Case Study

Analysis Saving Obamacare

Les Chambers, Director, Chambers and Associates Pty Ltd
www.chambers.com.au

CA-2014-CSA001 Version 1.0, April, 2014

Summary

We analyse the case study, Saving Obamacare and answer the question: "Could software engineering discipline have saved the U.S. government's HealthCare.gov software project?"

Our analysis is structured using the relevant software engineering knowledge areas specified in the IEEE Software Engineering Body of Knowledge (SWEBOK). It also provides an ethical analysis using the criteria described in the IEEE/ACM Software Engineering Code of Ethics.

Contents

1. Method

We chose the SWEBOK as our framework for analysis because software project failures are, most often, traceable to failures of software development process and the list of essential processes together with the knowledge required to perform them is codified in the SWEBOK.

The software engineering approach to problem solving is: fix the process and you will fix the product. To this end, we analysed the major problems that occurred on the HealthCare.gov project classifying each problem as belonging to a particular SWEBOK knowledge area. We then summarised the software engineering approach to executing the process thus avoiding the unwanted outcome.

We note that detailed information on the HealthCare.gov project is currently scarce. Further, the accuracy of dates and other figures published in newspapers cannot be verified. Notwithstanding we have done our best with the information available, accepting it at face value as "objective evidence". In drawing conclusions there has also been an element of reading-between-the-lines based on past experience.

2. Summary of Conclusions

2.1. Causal Analysis

The failure of HealthCare.gov - severe enough to require the President of the United States to apologise to the American people on national television - was fundamentally a failure of management. Its root causes are summarised as follows:

a) **Lack of competence in systems integration.** The designated systems integrator, CMS, lacked the competence to plan and execute a project of this size. A high level of systems integration skill is required to manage projects of this type. HealthCare.gov had 55 participating organisations, more than 200 unique interfaces to external systems and an extremely tight schedule. This class of project requires detailed planning based on substantial constructive experience and tight control, none of which were in evidence.

b) **Requirements churn.** Changes in requirements were not adequately controlled. This had a flow-on effect to project estimates and schedules. Cost and time became unpredictable and therefore uncontrollable.

c) **Underestimation of required capital.** A 71 percent project cost overrun indicates that insufficient effort was applied to developing accurate estimates at the start.

d) **Essential requirements delivered late.** Software requirements essential to the choice of architectural solutions were delivered late. The prime example was the late decision to have all users authenticated before they shopped for a health insurer. This created a predictable bottleneck that could have been dealt with by changing the architecture. At that late stage however, it was not possible. Uninformed high level design decisions were therefore made that caused the system to be unusable on delivery. As a result, corrective action on the system architecture may be expensive.

e) **Inadequate testing.** Given a perceived immovable deployment date, the time and effort allocated to testing was grossly inadequate. Tests that were run were poorly designed, failing to properly simulate expected system loads.

f) **Inadequate quality management.** The delivered system was not fit for purpose. It featured unacceptable response times and failure to complete user transactions. Quality control was grossly inadequate with common use cases not being properly tested.

g) **Failure to act.** Despite compelling evidence to the contrary HealthCare.gov was deemed fit for purpose and duly deployed to satisfy a politically convenient go-live date. This was done despite the fact that the problems listed in this analysis were known to many senior project operatives beforehand. There was no effective process for communicating these problems to the senior people with the administrative power to delay deployment until the system could be proven fit for purpose. President Obama was clearly not well informed or advised. The absence of a will to act does not speak well of senior management ethics.

We note that these causal factors are not unique to this project. They follow a constantly recurring pattern of behaviour in projects of this size. That is, requirements poorly managed, unrealistic commitments made, schedules crashed and testing curtailed, resulting in poor quality software that fails in service.

Philosophers tell us that we must believe in something, not just for reassurance, but to be capable of action. If we believe, we act, and our actions define us. Experience has shown that there is often a chasm between knowledge and belief which results in people who know better failing to engage with obvious problems. So it was with Obamacare: the knowledge of proper process was there but there was a lack of belief in its effectiveness and therefore no action.
(More on software engineering belief systems at *Introducing Software Engineering* [CA ISE 14]: http://www.chambers.com.au/glossary/software_engineering.php)

2.2. Could Obamacare Have Been Saved by SE?

Could applying software engineering (SE) disciplines have saved Obamacare? Our answer is: it depends on the value placed on this discipline in the target business environment. SE processes exist to avoid all the problems experienced on HealthCare.gov. To derive benefit however, an organisation must have the will to practice them. We therefore qualify our answer in the context of two cases:

Case 1: Software engineering is valued. Where software engineering processes are valued the answer is yes. The software engineering discipline features mature and proven processes that avoid all the problems listed above. Effective requirements management, married with project estimation techniques typically produce reliable estimates. Overall systems integration management skills are well established and effective. Problem definition and corrective action disciplines are also strong. Delivering a system that has not completed adequate testing is a cultural impossibility. Further, one would hope that ethical considerations, more prevalent in engineering cultures, would have prevented a transparently defective system from being deployed.

Case 2: Software engineering not valued. Where software engineering processes are not valued the answer is no. These environments feature critical decision-making by senior people lacking competence in large systems software development. Failures occur when decision makers are not provided with, or choose to ignore, competent technical advice.

This analysis argues that HealthCare.gov was an example of Case 2. An unachievable political deadline was enforced on a large-scale software project where people who knew better failed to follow proper processes, with highly predictable results.

3. Proposed Corrective Action

The evidence does not provide sufficient detail to propose detailed corrective actions. However one thing is clear, ignorance of the imperatives of large system project management in senior government officials contributed to the failure. The individuals responsible for making commitments to the public at large, allocating capital and making critical decisions were either not technically competent or allowed their better judgement to be overridden by profit or political concerns.

Train Program Managers

As a starting point the U.S. government, and indeed all governments engaging in large-scale software intensive systems development, should consider substantially improving the quality of training provided to its programme managers (indeed anyone making critical technical decisions). In short, we recommend creating a dedicated full-time school to train selected people in the pragmatics of managing very large software intensive projects.

This approach has already been implemented by the U.S. Department of Defense with the Fort Belvoir Defense Acquisition University (DAU) (https://www.belvoir.army.mil/). This facility provides intensive training for candidate program managers. It also provides follow-up resources such as the Program Manager's e-toolkit (https://pmtoolkit.dau.mil/).

Strengthen the Project Management Culture

We note that, courses in project management technologies such as earned value management have been jointly developed by CMS and the DAU. It appears that the appropriate practices were in place but the CMS senior managers responsible for HealthCare.gov were either unable or unwilling to apply them. There was knowledge but no belief and therefore no action. Corrective action should therefore be to strengthen the project management culture. This can only be achieved by the example of senior management.

Make Decisions on Hard Data

Channels of communication to decision makers should be more direct and the quality of performance information improved. As a minimum, senior managers should be taught the basic principles and rules of software development. To make good decisions they must be able to ask developers insightful questions and form opinions from hard data rather than the "spin" so often delivered by senior technologists. This will only be possible if they have the skills to interpret the complex performance information that emerges from large projects and be aware of common project failure modes.

Take Action to Reduce Risk

A core problem was the failure of senior management to act when serious situations presented. As part of the solution, government projects should adopt a practice from the U.S. space program. NASA has a concept it calls "flight rules"[Hadfield 13]. Flight rules provide practical guidance for recurring decisions. Distilled from a hard-earned body of knowledge derived from past, and sometimes painful, experience - NASA has been codifying flight rules since the early 1960s; many from fatal accidents - they identify situations and provide clear guidance on what to do if X occurs and why.

Flight rules create certainty when people need to make hard decisions. They provide clarity to senior managers because these rules are **never** to be broken, regardless of what cost, scope, time or technical pressures are being applied to a project. Chief among them is:

We shall NEVER launch a system that has not satisfactorily completed its test program.

4. Project Analysis

4.1. Introduction

As stated, our analysis of the HealthCare.gov project is based on relevant SWEBOK knowledge areas. Our assertion is that failure to apply codified software engineering knowledge was the root cause of the failure.

Each of the following subsections identifies the unwanted outcomes relevant to each knowledge area and summarises a typical software engineering approach to the problem.

4.2. Software Requirements

4.2.1. Problems

Architecture drivers specified too late. It was reported that:

[customer representatives were] debating whether consumers should be required to register and create password-protected accounts before they could shop for health plans,

... three months before the go-live date. Had it been captured earlier, this requirement would have had a significant impact on the architectural design of the system. It is therefore likely that, in the light of this requirement, the target architecture was not appropriate to deliver the required functionality with the required performance. Modifying the fundamental structure of the system to accommodate this requirement was not possible in the time available.

Evidence:

Another outcome of changing the policy was the complexity in rewriting what was already an unsustainable amount of code and the impact on website efficiency. Had the policy remained to only provide PII [personally identifiable information] when it was absolutely necessary to complete a financial transaction, it is quite probable many of the security issues would not have arisen.

- Testimony of Morgan Wright, CEO, Crowd Sourced Investigations LLC*, Before the House Committee on Science, Space, and Technology,*19 Nov, 2013

Requirements churn. It was reported that:

hardware and software requirements on the project had been modified seven times in the last 10 months.

This phenomenon typically creates rework in design and coding. Excessive rework is a common causal factor in late delivery.

Inadequate information for cost estimating. Unstable requirements make accurate cost estimating impossible - as evidenced by the cost blowout from $292 million to $500 million.

Non-functional requirements ignored. Critical non-functional requirements such as response times were either not specified, not validated or completely ignored. Evidence: the website registration page which took 71 seconds to load.

Lack of interface requirements. Requirements for interfacing with insurance companies and other government agencies were either not specified or ignored. Evidence: the HTTP 503 error codes that were being returned, but not handled.

4.2.2. Software Engineering Approach

Requirements engineering is a well-established and well understood software engineering discipline.
(for further details refer video: http://www.chambers.com.au/video_public/specifying_software_requirements.php)

Key requirements engineering practices relevant to HealthCare.gov are:

Capture architecturally significant requirements early. A small subset of the overall functional, quality and business requirements shape all systems architectures. Software engineers seek to capture these requirements early so some architectural design can proceed in parallel with requirements capture and specification. For example, in the case of HealthCare.gov, the relevant architectural pattern was a highly distributed service oriented architecture with many performance constraints. Early discovery of these critical requirements would have supported a more informed choice of target architecture and left time for prototyping and validation of alternatives.

Control requirements churn. Once the initial requirements are stabilised and baselined, further additions and changes are subject to formal approval through a Configuration Control Board (CCB). For further comment, refer to the section on Configuration Management below.

We note that the McKinsey & Co red team review identified requirements churn as a major problem and gave good advice:
Align on the scope of the initial release. Lock down the scope of HealthCare.gov version 1.0 by 8 April, 2013.
The advice was not taken.

Create a minimum viable product. Faced with an immovable deployment date, one of the key roles of the CCB is to select the minimal set of requirements that will result in a viable software product. This involves a process of "scrubbing" requirements, that is, preventing unnecessary gold-plating and feature creep by removing (or delaying implementation of) non-essential requirements.

Perform strict interface control. Software engineers recognise non-performing interfaces as a major risk factor in projects of this nature. The problem is solved by devoting engineers to interface management. This involves forming agreements between interfacing organisations, developing interface requirements documents and performing dedicated interface design and validation activities. Interface simulators are often used to decouple the development of individual subsystems from the overall system development, allowing off-line validation prior to system integration.
(for further details refer: http://www.chambers.com.au/glossary/interface_management_plan.php)

Quantify non-functional requirements. Non-functional requirements are an attribute of a system rather than a description of what it does. For example, response time, capacity, scalability, availability and usability. Some non-functional requirements, for example response time and availability, are critical drivers of choices in architecture. They therefore need to be quantified early to support architectural design and test strategy development.

4.3. Software Design

4.3.1. Problems

Architectural drivers not specified early. As discussed in the requirements section above, system architectures are chosen on the basis of a small subset of quality drivers that determine which architectural patterns and strategies are relevant to the target application. They therefore must be specified early as it is expensive for a project to recover from an inappropriate choice of architecture. A well designed architecture can scale and survive changes in user requirements.

With HealthCare.gov, the late decision to require user sign up before allowing browsing for health care plans drove massive traffic to one service (user authentication) causing a bottleneck. If this had been known earlier, more effective scalability could have been designed into this service to withstand the load.

Design analysis inadequate. The system architecture was not analysed to determine whether or not it would deliver the required response times with the projected user loadings. Evidence the speculation that:
The site basically does a distributed denial of service on itself.
This indicates a lack of performance analysis.

Architecture not scalable. The system did not accommodate the highly predictable initial rush of users. Evidence:
In the first 10 days, the site received 14.6 million unique visits.
In the context of scalability, HealthCare.gov is no different to road and rail transportation applications and electricity distribution systems. All these systems are designed to carry peak loads with the full expectation that they will be underutilised most of the time. The rationale is that if a system of this kind cannot carry peak loads it is not fit for purpose.

Inadequate design for usability. Not enough thought was given to making the system usable. Usability is a critical quality factor in design for web applications. Users are expected to interact with these systems with no training. Interactive websites therefore need to be highly intuitive. In the case of HealthCare.gov, users were frustrated by the system's tendency to present a part-loaded web page, giving the appearance of a complete on-screen form presentation, when in fact, this was not the case (it was still loading).

Inadequate interface design. The system was implemented with a service oriented architecture (SOA). It features interfaces with more than 100 external services. To work properly, service consumers must be provided with clearly specified and stable rules for the operation of each service. This is achieved with published application programming interface specifications (APIs). It is clear that the correct operation of all APIs was not investigated or properly validated. Evidence:
... [servers were returning] *HTTP 503 status code as the response to ... requests for service. ... this is a "Service Unavailable" response. It means that the HTTP server was able to accept the request but can't actually do anything with it because there are server side problems.*

4.3.2. Software Engineering Approach

Software engineering design practices feature architecture driven development and a strong focus on interface control.
(for further details refer: http://www.chambers.com.au/glossary/architecture.php)

Salient elements include:

Identify architectural drivers. Analyse the requirements and select the functional, quality and business requirements that shape the architecture. For example, HealthCare.gov needed to communicate with multiple external agencies (service providers). Its runtime performance was therefore a critical quality factor. Engineers would have ensured that the quality factors that drove choice of architecture were measurable. For example, user response times together with the transaction rates required for each service would have been quantified and thoroughly validated.

Form an architecture team. Common best practice calls for a dedicated team of architects responsible for designing, building and optimising the core architecture. Clustered around this team are various application development teams. It is unknown if this approach was taken on HealthCare.gov. However, it seems that communications were not effective with teams developing various web services, as poor service performance seemed to be a surprise to the overall systems integrator on deployment.

Choose an architectural style. The requirement for HealthCare.gov to communicate with multiple service providers indicated a service-oriented architecture style. This style was applied with HealthCare.gov, but there is little evidence of the rigour in modelling and analysis required to make it successful.

Choose design entities. Choose the "things" that will require detailed design. For example, given that HealthCare.gov interfaced with many services there existed a need for several architectural components that managed the interface with each of these services. Given the failure of many HealthCare.gov interfaces, interface control documentation (the repository of interface design) was probably deficient.

Analyse technical risk. Consider high risk components in terms of complexity, size, cost of development, cost of maintenance and potential performance problems. With HealthCare.gov high risk components would have been placed early on the development schedule to allow time to recover from mistakes in design approach. For example, critical interfaces such as the interface to the user authentication system (the EIDM) would be thoroughly specified, developed and tested early in the project. CGI took their eye off the ball in this regard, leaving no option but to apologise to a U.S. Senate oversight committee. Evidence:
The EIDM serves as the "front door" to the Federal Exchange that a user must pass through before entering the FFM. Unfortunately, the EIDM created a bottleneck that prevented the vast majority of users from accessing the FFM.

Develop alternative solutions. Create skeletal architectures to validate various architectural designs. Projects may choose to implement more than one architectural approach. Focus on the high risk elements of the system. This may be done in multiple cycles with stakeholder review of outcomes and modification of the design approach for the next cycle. With HealthCare.gov, this would have revealed a non-performing architecture early in the project.

Analyse the target architecture. Establish that the target architecture will comply with functional and non-functional requirements using analysis techniques such as reviews, simulation, prototyping and mathematical modeling.

CGI management seemed to believe that performance analysis is something you do after deployment (with the assistance of the user community) and that achieving the required performance was a simple case of system tuning.

Evidence CGI Senior Vice President, Cheryl Campbell's testimony:
Unfortunately, in systems this complex with so many concurrent users, it is not unusual to discover problems that need to be addressed once the software goes into a live production environment. This is true regardless of the level of formal end-to-end performance testing -- no amount of testing within reasonable time limits can adequately replicate a live environment of this nature.

In reality, having live users connected exposed major design deficiencies that should have been identified and corrected well before deployment.

Select and communicate a target architecture. The architecture group finalises the architecture description and communicates its structure and conformance requirements to application developers both in writing and in review meetings. With HealthCare.gov, this approach would have substantially reduced the number of interface anomalies with service providers.

Build application evaluation prototypes. When the architecture reaches a stage that will support layered applications, build application evaluation prototypes prioritising by risk. For example, choose applications that require challenging throughput performance. Further, given that some services used legacy code, SEs would have conducted compliance testing early to leave time to deal with problems. Solutions may include acceptance of graceful degradation under peak load.

Integrate early and often. As early as possible assemble a skeletal system. Conduct testing to identify problems with the design approach. This also gives the project something to show users. Progressively add application components and test for quality factors such as transaction rates. Evidence of multiple interface failures indicates that HealthCare.gov was "thrown together" at the last moment.

Conduct conformance reviews where application developers provide evidence to the architects that their application components conform to the requirements of the architecture. For example, in the case of HealthCare.gov, the architecture would dictate required transaction rates from each service. If that was not achievable it would be discovered early and rectified.

Make progress visible. Progressive integration and test makes project progress highly visible to project management and the customer. We note that HealthCare.gov project reviews were redolent with wishful thinking based on poor information. An uncompromising implementation of the earned value management discipline would have predicted that the mandated to go-live date was not realistic.

Instrument the system. On a project of the scale of HealthCare.gov a software engineer would have included system instrumentation functions to monitor performance. For example, functions that identified latency in data communications on the multiplicity of interfaces with remote services. This would have supported system tuning and rapid identification of problems, including the bottlenecks actually experienced when HealthCare.gov was deployed.

4.4. Software Construction

4.4.1. Problems

Inadequate time to write quality code. Coding started five months before the go-live date. Evidence:

... the government was so slow in issuing specifications that the firm did not start writing software until March 2013.

It is highly likely that the code is of poor quality given the crashed schedule. There would have been very little opportunity for code review and the associated rework and regression testing that it usually generates.

Insufficient reuse. There was speculation that too much of the system was developed from scratch. Evidence:

... It was reported that such a large "stash" of code could indicate that contractors “may be writing their own code in many places where they'd be better off relying on open-source external libraries."

There is no evidence to suggest that techniques such as product line engineering were applicable here, however there may have been opportunities for cost reductions, especially in the area of interface control.

4.4.2. Software Engineering Approach

Software engineering projects are typically conducted in fixed price, fixed scope, fixed time environments. Disciplines for negotiating these interacting parameters are strong. For example, if the customer requires the product in less time either productivity must be improved, usually at higher cost, or scope reduced.

Negotiate a minimum viable product. When a development team is presented with an immovable deadline, negotiation of project scope is usually necessary. If the deadline must be met and there is insufficient time to deliver all the required functions, some functions must be eliminated from the scope of the release. Software engineers would revisit the functional capabilities list and work with the customer to identify the minimum set of functions that would produce a viable product. If functionality cannot be reduced the date must be moved. The alternative is to take shortcuts, such as eliminating code reviews and curtailing testing, which all result in poor quality code and a very expensive failure in service (as demonstrated on HealthCare.gov).

Investigate reuse. Software engineers have developed formal processes for reusing code components. The motivation is to deliver high quality code with a shorter time to market. The core strategy is to develop a basic platform from which many unique end products can be derived by addition of small amounts of custom developed code. This discipline is called "Product Line Engineering". There is insufficient evidence to indicate whether or not this approach would have worked with HealthCare.gov. However, given that there were 55 organisations developing or enhancing web services there may have been an opportunity for a common web service platform developed by a central group and reused throughout the application.

A related discipline is model-driven development, where code is generated directly from detailed business models, with the aid of predefined production rules relevant to the application environment. This approach may have been relevant to transaction processing and interface control software.

4.5. Software Testing

4.5.1. Problems

Inadequate time in testing. The time ("weeks") allocated for final system testing was grossly inadequate for a system of this size and complexity (allegedly 500 million lines of code - we note that this figure is questionable, given the relatively short project duration).
Evidence:

... testing was conducted in the final weeks before the site went live.

This was largely the result of an immovable deployment deadline which resulted in quality control activities being curtailed and a system, unfit for purpose, being visited on its user community.

Inadequate test design. Testing of common use cases was not conducted.
Evidence:

... A week before the launch, the site hadn't been tested to see whether a single user could get all the way through the process.

Inadequate performance testing. The system was not adequately stress tested. The test environment was not capable of simulating the expected load which was of the order of hundreds of thousands of concurrent users.
Evidence:

... mere days before the launch date, [CMS] *tested the system's ability to handle tens of thousands of users at once. It crashed after a few hundred.*

Corrective action on defects not tracked. There was no formal system to verify that defects had been rectified prior to going live.
Evidence:

... [QSSI] *didn't have direct knowledge of how any fixes were implemented.*

Inadequate systems integration. CMS was not an effective systems integrator. The parallel schedules with many interfaces being integrated in a very short space of time belies wishful thinking. That is, that all interfaces will function correctly when systems are connected (known in the profession as the day the miracle happens). This is seldom the case. Professional systems integrators progressively integrate and test systems, leaving time for corrective action on defects and regression testing. It is likely that specifications were inadequate and testing of interfaces to external agencies ineffective. The McKinsey & Co red team review highlighted this issue.
Evidence the project threats:

Parallel stacking of all phases. Many activities were being conducted in parallel.
Perceived insufficient time for end-to-end testing given the go-live date.

Failure to act on unfavourable test reports. Despite the common knowledge of test failures among CMS, CGI, QSSI and many of the insurance companies who warned against deploying the system, no action was taken to delay deployment to allow more time for proper testing. Among the cohort of developers in a position to know, the presence of poor quality software became a wilfully unknown known.

Evidence:

... "We informed CMS that more testing was necessary," Andrew Slavitt, group vice president of Optum, the division of United Health that owns QSSI, told the House Energy and Commerce Committee on Oct. 24. He noted that a testing period for the project ideally would have lasted months rather than weeks

....
[QSSI] *made no* [formal] *recommendations about whether CMS should delay the rollout of HeathCare.gov because such advice wasn't in the scope of their work.*

4.5.2. Software Engineering Approach

In a software engineering project testing can represent between 15 and 30 percent of total project effort. In some critical applications it can go as high as 50 percent. It is a planned and tightly managed activity.
(for further details refer: http://www.chambers.com.au/glossary/software_testing.php)

Measures taken include:

Assign responsibility for testing. A test authority is appointed with overall responsibility for the test program. Responsibility for subsystem level testing is devolved to contractors.

The primary responsibility of the test authority is to determine, based on the evidence of test reports, that the system under test is fit for deployment. If such person did exist on HealthCare.gov, this responsibility was not discharged.

Plan testing. Test planning commences early in the project when requirements are stabilised. This is the responsibility of the test authority (in this case QSSI under the supervision of CMS). Some responsibility of for test planning is devolved to development organisations. For example, test plans for individual web services are created by contractors and provided to the test authority for approval. The outcome of the test planning exercise is an unambiguous statement of the overall approach to testing, test schedules, test resource requirements (people, data and tools) and test deliverables.
Of particular importance in a project of this size is planning for overall systems integration. Identifying the order in which software products will be integrated and progressively tested. An insightful analysis of the HealthCare.gov schedule probably would have revealed an infinite peak of activity at the end of the project. There was simply not enough time set aside for proper integration and testing.

Develop test environment. Large projects typically require extensive test rigs, test simulators and test tools to support effective testing. In the case of HealthCare.gov, designing and building a test environment to simulate millions of concurrent users would be a substantial project within a project. It is clear that CGI and QSSI did not feel that simulating the massive user loadings this system would experience on deployment was necessary. An engineering project would class it as a nonnegotiable necessity.

Conduct testing. Engineering projects conduct testing at the following levels:

- Module testing: where individual software modules are tested (usually by the developer).
- Unit testing: where many modules are integrated and tested as a unit (usually by an independent test group).
- Integration testing: where various software units are integrated into systems and subsystems and progressively tested.
- System testing: where a system is tested as a whole. This includes testing the system under stress to validate non-functional requirements such as transaction rates and response times.
- Acceptance testing: where the correct working of the system is demonstrated to customer.

- Regression testing: where elements of the system are retested subsequent to corrective action on defects identified in previous tests.

We note that the non-attendance of CMS representatives at end-to-end testing indicated abdication of its responsibilities in this area.

Record test results and monitor completion of corrective action. All test results, including defect reports are documented and corrective action tracked. Test logs give program management a clear view of the state of the product under test. High defect densities give early warning of problems. In the case of HealthCare.gov these reports either did not exist or were ignored. Further, the test authority is responsible for ensuring that corrective action is driven to completion. This was not the case with HealthCare.gov.
Evidence:
... a *QSSI representative testified that he shared testing results with CMS but had no direct knowledge of how any fixes were implemented.*

Summarize testing. The test authority produces reports on the quality of the software under test. The role of a test summary report this to evaluate the system's readiness for deployment to the user environment. In the case of HealthCare.gov a test summary report either did not exist or was ignored. Further, QSSI seemed unaware or disinterested in the status of the system under test, having no knowledge of whether or not defects had been fixed.
(for details of test documentation refer: http://www.chambers.com.au/glossary/test_documentation.php)

4.6. Software Maintenance

4.6.1. Problems

Expensive corrective action. The speculation that:
[Accenture's] technology 'do over' ... will likely result in hundreds of millions of dollars in additional spending",
is credible. It is likely that this expense is the result of poor architectural design caused by late delivery of essential requirements that determine the appropriate architectural approach. Experience has shown that inappropriate architectures are expensive to rework requiring large-scale redesign and redevelopment.

Excessive technical debt. Crashed schedules typically cause programmers to take sub optimal shortcuts producing programs that are expensive to maintain - a behavior now well known as placing your organisation in "technical debt". Funds allocated to paying down that technical debt will likely exceed the system's initial development cost. The act of "paying down technical debt" involves rewriting code to improve performance or maintainability without adding new functionality.

4.6.2. Software Engineering Approach

Highly complex software products are often expensive to own, sometimes to the point where the owner finds it attractive to abandon a system altogether. Cost of ownership must therefore be considered early in a project and maintainability designed into systems.

Consider maintainability in requirements. In the requirements phase software engineers consider the impact of owning a software product. For example, with HealthCare.gov the mechanisms by which CMS would integrate the inevitable post-deployment changes to various interfacing web services would be canvassed. Software engineers would have captured software requirements dealing with cost-effective methods for accommodating changes to web service APIs throughout the product's operational life.

Consider maintainability in design. Software engineers design systems to cater for predictable product evolution. The strategy is to localise the impact of change. That is, to prevent changes in an architectural component from rippling through the entire system. This is why object oriented design (OOD) practices have been so successful. For example, the OOD concept of data hiding calls for designs where individual subsystems maintain "private" internal data structures, which may change at will, while presenting and unchanging interface to the outside world. This localises change and substantially reduces rework across the system.

Simplify. Overly complex designs are expensive to maintain. Design reviews focus on complexity with a view to keeping things simple where possible.

4.7. Software Configuration Management

4.7.1. Problems

No pre-deployment functional audit. There was no formal process for checking that the system about to be deployed actually delivered its required functions.

System integrity at risk. HealthCare.gov was developed by 55 organisations. The evidence indicates that it ... *could contain about 500 million lines of code*. This, of itself, does not create problems if managed correctly, however, it raises questions regarding the integrity of all system artefacts. For example: who owns the software (that is, who is responsible for preserving its integrity)? Is it stored in a secure facility? Is it configuration managed? That is, who is responsible for recording which versions of which software make up the current release of the system. Who is responsible for integrating new releases of software, what specification documents describe which software products ... and so on. Given the disorganised and understaffed state of the designated systems integrator (CMS) it is likely that the integrity of the system has not been adequately secured.

A common outcome of poor configuration management is system failures on upgrades due to integrating the wrong versions of software with the existing code base.

4.7.2. Software Engineering Approach

Large projects often suffer from too little configuration management, introduced too late. Software engineering calls for configuration management activities to commence at project establishment.
(for further details refer: http://www.chambers.com.au/glossary/configuration_management.php)

Typical measures include:

Create a configuration management organisation. A system-wide configuration manager is appointed with appropriate staff.

Planning. Develop a Configuration Management Plan describing the administrative actions to be taken to identify the configuration of the system at discrete points in its development life cycle for the purposes of systematically controlling changes to the configuration and maintaining its integrity and traceability throughout its development life cycle.

Configuration identification. Consistent configuration identification practices are mandated across all contractors. This includes consistent code and documentation identification and identification of aggregates of development artefacts such as the various development baselines.
(for further details refer: http://www.chambers.com.au/glossary/configuration_identification.php)

Configuration control. A Configuration Control Board (CCB) is established to formally control changes to the configuration. The CCB is staffed by senior managers with business knowledge and budgetary control. The CCB includes representatives from customer and developer organisations. The CCB becomes the main vehicle for project scope control. In the case of HealthCare.gov a competent CCB would have made requirements churn visible at high levels of management and put a stop to it. Or if the requested changes were unavoidable - that is, driven by government policy made on-the-fly and not under the control of project management - the CCB's role would have been to ensure that the go-live date was extended to accommodate proper integration and testing of late-breaking requirements.
(for further details refer: http://www.chambers.com.au/glossary/configuration_control.php)

Configuration status accounting. A central register is maintained to record the status of the configuration in terms of current baselines and proposed changes to those baselines. HealthCare.gov test items that had failed system testing would have been tagged and prevented from being integrated with the production baseline.
(for further details refer: http://www.chambers.com.au/glossary/configuration_status_accounting.php)

Configuration audit. The configuration management organisation is tasked with insuring that functional and physical audits are performed on all subsystems prior to deployment. As deployed, HealthCare.gov would have failed a functional configuration audit as it could not demonstrate completion of common user transactions.
(for further details refer: http://www.chambers.com.au/glossary/configuration_audit.php)

Release. Engineering projects tightly control the release of software to the production environment. This includes providing evidence that a new release will not degrade system capabilities, developing release procedures and creating release notes.

4.8. Software Engineering Management

4.8.1. Problems

Over commitment. Commitments were made to the American people that could not be delivered. Systems of this size and complexity, engineered properly, are expensive and time-consuming to build. Further, cost estimates are typically imprecise at project commencement. It is clear that commitments were made without the necessary detailed investigation of scope, cost and schedule. When it became clear that the project could not deliver on its commitments, no action was taken.

Program office staffing inadequate. CMS, the designated systems integrator, did not apply adequate resources to program planning and execution management. The result was: critical systems integration tasks such as interface control were not performed.
Evidence:
... Dr. Donald M. Berwick, the CMS administrator in 2010 and 2011, said the time and budgetary pressures were a constant worry. "The staff was heroic and dedicated, but we did not have enough money, and we all knew that," he said in an interview.

Ineffective project tracking and control. The earned value management (EVM) discipline was well-known within CMS.
Evidence:
... According to a Project Auditor's report elements of earned value management (EVM) were applied. There is evidence that, as an organisation, CMS is committed to the earned value management discipline. EVM courses have been jointly developed by CMS and the Defense Acquisition University ...

However, evidence indicates that EVM techniques were not aggressively applied. This was probably due to deadline pressure and requirements churn, which destroys the stability of planned value estimates.

Crashed schedule. Late specification of system requirements made the go-live date unrealistic.
Evidence:
... Though the CGI contract was awarded 30 September 2011, the government was so slow in issuing specifications that the firm did not start writing software until March 2013, according to people familiar with the process. ...

Failure to extend the go-live date to compensate, caused the developers to take shortcuts, the most damaging of which was curtailing the test program and ignoring major defects detected in the tests that were conducted.
Evidence:
... He noted that a testing period for the project ideally would have lasted months rather than weeks. ... mere days before the launch date, [CMS] *tested the system's ability to handle tens of thousands of users at once. It crashed after a few hundred ...*
We note that crashing a schedule (that is, placing and unrealistic end date on a project and hoping a miracle will happen on the day) typically produces schedules for individual development organisations - back calculated from the delivery date - that are recognised as unrealistic and therefore ignored by the people doing the work. This defeats the purpose of the scheduling practice, leaving individual project managers without visibility of performance. Without a realistic plan there is no benchmark to judge performance and therefore no control.

Unrealistic estimates. The initial estimates were wildly optimistic. This was probably due to incomplete or incorrect requirements and ineffective estimating techniques. Evidence: the initial estimate of $292 million was overrun by $208 million.

Inadequate scope control. Incomplete requirements resulted in inadequate scope definition. Uncontrolled requirements change resulted in uncontrolled scope.
Evidence:
... As late as the last week of September, 2013, [eight weeks before deployment] *officials were still changing features of the web site.*

Inadequate systems integration – especially interface control. Multiple interfaces and multiple development organisations are classical high risk factors in large projects. Failure to manage interface requirements, design and testing and overall systems integration was a major contributing factor to system non-performance.
Evidence:
... no senior executive at the CMS was designated as the point person for integrating the various components of the system.

Ineffective risk management. A risk is defined as an unwanted outcome that has a finite probability of occurrence and a level of severity. Risks are considered when a project can take action now to

reduce the probability of occurrence at some point in the future. It was common knowledge throughout the project, including in the ranks of senior managers, that the system would not be ready for deployment on the scheduled delivery date.
Evidence:
... a test group of insurance companies had warned CMS a month earlier not to launch the site because of problems with the system.

Further, six months prior to the go-live date the McKinsey & Co red team review accurately identified all the risk scenarios that would ultimately come to pass.
Evidence:
The top risks identified were:

1. *The Federal Exchange would be unavailable on deployment due to system failure.*
2. *This would force long manual processing times.*
3. *Failure to resolve post launch issues rapidly.*
4. *Health care plan data would not be loaded in time. Several health care providers would therefore not offer health care plans.*

Despite this accumulated knowledge no one took action to reduce risk. That is, to get requirements churn under control and extend the go-live date.

Ineffective software acquisition and supplier contract management. CMS' software development vendor evaluation process was flawed. Despite its Software Engineering Institute Capability Maturity rating of five (highest capability), the prime contractor, CGI had a track record of non-performance on previous projects.
Evidence:
...[CGI] *had non-performed in a number of projects, including ... a Mississippi tax system. In* [this] *case,* [CGI] *had to pay $US474 million for its failure. Despite these red flags, CMS selected the firm for the project.*

4.8.2. Software Engineering Approach

A software engineering project typically expends at least 10 percent of project funds on management. With HealthCare.gov, using the actual cost at deployment as a guide, this would have created a project office with a minimum $50 million budget. Software engineering project office activities would include:

Developing an overall project plan. The core of the plan is a program work breakdown structure and a project master schedule. The work breakdown structure (WBS) integrates contributions from the WBSs of all contract organisations.
(for further details refer video: http://www.chambers.com.au/video_public/the_project_planning_process.php)

Develop estimates. Estimates are created using the program work breakdown structure. Estimates are updated on a monthly basis. Standard practice includes the use of management reserve, a "bucket" of money set aside for unforeseeable events during project execution. For example, non-performance of legacy software from an external insurance company, requiring an unplanned rewrite. With HealthCare.gov, the initial lack of precision in the estimates should have triggered allocation of substantial funds to management reserve.

Indicate estimate precision. Estimates of cost and time are typically imprecise in the early stages of a project. Their accuracy is largely dependent on the completeness and correctness of software requirements. Software engineers always provide an estimate and a precision (for example, $X plus or minus *Y* percent). This "realism in estimating" avoids giving customers unrealistic expectations. With the benefit of hindsight HealthCare.gov cost estimates would have been presented with precisions in the range 50 to 100 percent. This would have given the politicians pause when hanging their hats on an unrealistic delivery date.

Track progress. An earned value management discipline is imposed on all parties to the project to provide realistic and predictive information on project progress. The earned value discipline allocates the original budgeted value of a deliverable when it is completed, giving a realistic picture of the value of work actually performed (as opposed to the dollars spent).
(for further details refer video: http://www.chambers.com.au/glossary/earned_value_management.php)

Had HealthCare.gov implemented an effective earned value management discipline it is likely that a project overrun would have been predicted and dealt with early in the project.

Maintain the project schedule. An accurate schedule is kept up-to-date throughout the project focusing on the current best estimate for the end-date.

Supervise engineering specialty groups. A project the size of HealthCare.gov would have engineering support groups such as: verification and validation, configuration management and quality management. In some projects these responsibilities are delegated to a dedicated systems engineering management group responsible for technical oversight. If any of these specialty groups did exist on HealthCare.gov, they were ineffective.

Manage risk. Software engineering projects identify risks, develop risk management strategies, take actions to reduce risk and monitor their effectiveness. In the case of HealthCare.gov, late delivery of critical system requirements would be flagged as a high risk item and dealt with as early as possible.

Experienced software engineers are well aware of many classical risk factors. HealthCare.gov was redolent with many of the well-known risk raisers, for example: multiple development organisations on the critical path, multiple interfaces and business processes not well understood (this system had to comply with new legislation, with policies being developed on-the-fly). In the case of interfaces, the engineering response would have been to apply people and funds to ensuring that all internal and external interfaces were precisely defined and tested prior to deployment. In the case of policy formulation, the client would have been kept well appraised of the impact of new and changed policies on project cost, scope and schedule.
(for further details refer video: http://www.chambers.com.au/video_public/risk_planning_process.php)

Monitor subcontracted software development. Supervise the development of requirements for subsystem development. Issue bid packages and evaluate responses. This can involve evaluating not only a vendor's offering, but also the capability maturity of the project team that will do the work. For example, an evaluation of CGI's software development capabilities based on past performance (not on CGI's Capability Maturity Model rating) would have eliminated them from contention as the lead developer.

> ***We note that it is a common problem that, whereas a company may have a world-renowned brand and a stellar technical reputation, this can count for nothing if the people doing the work on a particular project do not have the appropriate skills.***

It is interesting to note that the CGI team developing the system's front end (the Federally Facilitated Marketplace - FFM) was rated by the Software Engineering Institute as CMM level 5.
Evidence Cheryl Campbell's testimony:

CGI is widely recognized by independent parties for its expertise in IT systems and software design, such as certification of the CGI Federal team delivering the FFM by Carnegie Mellon's Software Engineering Institute—the leading software certification body—as having the highest Capability Maturity Model Integration rating: 5 out of 5.

This rating was not reflected in their performance on HealthCare.gov.

One mitigating factor could have been that the CMM rating applied only to CGI, who were one of 55 organisations involved. Had this rating applied to all parties, especially CMS, there would have been a better outcome. It is clear that CMS' immaturity in systems integration had a major negative impact on the project.

Communicate with stakeholders. Develop and execute a strategy for communicating with all parties to the project. A key element of project communication is disseminating project performance information to the customer and all parties to the project. Active communication results in show-stopping problems being made visible to the people who can drive solutions. This did not happen with HealthCare.gov. Many people understood the problems but no one took action to communicate and resolve them.

4.9. Software Engineering Models and Methods

4.9.1. Problems

Inadequate analysis of design. Given the difficulty of creating a test environment to simulate millions of users accessing hundreds of internal and remote third-party services, substantial effort should have been applied to modelling and analysis and prediction of system performance prior to deployment. This was either not done or inadequate.

4.9.2. Software Engineering Approach

The software engineering approach is to understand how something works by modelling its behaviour. Models then provide a mechanism for problem analysis and design optimisation. With Healthcare.gov models would be applied as follows:

Create models. In the early stages of architectural design, models or prototypes would have been created to analyse the high risk elements of the architecture. An obvious risk factor with HealthCare.gov was the non-performance of interfaces to external systems such as other government departments and health insurance companies. Engineering projects deal with this problem by modelling, analysing and testing both parties to the interface separately prior to integration. This is often necessary because the development schedules of both parties do not align, leaving many unproven interfaces to be tested at the end of the project, requiring an infinite peak of unavailable resources ... resulting in inadequate testing or no testing at all.

In a software engineering project the HealthCare.gov software would have been tested by simulating the APIs of each external service. At the other end of the interface, the performance and functionality of each external service would have been validated with a test simulator. This would have eliminated surprises when external services were integrated with core system.

We note that this approach adds cost to a project, but, in the case of HealthCare.gov, the cost would have been several orders of magnitude less than the losses incurred by ineffective, interface testing.

Perform analysis. Interaction analysis would have been performed on the data communications and control flow relations between architectural components, followed by refinement of the architecture to deliver the required performance.

4.10. Software Quality

4.10.1. Problems

Poor quality system. The system, as delivered, was not fit for purpose, failing in service at a cost of hundreds of millions of dollars to the U.S. government.

Inadequate quality assurance. Problems encountered on HealthCare.gov, such as interface failures, where service provider errors were not properly handled by the requesting software, are symptomatic of inadequate quality assurance. These problems are typically found early in a project in the context of interface control document review.

Inadequate quality control. End-to-end testing constitutes the final quality control activity for any system. Inadequate testing, in concert with failure to rectify major defects indicates a failed quality control process.

4.10.2. Software Engineering Approach

Software engineering principles hold that quality is an essential product attribute that can only be reliably delivered by assigning people to the task. This is in stark contrast to the popular view that quality is everyone's responsibility and therefore does not require a dedicated organisation.

Software engineering quality activities include:

Developing quality plans. A quality plan describes a project's strategy for delivering a software product that is fit for purpose from the point of view of the development organisation and the customer. A quality plan describes quality criteria, quality methods and quality responsibilities. It also identifies people and other resource requirements to fulfil the quality management mission.

Appointing people with responsibility for quality. A quality representative ensures that standards and procedures exist, are up-to-date, reflect accepted best practice and are followed. The representative also verifies that planned quality assurance and control activities are adequate and are followed.
(for further details refer video: http://www.chambers.com.au/video_public/qms_framework.php)

Implementing quality assurance. Engineering projects introduce quality assurance processes to give confidence that defects will not be injected into software products in the first place. In the inevitable situation where people make mistakes, the role of quality assurance is to find the defects as close as possible to the point where they are injected. At this point they are inexpensive to rectify. Measures include standard operating procedures and documentation and code reviews. The problems encountered on HealthCare.gov, are symptomatic of a lack of all pervasive quality assurance.

Implementing quality control. Software engineering projects ensure that all deliverables (documentation and code) are reviewed and/or tested against predefined criteria and that

documented evidence of these tests is maintained.
(for further details refer video: http://www.chambers.com.au/video_public/qms_architecture.php)

It is unknown if HealthCare.gov had a quality manager. If that role did exist its incumbent would be ethically bound not to sign off on the end product.

4.11. Software Security

4.11.1. Problems

Users exposed to identity theft. The system, as deployed, was vulnerable to theft of personally identifiable information (PII) that would expose its users to identity theft. The system was vulnerable to both insider attacks and external attacks.

Evidence:
... [a bug detected after deployment] would have allowed an attacker to take over a customer's whole account in the insurance hub. ... To have discovered this major deficiency after launch only reinforces the conclusion that the site lacks both the proper security controls and comprehensive security test plan."
...
This shows a lack of understanding for the consequences to consumers and the protection of their PII. ... This creates massive opportunity for fraud, scams, deceptive trade practices, identity theft and more. Much of this is playing out right now.

No management focus on security. The project lacked a security lead, that is, someone whose sole responsibility was to develop security policy, security plans and security tests and ensure that security measures were adopted uniformly across the project.

Evidence:
For a system dealing with what will be one of the largest collections of personally identifiable information (PII), and certain to be the target of malicious attacks and intrusions, a lack of a clearly defined and qualified security lead is inconsistent with accepted practices.

No security testing. No third party end-to-end security testing was carried out.

Evidence:
... a lack of, and inability to conduct, an end to end security test on the production system. The number of contractors and absence of an apparent overall security lead indicates no one was in possession of a comprehensive, top-down view of the full security posture.

No policies to counter insider threats. There was no process to determine that the people operating the system, and in possession of the personal details of millions of Americans, were of good character.

Evidence:
... The most troubling aspect [of HealthCare.gov is] the lack of a personnel policy that requires background checks for individuals with access to PII or sensitive information systems.

During testimony on November 6, 2013, secretary Sebelius admitted that convicted felons could be hired as "navigators" and that no Federal policy existed to require background checks.

4.11.2. Software Engineering Approach

Security, like quality, maintainability and safety are attributes of a system. If these attributes are important to a customer the software engineering approach is to ensure they are delivered by tasking a human being with "baking" them into the product. The security manager/lead/advisor assures security by implementing a security development life cycle (SDL). An SDL overlays the overall development life cycle from the start of development. The SDL methodology identifies responsibilities, deliverables and points of interaction with the project team.

Security management measures include:

a) Appointing a security manager/lead/advisor for each project.
b) Educating management and developers on the importance of security.
c) Conducting a security threat analysis, identifying the most likely attack vectors.
d) Developing security requirements.
e) Review design and code to ensure that security requirements are fulfilled.
f) Conducting an "attack surface" analysis of designs.
g) Establishing secure coding practices.
h) Training programmers in secure coding practices.
i) Conducting security related code reviews.
j) Conducting end-to-end security testing.
k) Making plans for security response where vulnerabilities are discovered in production code or actual security breaches are detected.

4.12. Software Engineering Professional Practice

4.12.1. Ethical Analysis

If the cohort of development managers responsible for the HealthCare.gov project were formally bound by the *IEEE/ACM Software Engineering Code of Ethics and Professional Practice* [IEEE CoE], they would have been guilty of many violations. The following points quote the relevant principles (in italics) and provide evidence of non-compliance.

4.12.2. IEEE/ACM Code of Ethics Violations

Duty of care. *1.02. Moderate the interests of the software engineer, the employer, the client and the users with the public good.*
The public good was not served by knowingly deploying a system that was not fit for purpose.
Evidence:
In various emails dating July 8 and July 20 of this year, [CMS] *officials write that they "under oath stated we are going to make October 1," but other emails express that, "we believe that our entire build is in jeopardy."*
In one document, HealthCare.gov project manager Henry Chao writes, "I just need to feel more confident they are not going to crash the plane at take-off, regardless of price."

Compliance with specification. *1.03. Approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate tests, and does not diminish quality of life, diminish privacy or harm the environment. The ultimate effect of the work should be to the public good.*
It was well known that the system had failed end-to-end testing, that testing was inadequate and that the system as a whole was not fit for purpose.

Accurate public statements. *1.06. Be fair and avoid deception in all statements, particularly public ones, concerning software or related documents, methods and tools.*
The act of deploying a system, by default, is a statement of confidence that it is fit for purpose. This was incorrect, in fact the system comprehensively failed in service. Even after deployment, when that failure was evident to users and government alike, CGI executives persisted in making deceptive

and misleading public statements (commonly known as spin) regarding the status of a transparently failed system.

...[Subsequent to deployment] the emphasis shifted from software development to optimizing for FFM performance ...
- CGI Senior Vice President, Cheryl Campbell

We note that the President of the United States, Barrack Obama, behaved ethically by telling it how it was:

... There's no sugarcoating it. The website has been too slow. People have been getting stuck during the application process. And I think it's fair to say that nobody is more frustrated by that than I am. ...
- Barack Obama, President of the United States

Disclosure in the public interest. *2.05. Keep private any confidential information gained in their professional work, where such confidentiality is consistent with the public interest and consistent with the law*.
The unwillingness to speak out publicly was in compliance with employer confidentiality, but in this case, against the public interest. Those that did speak out in various U.S. Senate enquiries were ignored.

Deliver quality. *3.01. Strive for high quality, acceptable cost and a reasonable schedule, ensuring significant tradeoffs are clear to and accepted by the employer and the client, and are available for consideration by the user and the public.*
Unprofessional behaviour meant costs were never accurately estimated and schedules were unrealistic. The resulting tradeoff required to meet the deployment date (reducing the test program duration from months to weeks) did not serve the public interest.

Set achievable goals. *3.02. Ensure proper and achievable goals and objectives for any project on which they work or propose.*
Cost and schedule goals were transparently unrealistic.

Realism in estimating. *3.09. Ensure realistic quantitative estimates of cost, scheduling, personnel, quality and outcomes on any project on which they work or propose to work and provide an uncertainty assessment of these estimates*.
CMS' project office was under staffed and cost and schedule estimates unrealistic. If estimates were provided with precision these red flags did not trigger any decisive management action (that is, delaying the deployment date).

Adequate testing. *3.10. Ensure adequate testing, debugging, and review of software and related documents on which they work.*
Testing was transparently inadequate. With the crashed schedule it is unlikely that effective reviews occurred.

Support ethical behaviour. *5.12. Not punish anyone for expressing ethical concerns about a project.*
Failure to make serious project inadequacies known to the President, senior management and the public would have been driven by fear of job loss or commercial disadvantage.

Take corrective action. *6.08. Take responsibility for detecting, correcting, and reporting errors in software and associated documents on which they work.*
Serious defects were detected, but not corrected prior to deployment.

Indicate code violations. 6.12. *Express concerns to the people involved when significant violations of this Code are detected unless this is impossible, counter-productive, or dangerous.*
Although several organisations (for example, the insurance companies) pointed out that the system should not be deployed, it will never be known if ethics formed part of their supporting argument. We suspect not, it being an unfortunate reality in IT companies engaged in deadline death marches, that any discussion of ethics is almost universally *impossible, dangerous AND counter-productive* to one's job security.

4.12.3. The Value of Ethics

In performing this analysis we found the level of unethical behaviour prevalent on the Obamacare project astounding and disappointing. Further, it is uncomfortable to reflect that this behaviour is not unusual on any software development project. The value of ethics is to hold a mirror to our conscience, reminding us of what is meant by "doing the right thing". That said, we should also remember Saint Paul's warning:

> ***... the basic principles of this world, its rules, have an appearance of wisdom,***
> ***but they lack any value in restraining indulgence ...***

It is true that a code of ethics can play a part, but it is only part of the story. The full narrative includes a rigourous software engineering education, followed by applying best practices in real-world projects and witnessing the benefits, followed by belief in restraining our indulgence and doing the right thing.

5. References

All text presented in italics and tagged as "evidence" in this document may be found with corresponding citations in document [CA CS SO 14].

[CA CS SO 14] Chambers, Les (2014), *Case Study: Saving Obamacare*, Chambers & Associates Pty Ltd, [Online], Available http://www.chambers.com.au/public_resources/case_study/obamacare/saving-obamacare-case-study.pdf [22 Apr, 2014]

[CA ISE 14] Chambers, Les (2014), Introducing Software Engineering, [Online], Available, http://www.chambers.com.au/glossary/software_engineering.php [22 Apr, 2014]

[Hadfield 13] Hadfield, Chris (2013), *An Astronaut's Guide to Life on Earth*, Macmillan

[IEEE CoE] IEEE-CS;ACM, *Software Engineering Code of Ethics and Professional Practice*, ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices, [Online], Available, http://www.acm.org/about/se-code [17 Feb, 2014]

[SWEBOK] IEEE (2014), *Guide to the Software Engineering Body of Knowledge*, IEEE Computer Society, [On line], Available, http://www.computer.org/portal/web/swebok [17 Feb, 2014]