Practitioner-Customizable Clinical Information Systems: A Case Study to Ground Further Research and Development Opportunities

Cecily Morrison[1,*], Alan F. Blackwell[1], Alain Vuylsteke[2]
[1]Computer Laboratory, University of Cambridge, JJ Thompson Avenue, Cambridge, CB3 0FD, UK
[2]Papworth Hospital, Papworth Everard, Cambridge, CB23 3RE, UK

ABSTRACT
The uptake of electronic records and information technology support in intensive care medicine has been slower than many people predicted. One of the engineering challenges to overcome has been the subtle, but important, variation in clinical practice in different units. A relatively recent innovation that addresses this challenge is practitioner-customizable clinical information systems, allowing clinicians wide scope in adjusting their systems to suit their clinical practice. However, these systems present a significant design challenge, not only of added technical complexity, but in providing tools that support clinicians in doing many of the tasks of a software engineer. This paper reviews the use of a commercially available clinical information system that is intended to be practitioner-customizable, and considers the further design and development of tools to support healthcare practitioners doing end-user customization on their own clinical information systems.

Keywords: clinical information system, practitioner-customisation, end-user development

1. INTRODUCTION
Intensive Care Units (ICUs) are full of technology, from ventilators and blood gas machines to mobile computerized tomography (CT) scanners. Many ICUs, however, continue to capture the large volumes of data needed for assessing a patient's response to care on paper charts rather than in a clinical information system (CIS). Although the literature covers many organizational reasons that may account for the slow switch from paper to electronic records [1], a significant engineering issue is the subtle differences in requirements for each ICU. Industry has responded to this challenge in the past by engineering CISs that can be customised by Information Technology (IT) consultants. This kind of customisation is usually a costly process and often does not have the speed of iteration which allows clinical teams to fully adapt and adopt the CIS into their work [2].

*Corresponding author, cpm38@cam.ac.uk

In response, a number of recent CIS products have been designed to be customized directly by healthcare practitioners. Products of this kind are being made available both commercially (e.g., [3, 4]) and through open source projects (e.g., [5, 6]). A *practitioner-customizable CIS* is one in which a select number of trained healthcare practitioners are able to customize locally, by themselves and without prior programming knowledge, most or all aspects of the system with which they and their colleagues interact. This includes all data fields, work flows, and clinician decision support tools, such as alerts, as well as reports generated from data in the system. Such flexibility gives tremendous advantage to healthcare practitioners, by allowing them to adapt the system to their local ways of working. However, it creates a significant design challenge that healthcare engineering research needs to address.

Some of the challenge is technical, such as finding a mechanism to allow healthcare practitioners to write new fields into a database without compromising the integrity of the database or the software that accesses it. Much of the challenge however, derives from designing the tools that support healthcare practitioners in carrying out a range of practices normally done by a team of trained IT specialists, from interface design to updating software in a critical environment. The field of End-User Development (EUD) has begun to research the design of tools to facilitate end-user development and customization and is drawn upon throughout this paper (e.g. [7]). However, end-users customizing healthcare systems, or any critical systems, have not been researched to our knowledge.

In this paper, we provide a starting point for such research. We take advantage of access into an ICU which has been actively customizing for the past three years their commercially available, practitioner-customizable CIS. *Probing the concept of practitioner-customization, we examine the tools, practices, and organizational issues that influenced the customization of a practitioner-customizable CIS to enable further research and development of this emerging technology.*

2. STUDY DESIGN

The first two authors of this paper were invited as part of a 4-person multi-disciplinary research team to evaluate the introduction of a practitioner-customizable CIS into an ICU. The clinical team responsible for the introduction wanted to ensure that work practices were not disturbed and that job satisfaction among ICU staff did not fall. The research team, including experts in computer science, social science and information systems, developed a research program around these concerns, assessing changes in work practices between the paper and electronic records (e.g., [8]). However, throughout the three-year study, the research team realised that the ability of the healthcare practitioners to customise their system was both novel and an integral part of the success of the CIS and should be researched more explicitly. The study presented in this paper builds on our observations throughout the original study and uses them to contextualise a shorter, more focused study on customization.

The study specific to customization was designed to understand how end-user customization is carried out by healthcare practitioners. We felt a good starting point for

this exploratory study would be to investigate the customization tools used, Metavision Customization Suite in this case, and the social or cognitive practices that influenced tool usage. We soon discovered that both tools and practices are influenced by the organizations which maintain them, so we considered organizational issues as well. In order to probe the subtleties that affect the ways healthcare practitioners customize, we chose to use an elicitation technique – an activity that provides a way of gaining access to unarticulated knowledge of an informant [9].

The study was structured as a series of dialogues between the lead customizer of the CIS in the ICU of our case study ("the customizer"), who is a healthcare practitioner whose job description includes customizing the CIS, and a professional freelance Visual Basic programmer ("the programmer") who was recruited for the purposes of this study via a local online recruitment site. The catalyst of this interaction was to create a new report function, giving a customised summary of data and notes on patient care that was needed in the ICU. This particular report was beyond the capabilities of the customizer as it was outside the possibilities provided in the customization tools, and could only be carried out with some programming skill that the customizer had not yet acquired.

The dialogue-based elicitation technique applied here is known as 'constructive interaction'. This technique is theoretically grounded in [10] and has been previously applied to the study of Human-Computer Interaction (e.g., [11]). The constructive interaction approach to elicitation had a number of benefits for this study. It provided motivation for the customizer to participate and supplied a context within which the researchers could observe the use of the customization tools. Setting up the dialogue between the customizer and the programmer encouraged the former to verbalize practices and organizational issues surrounding customization, while avoiding bias introduced by the researchers having too much contact with the customizer. It also afforded a means to articulate and contrast the mindsets of a customizer and a programmer.

The study involved four working sessions between the customizer and the programmer, each four to six hours long. In the first session, the customizer introduced the customization tool to the programmer and explained the requirements for the report and why it could not be done with the customization techniques that she had previously used. The second and third sessions were for the programmer to work with the database of the CIS and the customization tools to produce the new report function. There were frequent, and revealing, interchanges between the programmer and the customizer when the former asked for clarification of the customization tools. The final session was the hand-over and explanation of the code created to that point and a general discussion of the outcome of the project.

All sessions were observed as well as recorded, transcribed and annotated independently by two researchers. A data analysis session was then held to draw out salient points that could contribute to the research and development of practitioner-customizable CISs. As the research does not consider any patient data or interaction with patients, it has been classified as an 'audit and service evaluation' activity by the Cambridgeshire Research Ethics Committee and does not require National Health

Service (NHS) ethics committee approval; however, all participants were aware of the purpose of the studies and the research was monitored by the lead clinician who is the last author on this paper.

3. BACKGROUND

3.1. The ICU

The case study was situated in an ICU of a cardio-thoracic specialist hospital. As in all ICUs, there are healthcare practitioners from many different disciplines working together to care for severely ill patients, including doctors, registrar (junior) and consultant (senior), three levels of nurses (head nurse, bay nurse and bed nurse), a dietician, a pharmacist, a microbiologist, and a group of physiotherapists. The strong leadership of this unit actively encourages communication across disciplinary boundaries, something notably difficult to achieve [12]. There are consequently a number of formal and informal practices that bring together multi-disciplinary teams to discuss changes in the ICU. This history of bringing all of the stake-holders into one room on a regular basis supported all aspects of the introduction of the CIS, from selecting it to customizing it.

Customizing the CIS was not the ICU's first experience with the design of a supporting information system. Prior to the introduction of the CIS, the ICU had re-designed their set of paper charts as described in [3]. The charts were streamlined so that the same information (e.g., chart name, patient name) was always found in the same place; the charts were adapted to the clinical work flow so that they were filled out in the same order in which the work was carried out; and the charts were designed to highlight problems in treatment at a glance. This experience of iteratively developing an information system, designing the charts and then reflecting on how their usage suited clinical practice, provided a grounding for the development of the CIS.

3.2. A Practitioner-Customizable System

The clinical head of the ICU, a consultant anaesthetist, proposed investing in a CIS to improve record keeping matters such as prescription legibility, and to provide research opportunities. Most importantly however, he was keen that the CIS should support clinical practice and encourage adherence to guidelines. Unlike many situations in which the focus is exclusively on the capabilities of database system to be a data repository [13], this ICU wanted to use the CIS as a clinical tool to improve practice. Hence there was significant concern that the CIS should match the workflow of the ICU rather than be an additional chore for the healthcare practitioners. The clinical leadership of the unit therefore selected the most highly practitioner-customizable system on the market, despite the expense and the considerable effort needed to do the customization, because it offered the greatest control over continuing to design and reflect upon the ICU's practices.

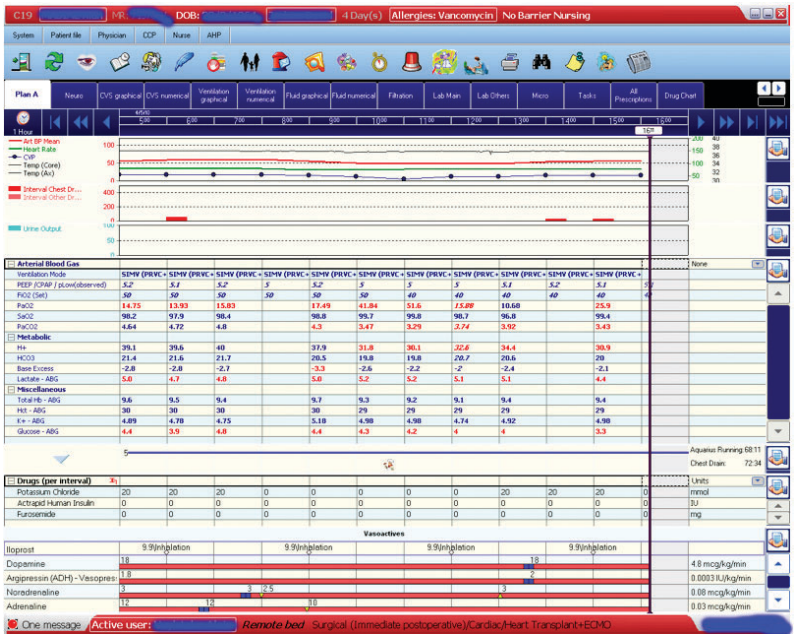

Figure 1. CIS main screen.

Metavision ICU[1], specialist software for intensive care, allows customization of every aspect of the system with which a user interacts. All data fields are created by the customizers and can either be linked to a field in the database or be an aggregate of other fields. Fields can be arranged and visualized on the screen in a range of ways depending on the customizers' choosing. Any number of tabs to display information can be created. This ICU has 15, of which the ward round tab is shown in Figure 1. Forms that allow healthcare practitioners to type data in or verify data sent from other devices can also be created. These are usually specific to a medical discipline with nurses having 25 but only 5 or 6 that they use regularly. An example of a form into which a nurse enters data can be seen in Figure 2.

The final two aspects of the system that can be customized are reports that extract information from the system and 'events' which cause something, such as a reminder, to appear when certain activities have or have not taken place. The customization tools, exemplified in Figure 3, provide an integrated environment for creating data fields, the

[1]http://www.imd-soft.com/

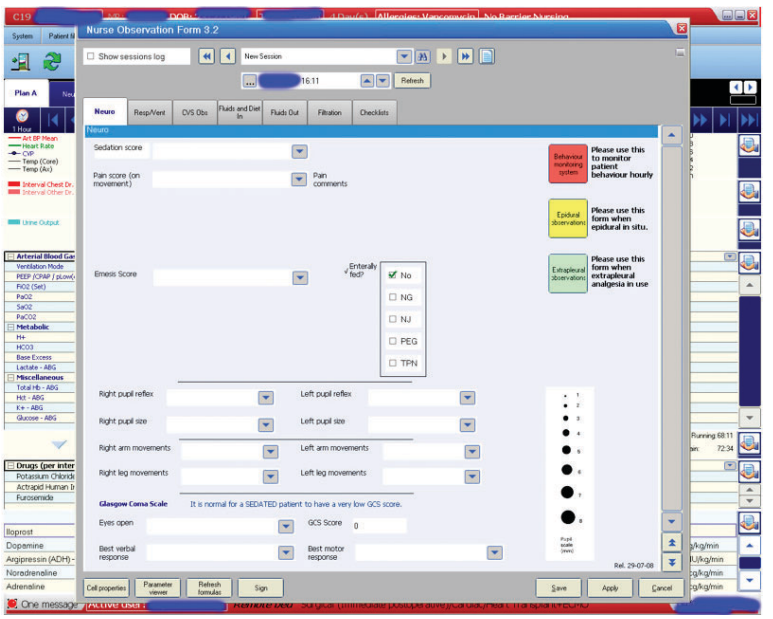

Figure 2. Example of a nursing form for entering data.

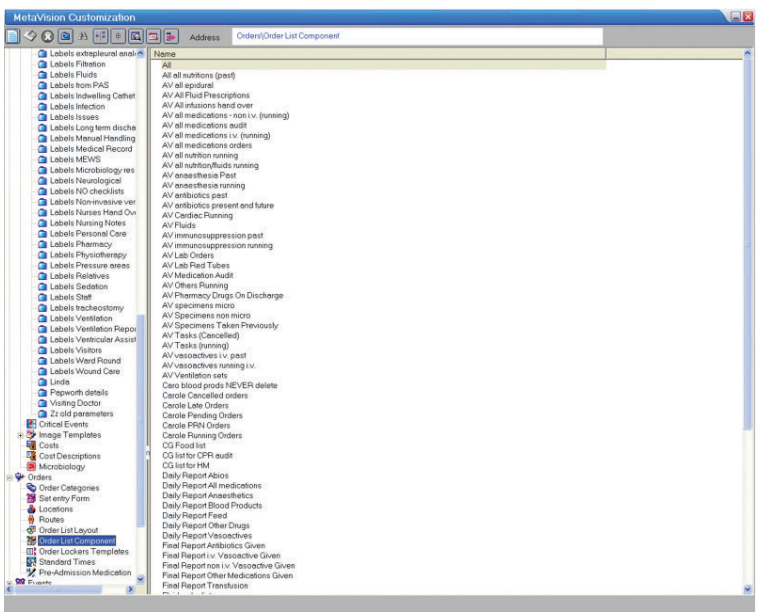

Figure 3. Metavision Customization Suite.

interface layout and forms. Separate tools support writing SQL queries to the database and building reports. These tools are loosely integrated, but also allow cutting and pasting between them.

4. CASE STUDY

The case study is broken into three sections detailing the customization tools, the customization practices and the organizational issues present. These themes are separated to provide structure for the case study, but then merged in the discussion section to examine how the findings of each theme influence the research and development of practitioner-customizable CISs.

4.1. Customization Tools

A practitioner-customizable CIS facilitates healthcare practitioners carrying out a number of activities normally done by software engineers. However, as healthcare practitioners are not trained as software engineers, there is ambiguity around assumptions of a customizer's needs and knowledge that determines the design of the customization tools. This section looks at the needs and knowledge of the customizer in this study as compared to software engineers.

4.1.1. Healthcare Practitioners as Software Engineers

As the customizer explained the customization tools, she related a number of recent projects that revealed the range of responsibility her job entails. One of the most common jobs she does is to create new forms for entering data into the CIS. The steps that she goes through to do this are as follows: she elicits the requirements from the relevant people and then considers what data fields are needed and how they relate to existing ones in the database. She then designs and builds the form in accordance with the requirements. Frequently, she has to debug the form, but once complete, she integrates it into the CIS. This description of her job demonstrates that customization goes beyond manipulating the software code to doing all the tasks of a software engineer from gathering requirements to testing and deploying the result.

Despite the need for the customizer to do a range of software engineering jobs, the customization tools only support the customizer in creating software code. This leaves the customizer to carry out the rest of her work as best as possible without any support. For example, the form development environment is 2/3 the size of the screens, which as the customizer pointed out, could lead to creating forms of a length in which people have to scroll – increasing the likelihood that data is missed. Furthermore, there is neither built-in version control nor a testing environment to ensure that more complicated customizations, such as events, are working properly before they are put into use. Although work-arounds can be found to address these problems in the ICU context, all professional software development environments would have those features built-in.

There are other situations in which software engineering tools are absolutely necessary, but missing. One such situation is the integration of new forms into the CIS while live. The customizer related the steps that she takes to do this. She goes into the

ICU and instructs the head nurse to tell all of the nurses not to use a particular form until told to do so. She would then search for all the buttons that led to the form and change them to point to the new form. Nurses would then be told to use the form again. This process is very stressful because the customizer has no way to verify that all of the buttons that go to a certain form have been changed, and similar looking buttons that go to other forms have not. The lack of appropriate tools to carry out updates encourages error.

We would suggest that the reason the product developers did not include these tools was that they did not perceive healthcare practitioners as doing software engineering, but rather just a bit of customization. However, the above examples illustrate that this is not the case and that a whole suite of tools beyond ones that help manipulate the code are needed. Not only would this make the job of the customizer easier, but would reduce risks, a significant part of software engineering. There are two significant areas of risk in these scenarios: human error in a safety critical industry and the concentration of the knowledge of the system (e.g., the data dictionary, software architecture) in a single person's head. Appropriate tools from software engineering [14] and design engineering [15] can reduce these risks substantially.

There were a number of tools lacking in this particular study, including: (1) Realistic Development Environment, (2) Appropriate Testing Environment, (3) Version Control, (4) Software Architecture Visualisation Tool, (5) Planned Automated Update, and (6) Data Dictionary. Tools for a specific product may vary, but we would encourage software engineers building practitioner-customizable systems in the future to consider the tools that they themselves would require if they were doing the same development work as the healthcare practitioners.

4.1.2. Healthcare Practitioners are not Programmers

The previous section illustrated that the types of tools that a customizer needs are largely equivalent to those used by an engineer overseeing a software project. However, the design of many of these tools assumes knowledge of the structure of the software, something a programmer would have, but a healthcare practitioner would not. A particular concern is a likely lack of awareness about hidden dependencies in software, as seen in the following example. The customizer can construct a database query in the Query Wizard (one of the customization tools) and then insert it into a report either by cutting and pasting it or by linking it. If she chooses the latter, and somebody, either herself or another customizer, changes the query in the Query Wizard, it would change in the report without notice. The former option did not have this behaviour.

Linking is likely to reduce 'repetition viscosity' – the effort involved in making future changes [16], and be useful for programmers managing large and complex systems. In a customization setting, the number of people using the customization tools and their lack of understanding of hidden dependencies however, can make it dangerously possible that someone accidently changes the query in the Query Wizard. This could then result in misinformation being calculated for a report and acted upon with significant human consequence. There are other examples that have less dramatic consequences, such as the recent feature that allows forms to share pages. Once the

pages have been shared, un-sharing them is very difficult, an outcome only discoverable by trial and error.

In knowledge of software, healthcare practitioners are not comparable to programmers. Steps should be taken to adjust. It is not clear whether designing out hidden dependencies or warning people of their presence is more useful. The optimal approach is likely to depend on the consequences of a hidden dependency going unnoticed. In the first example, the consequence could be human life, while in the second, an extra day's work. It would be useful to conduct a risk assessment of each feature of a customization program, employing standard risk assessment tools [17, 18] that would normally be used for software to be deployed in a critical environment. Normally these tools account for environment disruptions; one of those disruptions should be the knowledge of the healthcare practitioners customizing the system.

4.1.3. Healthcare Practitioners as Programmers in Training

The previous two sub-sections focused on the required tools and their top-level design. This section addresses the implementation work. Using appropriate language to name variables and classes to allow others to guess the meaning is taught in most programming text books. Nonetheless, there are still programmers who do not follow this practice [19]. This can cause problems for non-programmers when they look at the code. For example, the customizer discovered that if she changed the 1 to a 0 in the statement 'if x = y is 1, then...', she could use it in new contexts for a different purpose. Although experimentation is to be encouraged [20], the ambiguity of this statement, which should read: 'if x = y is true,' could lead to inadvertent errors if 1 and 0 have different meanings in other contexts.

A further problem that language can cause, if not chosen carefully, is miscommunication between customizers and professional programmers. In the product-supplier's customization tools, any medical data field is called a 'parameter'. It took some time for the programmer in our study to realise what the customizer meant by the word 'parameter'. His first expectation was that this word referred to the generic programming meaning, in which the term is used to mean a value passed to a function or subroutine. Realising that this was not the case, he attempted to map the word to other conventional programming concepts. Because it did not map directly, this meant that he had to understand the non-standard terminology of the customization tools before he could communicate effectively with the customizer.

If language is used more precisely, it could be a vehicle to build the skill of the customizer. Some technical language used by the programmer, such as the word 'environment', had no meaning for the customizer, but those words built into the customization tools, such as 'event', represented concepts that she could speak about fluently and with confidence. In these latter contexts, communication with the programmer was enhanced. It served the purpose of being a shared vocabulary around which to discuss both the requirements and problems of the customization tools, but perhaps more importantly, it was a way in which the programmer could recognise the contribution of what the customizer was saying in the terms of his technical 'object world' [21].

In the implementation of customization tools, it is useful to think of the healthcare practitioners who do the customization as programmers-in-training. Following good programming practice, such as using suitable variables and class names, not only encourages good practice among novice customizers through example, but allows for experimentation which increases their skills. Likewise, not re-using terms to name high level concepts that already have an exact low level meaning has a number of benefits. It helps customizers map their understanding of the customization environment onto other software tools not built specifically for customization. It also supports communication between customizers and programmers, promoting conversations that are likely to develop a customizer's conception of software and how to design and manage it.

4.2. Customization Practices

As pointed out in the previous section, customizers do much of the work of software engineers. However, their motivations for doing that work are quite different from programmers [22]. This section discusses three pertinent differences in perspective between the healthcare practitioner and the programmer that came through in our data.

4.2.1. A Job to Do

In the first interaction between the customizer and the programmer, the customizer presented the design of the report that she wanted made. Although the customizer presented a clear and well illustrated list of current requirements, the programmer immediately questioned future needs, because he wanted to ensure that she would learn how to customize the system without his assistance in future. He asked, for example, whether it would be necessary ever to draw data from two different fields in the database into a single field in the report (as would be the case if a data value was compared against a maximum value). The conversation was fraught with tension as the programmer steered the conversation towards possible future needs and the customizer drew it back to what was necessary now. This tension has been referred to as the Paradox of the Active User [23].

The Paradox of the Active User can be described in terms of the decisions that individuals make about allocating their attention [24]. In many contexts of computer usage, it is possible to learn more, or to customize the computer in some way, to make one's future work more efficient. But an active user often loses this opportunity, by focusing on the task at hand more than on future efficiency savings. A customizer, as an end-user of a tool, always has a specific goal to achieve, like the example of this case study in which a report needs to be designed. A programmer however, is trained to think through all possibilities so that design choices are made to limit the difficulty of making changes in the future. However, this can make projects take longer and lose focus. There is a definite tension between taking the short term view of solving the current problem and taking the long term view of solving problems that may appear in the future as well.

4.2.2. A Playground

The customizer clearly marks the boundaries of her work. She refers to anything that is not in the user manual as the backend. This includes any programming scripts in Visual Basic or SQL that could be used to access the database outside of the customization tools. She also sees a separation between problems that she needs to figure out and can do by better understanding the customization tools and those that need to be solved by the product-supplier. This divided view differs from the standard in programming, as exemplified by the programmer who wanted to examine and have control over the whole system rather than work within the bounds of the customization tools. One advantage of the separation between the customization tools and the backend was the confidence that it gave the customizer in 'tinkering.'

Tinkering is a concept thought to promote learning in end-user programming contexts [13]. For example, the customizer described an incident that taught her how to write event statements. These are logical statements that trigger certain behaviour in the system and in their simplest form look like: "if x = y is 1 then …" A number of these were done by the product-supplier's programmers during the initial customization, but the customizer was able to change parts of the statement to use it for other purposes. She could do this, because she knew that all of the system variables were locked and that she would not cause any harm to the system. As many customizations could be easily tested within the customization environment, she could use it as a playground to try out new possibilities.

4.2.3. And a Logical Mind

The customizer also distinguished between programming and logic. Several times she described her mind, saying "It's a very logical mind but I do not think it's a very programming one." Although she could not articulate what a programming mind was, we can draw conclusions about what she meant from the data. The customizer felt that SQL statements (e.g., SELECT * FROM patient WHERE diagnosis = heart failure) made a lot of sense, because although she did not know the terminology, she could guess enough of the meaning to experiment with the software code. However, other parts of the software code such as types (e.g., "dim" in Visual Basic) or software code that supports object orientation gave little scope for guessing and confused the customizer. For example, she made the incorrect assumption that changing the arrangement of definitions in the software code would change the order in which code was executed.

We would suggest that her understanding of a logical mind was a mind that could reason about the possibilities presented to her. In other words, through educated guessing and testing, she could figure out how to manipulate parts of the program. This proposition is supported by the customizer's enthusiastic reaction to debugging tools that the programmer introduced. Previously, she had no way to figure out what the software code was doing, having no formal instruction in object-oriented software. With the debugger, she could watch the order in which the software code executed and start to understand how it worked. This indicates that liveness, a concept proposed by

Tanimoto [25], which supports non software-engineers by providing immediate semantic feedback that helps them see how a program is functioning, is an important feature of software to support a logical mind without training in programming.

4.3. Organizational Issues

Organizations have incentives to develop products in certain ways. This section looks at those influences and their ramifications, considering the base design of the system and the customizer's relationship to the product-supplier.

4.3.1. System Design

A customizable system allows many elements of the interface and the reporting to be changed. System design however, will inevitably restrict those possibilities in some way. One example of such a limit is that the customizer cannot create a query on the currently displayed patient in the 'query wizard' end-user programming facility of the product. This is one of the most common needs that the ICU has, but the way to do it is not exposed in the end-user customization tools. The alternative of having the healthcare practitioner choose from a list of patients is considered risky with severe consequences if the wrong patient is chosen accidently. Limitations of this system in regards to the encoding of time in the system have also been noted [3]. Some limitations come from system design as the above examples, but others are likely motivated by the perceived ability of the healthcare practitioners.

One of the basic problems with the implementation of the new report function on which this study was focused was the inability from within the 'query wizard' facility to define the layout of the resulting columns so that data could be presented in a manner consistent with the clinical work. In programming terms, the customizer needed to define a declarative constraint on the column width. However, defining page layout by programming constraint relationships is challenging for non-programmers, as can be seen in the many web layout tools that provide only static absolute position layout rather than definition of dynamic constraints. This stance is typical of many end-user programming problems in which aspects of the system behaviour are fixed to directly-manipulable concrete values rather than abstract specifications that might allow more flexibility, but at the expense of additional cognitive complexity. Designing usable constraint programming languages is an active research concern [26].

4.3.2. Product-Supplier Update Lock-in

It is not clear however, whether building a user-centred customizable product is in the interest of the product-supplier. User research is expensive and often puts forward requirements which are more difficult to achieve technically. A company therefore needs incentive to follow this path. In many cases, the incentive is competition for a better product. However, in this situation there are a number of contrary incentives. The more easily customizable the product, the greater range of possibilities the company needs to support when upgrading the system. The product-supplier in this study recently limited their support to customizations done within their customization tools and not

those done directly in Visual Basic, as they found supporting the yearly updates of keen customizers very resource-consuming.

Backward compatibility of all Visual Basic functionality needs to be maintained for example, because customizers are free to use the language in any way that they like, including many ways that would be hard for the language authors to anticipate. If the behaviour of any language feature changes, it is likely that programs written using the language will stop working as intended. A further difficulty is that of service contracts. If the software is so well designed that it requires no further enhancement and upgrading, the company may lose service contracts. Many companies in the software industry derive a large part of their revenue from service contracts, 'maintenance' licences and consultancy sales to customers who have previously purchased their product.

We observed an interesting tension in this case study between the product-supplier and the freelancer. The product-supplier did not want to give out detailed documentation of the functions for calling the database, and the professional programmer criticised the company for their lack of transparency. Yet, he also encouraged the customizer to imagine more than what was possible in this small project and to re-hire him, once he had gained fluency in this particular piece of software, to do more work. The customizer was exasperated with both parties, although particularly the product-supplier, because she felt that within the budget restrictions of a public health provider, one of the benefits of a customizable system was that it could be adapted within the limited budget of the ICU. Ironically, the software professionals (both product-supplier and freelancer) wanted to protect their knowledge and found this to be in conflict with true practitioner-customisability.

5. DISCUSSION

5.1. Summary of Results

The present case study suggests that healthcare practitioners require more customization tools that help them carry out the software engineering tasks required of them, and not just support for programming. These results encourage a distinction between the technical skills of the programmer, not all of which are appropriate to practitioner-customizers, and the organizational practices of software engineering that support the creation of robust and usable software. Considering this distinction, customization tools need to be adapted to account for healthcare practitioners' lack of knowledge of software structure on a technical level, but still embody good software engineering practice at the organizational level to further the training of healthcare practitioners in appropriate software engineering practices.

Below we will explore how these separate findings can be merged to direct further research and development of practitioner-customizable CISs. We also draw on our regular participation and observation of the biannual regional user group for this software. Spanning 25 hospitals in the UK and Ireland, we have used the meetings of this group to ensure that the findings we present are not specific to the customizer whom our study is based on, but applicable to the broader community of customizers.

5.2. Implications for Design

5.2.1. End-User Software Engineering

The research community dedicated to the development of end-user customization tools has advocated End-User Software Engineering as opposed to End-User Programming as a basis for designing tools. They argue that the customizer's focus on the task at hand can discourage the customizer from carrying out important related tasks, such as proper requirements gathering or code-testing, that may not seem directly relevant [27]. The first step for suppliers of customizable CISs to take in rectifying this imbalance would be to provide appropriate software engineering tools. However, this step may not be enough to encourage the usage of tools that are understood to be valuable in software engineering, while not at all familiar to healthcare practitioners.

Wilson et al. [28] proposed the more directed approach of using a Surprise, Explain, Reward strategy to increase the likelihood of users testing their software code by drawing attention to what could be tested and then helping users do it. Our data suggest that end-users in healthcare are less likely to take risks with untested software than casual users of spreadsheets. We have seen examples across hospitals of testing of forms when they first go live, as well as attention from those using the forms to ensure their correctness. Indeed, many nursing staff in this ICU continued to calculate fluid balance manually for several months after the CIS was introduced, to ensure that the formulas were working properly. Nevertheless, there are many other aspects of software development that could be supported through behavioral interventions of this kind, particularly those that require some skill, such as interface design. How tools might help healthcare practitioners in all aspects of the software cycle is an open research question.

Some of those tools may be software development tools, but others may be tools that support design thinking. Despite the lack of tools to assist the customizer in her many roles, no severe incidents have been recorded that can be attributed to the customization of the CIS. We would suggest this is because of a strong design culture in the unit, as well as the skilled professional judgment of the many people involved with customizing the CIS. This was manifest in the ICU's previous experience in designing paper forms and encouraging multi-disciplinary input, as detailed in the background section. It was also guided by the product-supplier, which encouraged good requirements capture. Moreover, we saw a number of examples of the customizer devising her own tools, such as improvised version control and a testbed for newly developed forms. As this kind of design-orientation cannot be assured in every ICU, development of tools that support healthcare practitioners to carry out these tasks is an open area of research.

A final important difference between a software engineer's work and that of a healthcare practitioner is the attitude towards abstract planning. Software engineers try to foresee problems and likely future desires, while customizers focus on the task at hand. Here again the distinction between technical and organizational practices of software engineering comes into play. It is perhaps less necessary for a customizer to plan about future changes to the software code, because she always has ready access to it; however, planning against risk through designing thinking remains important. How to promote one type of planning while not stifling project completion and

experimentation is not clear, nor is the boundary between the two. This remains an open research question.

5.2.2. Experimentation

Experimentation helps customizers increase their skills. Both the customizer in this study and others in the user group have frequently commented on what they have figured out or happened upon through experimentation and how this has led to new possibilities for data usage in their ICUs. As mentioned in the case study, familiarity with technical conventions can support educated guessing and can also support good communication with professional software engineers as a way to increase skill. We also noted that tool characteristics such as Liveness [25] could provide a way for customizers to understand how programs execute. This approach could also be applied more broadly to make parts of how the customization tools link more transparent and guard against hidden dependencies. What such a tool would look like is another open research area.

In order for any experimentation to happen at all, regardless of how it is encouraged, the customizer must know that she cannot inadvertently cause the system to malfunction. There are a number of ways to do this from technical solutions to testing beds and version controls. These are essential to the success of any method for a program to promote experimentation.

5.2.3. The Software "Supply-Chain"

The previous sub-sections suggest four areas of research that could contribute to further development of practitioner-customizable CIS. However, an important parallel question is who develops such systems and whether they have the motivation to increase the ability of customizers. The case study suggests that at present companies may be insufficiently motivated to carry out user-centered design or to build tools that actively persuade customizers to experiment. The government, particularly in the UK (as demonstrated by the significant public resources spent on the National Program for Information Technology [NPfIT]) may also have difficulty supervising such systems, and may prefer to award long-term service contracts. It seems that for healthcare practitioners, the best way to gain from a customizable system is to ensure that the contract encourages customization.

6. CONCLUSION

This case study was conducted to provide guidance to software engineers, researchers, and practitioners on avenues for further research and development of practitioner-customizable CISs. The findings are summarised below:

6.1. Findings for Clinical Software Developers

- Customization tools should support every aspect of the software engineering cycle from requirements gathering to the integration of new elements.
- Good software engineering should be employed throughout the software, and particularly when the software code is visible.

- Hidden dependencies should be assessed, and either designed out or have appropriate warnings.
- Experimentation should be encouraged through a fail-safe environment and tools to support testing and version control.
- The most recent findings on encouraging good practice and experimentation should be included in the software.

6.2. Findings of Issues for Researchers

- How might tools help healthcare practitioners throughout the software cycle, beyond programming?
- How can tools be adapted from software engineering or design engineering to promote good design practice?
- How much, and what type of, planning should healthcare practitioners being encouraged to do?
- How can the notion of liveness be applied to the entire customization tool and not just to the programming environment?
- Are there other ways to prompt experimentation or upskill the healthcare practitioner with software engineering know-how?
- Are there alternative business models that would better support the creation of practitioner-customizable systems?

6.3. Findings for Healthcare Practitioners

- Build design capacity within your ICU.
- Encourage experimentation by customizers.
- Negotiate contracts for customizable systems with care.

ACKNOWLEDGEMENTS

We would like to acknowledge those who took time from their clinical work to help this study happen. We would also like to acknowledge the comments of Professor John Clarkson. This research was funded in part by Boeing and done in association with the Cambridgeshire and Peterborough CLAHRC.

REFERENCES

[1] Sellen, A. J. and Harper, R. H. R., *The Myth of the Paperless Office*, MIT Press, Boston, 2002.

[2] Martin, D., Procter, R., Mariani, J., & Rouncefield, M., 'Working the Contract'. *Proceedings of OzCHI*, 2007, pp. 241–248.

[3] Morrison, C. and Blackwell, A.F., Observing end-user customisation of electronic patient records, in: V. Pipek, M.-B. Rosson, B. de Ruyter and V. Wulf (Eds). *Proc. 2nd International Symposium on End-User Development, IS-EUD'09*. Springer Verlag (Lecture Notes in Computer Science - LNCS 5435), 2009, pp. 275–284.

[4] Metavision ICU. *http://www.imd-soft.com/metavision-for-icus*

[5] PatientOS – an open source health information system. *http://www.patientos.org/*

[6] OpenEHR. *http://www.openehr.org/home.html*

[7] Lieberman, H., Paterno, F., & Wulf, V. (Eds.) *End User Development - Empowering people to flexibly employ advanced information and communication technology*, Kluwer Publishers, Dordrecht, The Netherlands, 2006.

[8] Morrison, C., Jones, M., Blackwell, A. and Vuylsteke, A., Electronic patient record use during ward rounds: a qualitative study of interaction between medical staff. *Critical Care*, 2008, 12, R148.

[9] Johnson, J. C., and Weller, S., C., Elicitation Techniques for Interviewing, In J. F. Gubrium, J. A. Holstein (eds.) *Handbook of Interview Research: Context and Method*, 2001, pp 491–514. London. Sage.

[10] Miyake, N. Constructive interaction and the iterative process of understanding. *Cognitive Science, 1986*, 10, pp. 151–177.

[11] Kahler, H., Muller, M., Kensing, F., Methods & tools: constructive interaction and collaborative work: introducing a method for testing collaborative systems. *Interactions*, 2000, 7, pp. 27–34.

[12] Manias, E. and Street, A., Nurse–doctor interactions during critical care ward rounds. *J Clin Nurs*, 2001, 10, pp. 442–450.

[13] Scott, J.T., Rundall, T.G., Vogt, T.M., and Hsu, J., Kaiser Permanente's experience of implementing an electronic record: a qualitative study, *BMJ*, 2005, 331, pp. 1313–1316.

[14] Pressman, R.S., *Software engineering*, McGraw-Hill, Columbus, 2008.

[15] Pahl, G., Beitz, W., Feldhusen, J. and Grote, K.H., *Engineering Design: a systematic approach*, 3rd edition, Springer, London, 2007.

[16] Green, T.R.G. The cognitive dimension of viscosity: a sticky problem for HCI, in: D. Diaper, D. Gilmore, G. Cockton and B. Shackel (Eds.) *Human-Computer Interaction-INTERACT' 90*. Elsevier, Amsterdam, 1990.

[17] Habraken, M.M., Van der Schaaf, T.W., Leistikow, I.P. and Reijnders-Thijssen, P.M., Risk management: Prospective risk analysis of health care processes: a systematic evaluation of the use of HFMEA in Dutch health care, *Ergonomics*, 2009, Jul;52(7), pp. 809–19.

[18] Ericson, C. A. II, *Hazard Analysis Techniques for System Safety*, John Wiley & Sons, New Jersey, 2005.

[19] Blackwell, A.F., Church, L. and Green, T.R.G., The abstract is 'an enemy': Alternative perspectives to computational thinking, in: *Proceedings PPIG'08, 20th annual workshop of the Psychology of Programming Interest Group*, 2008, pp. 34–43.

[20] Beckwith, L., Kissinger, C., Burnett, B., Wiedenbeck, S., Lawrance, J., Blackwell, A. and Cook, C., Tinkering and gender in end-user programmers' debugging, in: *Proceedings of CHI*, 2006, pp. 231–240.

[21] Bucciarelli, L.L., *Designing Engineers*, MIT Press, Boston, 2006.

[22] Burnett, M., Cook, C., and Rothermel, G., End-User Software Engineering, *Communications of the ACM* 47(9), 2004, pp. 53–58.

[23] Carroll, J.M. and Rosson, M.B., Paradox of the active user, in, J.M. Carroll (Ed.), *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, Bradford Books/MIT Press, 1987, pp. 80–111.

[24] Blackwell, A.F. and Burnett, M., Applying Attention Investment to end-user programming, in: *Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments*, 2002, pp. 28–30.

[25] Tanimoto, S., VIVA: A Visual Language for Image Processing, *Journal of Visual Languages and Computing* 1(2), 1990, pp. 127–139.

[26] Bessière, C., Utiliser les contraintes sans rien y comprendre. *Presentation at Reconnaissance des Formes et Intelligence Artificielle RFIA 2008*. http://www.mis.upicardie.fr/rfia2008/index.php?id_page=20.

[27] Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Lawrence, J., Lieberman, H., Myers, B., Rosson, M.B., Rothermel, G., Scaffidi, C., Shaw, M., and Wiedenbeck, S., The State of the Art in End-User Software Engineering. Accepted for publication in *ACM Computing Surveys*.

[28] Wilson, A., Burnett, M., Beckwith, L., Granatir, O., Casburn, L., Cook, C., Durham, M. and Rothermel, G., Harnessing curiosity to increase correctness in end-user programming, in: *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2003.