Everything a medical innovator needs to know about developing software

Marcus Holmes

REVIEW

Please cite this paper as: Holmes M. Everything a medical innovator needs to know about developing software. AMJ 2013, 6, 1, 19-22. http://doi.org/10.21767/AMJ.2013.1579

Corresponding Author:
Marcus Holmes
Email: mfhholmes@gmail.com

Abstract

Software development is hard enough for specialist software companies to get right. For people outside the industry it can be a minefield full of hidden surprises. This articles hopes to explain why software development is so hard, how to go about approaching a software development project, and how to get the best out of any collaboration with a development team. It should be read by anyone who is considering developing a software product, including websites, from a non-development.

Key Words
Software development, Agile methodology, Software product

Introduction

The majority of software development projects fail. Software project failure happens when the result of the development project does not meet the expectations of the project sponsors; when the benefits of the software are not perceived to exceed the costs of development. There can obviously be many reasons for this, but they are almost all human, social reasons instead of technical reasons. In this article I attempt to outline an approach based on recent developments in the 'agile' software development methodology that reduces the risks of failure and allows doomed projects to be caught early. The article addresses the typical commercial startup scenario of an innovator with medical expertise, but no software development expertise, who needs a software product developed (a phone app, a website, a desktop application, or any other software product) by a third-party development team.

The technical creation of the software product is not that difficult; given enough time and free access to the internet, anyone with any aptitude for it can write almost any software system. The difficult parts are to do with communication, expectations and change:

- There is an understanding gap between the innovator (that is you) and the software developer, because the developer has no medical experience and you (presumably) have no development experience. This understanding gap leads to assumptions being made on both sides, which causes delays and defects in the software. Closing this understanding gap takes a lot of very good, very clear communication.
- The expectations of innovators are usually unrealistic when viewed from the perspective of a developer. This is usually because software development is expected to be a deterministic process, when it is in fact a creative design process and therefore subject to a lot of variance. Two developers will not write software the same way, in the same timeframe, or with the same results. Therefore it is not possible at the start of the project to determine the result at the end of the project.
- Software projects are subject to enormous amounts of change. This change comes from many different sources, but the most common is 'requirements creep'; the tendency for the requirements of a piece of software to change as the project proceeds. Requirements creep is a natural and unavoidable part of the development process, caused by the growing familiarity of the innovator and developer with the software, and the resulting changes in their perception of the best way to develop it.

A common preconception about software development is that it resembles housing construction: a detailed plan is drawn up, the developer proceeds to build the structure according to the plan and completes it after a definite amount of time, handing the completed structure over to the customer and leaving the project. Whilst there are software projects that can be built on this basis, they rely on a stable set of requirements and a thorough

understanding of the problem by the whole team. These conditions do not apply to innovative software products and so a different approach is needed (an example of such a comparison at British Telecom).[1]

The standard analogy used for this approach is that of building a pyramid. A pyramid builder is tasked with building a pyramid before his Pharaoh dies. He could start from the bottom and build up, and hope the Pharaoh does not die before he finishes, or he can build a small working pyramid and then enlarge it until the Pharaoh dies, at which point he is guaranteed to have a working pyramid.[2] This inability to have a predictable project because of the inherent unpredictability of the Pharaoh's lifespan mirrors an innovative software project, where the fact that it is innovative also means it is inherently unpredictable. Martin Fowler, one of the leading lights of the Agile movement has talked about the unpredictability of requirements.[3] The agile methodology deals with this unpredictability by making the smallest possible working version of the software and then adding to it in small steps or iterations. As the old joke has it; to eat an elephant you must take one bite at a time.

So as a medical innovator attempting to produce a software product, you probably find this approach can be difficult to understand. The basic process is:

- Start as small as possible; identify the most important function of your application, and concentrate on that.
- Build a 'user story' of how the user will interact with the application to use this one function. An example of a user story is given in Appendix 1.
- The developer builds this one function, and you assess with them how this works and if necessary refine it until it works to your satisfaction.
- Move on to the next function. Write another user story for how this function works and interacts with the first function. Hand it to the developers to write, then review it with them.
- Repeat this feedback loop until the system is able to be used for the purpose it was intended, then add user feedback to the loop. Find some test users and put the software in front of them and record their comments, suggestions and complaints. Adjust your 'user stories' to correct for the user's feedback.
- Each turn around the loop should be no more than 5-10 days, and the entire software program must be delivered for each loop (so full source code and an executable that runs or a website that works). Any defects or bugs need to be listed and communicated with the developer, and must be fixed first before any new functionality is added.

This process of defining small sections of functionality, and then delivering them completely, ensures a few very important things in the development:

- the development is constantly focused on delivery of working software. You are never stuck with a pile of useless expensive code that does not do anything.
- requirements (derived from your understanding of the subject area and user needs) are refined as the software is built, reflecting the actual use of the software by users;
- defects are spotted early and fixed early;
- if the software is not going to work as planned, for whatever reason, this is identified early before too much effort and money is wasted.

However, the method does make explicit the implicit uncertainties involved in software development, and you will not be able to set a definite budget before the project starts, or know exactly how long it will take. Large organisations can have problems with this approach.[4]

This process of defining a function and then reviewing it once completed involves a lot of communication. As the 'subject matter expert' you will need to communicate your understanding of the subject area to developers who probably have no experience with it at all. Likewise your technical knowledge of software development is probably limited and you will need to listen to highly technical explanations of possibly development options in order to understand the issue and make decisions during development. Please try to understand the issues, because from a developer's point of view there is nothing worse than being told 'whatever you think works best is OK' only to then find out that actually that decision was important and we made the wrong choice. There are no stupid questions, but there are stupid assumptions.

As communication is so important to the process, it is obviously essential to get the right development team. The clearer you can communicate with the team, the better the project will go. For this sort of development, good communication skills are more important than strong technical skills, as more time is wasted through miscommunication than through slow development. If you are using local developers, then ask to actually meet

with the team and talk about the project with them. If you are using offshore developers, ask to hold a teleconference with the development team (and if you do not share a language with the team, be prepared to spend a huge amount of time clarifying your user stories with them). Your 'gut feel' about these people is important, and you should be able to respect and trust the team to be doing their best for your project.

A software product is rarely 'finished', in that there are always updates and new releases to be made, and users will need support. You will need to consider this before finalising your business model; do not expect to sell a copy of your software to a customer and have no further interaction with them, this just does not happen (consider how often your phone apps update: you will need to maintain a similar release schedule for your software product). Likewise, do not expect to have your development team write your software and then walk away. It is much better to have the original team continue to be involved with the updates and support than to try and transfer the code base to a new team to maintain (they will almost certainly want to rewrite large chunks of the code to suit their style). So your relationship with the development team is likely to be a long one, and therefore prioritising the ability to communicate with them makes good sense over this long time period.

The commercial relationship must also be very clear and defined by a contract. Do not engage with a development team without a formal contract, it is just asking for trouble. Generally speaking the standard commercial arrangement is that a deposit is payable up front, with payments after that on completion of each feature or phase. Usually the developers will quote a fixed price to deliver each feature, although some will charge per hour (though ensure that if they charge per hour there are some clear clauses about what happens if their estimates are inaccurate). Other items to be aware of in the contract are:

- Copyright of the entire source code is clearly assigned to you. If the development team is planning to use a framework that is owned by them (or a third party) then ensure that the contract includes a non-revocable licence to use the framework for the product, and that the contract clearly delineates what code will be owned by which parties. It goes without saying, but if the development team is planning to re-use code paid for by you for other customers then you should not be paying the full cost for this development.
- Likewise, ensure that ownership of any domain names, brands, trademarks, and other intellectual property is clearly owned by you, or clearly not being paid for by you.
- If the project is a web site or otherwise requires hosting, that the host is a third party and not the development team. This is really important, as the ongoing hosting of the site needs to be independent of your relationship with the developers. The developers may host a development version of the site used during the project, but delivery of each project phase must include implementation of the site to your hosting provider.
- Responsibility for bug fixing and defect remediation is clearly defined. Not only the actual fixing of the code, but also bug tracking, customer communication and post-fix releases. Do not expect the development team to fix bugs for free once the product has been signed off, but agree a reasonable plan for post-release bugs. This is usually covered by a maintenance and support agreement with the developer, for an annual fee.
- What happens when third-party skills need to be sourced. Talk over the scope of the application with the development team and identify any areas that might be outside their expertise, and work out a plan to cover those gaps. It is usually easier for the developers to source the required expertise and incorporate it into the team, but even if not then make sure there is a clear clause about how any third parties are incorporated into the project.

Obviously, this is not legal advice and you should get a lawyer to advise you on the legal implications of any contract negotiations.

Lastly, developers are almost always craftsmen. They (we) really take pride in our work, and are usually driven to create elegant, functional, attractive, useful systems that users will enjoy experiencing. Your project will go much smoother if you treat the developers as craftsmen, as part of a team that includes you, engaged in a co-operative process to build, together, something that will create value. Developers do not tend to do hierarchical relationships or command structures well, and attempting to manage them by authority will tend to produce worse results than managing them through mutual respect and professional pride.

So to maximise your chances of your software development succeeding; take an iterative approach to delivery of the software, get it up and running every 1-2 weeks. Start with the core functionality and expand outwards, including some actual users as early as possible. Prioritise communication in your selection and management of the team, and be prepared to deal with the uncertainty inherent in the process. And good luck! Building and selling a software product successfully is a deeply satisfying experience.

Appendix 1: A User Story

User stories are narratives that describe the user's interaction with the system to be built. They don't describe what the system does, they just describe what the user does and expects to see.

A typical user experience is logging into a system. The user stories for this experience might look like:

> "Users have a user name and a password to log in to the system. The user specifies both the user name and password"
>
> "If a user has forgotten their password, they need an alternative mechanism for authenticating"
>
> "The password must be secure and not able to be intercepted, and the authentication process must not be able to be eavesdropped by a third party"

These stories are written by the customer of the system, usually in conversation with the developers, or as part of the initial brief for the system. The stories are then fleshed out by questions from the developers and further clarified, so for instance:

> "Users have a user name and a password to log in to the system. The user specifies both the user name and password. The user name must be at least 6 characters long, and the password must be at least 6 characters long, with at least one Upper-case, one lower-case and one numeric character in it."
>
> "If a user has forgotten their password, they can ask to have the password emailed to their email address. The email address must be specified when the account is created, and a confirmation email sent to it to ensure that the email address is valid."
>
> "The password must be secure and not able to be intercepted, and the authentication process must not be able to be eavesdropped by a third party. The system must not store passwords in cleartext in the database."

Once the stories have enough detail to be built, they are then implemented in the system and reviewed by the customer. There may well be further refinements to the story (now a feature of the system) once the customer actually gets to use the implemented feature.

References

1. Evans I. Agile Delivery at British Telecom. Method & Tools. Summer 2006. Available from: http://www.methodsandtools.com/archive/archive.php?id=43
2. Mayo J. Two ways to build a pyramid. Information Week. 2001. Available at: http://www.informationweek.com/two-ways-to-build-a-pyramid/6507351.
3. Fowler M. The New Methodology. 2005. Available at: http://martinfowler.com/articles/newMethodology.html#TheUnpredictabilityOfRequirements.
4. US Govt General Accounting Office. Effective Practices and Federal Challenges in Applying Agile Methods 2012. Available at: http://www.scribd.com/doc/101495164/SOFTWARE-DEVELOPMENT-Effective-Practices-and-Federal-Challenges-in-Applying-Agile-Methods.

PEER REVIEW

Not commissioned. Externally peer reviewed.

CONFLICTS OF INTEREST

The authors declare that they have no competing interests