8-14-2019

Making Usable Generic Software. A Matter of Global or Local Design?

Magnus Li
University of Oslo, magl@ifi.uio.no

Petter Nielsen
University of Oslo, pnielsen@ifi.uio.no

MAKING USABLE GENERIC SOFTWARE. A MATTER OF GLOBAL OR LOCAL DESIGN?

Research paper

Li, Magnus, University of Oslo, Oslo, Norway, magl@ifi.uio.no
Nielsen, Petter, University of Oslo, Oslo, Norway, pnielsen@ifi.uio.no

Abstract

Usability is widely acknowledged as a desirable trait of software, referring to how usable it is to a specific set of users. However, when software is developed as generic packages, aimed at supporting variety, designing user interfaces with sufficient sensitivity to use-contexts is a challenge. Extant literature has documented this challenge and established that solving usability-related problems are difficult, both during software development and implementation. Adding to this discussion, this paper contributes by developing a framework to analyze what characterizes usability-related design of generic software. This includes two levels of design; generic-level and implementation-level, and two types of design; design for use and design for design. We apply this conceptual framework on an empirical case based on an ongoing action research project where a global generic health software is implemented in a large state in India. From the analysis we argue that attempts to strengthen usability of generic software require a holistic intervention, considering design on both 'global' and 'local' level. Of particular importance is how usable the generic software and other design-resources are when implementers are customizing the software. We coin this aspect of design as meta-usability, which represent what we see as an avenue for further research.

Keywords: Usability, Generic Software, Implementation-level design, Meta-usability.

1 Introduction

A substantial portion of the software implemented in organizations today are 'generic' or 'off-the-shelf' type of software, developed to work across an array of organizational settings and use-cases (Baxter & Sommerville, 2011). Typical examples are Enterprise resource planning software (ERPs) (Dittrich, 2014; Dittrich, Vaucouleur, & Giff, 2009), and Electronic patient record software (Martin, Rouncefield, O'Neill, Hartswood, & Randall, 2005). While many argue that functional requirements can be made generic and that the same software thus can successfully serve different organizations (Pollock & Williams, 2009), usability is well documented as a major challenge in such software implementations (Martin, Mariani, & Rouncefield, 2007; Wong, Veneziano, & Mahmud, 2016). Usability refer to how usable a system is for the users in terms of efficiency, effectiveness, and user-satisfaction (ISO, 2018). For a system to be usable, a common argument is that the user interfaces (UIs) should be designed based on the existing practices, understandings, and mental models of the intended user group (Martin et al., 2005; Rosson & Carroll, 2009). There is thus a strong relationship between usability, users and the context of use, and there are many reports from generic software implementation projects where end-users are left with complicated UIs that fits badly with established practices (e.g., Koppel et al., 2005; Topi, Lucas, & Babaian, 2005; Wong et al., 2016).

In our empirical case, based on an ongoing Action Research project, we have observed similar mismatch between design and work practices. Our focus is on a generic health information software called DHIS2. Over the last two decades, the software has moved from a domain and organization-specific routine reporting system for health indicators implemented in a few countries to a generic software platform, designed to be used in any case of health data reporting, analysis, and presentation. In this regard, the software is highly successful with implementations in over 80 countries in domains such as

disease surveillance, patient follow-ups, health commodity ordering, and logistics management. While the software has shown remarkable flexibility in supporting highly varying functional requirements, usability remains a persistent challenge in many of the implementations. This is especially prominent when the software is implemented to be used in radically different situations from what was initially intended, which is increasingly the case. For instance, in new sub-domains of health with different domain specific procedures, terminologies and conceptual logics (Nielsen & Sæbø, 2016). As experienced by end-users, problems typically take the shape of complicated UIs with abstract and unfamiliar terminology, structured in a way that provides little similarity to existing practices.

As what makes sense to users may vary significantly across domains, countries and organizations, design to ensure usability cannot only happen at the global level of development, during what we term *generic-level design*. It must be addressed also on the level of implementation, through the process of *implementation-level design*. Thus, a challenging and reoccurring question related to DHIS2 is how to improve usability in the implementations of this software? In order to answer this, we need a better understanding of the nature of usability design in generic software projects, what makes key challenges, and a vocabulary suited to describe them and how to deal with them.

Existing research around generic software usability has mainly been concerned with identifying usability-related problems, often subscribed to misfits between the software UIs and existing practice and users' mental models (Atashi, Khajouei, Azizi, & Dadashi, 2016; Khajouei & Jaspers, 2010; Koppel et al., 2005; Topi et al., 2005; Wong et al., 2016). A few papers also illustrate how solutions to such issues are difficult due to conflicting priorities among project managers and the limited ability to shape the generic software as desired during implementation (Li, 2019; Martin, Mariani, & Rouncefield, 2004; Martin et al., 2007). In line with e.g., Dittrich (2014) and Dittrich et al., (2009), we argue that generic software implementation projects represents a significantly different enviroment for design and development than the typical in-house and product development projects in which methods for usability design are based. To advance research in this area, an explicit analysis of what characterizes design affecting usability in generic software is needed, which allow further research to address the aspects that could help to strengthen it.

In this paper, our aim is to address this gap by developing a conceptual framework to describe the usability-related design that unfolds on the generic and implementation level, and, based on this, discuss how usability can be strengthened. We base our framework on the concepts of *design for use*, and *design for design* (Ehn, 2008). Applying the framework on our case illustrates its relevance and enable us to say something about where effort could be put in with the overall aim of improved usability. Concretely, our research question for this paper is; *what characterizes design for usability in the implementation of generic software packages?* Through this understanding, we identify 'meta-usability' as an important aspect to the strengthening of usability. That is, how usable the generic software and other design-resources leveraged upon during implementation-level design are for ensuring usability. With this, we provide two contributions by 1) introducing a conceptual framework useful in analyzing and describing the nature of usability design in generic software projects, and 2) identify and discuss 'meta-usability' as an aspect of particular importance, which we argue should be subject to further research.

2 Related Research: Generic Usability and Customization

A system with good usability enables the intended users to achieve specific goals with effectiveness, efficiency, and satisfaction (ISO, 2018). This means that usability is not a result of the layout and structure of the UIs of a system alone, but how well these aspects work in relation to a particular set of users in a specific context of use. More concretely, usability is to us how well the UIs of software fits with existing practices, routines and mental models of the end-users (Norman, 2013; Rosson & Carroll, 2009). The tight relation to a specific set of users makes usability especially challenging in the context of generic software packages. While some aspects of design can follow universal principles (Grudin, 1992; Norman, 2013), varying practices, terminology, existing technologies, and culture suggests that one-solution-fits-all cannot be achieved (Soh, Kien, & Tay-Yap, 2000).

There is thus a tension between systems being both *generic* and *usable*, as the first emphasizes the general, and the latter the specific. As articulated by Norman (1998, p. 78); *"making one device try to fit everyone in the world is a sure path toward an unsatisfactory product; it will inevitably provide unnecessary complexity for everyone".* An array of literature has assessed and provided detailed accounts of usability problems in implemented generic software packages, often in ERP-systems (Topi et al., 2005; Wong et al., 2016), and in generic health software (Atashi et al., 2016; Khajouei & Jaspers, 2010; Koppel et al., 2005). Reporting from the implementation of a generic ERP solution used in several countries across the globe, Topi et al. (2005, p. 132) provides a colorful and aptly quote from a frustrated end-user:

> *"it was like the spaceship had landed, and these outer space creatures [trainers] got off, and started talking to us about how we were going to do our job, because nobody understood what they were saying. Now, they're talking about notifications, material numbers, document control, material masters -- you know, that wasn't in any of our language".*

A well-established means of making usable UIs is through the involvement of end-users in the process of design (Baxter & Sommerville, 2011; Kujala, 2003; Rosson & Carroll, 2009). Here, the nature of a generic software development project differs significantly from bespoke development practices. While development of context-specific software can emphasize local particularities of practice in design, it is near to impossible for developers of generic software to directly involve end-users and cater for the specifics of local practices across all implementations (Titlestad, Staring, & Braa, 2009). Thus, during implementation, generic software will typically need to be customized to the specifics of the use-case (Dittrich et al., 2009; Martin et al., 2004).

2.1 Customization during implementation

From a functional perspective, many see customization as an unwanted activity that complicates implementation and interfere with the ability to keep the software updated with new versions of the generic software. As such, customization is only to be a last resort if organizational practices are unable or "unwilling" to change. For instance, Light (2005) outline limited competence in the implementation team, and strategic motives such as maintaining the relevance of in-house developers as prominent rationales for local customization. Rothenberger and Srite (2009) subscribe the need for customization to factors such as users resistance to change, the implementer team's lack of authority in these manners, and their *"lack of opposition to customization requests"* (Rothenberger & Srite, 2009, p. 663). From such a perspective, usability receives limited focus, and users hesitant to change are to be blamed for problems associated with use.

Representing a more "user-friendly" strand of research, Dittrich (2014) acknowledge the need for customization during implementation if sufficient fit between software and organization are to be achieved. She argues that customization should rather be encouraged by designing 'half-way products', where local "customization development" is facilitated. Such design during implementation is however not straightforward. Martin et al. (2007) provide a detailed account of the process of implementation or "domestication" of a generic software. Often, usability problems are well known, but due to obstacles such as limited "tweakability" of the software and competition with other more functional requirements, they are not solvable. Martin et al. note that *"when straightforward technical solutions to usability problems cannot be found, they are inevitably turned into training issues"* (Martin et al., 2007, p. 55), resulting in situations similar to that of the end-user quoted above. Along the same line, Li (2019) argue that working with generic software during health software implementations represent an obstacle to ensuring usability as it may constrain the ability to design according to local needs. Martin et al. (2005) describe implementation work as an integration process, where software packages should be integrated with existing practices to be usable. The authors report from the implementation of a "customizable-of-the-shelf" system and illustrate how designers are faced with the choice of addressing usability-related problems in the system through customization (technically), or through the

change of user practices and training (socially). In their words, an essential factor is thus, *"how much it [the software] will have to be tweaked, and how tweakable it is"* (Martin et al., 2007, p. 48).

The issue of sufficient design flexibility to shape generic software according to the practices of local organizations is presented as a core issue by several researchers. Krabbel and Wetzel (1998, p. 46) present the nature of generic software implementation as a major challenge to user involvement activities, and Participatory Design more specifically. In the authors' view, the customization process on the level of implementation is of equal importance to that of traditional software development, but *"management [are] often misjudging this situation expecting an easy system implementation"*. The authors note that *"... systems can differ in their degree of flexibility to be adapted to the needs of the users. If adaptability is missing it means that the vendor has to change the system code for this customer"*. In other words, to achieve fit, and to respond to feedback from end-users, flexibility for adaptation or customization at the level of implementation is argued to be of essence. Similar arguments are made by Wulf, Pipek, and Won (2008), and Roland, Sanner, Sæbø, and Monteiro (2017, p. 8) arguing that *"end-user participation in development and adaptation of a software product can be enabled or constrained by the level of flexibility with the software itself"*.

We can see that usability problems of implemented generic software packages are frequently reported in the literature. While customization by some is presented as something to be avoided, as usability is tightly related to the specifics of each use-case, and global developers are unable to design UIs that fits all organizations, much research argues for the need of design on the level of implementation. This is not straightforward due to limited priorities of usability-related aspects at this level, and as designers deals with a pre-designed artifact, possibly constrained by the flexibility to shape it according to local practices.

3 Theoretical lens: Two levels and types of design

From the existing literature, we can conclude that design relevant to the usability of generic software is related to both the level of 'global' development, and the level of implementation where customization takes place. Based on this general understanding, we will in this section develop a conceptual lens for our analysis. To refer to the usability-related UI design processes on the two levels, we introduce the terms *generic-level design*, and *implementation-level design*. On both levels, different types of design unfold. To describe these, we adopt the concepts of '*design for use before use'* and *'design for design after design'* (Ehn, 2008), or more simply; 'design for use' and 'design for design'. The latter often referred to as meta-design (Fischer & Giaccardi, 2006). First, we will give a brief definition of our two levels of design, before relating them to the two types of design processes.

3.1 Generic-level and implementation-level design

We use the term *generic-level design* to refer to the design process unfolding during the development of the generic software product. This type of design and development has similarities to that of product development, where the emphasis is on creating a product to be used by a large audience (Grudin, 1991). Functionality and corresponding UIs are thus developed based on the anticipated need of this audience. However, as it is often recognized that both functional and non-functional requirements may vary, generic-level design also concerns the development of features and resources that allow customization of the product. In the words of Dittrich (2014, p. 1454) *"part of the design is deferred to other actors closer to the concrete use context"*. During *implementation-level design*, that is, the design process unfolding during the implementation of the generic software product, these features are leveraged upon to adapt the software to meet local user needs. At this level, design and development resemble that of in-house development (Grudin, 1991), however, with a basis in the traits of the generic software package at hand (Dittrich et al., 2009). Design is as such about integrating the software with local practice (Martin et al., 2005), and central to the designers or 'implementers' task is to mediate between capabilities of the software, and the needs of the end-users and other actors of the implementing organization (Dittrich et al., 2009).

3.2 Design for use and design for design

The design unfolding on the two levels can be categorized into two types. *'Design for use'* refers to design-activities that unfold before a new or updated artifact has been introduced in a working stage to the intended end-users (Ehn, 2008). Many widespread design methodologies are based on this principle, such as User-Centered Design (Norman, 2013), and Participatory Design (Bratteteig, Bødker, Dittrich, Mogensen, & Simonsen, 2012). In the process, designers attempt to understand the users' current needs and practices, and predict and anticipate how the artifact to be designed can fit into this context. Both generic-level and implementation-level design entails this type of process. At the generic level, this regards the generic UIs that will be delivered to the end-users without any customizations during implementation. Moreover, at the level of implementation, the customization-based design could be categorized as design for use.

A common critique of this type of approach, which is highly relevant to generic-level designers, is the limitations in trying to anticipate use before it actually unfolds (Ehn, 2008; Fischer & Giaccardi, 2006). As both technologies, users, and context of use are ever-changing and evolving, this predictive form of design runs the risk of making systems that quickly become irrelevant, or without a sufficient fit from the start. Based on the limitations of design for use, '*design for design'* is based on the idea that software should be designed as open and flexible systems, or "half-way products" (Dittrich, 2014), allowing further design at a later stage. For generic software, this design after initial design takes place during implementation-level design. In practice, this means that the role of the generic-level designers is not to provide a finished product, but rather a *design infrastructure* (Ehn, 2008) providing implementation-level designers with the means of continuing to shape the artifact before final use. Design infrastructure refers to both technical and social resources that may enable design after the initial design. The infrastructure is thus not merely technical but could encompass all types of social and material elements that would aid design at a later stage (Fischer, 2008). Table 1 illustrate the relation between the levels and types of design.

	for design	for use
Generic-level design	X	X
Implementation-level design		X

Table 1. Relation between the levels and types of design in our framework

Design for design has most popularly been conceptualized in Fischer and Giaccardi (2006) framework of 'Meta-design'. The framework has mainly been applied in research on end-user development (EUD) (Ardito, Costabile, Desolda, & Matera, 2017; Fischer, Fogli, & Piccinno, 2017). The rationale behind EUD is generally to empower end-users with the tools needed to customize or extend the software themselves during use-time. From a usability point of view, this is an ideal situation as the designers are actual users able to shape the technology to correspond to their world. This form of development poses a somewhat different situation than in the implementation of generic software packages. As pointed out by Fogli and Piccinno (2013), it is often the case that end-users are not interested in engaging in customization and development work directly. Furthermore, these users typically have significantly varying computer skills. From a software governance point of view, having hundreds or even thousands of different customized versions of the software in circulation will also imply difficulties with user support, training, and system updates. Accordingly, the utilization of the design infrastructure in the case of generic software implementation happens during implementation-level design, rather than during end-use. For usability, this again means that designers are not users, and that issue of usability design is relevant (Fogli & Piccinno, 2013, p. 421). Thus, implementation-level design is about design for use based on the infrastructure provided by the generic-level designers.

To summarize our conceptual framework, generic-level design entails design of generic UIs for use, and design for design by building a design infrastructure to support customization on the level of implementation. Closer to the use-context, implementation-level design adds another process of design for use, before the product is used by the end-users. Table 2 summarizes these levels and relevant

types of design. After presenting the methods for data collection, we will apply this conceptual framework on our empirical case, which illustrates its relevance and helps us identify where effort could be put in to ensure a more usable generic software for the end-users.

Level of design	Definition	General aim	Types of design
Generic-level design	The design process unfolding during the development of the generic software product	Support a variety of use through generic interfaces and customization features	Design (of generic UIs) for use and design for (implementation-level) design
Implementation-level design	The design process unfolding during the implementation of the generic software product	Appropriate the generic software to particularities	Design for use by leveraging upon the design-infrastructure built and maintained by generic-level design

Table 2. Definition, aim and types of design for the two levels of our framework

4 Methods

Our empirical case reports from an ongoing Action Research project concerned with health information systems development and implementation. Action Research is a methodology that allow researchers to understand organizational problems and attempt interventions to evaluate their effects (Baskerville & Wood-Harper, 2016). The process is cyclic, including phases of problem diagnosis, intervention planning, doing the intervention, evaluating the effects, and documenting the learnings. The project, called the Health Information Systems Programme (HISP), has over the last two decades been engaged in activities in a variety of developing countries (Braa, Monteiro, & Sahay, 2004). A central part of the project is the development of the generic software of focus in this paper, the health information software 'DHIS2'. When implemented, the software allows for the collection, storage, analysis and presentation of health-related data. To support implementation and continuous development of the software, an extensive network of nodes of local implementers has been established in countries such as South Africa, Tanzania, Uganda, and India. The nodes possess the required competence to configure the software to the data input and output needs in the use-case at hand. Within this network, there is ongoing research on several topics concerning systems development, integration, user participation, and ICT for development.

The authors of this paper have participated in the project several years, and been involved in activities at the global level of development as well as local implementations in many countries, including Uganda and India. Data for the concrete topic of this paper is collected through the diagnostic phase of a more specific Action Research initiative aimed at strengthening the usability of an implementation of DHIS2 in a state in India. The project was triggered by the implementing organizations explicit request of strengthened usability in their system. This allowed us as researchers to follow the HISP India team in diagnosing the usability problems experienced, and obstacles and possibilities for addressing these in the DHIS2 software. Related to these aspects, we have been in continual dialog with HISP India for several months, including spending a total of six weeks in the HISP India office, and on field-trips and meetings with end-users and managers in the implementing organization within the state. The experiences in India has simultaneously been discussed with generic-level designers. The project is now moving to the stage of action planning, where the findings from the diagnosis, partly presented in this paper will serve as a basis. Methods for data collection includes interviews, attending meetings, focus groups, and participatory observations at both the level of generic-level design and implementation-level design. Our aim is to improve our understanding of the overall process of design related to usability, and more specifically how local implementers and developers work, and the challenges faced when appropriating the software to local conditions. Table 3 summarize data collection with different actors and through various activities.

Actors	Activities	Number of participants
Global developers	Informal interviews and attending meetings and discussions	Approximately 6
Local implementers and developers in India	Formal and informal interviews (approximately 6), focus group, participation in design, planning, and development activities (approximately 4 months), attending meetings (approximately 10 meetings).	8
Project managers in implementing organizations	Attending meetings and discussions (3)	8
End-users (data entry operators and health managers) in the implementing organization	Focus groups (4)	5

Table 3. Summary of data collection

During the engagement in the project, data has continuously been analyzed through a hermeneutic process of documentation and reflection (Klein & Myers, 1999). Concretely, principles and techniques from thematic analysis (Braun & Clarke, 2006) has been applied to code and categorize notes taken during data collection. Further, codes and categories has been grouped into themes, and their relation have been drawn in figures and thematic maps. Throughout the process, these themes have been discussed in light of, and linked to the existing literature presented in the previous chapters.

5 Case and Analysis

Our empirical case concerns the implementation of DHIS2 in a large (estimated population of 200 million) state in India. Referred to as the 'HMIS portal', the system is mainly serving routine health data reporting from districts to higher levels. End-users are mainly health managers and data entry operators on the various levels. In this section, we will use our theoretical lens of levels and types of design to understand what affects the achievement of usability during design of the UIs of the software. We will first provide a rather general account of the generic-level design of the software before we move into more detail on the process of implementation-level design.

5.1 Generic-level design

The generic-level design of the DHIS2 software is performed by a team referred to as the *core developers*. These include about thirty people, in the roles of designers and software developers, mainly situated in Oslo, Norway. Their aim is to build a software that supports variety so that it can be implemented to serve different types of requirements and use-cases. A central part of this is a configurable generic software 'core' where organizational structures, data elements, and relations are configured during implementation. Also, a set of *bundled apps* are developed as standard alternatives for users to perform data entry, analysis, and presentation. The design of the generic UIs of these apps can be seen as *design for use*, as many aspects will eventually face the end-users as designed on the generic level, thus directly affecting the usability.

5.2 Design for design

Many aspects will vary greatly between use-cases, so empowering the design process that ideally will take place during the implementation phase with flexibility for customization is seen as important. To this end, the core developers implement customization features in the software to enable implementers to configure the software according to local needs. This includes features for configuring the generic core, and customizing the UIs of bundled apps. Furthermore, an application programming interface (API) is maintained to allow external parties and implementers in local projects to build custom *apps* using HTML, CSS, JavaScript and front-end frameworks. Finally, a 'dashboard' application is part of the generic DHIS2 package. The dashboard allows end-users to add and arrange the content of particu-

lar interest at the landing page of the software. Content can include links to applications and reports, graphs, maps, and tables. The design of the generic customization features, the API and the dashboard can be viewed as *design for design*, as the purpose is to allow further shaping of the application during implementation-level design.

To build capacity for the utilization of these technical features, extensive documentation is available online, and an educational certification system has been developed. This is called the 'DHIS2 Academies' and are arranged as regional conferences several times a year around the world with topics such as "Design and Customization", "Data use", and modules that are more specific to concrete aspects of the software. Members of the HISP network arrange academies locally and help to strengthen and share knowledge about best practices of implementation across projects. From an institutional perspective, the training resources and the DHIS2 academies also contribute to the design infrastructure built to support local customization. Furthermore, generic-level designers receive their requirements for further development and maintenance through the network of HISP-nodes. The requirements are mediated through digital channels such as email-lists and Jira, and through meetings with a consortium of "expert" local implementers on a regular basis.

To summarize, generic-level design of the DHIS2 software involves both design for use through the development of generic UIs used directly in a variety of use-cases, and design for design of the socio-technical infrastructure provided to the implementers.

5.3 Implementation-level design

In the implementation of focus in our state in India, the DHIS2 is configured to support state-wide reporting of routine health data. The local node *HISP India* is in charge of the implementation, working together with the implementing organization in the state. Several of the implementers in HISP India has attended DHIS2 academies, and are frequent users of the learning resources available online. They are as such utilizing the social component of the design infrastructure around DHIS2. The system implemented in India is referred to as the 'HMIS portal', and consist of a collection of generic DHIS2 components and a few custom-built apps developed by HISP India to support functional requirements that were not supported by the generic apps available. The generic components include a dashboard presenting particularly relevant graphs and tables of data per user-group, and the bundled apps for data entry, analysis, and presentation. Thousands of health facilities are using the system to report, analyze and present routine health data.

The process of implementation-level design for use has been ongoing through three "phases" since 2015. As the implementers are not actual end-users, the initial phase involved participatory activities aimed at establishing the data reporting requirements. That is, what data needs to be collected to satisfy the information need of the different actors involved, such as health managers at the district and state level. Based on this, data sets of around 4000 elements were defined, data entry forms were created and various output formats configured. During the initial phase, the system was introduced to the users throughout the state, and training sessions organized. As feedback and new requirements emerged, a new phase of customization and further development was taken on. A wealth of emerging requirement and limited time to meet them has been a characteristic of the implementation process. In the words of a HISP India implementer, *"requirement that comes on Friday, should be done on Monday".* The project coordinator elaborates on this explaining that *"If the [customer] likes it, they want it right away. They come up with all sorts of reasons for why they need it quickly".* This puts a lot of pressure on the implementing team to deliver new functionality and updates with limited time for thought on UI aspects. What data to be reported, and how to present this in the various bundled apps for data presentation was at the focal point of design in the two first phases. Technically, requirements related to this has been relatively easy to meet through configuration of the core, and the bundled apps of DHIS2. Where appropriate functionality was lacking, custom apps have been built.

In a newly initiated phase three, the usability of the UIs of the implemented software has received explicit attention by the implementing organization. Feedback from frustrated end-users and managers triggered this phase, where typical problems identified include inconsistent interface design, compli-

cated UIs, and frequent mismatches between the conceptual models of processes and terminology in the system UI and in the existing use practices. According to a project manager, this is particularly apparent in the amount of training they have to arrange with the end-users. Often users have to be *"re-trained several times on the same modules. It takes a lot of effort"*. Based on this, the implementing organization has provided a list of issues to be solved, and a set of suggestions for new layouts. The HISP India team has further explored these through focus groups and discussions with end-users at district health offices and other relevant locations. Moreover, an external usability expert has reviewed the UIs of the entire portal and provided comments on pressing issues. Summarizing these inputs, four areas have been given particular focus:

- Creating new dashboard content based on end-user needs. For instance, a way for data entry workers to easily see upcoming deadlines for reports.
- Improving the UI of the bundled app 'Pivot table', by removing unused menu options. The Pivot table is commonly used across implementations of DHIS2 by end-users to create different tables of data for analysis and presentation.
- Localizing the terminology of all apps in the portal to better align with terms familiar to the end-users.
- Making the design within the portal more consistent between apps (layout of buttons, lists, menus, etc.).

5.4 Design for use

Ideally, from a usability perspective, solving the issues outlined will require to localize the generic properties of the software to match the specific community of practice, and especially the UIs. As the HMIS portal is based on a generic software, doing so is not straightforward. Rather, the implementers have to find ways of leveraging on the material properties of the design infrastructure.

For DHIS2, the technical design infrastructure gives the implementers the choice of adapting the software UIs to local needs through three approaches: 1) 'generic customization', by configuring what is possible in the generic bundled apps. This was the main approach to design in the two first phases of the implementation project. 2) 'Forking' these apps, that is, downloading the source code openly available online and changing it as desired, and 3) developing custom apps using the API. Each of these approaches has both benefits and challenges. Generic customization is by far the fastest, most efficient, and least competence intensive choice. Albeit, flexibility is seen as limited in terms of UI design, which may constrain the ability to ensure sufficient usability. 'Forking' of apps gives more UI design flexibility but will require extensive software development competence and time. Further, future updates by the generic development will not be included in the forked app, and making the forked version of the app work with new versions of the software package may imply additional maintenance work. On this basis, forking apps are not seen as a good alternative. As argued by a HISP India implementer; *"the more we use generic functionality the less hassle with updating to new versions".* Finally, developing custom apps gives the implementers extensive flexibility to design the UIs as preferred by the end-users. On the downside, the development of such apps is time-consuming and require competence as all functionality has to be built from scratch. Table 4 summarizes the pros and cons of each approach.

Technical choice	**Benefits**	**Challenges**
Configuration of bundled apps	Fast, easy	Limited design-flexibility
'Forking' bundled apps	High design-flexibility, provides a starting-point in terms of functionality as based on already working app	Time and competence intensive, need to understand existing code-base, maintenance work with software updates

Building custom apps	High design-flexibility	Time and competence intensive, has to build everything from scratch

Table 4. Technical features of the design infrastructure

As mediators between the design infrastructure on one hand, and the end user's needs on the other, the HISP India team have started the process of addressing these issues and suggestions for improvements by discussing how these could be catered for technically in the software. As the HMIS portal consists of a combination of the DHIS2 dashboard, generic bundled apps, and custom apps developed locally, the design flexibility is varying. For instance, the generic dashboard app is highly customizable. Pre-defined components such as graphs and maps can be added and arranged on screen without the need of programming. In addition, custom 'widgets' can be created using HTML, CSS, JavaScript and APIs. This allows the implementers to address the posed problems and suggestions with relative ease. For this particular challenge, addressing the needs of the end-users and ensuring usability seems feasible, and the HISP India team is now working to create widgets. The idea is that end-users later can choose from these widgets and arrange them on the screen as desired. In contrast, the changes required to make the 'Pivot table' app more usable would involve customization beyond generic configurations, which leaves the option of forking the app. With the issues related to future updates and required time this is not seen as a viable option. The problems related to consistent design and terminologies poses an even greater challenge, which spans both generic and custom apps. For custom apps built locally, UIs can relatively easily be modified with consistent design elements and a terminology suited for the end-users. For generic components such as the Pivot table, the data entry app, and other apps for data analysis and presentation, the implementers are at the mercy of the generic configuration features available, if they want to avoid the demanding process and challenges associated with forking or custom app development.

How to proceed with addressing the issues affecting usability in India is yet to be determined. With the dilemmas and difficulties faced, it is likely that many of them will be solved through more end-user training, rather than by attempting to design the UIs to better align with existing practices.

5.5 Summarizing the Characteristics

In sum, we have seen that generic-level design of DHIS2 entails both design for use and design for design. Implementation-level design, which in our case is performed by HISP India, concerns design for use. During implementation phase one and two, DHIS2 was at large able to support the rapidly emerging functional requirements by using the generic configuration options, and development of a few custom apps. However, the process of making the UIs usable based on existing practices are faced with multiple obstacles and dilemmas as each technical choice implies significant pros and cons. While the standard configuration features are quite limited, extensive design flexibility in 'forked' or custom apps development comes at the cost of time, resources, competence and constrained updatability. As such, the implementation-level design is highly affected by the design-infrastructure it operates within. Generic-level design is thus highly relevant to usability both directly and indirectly; 1) directly through the generic UIs used by end-users, and 2) indirectly, through the design infrastructure forming the basis for implementation-level design, which will enable or constrain the implementers in localizing the software sufficiently on their behalf. Figure 1 summarize the levels and types of design.

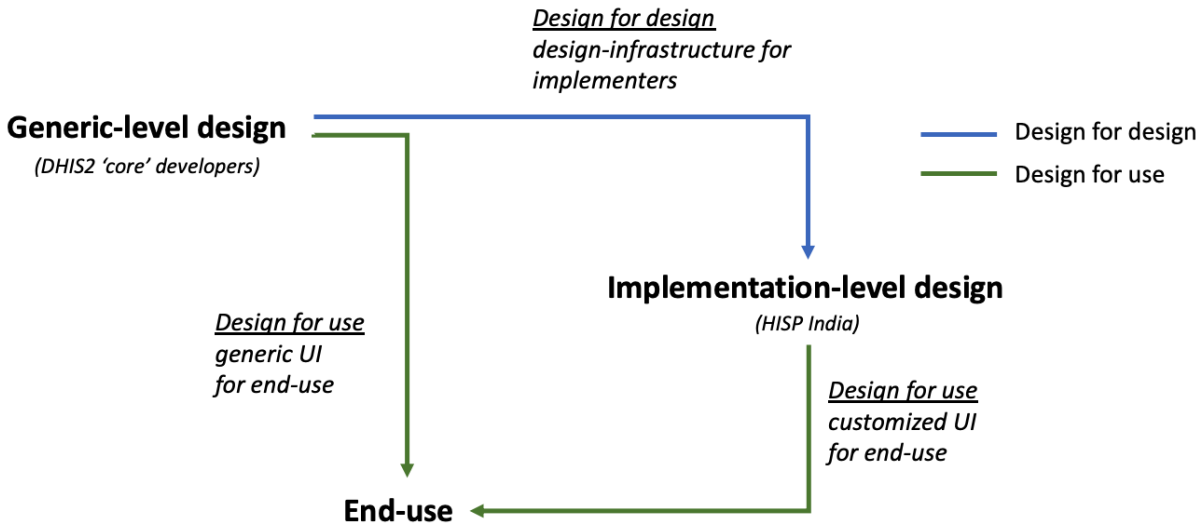

Figure 1. The two levels and types of design affecting usability

6 Discussion

In this section, we will discuss the characteristics identified, and based on this emphasize 'meta-usability' as particularly relevant to the strengthening of usability in generic software.

The end-users of the HMIS portal in India struggles with typical usability problems due to mismatches between UIs and existing work practices in terms of terminology, irrelevant menu options, and inconsistencies in UI layouts. This resembles challenges reported in prior literature on generic software implementations (e.g., Atashi et al., 2016; Khajouei & Jaspers, 2010; Koppel et al., 2005). Beyond outlining these problems, the conceptual framework we have developed in this paper has enabled us to analyze the key characteristics of usability-related design of the generic software implementation in our case. A strength of the framework is that it provides an explicit language to describe two types of design on two levels that we see as relevant to usability. In our case, the generic-level designers do design generic UIs to be used by end-users across use-cases. However, as implementations such as the one in India are filled with a variety of particularities, the generic-level designers cannot seek to sufficiently support everyone. In line with e.g., Martin et al. (2005), much is depending on the 'shaping' of the software during the implementation-level design process. It is thus important that customization is not neglected as an unwanted activity as portrayed by one strain of research on ERP system implementation (e.g., Rothenberger & Srite, 2009), where eventual problems are subscribed to end-users unwillingness to change. Rather, thought should be put into how implementation-level design best can be supported by the software and other components of the design-infrastructure to achieve sufficient 'shaping'. This illustrates the relevance of generic-level design for design. For usability to be attainable, in addition to developing usable generic UIs, generic-level design must focus on building technical features and relevant competencies for the implementation-level designers.

To suggest an answer to the title of this paper; achieving usability is about *both* global and local design. During 'global' generic-level design, design affects usability directly through generic UIs, and indirectly through the ability they give implementation-level designers to shape the software locally. The implementers leverage upon this design-infrastructure to ensure fit. Usability design is as such a joint effort between the two levels.

6.1 Meta-usability

As discussed by both Li (2019) and Martin et al. (2005), difficulties associated with 'shaping' the software as wanted represents a major obstacle for usability design during implementation. If designing for usability within this design-infrastructure is difficult as in our case, it may significantly halt the

process, and solving them through end-user training becomes a more viable option. How implementation-level design can best be supported through strategic design for design on the generic level is thus a prominent factor in the strengthening of usability. From an overall perspective, we see this as a key aspect to assuring a usable software for the end-users, which should receive more focus from researchers and practitioners. Representatives from the software industry have made similar arguments. For instance, Tao Dong, a User Experience Researcher at Google, recently articulated that *"the more usable developer tools are, the more energy developers can spend on delivering value to their users. Therefore, the UX [user experience] of developer products is just as important as for consumer products"* (Dong, 2017). Being largely dependent on the design-infrastructure provided by generic-level designers, this is particularly relevant for implementation-level design. Emphasizing its importance, we coin the term 'meta-usability' to refer to how usable the elements of the design infrastructure are in regards to achieving usability during implementation-level design. Two types of meta-usability are of particular prominence in our case:

1) How usable the software is in regards to customization and 'shaping' towards local practice, which could be referred to as *'design-usability'*
2) How usable the methods advocated through learning resources of the design-infrastructure are to aid the process of design, which could be referred to as *'method-usability'*.

First, what we refer to as *design-usability* concern how well the software supports implementation-level designers in the process of adapting it to local particularities. In addition to Martin et al. (2005) and Dittrich et al. (2009), this is in line with Singh and Wesson (2009), which describe *the ability to customize* as an important heuristic of generic software. In our case, design-usability consist of 1) customizability - what is customizable, 2) the degree of effort needed to utilize such features, and 3) to what extent utilization will collide with other desired aspects such as updates and maintenance (Light, 2001; Sestoft & Vaucouleur, 2008). This is particularly visible in our case where flexibility associated with the customization of *bundled apps* are too limited, and thus makes it impossible to shape the UIs according to local needs. At the same time, the creation of custom apps provides significant, or almost endless flexibility, however, at the cost of time, resources and need of competence. Both approaches has strong limitations when it comes to design-usability, with similarities to the *Turing Tar Pit* (Perlis, 1982) and its invert discussed by Fischer (2008, p. 368). Customization of bundled apps provides an environment "*where operations are easy, but little of interest is possible"* while during development of custom apps *"everything is possible, but nothing of interest is easy."* Also, if certain customization features significantly impact the work associated with updating to new versions of the global software package, they appear less usable from the implementers' point of view (Light, 2001; Sestoft & Vaucouleur, 2008).

Second, implementers are not end-users, so methods and techniques need to be used to understand how UIs best should be designed to integrate well with the use-context. Examples could be usability-inspections, user-centered design techniques or scenario-based design (Rosson & Carroll, 2009). Accordingly, in our case, such methods were applied to evaluate the UI to identify usability problems, and to gain knowledge on the real end-users' challenges related to UIs, and how to solve them. The methods used have often been conveyed to the implementers through the DHIS2 academies and learning resources that make up the social components of the design-infrastructure. Communication of methods well suited to achieve usability for the specific generic software, which are relatively easily adoptable by the implementation-level designers could be an important part of the design-infrastructure. As discussed by Baxter and Sommerville (2011), *method-usability*, that is, the usability of the method applied to aid the design process is thus relevant. For instance, the method needs to align well with the customization features of the software, the nature of the project, and the competencies and goals of the involved actors. Table 4 provides a summary of usability and meta-usability.

Concept	**Definition**
Usability	How usable the software is to a specific set of end-users

Meta-usability	How usable the elements of the design infrastructure are in regards to achieving usability during implementation-level design. Meta-usability includes: ***Design-usability:*** How usable the software is in regards to customization and 'shaping' towards local practice. Hence, how well the software supports implementation-level design. ***Method-usability:*** How usable the method applied to aid the process of usability design are, or, again, how well the method supports implementation-level design.

Table 5. Usability and meta-usability

6.2 How to strengthen usability?

Based on our analysis, it becomes apparent that an attempt to strengthen usability is not merely a matter of generic or implementation-level design, but rather will require a holistic perspective and intervention. Holistic in the way that interventions need to consider both generic and implementation-level design, and design for design and design for use. Particularly, design for design on the generic level will need to cater for design for use at the level of implementation. For instance, for DHIS2, a relevant intervention could be to extend the generic configuration options in the software to allow for the translation of terminology in the UI to correspond to an end-user familiar language. Also, strengthening design-usability by creating customization environments that are flexible, yet efficient and easy to use will be a great improvement. These technical interventions must be accompanied by the definitions and teaching of methods that correspond to these particular capabilities, thus increasing method-usability. This *software-method alignment* is in our opinion a particularly important factor, which has received limited focus in existing research. In sum, we argue that seeing the process as a means of strengthening meta-usability could be fruitful, where interventions to improve design-usability and method-usability should be an integrated process.

7 Conclusion and future research

We set out to explore the question; *what characterizes design for usability in the implementation of generic software packages?* In our case, the design process that affect the usability for end-users is characterized by two types of design (design for use and design for design) unfolding on two levels (generic-level design and implementation-level design). For usability to be attainable, generic-level design needs to focus on design for design, by building technical features and relevant competencies for the implementation-level designers. Implementation-level design needs to focus on design for use by mediating between the end-users existing practices and understandings, and the technical features of the generic software. Furthermore, strengthening usability will require a holistic intervention that involves both levels and types of design, where the generic-level designers' focus should lie on the strengthening of meta-usability. That is, ensuring that the elements of the design infrastructure are usable in regards to achieving usability during implementation-level design. This especially involves the alignment of software customization features and usability design-methods. Based on this, we suggest two topics suited for further research, which also are highly relevant to practitioners involved in generic software projects.

How to strengthen the design-usability of generic software?

This will require extended consideration of what needs to be customizable, how it can be made easy and efficient, while not interfering with the updatability from global releases. An interesting aspect is to find a balance between the "Turing tar pit" and its inverse. For this, a fruitful endeavor could be to explore the use of design-systems for app development to balance between flexibility, while keeping the barrier to take on such development as low as possible (Frost, 2016; Wulf et al., 2008).

How to strengthen the usability of design methods applied at implementation-level design?

Making methods more aptly would, as discussed, require and alignment between software and method, but also sensitivity to the nature of implementation projects and pressing issues such as scale, distribution and heterogeneity of users and practices within the "local" implementation level (Li, 2019; Sommerville et al., 2012).

References

Ardito, C., Costabile, M. F., Desolda, G., & Matera, M. (2017). A three-layer meta-design model for addressing domain-specific customizations. In *New Perspectives in End-User Development* (pp. 99-120): Springer.

Atashi, A., Khajouei, R., Azizi, A., & Dadashi, A. (2016). User Interface problems of a nationwide inpatient information system: a heuristic evaluation. *Applied clinical informatics, 7*(1), 89.

Baskerville, R. L., & Wood-Harper, A. T. (2016). A critical perspective on action research as a method for information systems research. In *Enacting Research Methods in Information Systems: Volume 2* (pp. 169-190): Springer.

Baxter, G., & Sommerville, I. (2011). Socio-technical systems: From design methods to systems engineering. *Interacting with computers, 23*(1), 4-17.

Braa, J., Monteiro, E., & Sahay, S. (2004). Networks of action: sustainable health information systems across developing countries. *Mis Quarterly*, 337-362.

Bratteteig, T., Bødker, K., Dittrich, Y., Mogensen, P. H., & Simonsen, J. (2012). Organising principles and general guidelines for Participatory Design Projects. *Routledge Handbook of Participatory Design*, 117.

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative research in psychology, 3*(2), 77-101.

Dittrich, Y. (2014). Software engineering beyond the project–Sustaining software ecosystems. *Information and Software Technology, 56*(11), 1436-1456.

Dittrich, Y., Vaucouleur, S., & Giff, S. (2009). ERP customization as software engineering: knowledge sharing and cooperation. *IEEE software*(6), 41-47.

Dong, T. (2017). Developer UX at Google. *Medium.com.* Retrieved from https://medium.com/google-design/how-i-do-developer-ux-at-google-b21646c2c4df

Ehn, P. (2008). *Participation in design things.* Paper presented at the Proceedings of the tenth anniversary conference on participatory design 2008.

Fischer, G. (2008). Rethinking software design in participation cultures. *Automated Software Engineering, 15*(3), 365-377.

Fischer, G., Fogli, D., & Piccinno, A. (2017). Revisiting and broadening the meta-design framework for end-user development. In *New perspectives in end-user development* (pp. 61-97): Springer.

Fischer, G., & Giaccardi, E. (2006). Meta-design: A framework for the future of end-user development. In *End user development* (pp. 427-457): Springer.

Fogli, D., & Piccinno, A. (2013). Enabling domain experts to develop usable software artifacts. In *Organizational change and information systems* (pp. 419-428): Springer.

Frost, B. (2016). *Atomic design*: Brad Frost.

Grudin, J. (1991). Interactive systems: Bridging the gaps between developers and users. *Computer*(4), 59-69.

Grudin, J. (1992). Utility and usability: research issues and development contexts. *Interacting with computers, 4*(2), 209-217.

ISO. (2018). ISO 9241-11:2018. In *Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts*: International Organization for Standardization, Geneva, Switzerland.

Khajouei, R., & Jaspers, M. (2010). The impact of CPOE medication systems' design aspects on usability, workflow and medication orders. *Methods of information in medicine, 49*(01), 03-19.

Klein, H. K., & Myers, M. D. (1999). A set of principles for conducting and evaluating interpretive field studies in information systems. *Mis Quarterly*, 67-93.

Koppel, R., Metlay, J. P., Cohen, A., Abaluck, B., Localio, A. R., Kimmel, S. E., & Strom, B. L. (2005). Role of computerized physician order entry systems in facilitating medication errors. *Jama, 293*(10), 1197-1203.

Krabbel, A., & Wetzel, I. (1998). *The customization process for organizational package information systems: A challenge for participatory design.* Paper presented at the Proceedings of the PDC.

Kujala, S. (2003). User involvement: a review of the benefits and challenges. *Behaviour & information technology, 22*(1), 1-16.

Li, M. (2019). *Usability Problems and Obstacles to Addressing them in Health Information Software Implementations*. Paper presented at the The International Conference on Social Implications of Computers in Developing Countries (IFIP WG 9.4), Dar es Salaam, Tanzania.

Light, B. (2001). The maintenance implications of the customization of ERP software. *Journal of software maintenance and evolution: research and practice, 13*(6), 415-429.

Light, B. (2005). Going beyond 'misfit'as a reason for ERP package customisation. *Computers in industry, 56*(6), 606-619.

Martin, D., Mariani, J., & Rouncefield, M. (2004). Implementing an HIS project: everyday features and practicalities of NHS project work. *Health Informatics Journal, 10*(4), 303-313.

Martin, D., Mariani, J., & Rouncefield, M. (2007). Managing integration work in an NHS electronic patient record (EPR) project. *Health Informatics Journal, 13*(1), 47-56.

Martin, D., Rouncefield, M., O'Neill, J., Hartswood, M., & Randall, D. (2005). *Timing in the art of integration:'that's how the bastille got stormed'.* Paper presented at the Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work.

Nielsen, P., & Sæbø, J. I. (2016). Three strategies for functional architecting: cases from the health systems of developing countries. *Information Technology for Development, 22*(1), 134-151.

Norman, D. (2013). *The design of everyday things: Revised and expanded edition*: Constellation.

Norman, D. A. (1998). *The invisible computer: why good products can fail, the personal computer is so complex, and information appliances are the solution*: MIT press.

Perlis, A. J. (1982). Special feature: Epigrams on programming. *ACM Sigplan Notices, 17*(9), 7-13.

Pollock, N., & Williams, R. (2009). Global software and its provenance: generification work in the production of organisational software packages. In *Configuring User-Designer Relations* (pp. 193-218): Springer.

Roland, L. K., Sanner, T. A., Sæbø, J. I., & Monteiro, E. (2017). P for Platform: Architectures of large-scale participatory design. *Scandinavian Journal of information systems, 29*(2).

Rosson, M. B., & Carroll, J. M. (2009). Scenario-based design. In *Human-computer interaction* (pp. 161-180): CRC Press.

Rothenberger, M. A., & Srite, M. (2009). An investigation of customization in ERP system implementations. *IEEE Transactions on Engineering Management, 56*(4), 663-676.

Sestoft, P., & Vaucouleur, S. (2008). Technologies for evolvable software products: The conflict between customizations and evolution. In *Advances in Software Engineering* (pp. 216-253): Springer.

Singh, A., & Wesson, J. (2009). *Evaluation criteria for assessing the usability of ERP systems.* Paper presented at the Proceedings of the 2009 annual research conference of the South African Institute of Computer Scientists and Information Technologists.

Soh, C., Kien, S. S., & Tay-Yap, J. (2000). Cultural fits and misfits: is ERP a universal solution? *Communications of the ACM, 43*(4), 47-47.

Sommerville, I., Cliff, D., Calinescu, R., Keen, J., Kelly, T., Kwiatkowska, M., . . . Paige, R. (2012). Large-scale complex IT systems. *Communications of the ACM, 55*(7), 71-77.

Titlestad, O. H., Staring, K., & Braa, J. (2009). Distributed development to enable user participation: Multilevel design in the HISP network. *Scandinavian Journal of information systems, 21*(1), 3.

Topi, H., Lucas, W. T., & Babaian, T. (2005). *Identifying Usability Issues with an ERP Implementation.* Paper presented at the ICEIS.

Wong, W.-P., Veneziano, V., & Mahmud, I. (2016). Usability of Enterprise Resource Planning software systems: an evaluative analysis of the use of SAP in the textile industry in Bangladesh. *Information Development, 32*(4), 1027-1041.

Wulf, V., Pipek, V., & Won, M. (2008). Component-based tailorability: Enabling highly flexible software applications. *International Journal of human-Computer studies, 66*(1), 1-22.