121st ASEE Annual Conference & Exposition
Indianapolis, IN
June 15-18, 2014

Paper ID #10829

# The Computer Software Compliance Problem

**Prof. Peter j Knoke, University of Alaska, Fairbanks**

Associate Professor of Software Engineering in the University of Alaska Fairbanks Computer Science Department for the last 25 years. Prior to that various positions in the computer industry from 1958 to 1988, mostly in the role of software engineer. Prior to that fighter pilot in the USAF for several years.

# THE COMPLIANCE SOFTWARE DEVELOPMENT PROBLEM: IMPORTANCE AND POSSIBLE SOLUTIONS

**Abstract**

**Successful modern software development often requires compliance with both ethical and legal standards. This creates the "computer compliance software problem". That is defined and discussed together with reasons for its importance. Some possible solution approaches are defined and discussed, with some related examples. There have been a few well documented past software disasters, and there exist recent but undocumented software disasters, but there are reasons for hope that progress is being made toward solving the compliance software development problem and some are briefly discussed. Hope resides in greater software knowledge among key non-technical software decision-makers and software engineering education improvements which include lessons learned and the use of software development processes that embed those lessons.**

## Background and importance

**The computer compliance software development problem is most easily considered in the context of the old software engineering development model called the "waterfall model". That model considers software development as a sequence of five phases, namely the requirements, architecture/design, construction, test, and maintenance phases. With the waterfall model it's best to address the compliance problem as early as possible in the requirements and architecture/design phases. However, there now exist many other models such as the popular "agile" models for which it isn't obvious how or where best to attempt solution of the compliance problem. The problem is important regardless of particular software development model context. It is important to software developers because lengthy software development times, large software development costs, poor software quality and high liability risks are very likely if it isn't well solved. It is important to the end users because this kind of software if properly implemented can greatly reduce compliance and compliance enforcement costs.**

## Compliance software development problem definition

**Table 1 below is evidence of the considerable current interest in "compliance" generally. The Google data reveals little about the specific reasons for that interest. However, the Wikipedia data shows 62 separate compliance categories which provide some insight on that score. Those categories include compliance(medicine), compliance(regulation), compliance cost, compliance(physiology), compliance(psychology), compliance and ethics program, and compliance professional among others. That shows that the term compliance**

is used in diverse fields, and further specific research shows that its meaning often differs considerably from one field to another.

Table 1 Google and Wikipedia Search (3 Jan 14)

| Engine | Search terms | #Hits |
|---|---|---|
| Google | Compliance | 119,000,000 |
| | Compliance software | 124,000,000 |
| | Compliance software development | 66,000,000 |
| | Compliance software development problems | 16,300,000 |
| | Regulatory compliance software development | 12,800,000 |
| | Regulatory compliance software development problems | 9,000,000 |
| Wikipedia | Intitle:compliance | 62 separate items |
| | Intitle:compliance software | 1 (tax compliance software) |

This paper is specifically concerned with how best to solve the compliance software development problem. A recent special issue of the journal IEEE Software[1] has the theme "Software Engineering for Compliance" and is mostly dedicated to that subject. That journal issue includes the following definition:

*The term compliance addresses the external regulations, internal policies, standards, and governance to which an organization must adhere. In general, compliance in the context of information systems means ensuring that an organization's software and systems comply with multiple laws, regulations, and business policies. Compliance imposes certain IT controls that focus on information creation and retention, as well as on its protection, integrity, and availability. This is a major issue in many organizations because non-compliance might lead to severe financial penalties and reputational risks.*

That definition and its rationale are adopted for the purposes of this paper because the author judges it good and because it is quite broad. It is broad because it includes not only external regulations (legal constraints), but also standards and internal policies (which might not be legally enforced and which could be considered as ethical constraints in some cases).

## Why the compliance software development problem is difficult

The general reason why regulatory compliance software is difficult to develop is that regulations are often complex, ambiguous, rapidly changing, and sometimes contradictory (e.g. IRS code and the Affordable Care Act code). While many intelligent people are able to cope reasonably well with this situation, it is difficult to teach computers (which could be

**described as "rapid idiots") to do the same. In this context, programming is viewed as the education of computers.**

**The broader definition of compliance includes policies and standards, and maybe also ethical issues. Here the problem gets worse because many intelligent people may have legitimate differences of opinion about compliance in these other areas. Examples of such areas are health care, privacy and security, gay marriage, and marijuana use. Examples of coming future computer systems which, with suitable software might have to cope with such problems include humanoid robots (cf. Isaac Asimov and his Three Laws of Robotics[2].**

**The name "wicked problems" has appeared in recent years and generated considerable interest. It was originally applied to the field of social planning, where it was defined by a 10 point list[3] That was later generalized to a 6 point list by Conklin as follows:**

1) ***The problem is not understood until after the formulation of a solution.***
2) ***Wicked problems have no stopping rule.***
3) ***Solutions to wicked problems are not right or wrong.***
4) ***Every wicked problem is essentially novel and unique.***
5) ***Every solution to a wicked problem is a "one shot operation"***
6) ***Wicked problems have no given alternative solution.***

**Reference 3 describes a number of other wicked problem definitions but their extensive discussion is beyond the scope of this paper. In all those various definitions the term "wicked" is used to denote resistance to resolution, rather than evil.**

**Also contained in reference 3 is an item on "wicked problems in software development" (1990, DeGrace and Stahl). The problem of constraint software development arguably satisfies the various definitions specified in reference 3.**

**While considering software development as a wicked problem, reference 3 includes the following statement which clarifies why that designation is appropriate:**

***Software development shares many properties with other design practices (particularly it seems that people, process, and technology problems have to be considered equally)***

**In other words, engineering design itself could be considered as a wicked problem in many cases. The development of software for the US healthcare.gov website[4] is a current high profile example of a wicked problem.**

**In summary, constraint software development is difficult because it is a wicked problem. It probably is getting more difficult because of rapid and significant changes to computer software technology and the increasing demand for new, larger and ever more complex**

**software (the healthcare.gov website software is complex and large, requiring an estimated 500 million lines of code).**

**Possible problem solution approaches**

**This section includes some possibly helpful ideas which have proven successful in solving past software development problems. They cannot be fully evaluated except in the context of a specific compliance software development case[1].**

1) **Use more multidisciplinary teams in early phases of development**

**Often software development solutions have been most cost-effective if applied at the requirements or architecture/design levels. Multidisciplinary teams could be effective at these levels. For example, lawyers who could be quite familiar with relevant legal issues that are complex, ambiguous and changeable could join software engineers on the development team during the early stages. The same goes for other domain specialists (e.g., from finance and medical domains). Specific team membership would depend on the sources of the specific constraint software goals (e.g., are the constraints primarily from the EPA, the IRS, the FAA, etc).**

2) **Develop better tools and processes and models for use throughout the development.**

**Better programming languages have in the past significantly reduced coding time and coding errors. Better tools and processes (e.g. agile processes) have speeded needed changes and reduced associated change errors. Better tools have speeded and simplified software test.**

**Some of the modularity and object definition ideas of David Parnas (arguably the father of Object Oriented Programming) might be applicable for compliance software. They could be effectively used if it were true that future regulatory and other requirements changes could be predicted with some degree of accuracy. The probable interpretation of some new laws might be estimated on the basis of past legal precedents.**

3) **Reduce the need for new software developments by more software reuse**

**Reuse has been attempted in software development for many years. One practical lesson learned in this area has been that reuse is most successful if reuse has been planned in the first place (there are distinctions between software development FOR reuse vs. software development WITH reuse).**

**Perhaps portions of successful older compliance software systems could be adapted for use in new compliance software systems.**

4) **Search for lessons learned from well- documented past software disasters, software related litigation reports, or other similar literature**

**Software law litigation documents can be a good source of relevant legal lessons learned. For example the monthly Thompson West Journal "Software Law" [5] is quite readable and affordable. Peter G Neumann's periodic report on "Risks to the Public in Computers and Related Systems" [6] is a highly regarded item in the ACM Software Engineering Notes (Neumann (an ACM Fellow) has been moderating that for many years).**

**The book "Software Runaways: Monumental Software Disasters" by Robert Glass[7] provides a good documentation of 16 software disasters of the past, together with lessons learned data. Those disasters were often found to be caused by a combination of problems including requirements (poorly defined, misunderstood and changing) and unrealistically short development times. The development times were sometimes too short because of political pressures. The new health.gov web software seems to have those same problems, plus some security and scalability problems which are now common for many current websites.**

**It should be noted that although good documentation of past and present software disasters prepared by knowledgeable insiders can be of great value as a source of lessons learned, such documentation is quite scarce. Reasons for this could include desires for maintenance of a good corporate image or for keeping a job.**

**Good lessons could also be learned from documentation of software successes, but in those cases corporations might be reluctant to share information about processes that work well, while intellectual property laws provide them with a means avoid sharing code details.**

5) **Improve the education of new Software Engineers and promote software engineering professionalism**

**Incorporate more materials such as lessons learned mentioned above into the standard undergraduate or graduate software engineering curricula.**

**Also, support emerging software engineering professionalism. A software engineering PE has recently been added to the NCEES Professional Engineer menu. The SWEBOK-based PE study guide now includes 15 study areas instead of the original 10 (SWEBOK[8] stands for Software Engineering Body of Knowledge, which was originally developed by the IEEE Computer Society).**

## Reasons for hope

**There are some reasons for hope that progress will continue to be made toward solutions for the compliance software development problem. Increasing numbers of well-documented, relevant and timely compliance case studies would help. An increasing number of US law schools now have programs addressing technology law, and an increasing number of lawyers, judges and lawmakers have gained significant technical understanding and expertise. There might be an increasing number of software engineers with awareness of software-related compliance issues. And the state of the art of software engineering continues to improve constantly.**

## References

1) IEEE Software May/June 2012 (Vol 29, No3) pp24-27.
2) Wikipedia(Three Laws of Robotics) 1 Jan 14.
3) Wikipedia(Wicked Problem)3 Dec 13.
4) Wikipedia(Healthcare.gov)4 Jan 14.
5) Westlaw Journal "Software Law" Litigation news and Analysis, Legislation, Regulation, Expert Commentary (Monthly Journal).
6) Wikipedia(RISKS Digest)(Peter G. Neumann)12 Dec 13).
7) Robert Glass, "Software Runaways", Prentice Hall, 1998.
8) Wikipedia(SWEBOK)6 Jan 14.