How Application of Systems Engineering Principles Would Have Led to a Better Rollout of the Affordable Care Act Website

Eric Mesa

November 23, 2014

Abstract

The rollout of the Affordable Care Act website was an unmitigated disaster. The original goal of the administration was to have the website available for an open enrollment season starting 1 Oct 2013. One month later the website was still having issues processing claims. This paper will examine the root causes of the site's failure and dicuss the ways in which an adherence to systems engineering principles would have led to a website rollout with far fewer issues.

1 Introduction

This paper examines the failure of the Affordable Care Act website to function as intended upon the website's launch and for months afterward as relates to systems engineering principles. The paper will begin by exploring the systems engineering princicples the project correctly implemented. The paper will then move on to where the project failed to implement systems engineering principles. These include a failure to require a requirements lock-in, to allow adequate time for testing, and to ensure management of subcontractors was handled competently. Finally, the paper will conclude with an examination of how proper utilization of systems engineering principles might have led to a more successful project completion.

2 Off to a Good Start

Congressional testimoney reveals the the contractors charged with developing the Affordable Care Act website were familiar with systems engineering principles. [US House(2013a)] Many models exist to encapsulate the systems engineering process; one of the classic models is the systems engineering *vee*. Systems engineering stresses the left side of the *vee* as being key to have locked-in before production begins if the developer is to meet projected schedule, cost, and levels of correct functioning. As can be seen in Figure 1, the processes on the left side are Feasibility Study, Concept of Operations, Systems Requirements, High-Level Design, and Detailed Design. Of the *vee* key processes, the most important are the Concept of Operations and the System Requirements. [of Transportation(2007)]

The Concept of Operations (CONOPS) describes what problem the new system or product is meant to solve. Without a well-written CONOPS, the rest of the development process becomes an ill-defined mess subject to mission or function creep. Mission or function creep describes a project whose boundaries are so blurry that additional functions or missions can be added to the project, sometimes taking it in a direction directly opposed to the original idea. The CONOPS defines preciesly the problem the system will solve and, in a well managed project, serves as an artifact approved by upper management to allow the manager to refuse additional functions. When these extra functions or missions have the best intentions in mind, a well-run project will add those to a CONOPS for version or release N+1. From the documentation available, it appears a well-defined CONOPS was in place describing what the Affordable Care Act website was supposed to do.[US House(2013a)] Not only does this reveal a well thought-out CONOPS, it also establishes what kinds of tests should be run for Sunny Day scenarios. [1]

If the CONOPS is the bedrock of the project, the System Requirements are the foundation. Shaky requirements will support a project as poorly as a shaky foundation. The reason for the intense focus on this process is that, in addition to being the foundation of the project, it is quite difficult to do correctly. The difficulty arises from the first step. An important skill for a system engineer to learn is how to create requirements from stakeholder needs. Stakeholders often try to define the solution rather than their actual needs. This can result in a substandard end product that meets their "requirements", but fails to meet the true requirements. As an example, a stakeholder may say he has a requirement for the system to posess 16 GB of RAM. If the system engineer takes this at face value, the system will have 16 GB of RAM, but may fail to accomplish its goal. It takes instruction to learn how to determine that the this same stakeholder actually needs X tasks to take Y minutes to complete. As the design progresses, it may be revealed that the key is to have a solid state hard drive and the system will actually perform perfectly fine with 8 GB of RAM. Additionally, it is important to document the requirements and present them to the stakeholders and get their buy-in. At this point, it is up to the organization's culture whether they will respect the process and allow requirements to be locked in until release or version N+1. The McKinsey Presentatoin to CMS shows that the contractors were well aware of how important the requirements phase was, but despite having the knowledge, the recommended process was not followed.[US House(2013a)]

3 Where Things Went Wrong

Despite being cognizant of the importance of the System Requirements phase, the aforementioned McKinsey presentation mentions "Evolving requirements" and "parallal 'stacking' of all phases". The graphic on this slide, which is like a simplified Gantt chart, shows "Define policy / requirements" lasting throughout all of the "Design" and "Build" phases, and through a large portion of the "Test" phase.[US House(2013a)] While the real world rarely allows a perfect progression through the stages of Systems Engineering, it is accepted that while

[1] A Sunny Day scenario describes an ideal sequence of events demonstrating interaction between users and systems

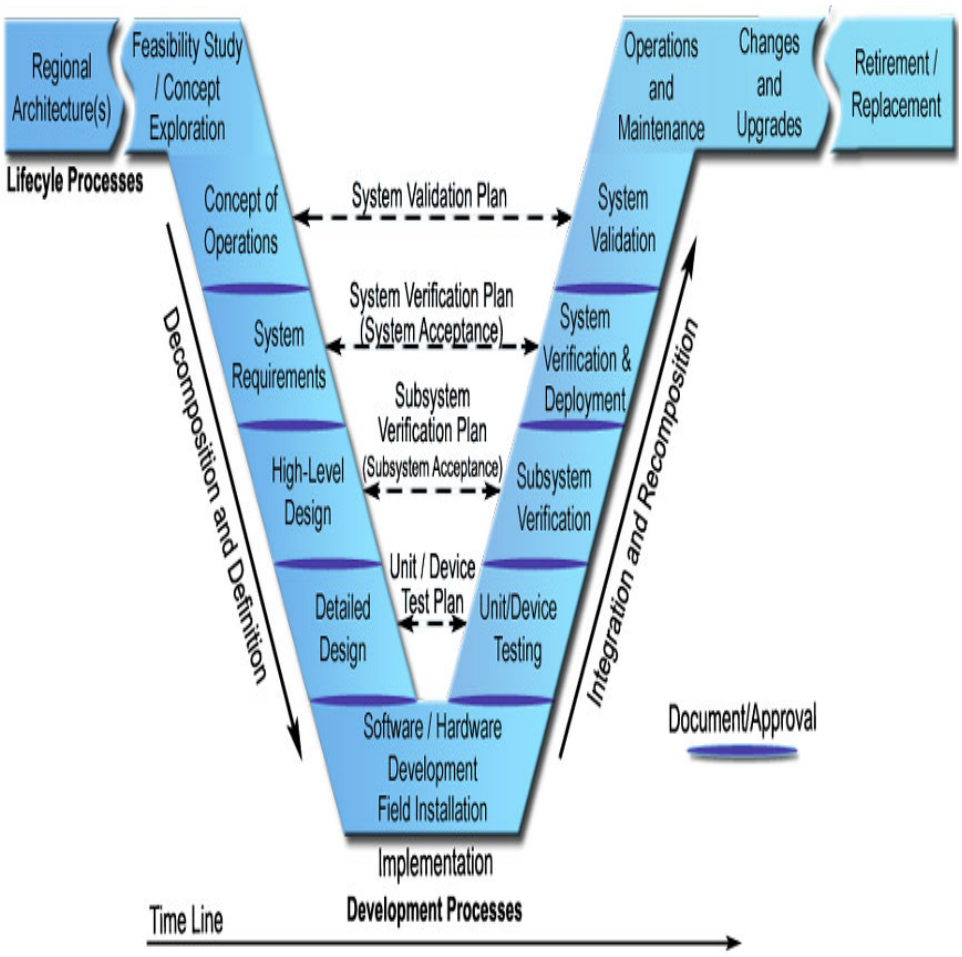

Figure 1: Systems engineering *vee*

the design phase can lead to a review of the requirements, only in extremely rare circumstances should requirements be changing during the build phase. The requirements must be completed before testing can begin because integration and testing requirements are derived from the original requirements.

While the importance of locking in the system requirements cannot be understated, there were still other issues affecting the project. On the other side of the systems engineering *vee*, verification (system meets system requirements) and validation (system meets stakeholder needs) were impossible to do correctly if the requirements weren't firm.[of Transportation(2007)] Because requirements come from an understanding of stakeholder needs, the constantly changing requirements either signaled a constantly changing set of stakeholder needs or requirements being changed without confirmation of a change in customer needs. Another consequence of changing requirements (or adding new requirements) late in the development process is a need to compare requirements to ensure contradictory requirements are not being added to the project. Even in the case where the new requirements are not directly contradictory, they may affect

an existing requirement in a way that makes adequate testing difficult. For example: requirement A says states that "the system shall complete the task in 1 second." New requirement Z states that "the system shall complete the task after sending an email." What if this new requirement causes the system to take more than one second? Is it now failing its requirements or does the new requirement supercede the old one? And this new requirement might cause the entire code base to be rewritten, making it a better candidate for a future release. Because success on a software project can be defined in many different ways, verification and validation testing provide a strong basis of knowing when the system is sufficiently correct. A system passing these tests should not have any more work done as regressions could end up in the code.

Given those difficulties, the biggest culprit of the end-user failures which made the headlines upon the launch of the website was the insufficient time given over to completed system testing; this is also called out in the McKinsey presentation.[US House(2013a)] The high necessity to test the final design of the Affordable Care Act website was due to two aspects of the site. First of all, the site's frontend needed to remain responsive in the face of the high number of expected users and the database backend needed to cope with this great number of users. Modern websites are able to use content delivery networks (CDN) to reduce the strain on the server for static content. However, highly dynamic sites like the Affordable Care Act website require many accesses against databases and such content cannot be cached by a traditional CDN.[CDN()] Thus, load testing becomes incredibly important for websites expecting users on the order of magnitude of the number of households in the United States of America. A reduced testing timeline combined with incomplete designs and builds left the contractors unable to fully load test the website. For the first month the website was unresponsive to users.

Second, the system's backend needed to be able to connect to the systems of the insurance providers to enroll the users. Systems which were not originally designed to talk to outside systems need extra testing. The Application Programming Interfaces (APIs) may have been built with internal processes in mind and it would be incredibly difficult for the creators of the Affordable Care Act website to anticipate differences in the way different terms were used internally. Additionally, the system would be vulnerable to edge cases that would not have existed with only internal systems. Black box testing of the edge cases would have been extremely helpful in this case. Again, without a long enough testing cycle and without the build being done on time, it would be impossible for the website to reliably connect to the insurance backend computers; the final step in ensuring the enrollment process had completed. At the point users were able to connect and go through the website, but were not certain of being registered with the provider because of the backend problems.

The project was also plagued with managment issues. Without good management, the principles of Systems Engineering cannot be enforced. For one thing, it appeared there was a lack of accountability from the subcontractors on the project. Emails released as part of congressional testimoney show that the FM (figure out what this is) build was not adequately staffed. Only ten developers were working on the project and only one of them had the required skills to complete the project.[US House(2013b)] When a project is being subcontracted, the proper Systems Engineering princple to apply would be to consider the system as being broken up into multiple systems and hold systems integra-

tion meetings to ensure the final product would be ready to integrate into the final product. This includes guarantees that the project is being adequately staffed. Additionally, if the integration is taking place across different companies rather than different divisions within one company, the contracts should be written to enforce accountability with fiscal penalties for for lack of delivery at the expected quality at the expected milestones.

4 How Things Might Have Gone Better

Systems Engineering principles do not only cover the ideal situation in which all steps are properly followed. It also provides for coping mechanisms to try and correct deviations from the ideal in an attempt to achieve the highest quality at the lowest cost given the suboptimal situations. The most important deviation is the inability to lock in the requirements during early phases. The primary contractor should have pushed for a contract with the US Federal Government which recognized the importance of systems engineering and empowered the contractor to set a date beyond which no new requirements would be added to the project and no existing requirements could be substantially changed. At that point they would have continued to collect new requirements, but would have committed to rolling them into Affordable Care Act Website 2.0. This would give the developers a fixed point from which to develop and a stable API for connecting to the insurance provider backend. Such a process is not unheard of in software development. It is the main process by which nearly all open source projects, such as the Linux Kernel, are developed. At predefined intervals the coders decide which issues they will tackle in the next kernel version. At that point, any new proposals will have to wait until the N+1 version. Once a version is released, a small team continues to work on bugfixes while the majority of the team moves on to the next version and the new requirements.[Corbet(2013)] Such a lock-in would have given the contractors on the Affordable Care Act website a solid foundation for development and future testing. Additionally, while committing to such a schedule, the stakeholders would have felt assured their needs wold be met in a timely matter while allowing the website to launch with as few issues as possible.

By locking in requirements early, the developers would have also bought more time for testing. A rigid lock on requirements would keep the code from gaining feature creep and testing could be planned out correctly. While it might be fair to suggest the contractors were working on a website without precedent, the developers could have looked to similarly popular events for the necessary testing scale. Early in its life Twitter would often become unresponsive during large cultural events. The website has now built in the ability to scale with use, resulting in a near lack of unavailability during even the largest events.[Riley(2007)] Cloud technologies like Amazon Web Services (AWS) allow websites to scale dynamically obviating the need to run and maintain a large number of servers when there will only be spikes during open enrollment season. [Amazon(2014)] Running a service like AWS would allow the site to scale beyond any number the constractors assumed for the popularity. Massachussetts' healthcare law was considered the model for the national healthcare law, but the website had to be redesigned to interface with the ACA website. The same contractors built the new Massachussetts site and it was just as unusable as the national

site.[Conaboy(2013)] Even when faced with a reduced set of users, the site did not perform well. A new site is being built and is expected to be able to handle 12,671 concurrent users triggering actions. With 306,000 people needing to enroll, they may still face issues that could be obviated by a dynamically growing site. But, at least they are basing their numbers on being able to have 74,000 users on the site at once (although only 12 thousand of them triggering actions at once) which is twice the number that accessed the site on opening day last year.[Bebinger(2014)]

While an Agile software development process might have better dealt with changing requirements, it may not have worked well for the Affordable Care Act website. The Agile Manifesto contains the following two points: "Customer collaboration over contract negotiation" and "Responding to change over following a plan".[Beck et al.(2001)] As has been discussed above and will be elaborated on below, because of the subcontracting within this project, it appers stronger contract negotiations were required. As for the second point, software in the commercial world often slips its release date and software in the free software world is often released "when it's ready". Both styles lend themselves well to an Agile development process. Because of federal open season deadlines and because the Affordable Care Act had tax penalties for the uninsured, a deadline slip was not possible. Instead, testing was simply reduced to the minimum. WIthout being able to use the key advantaged of agile programming, its ability to deal with constantly changing requirements would have been hobbled.

Often when development work is subcontracted, the work becomes a black box to be delivered at the integration phase. If communication has not been at an optimal level, this risks the project being held up at the integration phase as both entities focus on debugging incompatible interfaces and a realization that the sub-contracted code does not work as desired. Systems Engineering presents two possible approaches to help mitigate the effects of subcontracting and which would have helped the Affordable Care Act prime contractor discover issues at an earlier stage in which remedial steps could have been taken; including finding a new subcontractor or alerting the customer that the website would not be able to make the deadline.

The first technique recommended by systems engineering is related to the main reason the project was off track - clear sub-requirements and definitions of success needed to be submitted to the subcontractor. By doing so, the the subcontractor loses the ability to claim they did not understand what they were tasked to do. Additionally, it creates a point, early in the process, for the subcontractor to ask for clarification to ensure they have the same understanding of the definition of success and sub-requirements. The goal is to set up the honest subcontractor for success and create a papertrail for the dishonest subcontractor to be sanctioned.

In a situation in which a subcontractor's work is either going to be key to the success of a project or there is suspicion a subcontractor may not intend to give a project the proper resources, one way to have a set of checks against their progress is to have a series of early integration testing steps. Rather than waiting for the final integration and integration testing phases, the subcontractor can be required to try and integrate their code more often. Beginning with stub functions, this serves a very important role of ensuring interfaces work as intended. If one contractor is expecting to be able to pass three variables and the other is only expecting two for its input, this can be discovered at the stub

stage before major coding begins. At each successive integration testing phase, more work should be complete and this would reveal several issues. Within the context of this specific project, the lack of engineers on the project would have been revealed by a lack of adequate progress between phases. This would also be key in alerting the team to potential schedule slippages. In general, these integration phases would be a good time to do basic code reviews to ensure the code quality and/or code speed was adequate. There are many ways to code a solution, but there are different scales of the Computer Processing Unit time needed to arrive at the solution. Some of them scale linearly with inputs, but others increase exponentially. For a website needing to handle hundreds of millions of patients, this scale is incredibly important.

5 Conclusion

This paper has shown the root causes of the failure of the Affordable Care Act website. The paper examined how the failures stemmed from a lack of adherence to systems engineering principles. Finally, the paper explored how systems engineering princicples could have helped alleviate the issues, including ways to get around real world issues that face implementation of systems engineering principles.

References

[CDN()] "What is CDN?" http://www.cdn-advisor.com/what-is-cdn/, ????

[Amazon(2014)] Amazon. "About AWS." http://aws.amazon.com/about-aws/, 2014.

[Bebinger(2014)] Bebinger, Martha. "Health Connector Website Update, By The Numbers." *WBUR* .

[Beck et al.(2001)] Beck, Kent et al. "Manifesto for Agile Software Development." http://agilemanifesto.org/, 2001.

[Conaboy(2013)] Conaboy, Chelsea. "Snarls in retooled Mass. insurance site." *The Boston Globe* .

[Corbet(2013)] Corbet, Jonathan. "A GUIDE TO THE KERNEL DEVELOPMENT PROCESS." https://www.kernel.org/doc/Documentation/development-process/2.Process, 2013.

[of Transportation(2007)] of Transportation, US Department. "Systems Engineering for Intelligent Transport Systems." http://ops.fhwa.dot.gov/publications/seitsguide/seguide.pdf, 2007.

[Riley(2007)] Riley, Duncan. "Twitter Back On The Straight And Narrow: Interview With Founder Biz Stone." http://techcrunch.com/2007/05/30/twitter-back-on-the-straight-and-narrow/, 2007.

[US House(2013a)] US House, Committee on Energy and Commerce. "CMS Red Team Report." http://energycommerce.house.gov/sites/republicans.energycommerce.house.gov/files/Hearings/OI/2013 CMS-Red-Team-Discussion-Document.pdf, 2013a.

[US House(2013b)] US House, Committee on Energy and Commerce. "HHS Emails." http://energycommerce.house.gov/sites/republicans.energycommerce.house.gov/files/20131115-HHSemails-July2013.pdf, 2013b.