18th Annual Health Informatics Conference, HIC 2010
D.P. Hansen, L.K. Schaper & D. Rowlands (Eds.)

On the maintainability of openEHR based health information systems – an evaluation study in endoscopy

Koray Atalag, Hong Yul Yang, Jim Warren
Department of Computer Science, the University of Auckland, New Zealand

Abstract.
Maintaining health information systems over time requires a considerable amount of effort and time. This is especially marked in systems involving structured clinical data such as electronic medical records. Inevitably most of software is then dictated by volatile healthcare concepts and processes. Literature indicates that maintenance tasks alone may usually constitute 70-80% of the whole software cost. It has been suggested that openEHR based systems will effectively tackle this by separating domain knowledge from software code. The objective of this paper is to investigate whether it is easier to introduce changes to an openEHR based clinical application. An endoscopy reporting application driven by openEHR archetypes has been implemented in .Net/C#. It has the same functionality and appearance as the existing application which has been developed using traditional methods. Afterwards a number of change requests have been implemented in both systems while recording the development effort using ISO/IEC 9216 series standard software maintainability metrics. This paper presents the implementation methodology and preliminary results of the larger evaluation study employing formal software measurement techniques. On average, the openEHR based application took approximately nine times less effort to implement the change requests. This is the first quantitative comparative study in the literature and has the potential to affect how we develop health information systems in future.

Introduction

Modifications in software become inevitable after the product has been deployed which may include corrections, improvements or adaptation to changes in environment, requirements and functional specifications. Maintenance comprises activities needed to carry out these tasks and is a major part of the software lifecycle. Software maintainability, on the other hand, is a software quality attribute which is defined as the capability of the software product to be modified (ISO/IEC 9126-1: 2001).

ISO 9126 present a comprehensive quality model which comprises three distinct aspects of quality: internal, external and in-use. Internal quality refers to the characteristics of software from an internal view and is attributable to requirements, design and implementation artefacts such as code. External quality refers to the characteristics from an external view when software is executed, which is typically measured and evaluated while testing in a simulated environment. Quality in use is the user's view of the quality of software. This study evaluates maintainability as an external quality characteristic.

A typical software development project starts with the requirements collection phase where developers interact with domain experts and users to identify and document them. Then comes the design phase where the blueprint of software is laid out based on the elicited requirements. There is a *handover* in traditional development methodologies where domain knowledge is extracted from the problem domain and then transferred to the solution domain by means of formal models and programming constructs. Development then continues with coding, testing, verification and validation, deployment followed by the maintenance phase. The essential difficulty in maintenance arises from the fact that if requirements change or new requirements are introduced then the whole development cycle has to be repeated again – from redesign to redeployment. It is a well known fact that maintenance usually exceeds initial development costs (around 70-80% of total cost) (Sommerville 2000).

Classic software development approaches, typically the Object Oriented formalism together with relational databases, work well in domains where domain concepts are stable and that most of the requirements can be elicited at the outset. However, healthcare is a "wicked" domain – that is the size and complexity of Medicine negatively affects this process (Wicked Problem n.d.). Moreover, the rapidly changing body of knowledge plus the non-deterministic nature of Medicine, that is the *Art of Medicine*, adds more to the problem. Not only is the body of knowledge highly variable but also medical conduct changes from time to time and place to place across different organisations and jurisdictions (Rector 1999; Beale 2005; Paech and Wetter 2008). In a typical development project clinicians, without much idea about the technologic limitations or possibilities, express their needs. IT professionals on the other hand, without much background in biomedical and clinical sciences, try to comprehend these requirements. This has fundamentally two consequences: 1) Not all requirements can be elicited in the first place or they may be wrong resulting from the cognitive limitations of *handover* between healthcare and technical professionals, 2) Elicited requirements then frequently need to be altered or more likely new ones come into play. Thus is it only natural to expect more effort and cost

CONTENTS
PRINT

associated with maintenance, especially in clinical applications. Solid evidence is scarce in literature but Girosi et al. reports that most long term costs associated with electronic medical record (EMR) systems are due to maintenance. This obviously creates a very large room for improvement and that tackling this will certainly impact on how we will develop systems in future. This is the main motivation of this study.

Traditionally, both technical and domain knowledge expressed as software requirements are hard-coded into the program code, database schema and user interfaces. While the former type of knowledge is mostly static, the latter is highly volatile and often causes modifications in software. Thus it is a compelling idea to separate domain knowledge from technical environment, embed in a formal model and then develop software accordingly. This has been shown to improve software maintainability (Cao, Ramesh and Rossi 2009).

openEHR has developed a methodology which here we will denote "the Multi-level Modelling and Development Methodology (MLM/D)". This can potentially minimise, if not totally eliminate, the *handover* paradigm (hence dealing with incomplete or wrong requirements). It does this by separating domain concepts from software code using domain specific models called Archetypes and a set of stable technical reference models (RM). Archetypes formally constrain RM entities to build clinically valid domain concepts such as blood pressure measurement. In the runtime software is driven by these models for dynamic graphical user interface (GUI) creation and data validation (Beale 2002). As a result altering software after deployment mostly involves remodelling done by domain experts.

A clinical reporting and analysis application in the field of gastrointestinal endoscopy has been formerly developed by the primary author for use in a real clinical setting circa 1999. Most of the functional specifications were defined by the Minimal Standard Terminology for Digestive Endoscopy (MST) which formally prescribes the structure, well defined terms and rich semantics in this domain (Delvaux 2000). During development, this domain knowledge has been embedded into the software. Object-Procedural programming has been employed using Microsoft Visual Basic 6 together with Microsoft Access for data modelling and persistence. After deployment it became apparent that modifying the application to meet new requirements was quite cumbersome. This has been the starting point of the quest for finding a better means to build future-proof clinical applications. In an earlier study this application has served as a research prototype (GST) where the objective was to validate that the openEHR MLM/D can faithfully represent the problem domain (Atalag 2009).

The aim of this paper is to present the initial results of the larger evaluation study, and also to unravel a generic implementation path for clinical applications using openEHR and .Net/C#.

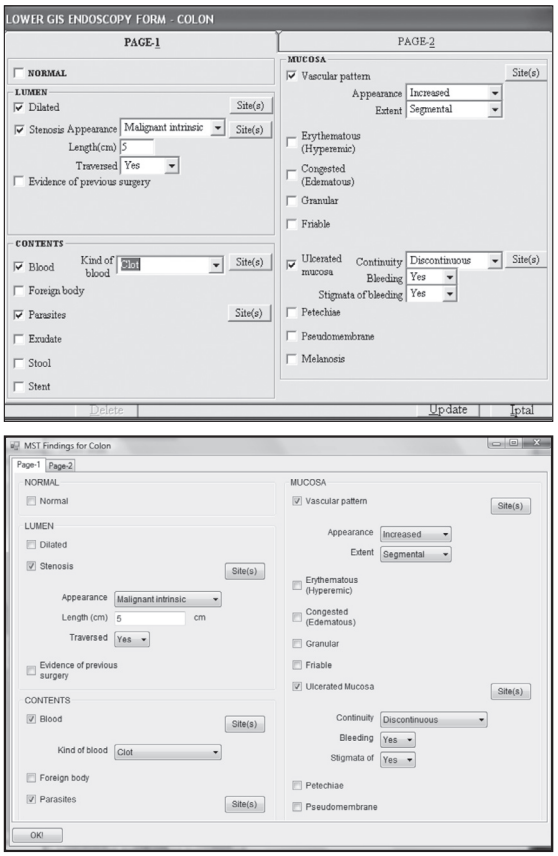

Figure 1. GST and GastrOS GUI for MST Findings for Colon

Methodology

The study comprises: 1) Defining software requirements for GastrOS with reference to GST by means of preparing a formal Software Requirements Specification document (SRS); 2) Archetype modelling based on MST; 3) Design, implementation and testing of GastrOS; 4) Introducing change requests (CR) and implementing them; 5)Measuring maintainability using external metrics, and evaluation.

The open source openEHR Archetype Editor was used to model archetypes based on MST. Then an openEHR template representing the full endoscopy report was modelled using Ocean Template Designer. This is the sink of all domain knowledge bringing together all MST archetypes. Then three operational templates were automatically generated representing each examination type. To explicitly define the scope and functionality of GastrOS the SRS document was prepared. We have excluded parts such as patient administration, detailed search and reporting.

GastrOS consists of two distinct sub-systems: 1) openEHR based **Structured Data Entry (SDE) Component** which provides all data entry and validation functionality driven by underlying MST Archetypes; and 2) a **Wrapper Application** to provide a minimum level of functionality to drive the SDE component (performing functions like patient and visit data entry and sign-off/reporting). Both components are designated to be free and open source.

The SDE component (hereafter referred to simply as 'SDE') is a programming library that takes in an operational template in XML form as input and dynamically constructs an appropriate graphical data entry form. It has the capability to store and validate user-entered data. Thus if a clinician wanted modifications in the form, he or she needs only to change the model and then SDE would automatically generate an updated form that reflects these changes. This component is also targeted to be reusable across clinical domains – in applications that require hierarchical data entry and validation.

To make this work, SDE first parses the input operational template into a tree-like data structure, called archetype objects. Each archetype object acts as a blueprint for a specific part of the data to be entered and stored, as well as the GUI widget to represent the data (Figure 1). It can be as atomic as a single textual entry or as complex as an entire group of findings. SDE defines a set of mapping rules to determine what kind of GUI widget to create for what kinds of data elements. For example it would create a text field for a textual entry (i.e. name of a drug), a drop-down list for a restricted range of values (i.e. organ types), or a panel for a cluster value that further contains sub-values (i.e. a diagnosis entry). These rules are fairly generic so as to accommodate as wide a range of usage domains as possible. The consequence of this is that without any external customisation, the aesthetics and visual behaviour of the GUI generated by SDE would be uniform across different usage domains. For this reason we introduced what we call 'GUI directives', which allows the user to encode additional instructions to customise the appearance and behaviour of the generated GUI widgets. An example of a simple directive would be to put a border around a panel representing a specific cluster; a complex example would be adding the option of dynamically showing and hiding a group of values. Because the Archetypes are not designed to hold presentation related information, we have chosen to embed the GUI directives as template annotations to feed into the GUI generator. Interestingly, all the directives we used turned out to be generic enough to be applicable to any other clinical domain.

The change requests used in this study for evaluation comprise real cases which caused GST to be modified. These are mostly due to errors detected in MST and local extensions (Table 1). Each CR has been assigned as a maintenance task to both GST and GastrOS. Then the primary author performed necessary programming and testing tasks on GST who is the original developer, and similarly the second author performed these tasks on GastrOS while the primary author, who is also a domain expert, made necessary changes in the models.

The following ISO 9126 maintainability external quality metrics related with changeability have been selected:

Change cycle efficiency (CCE) is used to answer the question: Can the user's problem be solved to his satisfaction within an acceptable time scale? It is computed by measuring the time from initial request to resolution of the problem.

Modification complexity (MC) is used to answer the question: Can the maintainer easily change the software to resolve problem? It gives a ratio that is computed by sum of time spent on each change per size of software change divided by total number of changes.

The measurements were done while modifications were being made to both applications. A Subversion repository (SVN) and an issue tracking system (Atlassian Jira) have been setup to document and help with data collection.

Table 1. The change requests (CR) and measurement results used in the study

No	Type	Description	Time (min)		Changed LOC	
			GST	GastrOS	GST	GastrOS
1	Fix	MST: Anatomic site (colon): add site **anal canal**	3	10	1	83
2	Ext	MST: Findings (stomach): add term **rapid urease test** | add attribute **result** | add attribute values **positive** and **negative**	30	5	45	37
3	Fix	MST: Findings (stomach and colon>protruding lesions>tumor/mass): add attribute: **Diameter** | change attribute value **diameter in mm. ➔ in mm.**	50	13	92	2
4	Ext	MST: Findings (colon>protruding lesions>hemorrhoids): add new attribute **type** and add attribute values **Internal** and **External**	30	7	46	23
5	Fix	MST: Therapeutic procedures (Sphincterotomy>Precut): add attribute value **No**	6	5	1	4
6	Ext	MST: Therapeutic procedures (Thermal therapy>Device): add attribute value **Heat-probe**	11	5	1	4
7	Ext	MST: Diagnoses (stomach): add main diagnoses **Antral superficial, Pangastritis, Atrophic, Alkaline reflux** and **Somatitis**	6	8	4	20
8	Ext	MST: Diagnoses: Add free text description for each organ	50	10	60	20
9	New	Other: Split **lower gastrointestinal examination** type into **colonoscopy** and **rectoscopy**. Bind both types to **Findings for colon**	30	10	6	2
10	New	Other: Localise MST Findings for Stomach form to English	1116	70	722	499
		TOTAL	**1332**	**143**	**978**	**694**

Table 2. Comparison of changeability metric values

Metric	GST	GastrOS
CCE	133.20	14.30
MC	0.14	0.02

Results

The CRs used in the study are presented in Table 1, along with the measurements for the elapsed time (in minutes) to complete each CR as well as the size of change – in lines of code (LOC) for both software source code and archetype model expressed as Archetype Definition Language statements. The 'Type' column depicts whether this caused a correction or extension of clinical concepts described in MST or addition of a new feature. The short-hand notations in the 'Description' column point out to their location in MST. The CCE and MC metric values computed from the time and change measures are shown in Table 2.

GastrOS showed lower values for both CCE and MC than GST – by factors of approximately 9 and 7 respectively, the former indicating 9 times less effort overall to modify GastrOS, and the latter indicating the changes were on average 7 times less complex.

Discussion

GastrOS is the first openEHR implementation of a desktop clinical information system using .Net platform and C#. Having a clinically validated clinical application at our disposal was the distinguishing feature of this study which gave us the capability to design a comparative study. The paper presents the implementation methodology only superficially due to space constraints. All artefacts and source code are available on GastrOS Project Website (http://gastros.sourceforge.net).

The metrics used in the study are very simple. It is worth underlining that it is the external quality characteristics of software which determine the final maintenance effort, and that these metrics are far more accurate and have a higher predictive power than internal metrics. We also believe these results are potentially generalisable to clinical applications because they don't rely on internal attributes.

Better maintainability of the openEHR based application can be attributed to a number of factors: first the CRs were mostly related with the domain knowledge. This confirms our prior assumption about the source of changing requirements. Second, much less handover was needed between the clinical and technical roles due to the clear separation model vs. software. Third the modelling formalism itself was quite intuitive and that the high level graphical modelling tools proved to be very efficient when making necessary changes. Moreover it is possible to ensure backward compatibility of data using openEHR as it provides a semantically coherent specialisation mechanism.

We have chosen to exclude search and reporting parts due to resource constraints. In fact we believe that it is these aspects of the software that openEHR would have the highest impact on maintainability because a powerful querying mechanism is also provided. The closest study where formal software engineering techniques have been employed is the study by Arisholm et al. (2001). They have evaluated changeability of two different programming approaches by writing software with same functionality (coffee machine). A full-blown clinical application using real-life CRs has been implemented in this study. These results, when supported by further work and other independent studies, may have a large impact on software cost and schedule estimation in health IT. Work is still underway to expand and validate these preliminary results within the context of the larger evaluation study where we are investigating how internal quality reflects on external quality and the cause of better maintainability using openEHR.

Narrowing the scope to clinical aspects was a limitation. However this should not be all too disturbing revisiting the fact that the source of difficulties is the volatility of domain concepts and not administrative aspects. It is noteworthy to point out that the primary author has performed maintenance tasks in both systems. These are indeed two distinct roles and neither of the authors had technical experience on the other application. So any bias is unlikely to have propagated to the maintenance tasks. It can then be argued that parameterising software can have equal effects on maintainability. However this type of approach might inevitably be too application specific and cannot be reused in other domains due to the aforementioned nature of the problem domain.

Conclusion

We have discussed how the maintainability of health information systems has a major impact on their cost and argued that this is mainly due to the essential difficulties in healthcare domain. Constantly changing knowledge translates into changing requirements. In order to evaluate the implications of openEHR formalism on software maintenance an openEHR based endoscopy reporting application has been developed using .Net/C#. It is based on the same requirements of an existing application developed with a traditional approach. Then the maintainability of both systems has been assessed using standard metrics. Our results indicate that the openEHR based application on average took nine times less effort, thereby making it significantly more maintainable.

Acknowledgements

This work was supported by a research grant from the University of Auckland (Project No: 3624469/9843). Special thanks to Dr. Heather Leslie and Dr. Louis Korman for their help during modelling. We also acknowledge Dr. Ewan Tempero for providing us with rigour in software engineering aspects. Our deepest gratitude goes to the Bedogni family who founded the Clinton Bedogni Fellowship in Open Systems which enabled this research. Training, support and C# openEHR Reference Model library have been provided by Ocean Informatics Pty. Ltd.

References

Arisholm E, Sjøberg DIK, Jørgensen M. 2001. Assessing the Changeability of two Object-Oriented Design Alternatives--a Controlled Experiment. Empirical Software Engineering; 6(3):231-277.

Atalag K. 2009. *Archetype Based Domain Modeling for Health Information Systems*
GastrOS: Case Study on Digestive Endoscopy and Validation of the Minimal Standard Terminology (MST 2). VDM Verlag. http://www.amazon.com/exec/obidos/ASIN/3639134168/ejelta5-20.

Beale T. 2002. Archetypes: Constraint-based domain models for future-proof information systems. In *Eleventh OOPSLA Workshop on Behavioral Semantics: Serving the Customer*, 16-32. Seattle, Washington, USA: Northeastern University.

Beale T. 2005. The Health Record: why is it so hard? In *IMIA Yearbook of Medical Informatics 2005: Ubiquitous Health Care Systems*, eds. Haux R and Kulikowski C, 301-304. Stuttgart: Schattauer.

Cao L, Ramesh B and Matti R. 2009. Are Domain-Specific Models Easier to Maintain Than UML Models? *IEEE Softw.* 26, no. 4: 19-21.

Delvaux, M. 2000. Minimal standard terminology in digestive endoscopy. *Endoscopy* , 32(2), 162-88.

Girosi F, Meili R and Scoville R. 2005. *Extrapolating Evidence of Health Information Technology Savings and Costs*. RAND Corporation.

ISO/IEC 9126-1: 2001. *Software engineering – Product quality – Part 1: Quality model*. International Standard. Software engineering – Product quality. Geneva, Switzerland: International Organization for Standardization.

Paech B and Wetter T. 2008. Rational quality requirements for medical software. In *Proceedings of the 30th international conference on Software engineering*, 633–638.

Rector A L. 1999. Clinical terminology: why is it so hard? *Methods of Information in Medicine* 38, no. 4-5 (December): 239-252. doi:10.1267/METH99040239.

Sommerville I. 2000. *Software Engineering*. 6th ed. Addison Wesley.

Wicked Problem (n.d.). Retrieved April 29, 2010, from Wikipedia: http://en.wikipedia.org/wiki/Wicked_problem