technology that fulfills more than just instrumental needs in a way that acknowledges its use as a subjective, situated, complex, and dynamic encounter. To achieve a good user experience of their products, companies must establish processes and teams responsible for user experience design.

UXD is the central concept of this paper. The term has no commonly accepted definition in academic literature. Therefore, for this paper we use our own definition:

UXD is the collection of practices, processes, and people used in and responsible for defining, implementing and managing user experience features, components and platforms of products.

In the scope of this paper we only discuss UXD of software products that are part of software product lines (SPLs). We study this specific subject because it lacks academic research, and yet carries significant industrial interest.

We use the general **SPL** definition by of CMU's Software Engineering Institute (SEI)[1] coined by Clements and Northrop [4]:

"A software product line is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way."

We also use the SPL **Engineering** definition by Pohl et al. [18]:

"Software product line engineering is a paradigm to develop software applications (software-intensive systems and software products) using platforms and mass customization."

SPLs have special engineering requirements due to their distributed nature. Based on centralized software platforms [18], SPLs can include hundreds or thousands of applications or products that have both common and individual features, highly coupled functional interdependencies, and common end-users. In such a context, the need for a common, yet expandable UXD is imperative for an SPL's success [2]. Designing user experience for SPLs can be more challenging, complex, and costly compared to UXD for individual products. SPLs have distributed products and development teams, often organizationally far apart and decoupled.

Industry has not and does not commonly follow accepted best practices for UXD in SPLs. This further motivates the paper, for which we analyzed the successful

[1] Software Engineering Institute at Carnegie Mellon University - http://www.sei.cmu.edu/productlines/

URI: https://hdl.handle.net/10125/60187
ISBN: 978-0-9981331-2-6

UXD practices of two SPLs and proposed derived best practices for other software companies to follow, as well as for researchers to validate and build upon.

We performed and now present exploratory research into UXD in SPLs. The goal was to derive a preliminary theory consisting of a set of industry best practices that can be applied to software product lines wanting to establish or improve their user experience design. Hence, the overarching research question was:

How should companies perform user experience design in software product lines based on existent industry best practices?

We broke down this question into subquestions focused on UXD definition, implementation, and management. These are the major aspects of UXD in SPLs identified through our initial literature review and confirmed in our expert interviews:

- How should companies define UXD in SPLs?
- How should companies implement UXD in SPLs?
- How should companies manage UXD in SPLs?

To answer these questions, we performed exploratory multi-case case study research. We derived a set of best practices of UXD definition, implementation and management. These best practices synthesize case study knowledge into actionable advice for software product lines.

We performed case study research of two independent product lines at separate company divisions of Siemens AG operating in the domains of healthcare and industry automation. We conducted three semi-structured interviews with UXD experts in similar roles within each SPL. We then employed qualitative data analysis (QDA) to analyze the data collected during the case studies, which ensured traceability between data and our findings. We then summarized the abstract findings of our research, while presenting some of the key findings in form of best practices. Our best practices are presented as patterns [5] with a Context-Problem-Solution structure at the core.

We deliberately made the choice to use a pattern structure, as opposed to a process-oriented best practices structure, because our findings do not focus on the process of UXD, but rather on individual best practices employed at distinct stages of UXD. We therefore define a best practice as the abstraction from a common solution to a recurring design problem in a given context. This form enables practitioners to benefit from our research, as argued in our previous work on benefits of using design patterns in an industry context [19].

Our contribution is a preliminary and partial theory of UXD in SPLs based on the knowledge synthesis from two case studies with different contexts and product lines, consisting of common industry best practices, including:

- 6 best practices of UXD definition
- 6 best practices of UXD implementation
- 9 best practices of UXD management.

In section 2, we present related work and literature about the topic, while identifying gaps and open questions. In section 3, we present our research approach and methodology, including case study preparation, case context, data collection, analysis methods and quality assurance. In section 4, we present the research findings as a discussion of our user experience design theory in software product lines. We present the summarized results, as well as illustrative best practices of our theory. We do not present every single best practice of the theory in detail, but rather one best practice from each high-level category: UXD definition, implementation and management. In section 5, we discuss research limitations, including threats to external validity and internal validity. In section 6, we conclude the paper.

2. Related work

We found little existing research about user experience design in software product lines. Therefore, we review general UXD research literature and compare and contrast it with our contributions to UXD in SPLs.

UXD does not have a commonly accepted definition, because it is an evolving concept in literature. UXD is rooted in the broader concept of Human-Computer Interaction (HCI) and usability research. Two decades ago, given the limited computing capacities, HCI was limited in its functionality and the focus was on enriching usability, rather than user experience [1]. We addressed this in our own definition of UXD presented in the Introduction section of this paper.

Along with the industry adaptation of UXD, papers emerged trying to establish a shared understanding of UXD, its characteristics and research basis. As there is no commonly accepted framework for the study of UXD in software product lines, we derived our own theoretical framework based on the conducted case studies. The theoretical framework consists of the three main categories derived from our data: UXD definition, UXD implementation and UXD management. These categories are also among the key UXD concepts highlighted across research papers and books [2, 6, 8, 10].

Bergaus was one of the few researchers that studied the interdisciplinary field of UXD in SPLs in a 2015 book on UXD design issues for service delivery platforms, where he focused on the study of individual user needs as the basis for a platform's UXD [2]. In line with Bergaus' findings, our theory recognizes user needs as the central topic of UXD definition in SPLs. Our best practices UXD-DEF-4 and UXD-DEF-5 suggest focusing on user needs and feedback when defining the UXD of an SPL. Bergaus focused on services and service platforms, using SPICE (Service Platform for an Innovative Communication Environment) specifications as a major input for his grounded theory research. In contrast, we don't focus on any given specification as a starting point, but instead discover only those best practices that we can trace back to the collected data.

Researchers highlight that UXD characteristics go beyond the instrumental functionality of products. According to Gaver and Martin [7], diversion and intimacy are important aspects of user experience design. Our theory does not confirm this claim as a UXD best practice for SPLs. Instead, according to our data, users of the studied SPLs gave higher priority to the functional UXD aspects rather than non-functional aspects like diversion and intimacy. Our initial theory encompasses the UXD definition best practice UXD-DEF-4 that focuses on the high-priority functional UXD aspects of an SPL. The practice suggests using SPL customer needs as the main input for UXD definition requirements, while matching them to an SPL's UXD competences.

Emotional aspects of UXD are considered to be its integral part. Several authors argue that future HCI must consider emotional aspects like stimulation, identification and evocation as part of UXD [9, 14]. The emotional state of the user is hard to research because the rigorous research methodologies (such as study of brain responses to different experiences) is complex. However, there are attempts to develop more accessible, yet rigorous methods of evaluating digital user interfaces (UI), which are part of UXD [15]. Our research results indicate that evaluation of UIs is an essential part of the UXD implementation process. However, advanced methodologies, such as evaluation of brain responses, are not widely used for software product lines. Instead a key UXD implementation best practice we found is customer evaluation of the UI and UX. Our best practice UXD-IMP-4 focuses on SPL-wide usability and UXD tests as the main evaluation technique, while the best practice UXD-IMP-5 focuses on the external evaluation of SPL's products or complete SPLs.

SPL engineering fundamentally suggests a high initial investment in developing a set of products with certain commonalities, followed by systematic and efficient derivation of products. In this context, Pleuss et al. suggest and test an approach based on using techniques from model-driven development, in order to automatically derive concrete products from a given configuration (e.g. selection of features) without the traditionally recognized drawback of lacking UXD in such setting [17]. Our data confirms the common use of feature configurations within an SPL to derive individual products, while not focusing on the automation of this process. This may have been caused by the specifics of our cases, which are long-established SPLs where such product configurations are done manually.

Like Pleuss et al., Kramer et al. also focus on automated UXD configurations for SPL products, suggesting a theory and a practical method for automated graphical user interface (GUI) generation at compile time [13]. While no specific best practice in our theory focuses on this aspect of UXD in SPLs, our best practice UXD-IMP-1 recommends establishing a common development platform for consistent UXD implementation in all products of an SPL. This includes UI implementation as a key UXD component. The implementation is automated whenever possible to increase efficiency and consistency.

Another important aspect of user experience design in software product lines is its economic valuation by the company, which is a task of an SPL's UXD management team. According to our case study partners, UXD components are measured by their economic value before inclusion into the software. Such economic valuation can be realized by ranking various UXD components based on their perceived value to the user [21]. Our findings confirm this statement, while adding additional valuation parameters – the development effort and risk of the given UXD component. This is expressed in the UXD-MGM-6 best practice suggesting effort-value mapping for UXD requirements and its specifics.

Sottet et al. studied another aspect of UXD management suggesting a variability management approach integrated into a UX prototyping process, which involves the combination of Model-Driven Engineering and SPLs. Their theory is based on the separation of concerns through multi-step partial configuration of UX features enabling each stakeholder of the UXD process to define the variability of the assets she manages [20]. Our theory suggests a best practice that in part addresses the issue of variability management and internal collaboration. The best practice UXD-MGM-4 suggests encouraging close internal collaboration on UXD variability across an SPL with clearly defined UXD stakeholder roles, responsibilities and assets.

The non-UXD aspects of SPL engineering have been researched more extensively. For example, Henard et al. studied test configurations for large SPLs suggesting automated generation and prioritization of product configurations [11]. Our findings do not confirm or reject the requirement of automated test configurations for UXD component implementation. Out of scope for this paper, this can be the subject of further research.

3. Research method

3.1. Case study methodology

We studied two independent software product lines at Siemens AG using qualitative, exploratory multiple-case case study research informed by Yin [22]. We followed Yin in preparing for the case studies, in defining the expectations from the research, in selecting the cases, in choosing of data collection methods, in collecting data, and in analysis and deriving the results.

The first step was to prepare for the case studies, which included the creation of case study protocol. Using a template for the case study protocol based on the guidelines by Yin [22] enabled us to specify the process of answering research questions in detail [3]. Following the protocol also ensured the consistency and the predefined structure of the research approach. For our case study protocol, we used the template suggested by Brereton et al. [3]. The

case study protocol addressed case study research background, design, case selection, case study procedures, data collection, analysis, validity plan, study limitations, reporting and schedule.

To ensure internal construct validity of the results, we employed Höst's and Runeson's Checklist for Software Engineering Case Study Research [12], which included key questions on case study design, preparation for data collection, collecting evidence, analysis of collected data and reporting.

We chose the product lines for our cases from the list of our industry partners. The main dimension for our case selection was the diversity of SPL domains. We selected an SPL in the industry automation software domain and another one in the healthcare software domain. We refer to the industry automation division as Case 1, and to the healthcare division as Case 2.

As the main data collection method, we chose semi-structured interviews. We conducted six expert interviews with employees holding different UXD-related roles at two distinct SPLs:

- For UXD definition questions
  - senior product manager, Case 1
  - senior usability product manager, Case 2
- For UXD implementation questions
  - software architect, Case 1
  - software developer, Case 2
- For UXD management questions
  - head of the UXD team, Case 1
  - project lead of the user interface, Case 2.

Data collection also included observation of style guidelines (guidebook of fonts, colors and other key UXD components that are common within product lines) and other UXD-related documentation. According to our case study partners, such guidelines serve as the main point of reference for developing consistent UXD for their SPLs. The following data was collected:

- 6 expert interviews, notes and follow-up discussions
  - 2 interviews with UXD definition experts
  - 2 interviews with UXD implementation experts
  - 2 interviews with UXD management experts
- materials that document UXD practices
  - UXD definition style guidelines
  - UXD implementation style guidebook
- other information on case product lines and their context (e.g. SPL and product information from Case 1 and Case 2 websites, their marketing materials etc.).

The interviews with the individuals responsible for UXD helped us understand various stages of user experience design at Siemens. The perspective of the UXD definition team was a high-level one, as they oversaw UXD as a whole. As presented in the related work, UXD currently goes beyond usability and functionality aspects of a product, while focusing also on interactions, user time and contexts etc. This became especially apparent from interviews with UXD definition experts.

The UXD managers provided perspective on product management in their product lines. They oversaw the cross-product use of central UXD components, such as common UI interfaces, sets of fonts, buttons, and their expected functionalities within an SPL. These managers were also responsible for making the decisions on modification or addition of new features and their UXD solutions. Furthermore, they prioritized the new functionality of upcoming versions of the software and provided a consistency check for UXD within a product line.

The interviewees from the UXD implementation teams illustrated the perspective of developers and software architects who were mainly involved in the implementation of new functional requirements and their UXD aspects, as well as to some extent the UXD testing and evaluation with the customers. This provided low-level and technical insights relevant for the UXD implementation best practices.

Data collection was performed parallel to qualitative data analysis (QDA) thereof. QDA consisted of coding the interviews using an external tool (MAXQDA 12) and a tool under development by at our research group (QDAcity[2]). First, we developed a codebook for the interview analysis. This helped us identify and categorize key UXD concepts, as well as to abstract from our data and to consolidate the basis for the resulting best practices. We improved our codebook iteratively by including newly identified UXD concepts from the expert interviews and by removing unused codes. The final codebook included 3 code categories and 28 codes, where the code categories corresponded to UXD definition, implementation and management, and the individual codes served as building blocks for the final best practices of our theory.

As a quality assurance measure for our QDA, we had more than 40 of our students recode our data using the original codebook in 2017. This resulted in intercoder / interrater agreement of between 0.3 (30%) and 0.4 (40%). Students performed the recoding and the agreement was automatically calculated (as the harmonic mean of coding precision and recall) using our tool – QDAcity. The intercoder agreement between original and student coding confirmed the acceptable and reproducible quality of our analysis [16]. More importantly, it helped us improve our code system and review controversial codings for the final QDA and theory building phase, as we analyzed the cases where our coding deviated from student code applications.

3.2. Case context

We conducted Case 1 in a healthcare software product line of Siemens. The SPL included more than ten business lines (BL) and the core platform team. Different business

[2] QDAcity - https://qdacity.com/

lines specialized in developing various healthcare scanner products, as well as software applications for these scanners based on the common platform developed by the SPL team. This core SPL team developed the basic applications used by different BLs, as well as the platform used by the BLs to develop more advanced applications. Through the centralized platform, the SPL had to support 65-70 application-specific tasks and 100 workflows consisting of these tasks. To achieve this, this product line had a centralized UXD team responsible for the core usability and for UXD components that were available for SPL products. These components ensured the consistent user experience for the user.

Case 2 was realized in an industry automation software product line at Siemens. While Case 1 had one SPL with many products in it, Case 2 had an SPL that itself consisted of 20 product lines, which then had individual products or so-called plug-ins that would connect to the central product and enable the user to centrally configure and monitor the automated hardware connected to the system. Similar to Case 1's UXD team for the central platform, Case 2 had a core team responsible for the software solution / platform connecting the automation hardware, central software system and individual plug-ins. The main approach was to have a centralized team responsible for core UXD building blocks such as tables, trees or text boxes with predefined UI and set functionalities. These components could then be applied by each product / plug-in team. While there were centrally defined UI aspects, such as the properties of the plug-in windows and their interactions, the semantics of each plug-in had an independently developed UXD by each product team.

Both cases separated the SPL-wide central (platform) UXD team from other UXD teams working on individual products of the SPL. This organizational structure was a result of the internal evolution and iterative refinement of the UXD approach. Thus, many best practices were affected by this structure, including the need for central management of UXD components and UXD testing, as well as the centrally planned and aligned implementation of UXD components.

4. Results

Through our data analysis three overarching high-level categories of UXD best practices emerged and structured our baseline, preliminary theory: UXD definition, implementation and management. Here we will present the major best practices and their relationships derived from the QDA of the collected data from Case 1 and Case 2.

4.1. Overview

Our preliminary findings suggest that for companies with new SPLs, the first step should be to establish an overall strategy for the given SPL's UXD definition, implementation, and management. The company also should define the corresponding UXD roles, responsibilities, and guidelines for the SPL. Companies should consider separate teams for UXD definition and implementation working under a management team. Each team should have clear cut responsibilities but coordinate with the other teams. This best practice was expressed, in one instance, in our interview with the usability product manager from Case 2 who manages a team that manages UXD definition and implementation. He briefly explained how their SPLs evolution shaped this separation of UXD responsibilities and his current role:

"I am responsible for the usability perspective of product management. So, in the evolution of the product we have clear goals - product management and development. Between product management, which is formulating the requirements, and the development, who implements the requirements, there was a need to have some type of [SPL-wide] glue regarding the usability. Somebody, who is taking care of the UXD from the customer perspective, but at the same time joins the development process. That's exactly my role [as head of usability product management team]." [Case 2, Interview 2]

Our data yields that companies with established SPLs usually have existing teams working on user experience design. Such companies can use our best practices to improve their UXD processes. This includes clear separation of common and product-specific UXD components during definition and implementation phases. These SPLs need to establish detailed guidelines and handbooks of UXD, including definition and implementation style guides with key UXD principles and rules. Both the SPLs of Case 1 and Case 2 had such guides. Here is a quote (an example trace) from Case 2's project lead of the user interface talking about the specifics of an implementation style guide, which was coded using the "style guide" code in UXD implementation code category:

"We have different levels of contribution [by the central platform and by each plug-in / product]. You say [as a plug-in UXD developer], "I have a plug-in, I have a software component. What interfaces [to the central platform] do I have to use?" This is described in the internal wikis. The style guide is the wiki documentation of all these common [UXD] interfaces you have to use. You have best practice samples. You can see how certain functionality has to be realized, if you are a beginner. The style guide only has the rules for your [SPL] plug-in. .... there are [plug-in specific] guidelines if you have to use [a specific UXD component]. It's more of documentation of [UXD implementation] best practices of how you should do it inside of the plug-in. The role of the style guide is to help you do things similar inside of the plug-ins." [Case 2, Interview 1]

Companies with advanced UXD processes in their SPLs should work on optimizing UXD management, according to our expert interviews. Some best practices here are to establish channels for UXD information exchange between product teams in an SPL. This would proposedly

reduce redundancy of developing a similar UXD component for two or more products. Instead, such components can be incorporated into the core platform of the SPL.

Another management-related industry best practice is to define value-effort mapping for UXD. This could range from simply calculating the cost of development and the customer's perceived value to a more elaborate value evaluation model, considering the possibility of having a pilot customer and then using the solutions across the SPLs. For example, the best practice of value-effort mapping for UXD (evaluating the economic value of a UX component as compared to the effort and costs needed to develop the component) can be traced back in part to our interview with the software architect from Case 1:

"As for the usability, we [SPL's central platform UXD implementation team] are the guys who implement what people [SPL's central platform UXD definition team] envisioned. Of course, in this phase, we see some issues that might have been overseen. Here we start a collaborative approach with the guys from the usability [definition] department. This becomes a dialogue in order to get their feedback and to tell them what's possible and what's cheaper [for what perceived user value]." [Case 1, Interview 2]

Our findings suggest that, for many industry experts, distinguishing between core and product-specific UXD is a key best practice. According to our data, the UXD definition team is responsible for defining the boundaries of a core platform for SPL engineering and product-specific engineering. Core and product-specific UXD components need to be treated separately and defined using different rules. Core components must include the UXD of SPL-wide functionality, such as saving or closing a document. They need to have a consistent and simple format that does not contradict the layout of specific products. Their UXD implementation should also be done separately from that of product-specific UXD development. Thus, the resulting best practice is to set up a core UXD implementation team that is responsible for the core platform used across an SPL, while each product division has a team for UXD implementation of the product-specific components. The UXD management team is responsible for deciding which components are to be part of the common core and which ones should be part of specific products. Some of our interview partners argued that UXD components can often be developed within one product, but can eventually be adapted into SPL's common core. This can happen when a functionality and its UXD component are required for several products. For the latter best practice, here is an example trace from our interview with the software architect from Case 1:

"Interviewee: [Answering to whether the SPL has experience with centrally reusing UXD components developed by individual product teams] Yes, this is a practice we use here. When you [UXD implementation engineer in an individual SPL product] integrate something like this, you need to ensure that it is a commonality for everyone [in the central SPL platform]. Sometimes one department develops something, but the others need it a little differently. In this case, we need to integrate it in a commonly acceptable way [into the central SPL platform]. Usually if it [the UXD component developed by a product team] works [for the central SPL platform], we reuse it. We try to fetch it." [Case 1, Interview 2]

UXD documentation is another essential best practice, derived from industry interviews, for consistent and efficient design. Our interview partners highlighted the importance of having documented guidelines for all responsible teams including UXD definition, implementation and management guidelines. For example, UXD definition guidelines are broad UXD principles, but give the overall direction of the user experience design and some useful techniques.

The rest of the best practices are specific to various aspects of UXD presented in the following subsections. There we answer our research question of how companies should perform UXD in SPLs by addressing three key subquestions identified in the research question section: UXD definition, implementation and management.

4.2. UXD definition

The definition of user experience components is the first phase of applying UXD to a software product line. In this phase, a team of experts conceptualize new user experience components.

Our industry-derived best practices on UXD definition suggest creating a unified UXD definition process within a product line that is used for capturing customer needs and deriving corresponding UXD concepts specific to the SPL, as a special consideration only applicable to UXD in SPLs. Industry experts suggest using consistent templates across product lines to define and communicate new UXD concepts. Another proposed best practice encourages the establishment of a common UXD glossary for a product line. Among other best practices, we propose SPLs to separate their UXD definition for the core (platform) components and product-specific ones.

This section is mostly based on our interview data from Case 1, Interview 1 (a product manager), Case 1, Interview 3 (a head of a UXD team), and Case 2, Interview 1 (a project lead of the user interface).

Our analysis suggests the following key best practices companies should use:

- **UXD-DEF-1**: During SPL creation, develop UXD definition guidelines and reinforcement mechanisms for these guidelines.
- **UXD-DEF-2**: Develop SPL-wide templates for new UXD concept definition, improve them over time and use them consistently.
- **UXD-DEF-3**: Distinguish between UXD for core functionalities of the SPL and product-specific ones. Create separate teams and processes for each.

- **UXD-DEF-4**: Focus on customers by suggesting UXD definitions that correspond to their needs, while matching SPL's competences in UXD.
- **UXD-DEF-5**: Link UXD definition to the customer needs and collected customer requirements. Keep track of requirement-UXD solution pairs across the product line.
- **UXD-DEF-6**: Define common user scenarios for most SPL products. Based on user scenarios and needs, define UXD modes (beginner mode, professional mode) for the SPL's core functionality.

Let's consider UXD-DEF-2. UXD definition templates include well-defined text fonts, colors and sizes for different functions in a software product, common and expected triggers and UI reposes (e.g. mouse clicks, shortcuts). For example, a warning message across all SPL products could be defined as a pop-up window in the middle of the working space with RGB (255,0,0) red 12-point text. Such components should be defined in a clear and detailed way to ensure efficient communication between customers and UXD implementation team. For the necessary level of detail and consistency in the UXD definition phase, companies should define and use concept templates for new UXD component definitions. We suggest a best practice for this problem in Table 1.

Table 1. Example UXD definition best practice

ID: UXD-DEF-2
Name: Develop SPL-wide templates for new UXD concept definitions, improve them over time and use them consistently.
Problem: How to create and formulate new UXD concepts in a detailed, consistent and efficient way across an SPL?
Context: UXD definition is considered a creative process, so definition teams often don't have templates for suggesting new UXD components. Each UXD engineer in the SPL uses the tools he prefers to create UXD concepts and mock-ups, for example PowerPoint presentations. However, often there is a need to compare various UXD concepts in the SPL, which can be difficult, if presentation formats and levels of detail are very different.
Solution: Even though templates are often considered as creativity killers, according to our case studies, if well designed, they can improve the creative process, by stimulating it and putting the necessary limitation and technical constraints in place. The best practice is the development of templates for UXD concepts that would include the technical details of the concepts, its mock-ups and description consistent across the SPL. These templates need to be evaluated and improved continuously to ensure that they are a stimulating tool for the concept development and not another documentation step that is perceived unnecessary and time consuming. The SPL-wide usage of such templates ensures that they evolve and lead to better UXD design. In an SPL such templates will ensure a common approach to the conception of new features and a common UXD definition. Templates save time by avoiding redesigning the basic concept structure every time. This can be a significant benefit. However, the use of the templates for UXD concepts should not eliminate the use of more sophisticated prototypes in the further phases of development. Beyond the UXD concept, there is a need for prototypes, static or dynamic. For these instances, our data suggest a freer approach in terms of the toolset used to formulate the UXD. In these stages the use of templates is not recommended.
Traces in our data: [Case 1, Interview 3] [Case 2, Interview 1] **Example trace in our data:** The head of UXD team from Case 1 explains the practice: *"So we have a template for concept definitions. Basically, all UXD changes are done with use of these concepts. Not only do we define concepts, but there are also some technical aspects to be clarified and there is basically a template that explains what the contents are on the information that needs to be gathered."* [Case 1, Interview 3]

4.3. UXD implementation

The implementation of user experience components follows the UXD definition of a software product line. In this phase a team of developers and UXD engineers implement the predefined UXD components for functionalities in either SPL-wide (product agnostic central platform) or product-specific components.

According to the analysis of our expert interviews, one crucial best practice of UXD implementation is to establish a commonly accepted style guide that details how UXD definitions should be implemented by software development teams. This includes technical details, such as development tools and environments preferred across an SPL, as well as common testing practices, guidance on prototyping, and usage of customer feedback. Another best practice highlights the use of an external evaluation of an SPL's products and their UXD. This could potentially be best achieved by developing early prototypes or working with pilot clients.

This section is mostly based on our interview data from Case 1, Interview 2 (a software architect), Case 2, Interview 1 (a project lead of the user interface), and Case 2, Interview 3 (a software developer). Our analysis suggests the following key best practices companies should use:

- **UXD-IMP-1**: Establish a common development platform for consistent UXD implementation in all products of an SPL.
- **UXD-IMP-2**: Independently implement UXD components for core functionalities of an SPL and product-specific ones. Create separate teams and processes for each. Use a centralized development platform for core UXD components. Define easy to use interfaces for product-specific component integration into the core platform.
- **UXD-IMP-3**: Define and regularly update an SPL-wide UXD implementation style guide, including precise solutions to common implementation problems, common user scenarios, corresponding UXD solutions, and references to previously developed UXD components linked to their requirements. Within the style guide, encourage simple and direct interactions, whenever possible.

- **UXD-IMP-4**: Develop a central repository for UXD tests across the product line to reduce redundancy and enable internal collaboration.
- **UXD-IMP-5**: Have an external evaluation of an SPL's key products or of the complete SPL, collect feedback from customers and systematically analyze issues caused by poor UX.
- **UXD-IMP-6**: To realize external evaluation, use mock-ups for simple UXD components and dynamic prototypes for complex or critical UXD components. Develop and share knowledge on prototyping and external evaluation across the product line.

The most essential UXD implementation best practice is to distinguish between implementation of core (central / common) and product-specific UXD components. Companies should develop a core platform in an SPL with core UXD components. Single products must be connected to the common core through interfaces, shared databases and integrated processes. There should not be a possibility for specific product teams to change the core UXD components, but there should be a possibility to suggest inclusion of some product-specific UXD components into the common core. This is demonstrated in Table 2.

Table 2. Example implementation best practice

ID: UXD-IMP-1
Name: Establish a common development platform for consistent UXD implementation in all products of an SPL.
Problem: How to organize an efficient UXD implementation of SPL's core platform?
Context: There are many approaches to UXD development in products. For an individual product, this problem is often not essential, as the implementation team can use any development tool or language to achieve their UXD goals. However, in case of a product line with many products it can be redundant and challenging to have many different UXD implementation practices and environments. This can result in inconsistencies and incompatibilities across the SPL.
Solution: The solution is to have a common development platform for all the products of a product line. Having a common development platform and environment for UXD implementation means having the same framework and the same development toolset that would ensure the UXD commonality for all the products. Moreover, this means to make a common implementation and debugging system available across the SPL. According to our data, such a platform is to be set up initially, when the product line is created. It is then utilized for developing the SPL's common core and software products within a common framework. At the same time, a common platform including tools, such as easily passing data from one application to another, are a feature required by customers. Having a common development platform can ensure that easy data passing can be implemented in each software application. Once the common platform is established, developers can work on building modules following the style guide. Such modular development pattern can be used by any software product line in case of a well-established development platform. The centralized platform ensures that the core UXD components are similarly implemented throughout the whole product line. One tool to achieve this could be the use of so called contracts (interface requirements) that each product must fulfill, when implementing a core UXD component.
Traces in our data: [Case 1, Interview 2] [Case 2, Interview 1] [Case 2, Interview 3] **Example trace in our data**: A product manager from Case 2 explains*: "If you have central functionalities, you have to fulfill contracts, so that your plug-in can contribute to the central functionality. For example, if you open and close the project, or go online, you must fulfill the contracts so that your plug-in works. If user says "Store" then all the data must be stored according to this contract."* [Case 2, Interview 1]

4.4. UXD management

The management of user experience design includes coordination between UXD definition and implementation, as well as management of customers. This category also covers decision making about the economic value of UXD components.

Our analysis of the gathered case study data suggests that one best practice of UXD management is to establish a designated UXD team within a product line. This team coordinates user experience design definition and implementation teams, as well as allocates SPL's resources and control the UXD quality. Such a team uses its crosscutting structure in an SPL to harmonize UXD across products and to reuse successful practices. Among other proposed best practices, industry experts suggest models for value-effort mapping when deciding on development of certain UXD components.

This section is mostly based on our interview data from Case 1, Interview 3 (a head of the UXD team), and Case 2, Interview 2 (a usability product manager).

Our analysis suggests the following key best practices:

- **UXD-MGM-1**: Establish a clear decision-making mechanism and define which UXD related issues should be handled at which level in an SPL. Clearly separate responsibilities of UXD definition and implementation teams.
- **UXD-MGM-2**: Establish a dedicated UXD team that manages and oversees user experience design in a product line. This ensures UXD consistency, efficient use of resources, and productive internal collaboration.
- **UXD-MGM-3**: Give UXD management team a final say in UXD-related decisions across an SPL. If no decision is made on UXD definition or implementation levels, the issue should be escalated and solved on the level of UXD management.
- **UXD-MGM-4**: Encourage internal collaboration on UXD across an SPL. Create collaboration channels, tools, and opportunities for UXD definition and implementation teams, such as centralized shared code repositories, knowledge sharing documentation, and SPL-wide meetings and discussions.

- **UXD-MGM-5**: Ensure responsive and proactive external communication with potential and existing customers focusing on UXD solutions and use cases the SPL offers. Start pilot projects, create channels for feedback, and show that this feedback is being used in an SPL.
- **UXD-MGM-6**: Create a framework for value-effort mapping for designing UXD components. Communicate common principles for value-effort mapping across an SPL and define decision making model based on the economic value.
- **UXD-MGM-7**: Offer customer training for your SPL focusing on UXD components and how to make best of the UI. Offer training together with purchase, if possible.
- **UXD-MGM-8**: Allocate 20% of an UXD expert's work time for new UXD concepts and tools exploration. This ensures innovation and employee satisfaction.
- **UXD-MGM-9**: Benchmark UXD solutions of an SPL with those of the competitors. Analyze competitors' UXD solutions and their value to the customers. Focus especially on SPL-wide UXD components corresponding to customer needs.

UXD management should focus on internal and external issues. Internal ones include defining responsibilities and collaboration between UXD definition and implementation teams, as well as assigning UXD component ownership in an SPL. Other internal aspects include managing internal suppliers, using wikis and other tools for knowledge sharing and realizing economic evaluation of UXD components. On the other hand, external aspects of UXD management include communication with existing customers and the market. It also includes initiation of pilot projects that encompass various products of an SPL, as well as customer training. One best practice on how to manage user experience designer's time is presented in Table 3.

Table 3. Example management best practice

ID: UXD-MGM-8
Name: Allocate 20% of a UXD expert's work time for new UXD concepts and tools exploration. This ensures innovation and employee satisfaction.
Problem: How much time should UXD experts have for learning new UXD concepts, practices, and tools?
Context: UXD is a relatively new domain in software development and it is becoming more and more widespread quickly. This means that a lot of new concepts and tools are being constantly developed, thus driving the demand for innovative UXD by clients. This means that UXD experts in product lines cannot spend all their time only improving the existing UXD components. They need to spend some time learning new UXD concepts and tools.
Solution: Based on our data we suggest allocating 20% of UXD expert's work time to learning new UXD concepts and acquiring new skills. This is both motivating for the employees and useful for the company that keeps up-to-date with current UXD developments. UXD management should also create an internal platform to share this knowledge across an SPL, and to use innovative ideas and tools in pilot projects or prototypes. Building skills and experience in modern UXD techniques ensures the competitive advantage of the whole SPL and the satisfaction of customers.
Traces in our data: [Case 1, Interview 3] [Case 2, Interview 2] **Example trace in our data**: A usability product management from Case 2 says*: "Everybody should have 20% time to see other things, to experience and try other things. So, everybody goes that way and then one comes and says I saw this solution, it's not bad, let's see."* [Case 2, Interview 2]

5. Limitations

External validity. The main limitation of our research is the study of only two cases. As a threat to external validity, this limits the generalizability of our theory. However, in future research we intend to add more cases and apply our exploratory theory in real-life settings to evaluate it.

Another limitation is that our theory, presented as a set of industry best practices, is still under development and in its preliminary state, while being a valuable baseline study.

Internal validity. One limitation to the internal validity of our research can be the use of only qualitative data analysis as our research method alongside case study research method for extracting the theory from our data. We addressed this limitation by introducing a quality assurance measure for our QDA. We had more than 40 students of our research course recode our data using the original codebook in 2017, after teaching them QDA techniques and its use in research.

6. Conclusions

We performed two-case case study research on user experience design (UXD) in software product lines (SPL). Our results confirm that like in general UXD, UXD for SPL can be split into three main aspects: definition, implementation and management. We describe the resulting (preliminary) theory and provide examples of best practices, which we derived from the theory to provide actionable advice to our case study partners.

Our work is the first to use an established presentation format, best practice descriptions derived from the patterns community, to present the results of theory development using case studies in software product line research.

This work is the first to combine user experience design with software product lines. In future research we plan to add more case studies to our exploratory studies in order to extend the scope and detail of our findings. We then plan to evaluate our theory of UXD in SPLs though confirmatory case studies at other companies evaluating and adjusting the proposed best practices.

To conclude, this paper serves as a baseline for future more in-depth research in the design, implementation, and management of user experience design in the context of software product lines. We identified that future research should extend the scope of the study by adding further case studies and subdomains related to the above mentioned overarching topics. Furthermore, we identified the need for practical implementation and validation of our proposed best practices through case studies informed by Yin [22] or through hypothesis testing for select best practices.

7. Acknowledgments

We would like to acknowledge our case study partners for their collaboration. We would like to acknowledge Michael Dorner, Shushanik Hakobyan and the anonymous reviewers for their valuable feedback that helped us improve the paper significantly.

8. References

[1] L. Alben. Defining the criteria for effective interaction design. *Interactions, 3*(3), 11-15. 1996.

[2] M. Bergaus. Design Issues for Service Delivery Platforms: Incorporate User Experience: A Grounded Theory Study of Individual User Needs. *Springer Vieweg*. 2015.

[3] P. Brereton, B. Kitchenham, D. Budgen and Z. Li. Using a protocol template for case study planning. *In Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*. 2008.

[4] P. C. Clements and L. Northrop. Software Product Lines: Practices and Patterns. *Addison-Wesley*, 2001.

[5] E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns. *Addison Wesley*, 1995.

[6] J. J. Garrett. The Elements of User Experience: User-Centered Design for the Web and Beyond. *Pearson Education*, 2010.

[7] B. Gaver and H. Martin. Alternatives: exploring information appliances through conceptual design proposals. *In Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 209-216. ACM. 2000.

[8] R. Hartson and P. S. Pyla. The UXD Book: Process and guidelines for ensuring a quality User Experience. *Elsevier*. 2012.

[9] M. Hassenzahl. The thing and I: understanding the relationship between user and product. *In Funology*, 31-42. *Springer*. 2005.

[10] M. Hassenzahl and N. Tractinsky. User Experience – a research agenda. *Behaviour & information technology*, 25(2): 91-97, 2006.

[11] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans and Y. Le Traon. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Transactions on Software Engineering*, 40(7), 650-670. 2014.

[12] M. Höst and P. Runeson. Checklists for Software Engineering Case Study Research. *In ESEM*, 479-481. 2007.

[13] D. Kramer, S. Oussena, P. Komisarczuk and T. Clark. Using document-oriented GUIs in dynamic software product lines. *In ACM SIGPLAN Notices*, vol. 49, no. 3, 85-94. 2013.

[14] R. J. Logan, S. Augaitis and T. Renk. Design of simplified television remote controls: a case for behavioral and emotional usability. *In Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 38(5): 365-369. SAGE Publications. 1994.

[15] L. Longo and B. Kane. A novel methodology for evaluating user interfaces in health care. *In IEEE Computer-Based Medical Systems (CBMS), 24th International Symposium*, 1-6. 2011.

[16] J. F. Marques and C. McCall. The application of interrater reliability as a solidification instrument in a phenomenological study. *The Qualitative Report*, 10.3: 439-462, 2005.

[17] A. Pleuss, B. Hauptmann, D. Dhungana and G. Botterweck. User interface engineering for software product lines: the dilemma between automation and usability. *In Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, 25-34. 2012.

[18] K. Pohl, G. Böckle, & F. J. van Der Linden. Software product line engineering: foundations, principles and techniques. *Springer Science & Business Media*. 2005.

[19] D. Riehle. Lessons learned from using design patterns in industry projects. *Transactions on pattern languages of programming II. Springer*, 1-15, 2011.

[20] J.-S. Sottet, A. Vagner and A. G. Frey. Model transformation configuration and variability management for user interface design. *In International Conference on Model-Driven Engineering and Software Development. Springer*, 390-404. 2015.

[21] K. M. Sheldon, A. J. Elliot, Y. Kim and T. Kasser. What is satisfying about satisfying events? Testing 10 candidate psychological needs. *Journal of personality and social psychology*, 80(2): 325. 2001.

[22] R. K. Yin. Case study research: Design and methods. *Sage publications*, 2013.