Lessons of Software Engineering for eHealth System Developers

Ten Not-so-Easy Pieces

H. Dominic Covvey, with Dr. Donald D. Cowan

H. Dominic Covvey, Associate Editor, is a professor in the Faculty of Science at the University of Waterloo, the President and Director of The National Institutes of Health Informatics (NIHI), and a Fellow of the American College of Medical Informatics and the Healthcare Information and Management Systems Society, in Waterloo, Ontario.

For many years, I've had the privilege of working with and around Professor Donald Cowan at the University of Waterloo. Don launched and was the first Chair of the Department of Computer Science at UW and he is a highly respected CS researcher and Software Engineering (SE) Expert. My speaking with Don was motivated by my observation of a number of software development efforts that seemed to be virtually uninformed about what we have learned about SE. There was also a proposal presented to me regarding an 'eHealth Development Education Program' that did not even mention SE as part of the curriculum. My sense was that this cried out for comment. I snaffled Don from his busy schedule and this is what emerged.

Don, welcome and thank you for this opportunity! First of all, what is a simple definition of Software Engineering and how does one become competent in it?

DON: An excellent definition adapted from Wikipedia is: Software engineering (SE) is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches. SE is the application of engineering principles and practice to software and it integrates significant mathematics and computer science [11]. One can study SE at a university or college, but, as in many young disciplines, there are many people building software systems who have never had the opportunity to study the field; rather, they learned on the job. They could enrol in a graduate or continuing education program or try to associate with a team that is practicing good SE techniques. There are many books on the subject, but someone learning SE needs guidance through this morass of literature.

What is it that convinces you that Software Engineering is crucial when we develop eHealth systems?

DON: The delivery of pro-active health care to the individual and society as a whole is an extremely complex challenge. We have many interacting and interdependent stakeholders or agents, from patients, to care providers, to funders and regulators, all with different reasons for participating in the healthcare system ranging from those who need adequate care to those who deliver and profit from it. In addition to these agents, there are many shared information objects such as distributed records, locations, interventions, as well as protocols and guidelines related to dealing with both the patient and the caregivers.

Considering the complexity of delivering preventive, acute, and chronic healthcare, it seems to me a miracle that the system works at all! Remember that the modern healthcare system, like our power and transportation networks is over 100 years old. During that time the wisdom of the crowd has prevailed [5], gradually molding the healthcare system and the road and power networks [6] to what they are today.

To address this complexity and often to add to it, we are now trying to introduce information systems that we hope will make the healthcare system better. One improvement we seek is the ability to share a patient's records across the entire spectrum of healthcare providers and loci of care. But what about all the factors that frustrate success: the incredible busy-ness of the ER, the limitations of our technology and interfaces, our desire for privacy, the disruptive influences of technology, and so on.

Because of the complexity of human health and disease, of the healthcare system, of the human-system dance, and of the interconnectivity of the many manual and automated systems, it will not be simple to transform it dramatically. Rather, transformation must be a gradual process where we experiment, perhaps discarding less-than-adequate approaches, and work our way stepwise toward better solutions.

We know we must move forward to transform health care, and we are convinced that modern information and communications technology is the tool that will enable productive evolution. However, to do this we must learn from decades of SE research, building what we do on a solid foundation of knowledge and experience. We all know, too, of many disappointments and failures…these alone should cause us to ask if there are better ways to do things!

How did we develop software in the past?

DON: The traditional approach to building application software used the "Waterfall Model", which has a number of sequential stages usually labelled requirements, design, implementation, verification and maintenance. Each step, typically, is transited only once. In the requirements stage, the problem to be solved is analyzed and a specification is created. In the design stage the details of the application are created often using diagrammatic tools and other aids. Next, at the implementation stage the design is transformed into working code through programming and the reuse of components (pre-programmed software that does specific things) often stored in libraries. The integrated new code and the components of the finished software product are then tested and, possibly, parts mathematically verified, mistakes corrected, and then the application is deployed. Finally, during the maintenance stage, previously undetected errors are fixed or new functionality is added once the system is in use.

OK, but what's the problem with the Waterfall Method when it is applied to eHealth development?

1 DON: The first lesson of Software Engineering is that our approach to development needs to be 'agile'.

Under the 'Waterfall', so to speak, once the requirements for a system are specified and "signed in blood" by the organizations requesting the application, the software team "disappears" and creates the application. Several days, weeks, months or years later, the application is presented to the requesting group for final deployment. There's no real opportunity to see part of what's been done and, if necessary, correct it.

Such an approach to software development can work well when the problem is well understood and not complex. For example,

a payroll system or an ATM are examples of complicated (not complex) but well understood systems. However, because of the evolution of computing technology (Moore's Law [7]) and the way it is being used, we are now pushing the boundaries of well-understood, non-complex problems. The things we need to do are less well-defined and their behavior is not fully predictable.

We usually have some idea of what we would like to accomplish with various information processing devices, but there are often 'wheels within wheels' and we lack a deep understanding of both the problems we are trying to address and the appearance of competent solutions. For example, we are now trying to create systems that emulate human reasoning and that can 'co-operate' with busy, intuitive, somewhat unpredictable humans.

Based on these observations, we believe that the whole way in which we develop complex applications must change.

What does the new approach look like?

The software team and the users must work together during the whole project to sculpt the software as we identify new requirements and uses during the development process. In fact, the user group working with the software developers might change as the project evolves and the use of the software becomes clearer. One form of this iterative approach is called 'agile software development' [8]. This methodology for development has a continuous interaction among all the steps of the development process. For example, during the design phase, new requirements may emerge and will need to be fed back to the requirements phase; new design issues may arise during implementation and testing and also need to be cycled back up the chain.

The programming team must work continuously with users to produce what will really work for the users. Both users and developers need to take into account that new software that often obeys the "law of unintended consequences", producing results both good and bad that were not anticipated [4]. One should consider software development for health care, where the use of information technology is not well understood, as following the 3 'I's: iterative, incremental and interactive. Software should be developed in small steps (incremental), which are adapted and evolved (iterative) based on frequent user input (interactive).

Ideally many software development tools should be so easy to use that the group needing the application should be able to prototype (mock-up) or create a pilot version of most of the system. Only the hard parts, such as complicated data analysis or modeling, need be left to professional programmers.

That's important food for thought and action! What are some of the other lessons of Software Engineering?

2 **DON: A second lesson of Software Engineering is to do development 'experiments', starting small and evolving towards full solutions.**

Governments and other health management organizations often make grandiose announcements about new projects that will revolutionize the way we deliver health care. The Electronic Health Record (EHR) is one big example and, on a slightly smaller scale, the Ontario province-wide diabetes registry is another. Let's look at some fundamental questions about the registry:

- What will we store about a diabetic – diabetic's record, just basic information or information more closely related to the condition and the person?
- Who will ensure the record is kept up to date – is the entry one-time or it is updated regularly?
- Who will do the work and pay the cost of entering, validating, correcting and maintaining the information?
- How will diabetics and physicians be informed of this program? Will all or most of them use it?
- Will the system really help diabetics? Will it motivate patients appropriately?
- What value does the record provide for the diabetic, the diabetic's family, attending physicians, the Government of Ontario?

In many projects of this magnitude, questions like this are seldom asked or fully answered. The same questions apply to the EHR and the development of the Public Health Information System. Often, a high-level decision is made to announce initiatives that will impress the public and increase political capital. Software Engineering and practical experience has taught us that carrying out limited experiments within a small area with a manageable population is a superior approach that limits risks and gains the most success. Another aspect of this is friendly competition (co-opetition in today-speak): allowing the best solution to emerge, then integrating this with other experimental results, and finally scaling-up. This is a superior approach when addressing complex systems. This approach allows gradual discovery and rewards innovation and demonstrated performance.

This makes sense, as it takes advantage of the grassroots where innovation often sprouts. What about lessons related to our legal frameworks for projects?

3 **DON: The third lesson of Software Engineering is that our development contracts need to be re-thought and restructured.**

Healthcare organizations often engage outside organizations to build software systems. Thus, a contract needs to be created and executed to ensure that both parties understand the final product to be created. The contract serves as the basis for determining if the contracting parties are living up to defined obligations by establishing milestones against which to assess performance. Often, though, these contracts, even today, usually embody the Waterfall Model.

Because of what I've said already, and given that the new system is usually quite complex - no one fully understands the design or its implications – then using the Waterfall Model is completely inappropriate. Building such a system is a learning experience quite different from building a well-defined building or device. It even has some of the characteristics of a research project! We must find a new way of measuring progress on a contract, one that reflects the reality of agile software development for complex eHealth information systems.

Then there's my favorite frustration: Project Management. What does Software Engineering have to say about how we should manage projects?

4 **DON: Lesson 4 tells us that we need to manage projects that develop complex software development more like we manage innovation: not top-down and not command and control.**

There is a classic book written in the 70s by Fred Brooks [3] that details his experiences in creating and deploying the operating system for the IBM 360. The book is still in print and contains many valuable software development management lessons for today. Two messages, which are obvious when you see them in print, come to mind:

- The larger the team, the more is lost to management rather than development – large teams of software developers are not necessarily more productive than small teams. With large teams, significant resources are directed toward coordination and communication. In other words, we have to ensure that everyone on the team is not only on the same page, but on the same paragraph and preferably even the same word.

- We can't speed up development by adding people late in the project – that'll slow it down. These people must be taught all that came before. This takes away from the productivity of the current members of the team as they pass on their knowledge.

Further, more 'power' must be given to the team to change or adapt its methods. More flexibility must be added to deal with discovery, revising previous phases and allowing innovation to emerge. In one recent eHealth project, the team was already well into the development when an expert in motivation noted that nearly half the users targeted by the system were depressed, making it automatically ineffective if this fact was not addressed. Depressed people aren't big on using systems.

We always seem to be looking at other countries and trying to figure out why we're 'behind.' Is this just a self-esteem problem?

5 DON: Well, when it comes to challenges, Software Engineering tells us that, if you've seen one; you've seen one. That's the fifth lesson.

Often successes are touted when another jurisdiction seems to be far ahead in implementing information technology that supports its healthcare system. Two specific examples come to mind, Denmark and the Veterans Administration in the United States. Both these examples appear to have a more homogeneous population and a different approach to organizing their health systems – for example, Denmark makes primary care the central point of its care system – and its incentives are different from those in Canada. We can use these as exemplars, but we must find out why they work and if the lessons learned are applicable.

I have observed many examples of the 'NIH' (Not Invented Here) syndrome. What does Software Engineering have to say about that?

6 DON: Our sixth lesson is that, if we want real productivity, we must build on and/or reuse anything out there of adequate quality; we must use all the tools we can get.

There are many excellent projects applying information systems to health care with a view to making them widely available. However, when they are ported to another jurisdiction, they do not "quite fit" and, so, they are reinvented at great expense. When designing a major application, part of the budget should be devoted to discussions with other "influential" organizations that have built like systems or that might adopt the system. If we proceed in this way, the design could be flexible initially and be adaptable. Such conversations are not easy and could add to the expense of a project, but would save money in the long term.

On the other hand, the software team should seek out good ideas and developments from elsewhere and copy and improve on them. Just because your team did not think of the idea does not make it bad. Obviously, giving credit and, perhaps, paying may be necessary. Sometimes, returning components that have been improved will be 'payment' enough.

It seems that one of our great challenges is developing systems that can evolve and be adapted as the applications environment evolves. Yet, computer-based systems can be harder to adapt organically than the original manual system was. What can you tell us about that?

7 DON: Lesson 7 (maybe this is the Seventh Seal?) of Software Engineering is that architecting for adaptability is job 1.

As eHealth, which models the complexity of health and the health system, is so complex, we do not truly understand initially (and sometimes a lot longer) what is really needed and what will actually work and be used productively. Therefore, systems must be built to be flexible and organic (adaptable and survivable) in design. Not only input and reporting may evolve, but also which data is stored, how it's stored, how it's processed and how it will be accessible may evolve. Further, data may need to be redundantly stored – for efficiency of access and processing. Data redundancy is not usually a desirable property, as duplicate data can become inconsistent unless great care is taken, but redundancy can be crucial. Any design must consider these factors. Thus, any system design should include tools to make it easy to:

- Change reports, inputs, rules, processing methods, vocabularies, screens, workflows, reports and other aspects of the system subject to change.
- Alter database structure and content to address evolving 'data models' (how the data is structured to represent the environment) without requiring re-programming (this is often termed 'active data dictionary').
- Develop software agents that support type-once (enter data once), use-many (use it in many applications) and that synchronize multiple databases on a pre-determined schedule or whenever a change occurs.

One of the 'hot buttons' of eHealth is privacy and the concomitant need for security, where is Software Engineering on that issue?

8 DON: Security is crucial and the eighth lesson of software engineering is that it must be addressed at only one point in a system, be designed into the system architecture and be fully tested.

Ensuring secure and authorized access to records with personal information is a difficult process at best. A userid and password are not enough. There needs to be a structured form of access control similar to RBAC [9] or the Tees confidentiality model [10]. Architectural approaches like Aspect Oriented Design (AOD) are essential. AOD enables the centralization of all security management at one place, serving as a gate into the fortress.

eHealth systems can be implicated in causing serious bodily harm or even death if the incorrect data is entered or incorrect or inappropriate lab results (e.g., of another person) are delivered. We must build systems to monitor entered and reported data for reasonableness. Similarly, we must subject crucial pieces of code to some form of highly structured and exhaustive testing or even proof of correct operation. Understanding and applying the theory and practice of Information Systems Assurance [12] and risk mitigation are essential.

You've talked a lot about the technology aspects of systems, what does Software Engineering tell us about the human side?

9 DON: Software Engineering's ninth lesson tells us that we must recognize that systems are developed in the context of humans, who must be deeply involved in the development and implementation process.

We need to remember that the delivery of health care is a collaboration among members of a team with many participants. This team includes the care providers, the patient and many other stakeholders. Their direct and effective involvement must be assured and any products must genuinely satisfy their needs. Health care could also benefit from the many social media strategies that are being developed and deployed, although not necessarily in their current form. Mediated social networks [1, 2] with adequate security must be considered as tools to be integrated into modern healthcare processes. In other words, the realities of humans and their professional and social interactions must be embodied in development processes and the systems produced.

We also need to recognize that eHealth information systems are developed and operated by people for people, so a significant amount of the design effort must be directed toward how people will

use them and how the systems fit into the care process. The introduction of a system will affect care processes; the care processes must be studied carefully to ensure that the new system will improve the situation, not get in the way and have to be circumvented by 'work-arounds'. For example, centrally capturing data about patients may be valuable for a hospital, but it should also help floor personnel do their job better, not force them to create a spreadsheet kept in a desk drawer that contains the "real" data. Thus, systems need to be assessed for usability and utility [13].

DON: What about education and training? How important are they to Software Engineering?

10 Lesson 10 tells us that Software Engineering is a growing and evolving discipline impacting many different areas; it's not just a matter of writing code; constant upgrading and assessment are involved.

Although we are building software for many different applications, we are still discovering new lessons and techniques particularly in areas such as eHealth that involve interacting and interdependent stakeholders. Any software development team should have the education budget to ensure that they stay on top of the latest usable technologies. Further, teams should continuously measure their ability to perform software development against some "standard" such as the Capability Maturity Model [14]. These may seem like frills when the job is to create a software product, but remember that people's safety and lives often depend on the result.

Well, Don, that's quite a syllabus in itself? Any other thoughts?

DON: These are just the high points of Software Engineering. Given the depth underneath all this, someone (if not the entire development team) must be expert in the breadth and depth of Software Engineering. We should no more develop software without this expertise than build a bridge without the appropriate engineering team! Software development is not an amateur DIY job; too much depends on the lessons described earlier. I've just provided some of the thoughts based on my experience in designing, implementing and observing software systems in many fields including eHealth. If anyone would like to discuss any of this, contact me at dcowan@uwaterloo.ca. Obviously one could write a book similar to the one by Brooks [3], but this article will have to suffice for the moment. Maybe someone who reads this could take on the challenge of writing a book on eHealth Software development!

References

Donald Cowan, Paulo Alencar, Fred McGarry, Carlos Lucena. A Web-based Framework for Collaborative Innovation Technical Report CS-20121-02, David R. Cheriton School of Computer Science, University of Waterloo, 2012. http://www.cs.uwaterloo.ca/research/tr/2012/.

D. Tapscott and A. D. Williams. Wikinomics: How Mass Collaboration Changes Everything. Portfolio, Penguin, 2006.

Frederick P. Brooks. The Mythical Man-Month: Essays On Software Engineering, Anniversary Edition, Pearson Education, 1995.

Law of unintended consequences See http://en.wikipedia.org/wiki/Unintended_consequences.

J. Surowiecki. The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations. Random House, 2004.

Steven Strauss blog: http://www.cbc.ca/health/story/2010/01/06/f-vp-strauss-electronic-health-records.html.

Moore's Law: http://en.wikipedia.org/wiki/Moore%27s_law

Agile Software Development: http://en.wikipedia.org/wiki/Agile_software_development?

utm_source=twitterfeed&utm_medium=twitter.

E. Bertino, L. Martino, F. Paci, and A. Squicciarini. Security for Web Services and Service-Oriented Architectures. Springer, 2010.

J. Longstaff, M. Lockyer, and J. Nicholas. The tees confidentiality model: an authorisation model for identities and roles. In Proceedings of the eighth ACM symposium on Access control models and technologies, 2003.

Software Engineering Definition: http://en.wikipedia.org/wiki/Software_engineering.

Information Systems Assurance: accounting.uwaterloo.ca/uwcisa/Annual.../annual_report_07.pdf.

Books on Usability http://www.amazon.com/Best-Usability-books/lm/3IL8FPQIHFB4O, http://www.usabilityfirst.com/training-workshops/books-on-usability/.

Capability Maturity Model http://www.sei.cmu.edu/cmmi/.