

Assessment of Software Maintainability of openEHR Based Health Information Systems – A Case Study in Endoscopy

Koray Atalag^{1,2}, Hong Yul Yang³, and Jim Warren^{1,2}

¹Department of Computer Science, The University of Auckland, New Zealand

²National Institute for Health Innovation, The University of Auckland, New Zealand

³Ocean Informatics Pty. Ltd.

Abstract

Maintaining health information systems over time requires significant effort and time. This is especially marked in clinical information systems where most, if not all, functional software requirements are dependent on healthcare concepts and processes which are prone to high rate of change. Software engineering literature indicates that maintenance tasks alone may constitute 70-80% of the total development 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 assess the maintainability of an openEHR based clinical application with comparison to another application based on the same functional requirements but implemented using traditional development methods. An endoscopy reporting application (GastrOS) driven by openEHR archetypes has been implemented using .Net/C#. It has the same functionality and appearance as an existing application which has been developed using Object/Procedural approach with relational data modelling. Afterwards a number of change requests have been implemented in both systems while assessing maintainability using metrics defined in the ISO/IEC 9216 and 25000 software quality standards. This paper presents the implementation methodology and preliminary results of the larger evaluation study using a more comprehensive set of change requests. These results indicate that, on average, the openEHR based application took approximately nine times less time to implement the change requests and were seven times less complex compared to the other application. While essentially a quantitative study it equally presents qualitative findings about opportunities and limitations of taking a model driven approach in development.

Keywords: *Electronic Health Records; Health Information Systems; Endoscopy; Software Maintainability; Standards; openEHR*

1 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 soft-

ware quality attribute which is defined as the capability of the software product to be modified [1].

ISO/IEC 9126 and 25000 standards present a comprehensive quality model which comprises three distinct facades 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 artefacts and code. This is closely related with maintainability prediction because the assessment is

often performed before building software for the purpose of estimating time and effort required – hence the cost. 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. This is a direct measurement of maintainability performed by observing maintenance activities. Quality in use is the user’s view of the quality of software when deployed in a real world setting. This study assesses maintainability as an external quality characteristic by using external maintainability metrics.

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. Both technical and domain knowledge expressed as software requirements are hard-coded into the program code, database schema and user interfaces. 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 (up to 70-80% of total cost) [2].

Mainstream software development approaches, typically the Object Oriented (OO) formalism together with relational databases with complex data models, work well in domains where domain concepts are stable and that most of the requirements can be elicited at the outset [3]. However, healthcare is a “wicked” domain – that is the size and complexity of Medicine negatively affects this process [4]. 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 [5-7]. In a typical development project clinicians, without much idea about the technological 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 cog-

nitive limitations of the *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 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 [8]. 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.

There is no doubt that software maintenance, together with other quality characteristics, are highly dependent on the architecture. In order to tackle the difficulties in software maintenance a set of software development methodologies and best-practices have emerged which include multi-tiering (e.g. dividing presentation, business and persistence into different tiers), following good OO design and development practices (such as loose coupling etc.), model-driven architecture/development, Model-View-Controller (MVC) approach for GUI generation and parameterising software. The openEHR formalism includes elements of these approaches and it is essentially a typical example of Model Driven Architecture where domain knowledge is captured using a domain specific language (Archetype Definition Language – ADL) and software are driven by these models. This has been shown to improve software maintainability [9].

Moreover there are additional benefits of openEHR which have led to its selection in the study. These are:

1. openEHR is a world-wide recognised *de-facto* standard which has led to the development of the *de-jure* CEN EN13606 and ISO 13606 suite of health informatics standards. This is critical for enabling semantic interoperability among health information systems.
2. openEHR provides an intuitive way of capturing clinical requirements which in effect helps engage clinicians, enables domain knowledge governance and sharing of reusable artefacts.
3. Through Archetype specialisations and formal versioning it is possible to achieve a high level of backward data compatibility.
4. openEHR also provides the means to write semantic queries which enable querying of data independent of the underlying data model (Archetype Query Language – AQL).

5. There is good tooling support which are mostly free and open source; such as Archetype Editor, Archetype Workbench, Template Designer, Clinical Knowledge Manager which are all available from the openEHR Website (<http://www.openEHR.org>).

A clinical reporting and analysis application in the field of gastrointestinal endoscopy (called as GST from now on) has formerly been developed by the first 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 describes the structure, well defined terms and rich semantics in this domain [10]. During development, this domain knowledge has been embedded into the software by means of program code and database schema. 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 GST has served as a research prototype where the objective was to validate whether the openEHR formalism can faithfully represent the problem domain [11].

Existence of a working and clinically validated application with which we could compare the new openEHR based application (called as GastrOS from now on) gave us the ability to conduct a comparative study. We believe that such comparative results provide more meaningful and convincing evidence than absolute maintainability measures alone. The aim of this paper is to present the initial results of the larger evaluation study, and also to unravel a generic implementation methodology for building desktop clinical applications using openEHR and .Net/C#.

2 The openEHR Formalism

openEHR refers to the publicly accessible engineering specifications to build complete EHR systems and also to the governing body – the openEHR Foundation [12]. openEHR have developed a methodology which here we will denote ‘**Multi-level Modelling and Development Methodology (MLM/D)**’. This can potentially minimise, if not totally eliminate, the *handover* paradigm (hence tackling with incomplete or wrong requirements). It does this by separating domain concepts from software code using domain specific models

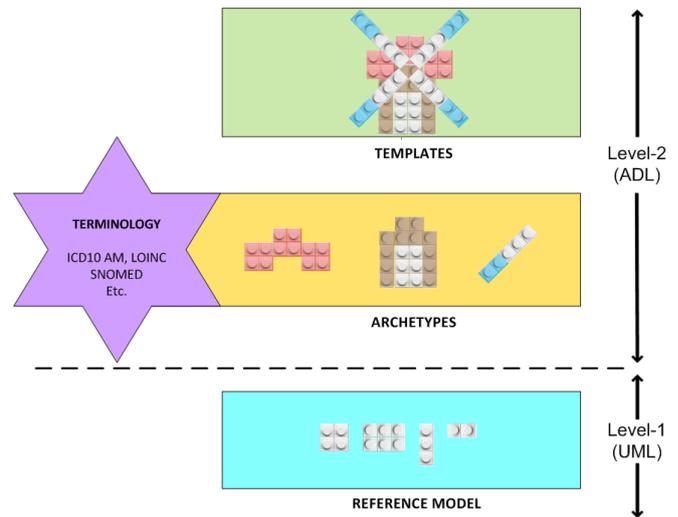


Figure 1: Schematic representation of the openEHR Multi-Level Modelling and Development Approach

called *Archetypes* which formally constrain a set of stable technical reference model (RM) entities. Archetypes represent clinically meaningful concepts such as blood pressure measurement or APGAR score. Another layer of modelling is the *Templates* which bring together a number of related archetypes and further constrain them for local use. In the runtime software is driven by these models for dynamic graphical user interface (GUI) creation, data binding and validation [13]. openEHR also defines the means to model care processes and behaviour via INSTRUCTION type Archetypes which allows for modelling of state and workflow steps of care processes. As a result altering software after deployment mostly involves remodelling done by domain experts. Thus there is hardly any need for redesign, coding, testing and deployment.

A good analogy to understand how RM, archetypes and terminologies relate to each other is using a limited set of standard LEGO® blocks to assemble many different structures (Figure 1).

RM consists of a small set of object oriented classes which depict the generic characteristics of health records (e.g. data structures and types) and the means to define context information to meet ethical, medico-legal and provenance requirements. Thus a blood pressure measurement will be represented as an instance of RM OBSERVATION class, which is a sub-class of the generic ENTRY type of class used for introducing all types of contributions to the health record. These carefully engineered classes can faithfully capture results of all medical entries together with necessary contextual information such as cuff size (of a sphygmomanometer) and position (i.e. sitting, lying) in order for correct medical interpretation. The stable and well-defined RM entities usually correspond to individual GUI widgets;

such as a CLUSTER container data structure can be represented as a frame around its children elements or the DV_TEXT data type corresponds to a text box control or a drop down list.

Archetypes provide the full semantics and structure of domain concepts by constraining relevant classes from RM and then use them as building-blocks to form a computer processable clinical model. Practically they specify particular record entry names, data structures, data types, prescribed value ranges and values for some of the context attributes. They also make possible to leverage the vast amount of standardised terms and semantics by linking to biomedical terminologies by means of terminology bindings.

On the top level Templates further constrain a group of Archetypes; such as changing names of certain data elements or omitting some altogether. During implementation *operational templates* provide the means to automatically create GUI and persistence methods which contain the whole set of information contained in the contained archetypes and terminology bindings as well as language translations.

An often overlooked feature of openEHR is the provision of a mechanism for evolving and maintaining domain models. Here the Archetypes can be modified safely without breaking original semantics and data-level compatibility by a formal method called Archetype Specialisation [13]. New data elements or values can be added as well new constraints can be introduced which are narrower than existing constraints if any. Figure 2 shows how we have created two specialised versions of the MST Findings for Stomach Archetype to meet international and then local needs. In first specialisation new terms were added where 'Rectal exam' has been defined as free text. In second specialisation not only new extensions were added but also the 'Rectal exam' data type has been further constrained to be coded text which imposes a tighter constraint than the previous. Any coded text entered using the most specialised archetype will still conform to the original and can be persisted and queried alike.

The high quality openEHR specifications have recently been adopted by both CEN and ISO, and released as 13606 suite of *de jure* health informatics standards [14,15].

3 Methods

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)

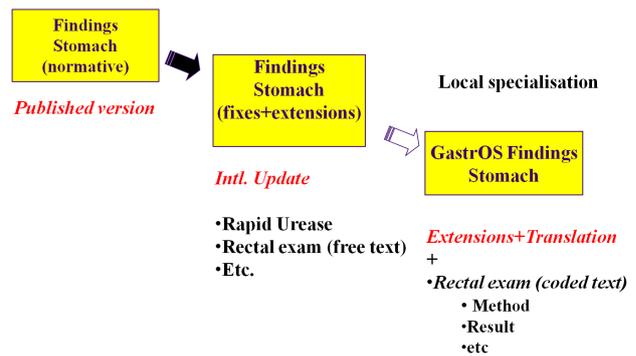


Figure 2: Archetype specialisation example from GastrOS

design, implementation and testing of GastrOS; 4) introducing change requests (CR) and implementing them; 5) measuring maintainability using external metrics, and evaluation.

3.1 Architecture of GST and its Evolution

It is worth mentioning about the old application being compared to – the GST. It had started as a commercial project in 1999 and later became the first author's Ph.D. research prototype which has been used to clinically validate MST [11]. Developed by MS Visual Basic 6 using a mix of procedural and object oriented programming techniques, MS Access (version 97 first and then 2000) has been used for data modelling and persistence. After deployment in a live clinical setting it has been used as the only means for endoscopy reporting until August 2003 (later the users preferred to use it for two more years but no data have been collected during this period). Feedback from the users (physicians and nurses) has been collected and the application development continued iteratively. This provided us with valuable insight into the domain and gave the chance to further refine requirements and observe change requests over time.

Considerable part of MST items and other extra information (e.g. patient demographics, encounter information, infection markers and etc.) were incorporated into the program code, database schema and GUI. For example the endoscopic findings for each organ, which form the majority of MST content, have been represented as separate database tables with each data item corresponding to a database column. As the requirements frequently changed due to altered domain concepts, it took very long time and serious effort to modify the software and redeploy. This clearly showed that the system was not feasible to maintain and indicated what shouldn't be done when developing such systems from a technical point of view (e.g. embedding highly complex clinical model into software).

3.2 Modelling and Development

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 detailed clinical search functions from the study which include searching by findings or procedures at data item level. The rationale for doing so was two-fold: 1) these functions were seldom used by clinicians over the course of GST's lifecycle; 2) we had to limit our scope due to time and resource constraints.

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

The 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 3). 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 (e.g. name of a drug), a drop-down list for a restricted range of values (e.g. organ types), or a panel for a cluster value that further contains sub-values (e.g. 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.

3.3 Measurement and Evaluation

The change requests used in this study for evaluation comprise real cases which caused GST to be modified in past. 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 first 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. An issue tracking system (Atlassian Jira) and two Subversion repositories have been set up to capture and document changes in code, models and databases for both applications. A different branch has been created for each CR and there were multiple commits for each CR. The data which forms the basis for software size change have been obtained from the start and end of each CR branch in corresponding Subversion repositories. For determining software size we have counted the total lines of code (LOC) for program code, ADL LOC for models and any added database column or row numbers and added them up. Similarly we have obtained the time lapsed for implementing each CR by recording start and end time and date of each CR. Any break between coding sessions has been noted carefully and excluded; thus the measured times reflect net programming time. This data also include specific comments for implementation, difficulty levels and type of programming task such as requirements, coding, database changes and debugging which will be used in the larger study. At the end of each CR implementation each programmer tested the other application using detailed CR descriptions as the reference and then debugging was made when necessary. Then the data collected on soft-

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

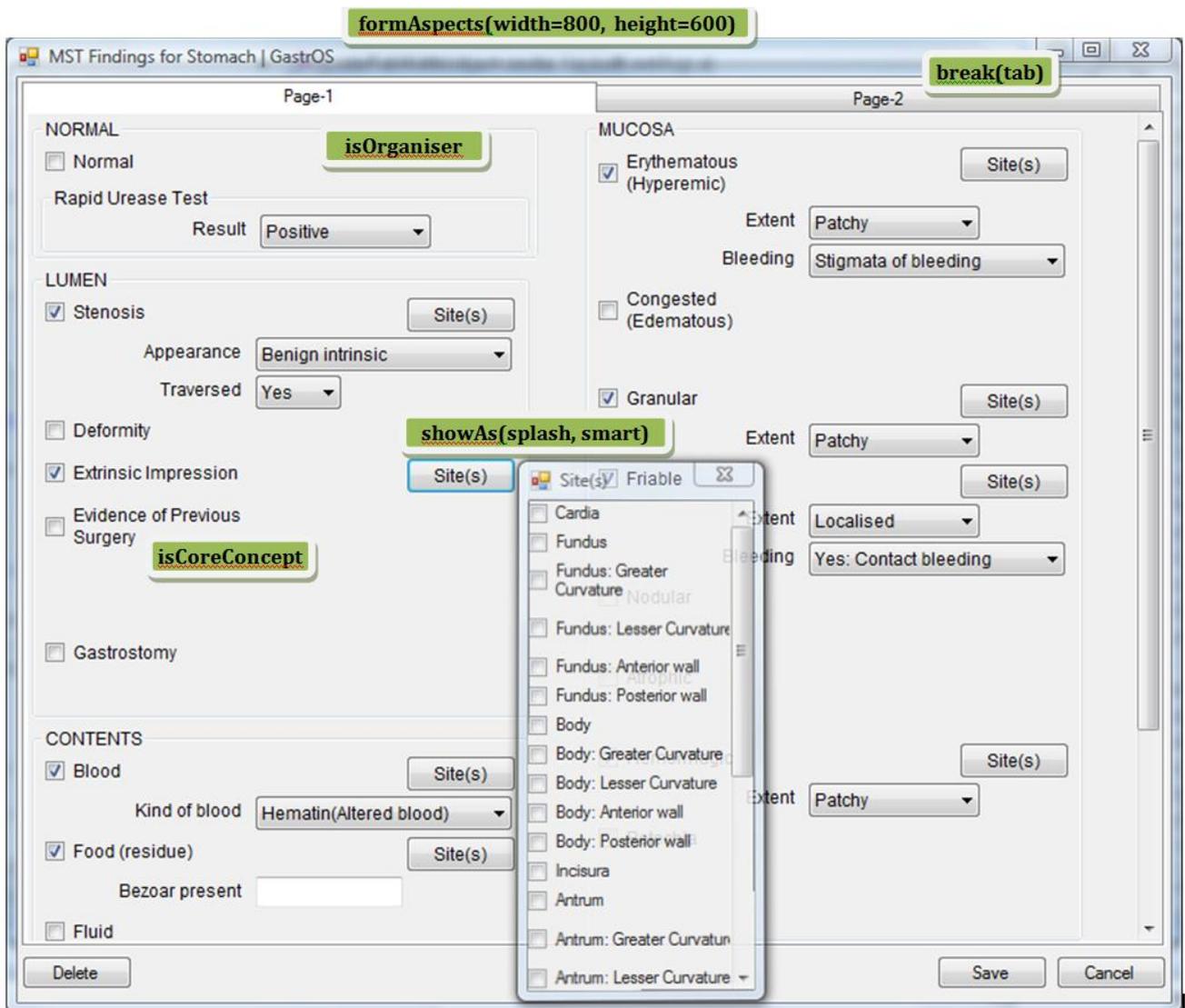


Figure 3: Endoscopy findings GUI from GastrOS with examples of used GUI Directives

Metric	GST	GastrOS
CCE	133.20	14.30
MC	0.14	0.02

Table 2: Comparison of changeability metric values

ware size change and time elapsed for each CR have been fed into the following metrics to obtain results.

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.

4 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 total size of software changed – in lines of code (LOC) for both software source code and archetype model expressed as Archetype Definition Language (ADL) 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 nine times less effort overall to modify GastrOS, and the latter indicating the changes were on average seven times less complex.

In addition to these quantitative results which applies to this specific case study we would like to emphasise the qualitative results including our design decisions and implementation methodology using openEHR; most notably the novel GUI Directives coupled with openEHR models which we have defined when generated GastrOS' user interfaces.

5 Discussion

GastrOS is one of the first openEHR implementations 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 high level implementation methodology; however all documents, design artefacts and source code are available on GastrOS Project Website (<http://gastros.codeplex.com>).

We used a combination of time and size to reflect maintenance effort, as evident in the metrics description. As with any software measurement exercise, we cannot discount the possibility of errors. One way in which the accuracy of effort measures could be compromised is if there was significant disparity between the two implementers (in terms of expertise, familiarity and productivity). This threat must be recognised, and although it would be ideal for the effort measurements to be normalised in some way to cancel out any bias, we believe such bias was already minimised due to the implementers having similar levels of expertise with their respective programs.

Notwithstanding the above threat, there is still sufficient reason to believe the results, which indicate that the openEHR based application is more maintainable, due to a number of factors inherent in the application and in the study design. First the CRs were mostly related with the domain knowledge which was taken out of GastrOS code and put into the model. There was almost no need for redevelopment when implementing these changes. 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.

This study has not been conducted in-vivo – that is the resulting changes were not tested by real clinical users. However these CR have been raised by real users over the course of live clinical usage and also validated by senior experts in the field.

There are studies similar to ours that evaluate the effects of differing software designs on maintainability. For example Arisholm et al. [16] have compared two different programming approaches by measuring the effort that it takes to implement the same application (a "coffee machine" application, which is not a real-life system despite its non-trivial complexity) using each. The main advantage of our study is that in contrast to such studies based on what are arguably "toy" systems, an application with clinically validated requirements

has been implemented and extended using real-life CRs. These results, when supported by further work and other independent studies, may have a large impact on software cost 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 further justify our results.

We did not include functions related with dynamic behaviour in the study simply because the source of domain knowledge used, MST, defines only static information requirements. It was, however, possible to model clinical workflow and state of processes using openEHR.

We have excluded detailed clinical search functions from the study due to the fact that: 1) they were seldom used by clinicians; 2) we had to limit our scope due to time and resource constraints. It is worth stating that if we had chosen to implement this functionality we would expect this actually favouring openEHR based GastrOS and not the old application. openEHR provides a semantic querying language, Archetype Query Language (AQL), which allows for designing queries independent of the physical data model but rather uses the archetype paths which have been used to collect data. Thus if the archetypes are backward compatible there is no need to make major changes to existing queries as the system evolves. On the other hand since the data model of GST is based on relational model this means all queries need to be redesigned as the application is modified which affects the data model. This inevitably will cause significant maintenance effort. In summary we believe if we could have included detailed clinical search functions the quantifiable maintainability difference between GastrOS and GST would be considerably higher in favour of GastrOS.

Finally it should be noted that using some other well-known software development techniques and best practices (such as good OO design, multi-tiering, parameterising software and MDA in general) could have caused similar effects on maintainability in this particular case study. However, as stated, openEHR has not been selected on the basis of better maintainability alone but due to many other aforementioned benefits – especially being a relevant standard. We believe a very important finding of this study is that our results indicate implementing standards based software can actually lower the development cost while this is commonly presented as a barrier to their adoption in the Sector.

6 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 maintainability 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 and were seven times less complex to implement; thereby making it significantly more maintainable. This indicates that, which applies to our case study (but with the potential for generalisation in similar clinical domains), the implementation methodology we set forth using openEHR provided better maintainability over a traditionally developed software in addition to its other benefits mentioned in the paper.

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 established 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

1. ISO/IEC 9126-1 [standard]. Software engineering - Product quality - Part 1: Quality model. International Standard ISO 9126: 2001, International Organization for Standardization (ISO), Geneva, Switzerland.
2. Sommerville I. 2000. Software Engineering. 6th ed. Addison Wesley.
3. Booch G, Maksimchuk R, Engle M, Young B, Conallen J, and Houston K. 2007. Object-Oriented Analysis and Design with Applications, Third Edition (Third ed.). Addison-Wesley Professional.

4. Wicked Problem (n.d.). Retrieved April 29, 2010, from Wikipedia: http://en.wikipedia.org/wiki/Wicked_problem
5. Rector AL. 1999. Clinical terminology: why is it so hard? *Methods of Information in Medicine* 38, no. 4-5 (December): 239-252. doi:10.1267/METH99040239.
6. 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.
7. Paech B, Wetter T. 2008. Rational quality requirements for medical software. In *Proceedings of the 30th international conference on Software engineering*, 633-638.
8. Girosi F, Meili R, Scoville R. 2005. Extrapolating Evidence of Health Information Technology Savings and Costs. RAND Corporation.
9. Cao L, Ramesh B, Matti R. 2009. Are Domain-Specific Models Easier to Maintain Than UML Models? *IEEE Softw.* 26, no. 4: 19-21.
10. Delvaux, M. 2000. Minimal standard terminology in digestive endoscopy. *Endoscopy* , 32(2), 162-88.
11. 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.
12. Kalra D, Beale T, Heard S. The openEHR Foundation. *Stud Health Technol Inform.* 2005;115:153-73.
13. 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.
14. CEN EN 13606 [standard]. Medical informatics—Electronic healthcare record communication. European Standard EN 13606: 2007,2008, European Committee for Standardization (CEN), Brussels, Belgium.
15. ISO 13606 [standard]. Health informatics – Electronic health record communication. International Standard ISO 13606: 2008-2010, International Organization for Standardization (ISO), Geneva, Switzerland.
16. 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.

Correspondence

Dr. Koray Atalag

M.D., Ph.D., FACHI

Clinton Bedogni Research Fellow

Department of Computer Science

National Institute for Health Innovation

The University of Auckland

Auckland, New Zealand

Phone: +64 (9) 373 7599 ext 87199

Fax: +64 (9) 308 2377

<http://koray.asklepion.org>

k.atalag@auckland.ac.nz

Dr. Hong Yul Yang

Ph.D.

Research Fellow

National Institute for Health Innovation

Research Programmer

Department of Computer Science

The University of Auckland

Auckland, New Zealand

Phone: +64 (9) 373 7599 ext 88237

Fax: +64 (9) 308 2377

<http://www.cs.auckland.ac.nz/~hongyul/>

hongyul@cs.auckland.ac.nz

Dr. Jim Warren

Professor of Health Informatics

Department of Computer Science

National Institute for Health Innovation

The University of Auckland

Auckland, New Zealand

Phone: +64 (9) 373 7599 ext 86422

Fax: +64 (9) 308 2377

<http://www.cs.auckland.ac.nz/~jim/>

jim@cs.auckland.ac.nz