Towards a Discipline of Software Engineering Forensics Analysis

Paul Bailes[1], Christine Cornish[1,2], Toby Myers[1,2], Lou Rago[2], Nick Tate[3] and Mal Thatcher[4]
[1]School of ITEE, The University of Queensland, St Lucia, QLD 4072, Australia
[2]BCI Technology, Level 2, 167 Eagle St, Brisbane, QLD 4000, Australia
[3]RDSI Project, The University of Queensland, St Lucia, QLD 4072, Australia
[4]Mater Health Services, Raymond Tce, South Brisbane, QLD 4101, Australia

Keywords: Forensics, Process, Software.

Abstract: Software development and procurement continues to be a source of great disappointment for its social and economic stakeholders, with literally billions of dollars being expended for little ostensible benefit. But significant progress can be made in engineering domains that match software for complexity and novelty: the international regime of aviation accident and incident reporting has been the basis for a wide range of evidence-based technical and process improvements in applied aeronautical engineering. Accordingly, we set out to characterise the knowledge, activities and structures that would promise to deliver analogous benefits to software engineering. While we are hopeful of early positive outcomes, a significant research agenda lies before us.

1 INTRODUCTION

Despite the near half-century that has elapsed since the need was explicitly recognised for software systems to be developed to the same standards as other engineering artefacts (Naur and Randell, 1969), the procurement of computer-based information systems remains unsatisfactory. Too many significant software development projects continue to fail in one or more of the dimensions of: on-delivery; within-budget delivery; client satisfaction; or outright cancellation. It does not however have to be the case that innovative high-technology engineering projects are so destined for failure. Taking aeronautical engineering as our exemplar - and we are not alone (Charette, 2005) - it indeed seems possible to accelerate successfully the inculcation and adoption of the high standards of performance and conduct in a relatively new engineering field. Instrumental in this outcome has been the forensic analysis of departures from these standards (i.e. across the spectrum from aviation incidents to disasters), including both the appropriate technical and sociological apparatus to ensure the effectiveness and impact of these investigations (ICAO, 2001).

Consequently, the question arises of how software engineering can learn from aviation/aeronautical engineering in this regard. Accordingly, the goal of this paper is to envision and strategize the creation of a new branch of software engineering, which we name at least provisionally "Software Engineering Forensics Analysis" (SEFA). Taking both the technical and sociological aspects of aviation investigations as our broad model, we seek:

- to advance the theory and practice of software engineering through the analysis of software development projects by distinguishing between the characteristics of successful versus failed or failing projects;
- simultaneously, to develop tools and techniques to facilitate these analyses;
- equally, to foster the development of social institutions and practices (both voluntarily and by regulation/legislation, as appropriate) that will encourage the adoption and application of the above;
- thereby engendering improvements in the timeliness, cost and effectiveness of significant software procurement exercises;

and thus, to achieve the economic and social benefits resulting from all the foregoing.

Bailes P., Cornish C., Myers T., Rago L., Tate N. and Thatcher M..
Towards a Discipline of Software Engineering Forensics Analysis.
DOI: 10.5220/0004970002350240
In *Proceedings of the 9th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2014), pages 235-240
ISBN: 978-989-758-030-7

2 SOFTWARE DEVELOPMENT AND PROCUREMENT FAILS

Despite nearly half a century of research aimed at performing software development at levels of effectiveness and efficiency matching those of established engineering disciplines, software development (or more generally, procurement) remains plagued by failures, some of a spectacular nature. An internet search on terms such as "failed and overbudget custom software projects" is sadly fruitful, some highlights of the results of which are as follows.

- The US Air Force has abandoned 7 years of development of a new integrated ERP system - the Expeditionary Combat Support System (ECSS) - at a cost of US$1B(illion) wasted (Kanaracus, 2012a). Instead, reversion to the separate legacy systems that predated ECSS seems to be the basis for future development.
- The UK government's "Universal Credit" consolidated welfare payments system (NAO, 2013) has so far incurred development costs in excess of UK£300M(illion) but with a now indefinitely delayed roll-out and unquantified value proposition.
- The failed Queensland (Australia) Health Department's payroll system (Chesterman, 2013) is incurring significant costs as a result of the need to employ manual work-arounds to compensate for its deficiencies: estimates of AU$416.6M(illion) for the 2009-12 period; and further estimates of AU$836.9M from 2012-17 (i.e. total cost of AU$1.2535Billion); and that's without costing the required replacement systems development.
- The UK National Health Service abandoned its National Programme for IT (NPfIT) in 2011 (Mathieson, 2011) after spending UK£12B(illion).
- The US government's "Obamacare" Health Insurance Exchange website (healthcare.gov) has been plagued by start-up problems (Ford, 2013) but costs are not clear.
- Charette's (2005) already-cited survey includes a "Hall of Shame" where nine-figure losses (in major world currencies) are unexceptional.

Lest we be tempted to think that the above represent ancient or isolated history, the tale of woe is sadly contemporary (Kanaracus, 2012b).

3 WHAT IS THE PROBLEM?

At first glance, it seems inexcusable, or at least inexplicable, why software development should be so problematic, in view of the long-standing and extensive educational and research literature on the subject. From a comprehensive point of view: Sommerville's (2011) classic text cites almost every conceivable technology, technique and process; and if coverage of individual topics may sometime be a little sketchy, the extensive bibliography offers a portal to the fruits of the global effort in the field.

This problem (of practical failure to apply appropriate software engineering tools techniques and processes) is definitely not one of professional community ignorance of the wider, non-technical aspects that transform computer science into software engineering. For example, the importance of management skills (including project planning, scheduling, risk management and personnel management) as complementary to technical expertise is well-established (Sauer and Cuthbertson, 2003), and indeed well-represented by Sommerville.

Our hypothesis is that it's not the case that there are too few solutions to the software development/procurement problem at hand, but rather that in a sense there are too many. More specifically, for many if not most generic situations that arise in a software project, multiple approaches are possible. Consider the following choices relating to various aspects of software engineering (Sommerville, 2011).

- Requirements: structured natural language vs. mathematical logic vs. graphical languages; in the latter case examples of choices are between UML (OMG, 2011) and Behaviour Trees (Dromey, 2006).
- Specification: examples of choices are between model-based and abstract, executable and non-executable, mathematical vs. graphical; for example Z (Spivey, 1992) is abstract, non-executable and mathematical; Petri Nets (Peterson, 1977) are abstract, executable and graphical.
- Architecture/Design: e.g. transaction processing vs. event processing vs. object-oriented vs. client-server vs. distributed.
- Implementation: e.g. choice of programming language C++ vs. Java vs. Scala etc etc etc.
- Verification and Validation: formal methods vs. testing; in the latter case black box vs white box.
- Overall process: waterfall vs. iterative (including "agile").

Under such circumstances, the status quo of confusion and under-performance is not surprising.

4 THE CHARACTERISTICS OF A SOLUTION

The simple way to address the above "too much information" problem is to clarify the situations under which one is applicable over the alternatives. Developing our hypothesis, the current position in the professional development of software engineers is all too much one of a lack of evidence-based authority about the circumstances under which one software development options (tool, technique, process, etc.) is more applicable than the alternatives.

4.1 Evidence-based

The aeronautical engineering precedent is, as above, evidently attractive to software engineering because of its complexity and relative novelty. More fundamental however is the distinctive way in which aeronautical engineering failures, as represented by aviation accidents and incidents, are treated. Most distinctively, the accidents/incidents are the subjects of (depending upon severity) extensive and disciplined forensic investigations that yield hard evidence about the appropriateness or otherwise of various aeronautical engineering techniques to operational circumstances. For example:

- square-shaped windows are acceptable in unpressurised aircraft, but pose unacceptable metal fatigue risks in pressurised aircraft (MTCA, 1955)
- gyroscopic precession of propellors ("whirl mode") has a risk of resonating with the natural frequency of the wing, leading to catastrophic flutter (Job, 2001);
- incorrect attachment of pod-type jet engine mounts to wing pylons can lead to engine separation and collision with airframe leading to loss of control and destruction (Job, 1998a).

4.2 Transcending the Technical

There is however a potential objection to the validity to software engineering of the aviation precedent, in that aeronautical engineering is dominated by physical science, viz. the above examples. The corresponding class of software engineering examples might be concerned with correctness against specifications, but in the above-cited examples of software project failures, simple coding errors are evidently sparsely-represented. Rather, even a cursory reading of failed software project reports reveals process failures far beyond the implementation/coding stage (or its equivalent in non-waterfall processes). What example does aviation forensics have to show for this class of software engineering problem?

We are able to give a strong positive response to this question, because the scope of aviation forensic investigations is indeed considerably wider than the above examples indicates. From ICAO (1991), we read:

> *FORMAT OF THE FINAL REPORT*
> *...*
> *1.17 Organizational and management information.*
> *Pertinent information concerning the organizations and their management involved in influencing the operation of the aircraft. The organizations include, for example, the operator;*
> *the air traffic services, airway, aerodrome and weather service agencies; and the regulatory authority. The information could include, but not be limited to, organizational structure and functions, resources, economic status, management policies and practices, and regulatory framework.*
> *...*
> *3. CONCLUSIONS*
> *List the findings and causes established in the investigation. The list of causes should include both the immediate and the deeper systemic causes.*

It is thus abundantly clear that aviation accident and incident investigations cover the full range of the aviation "process model" (as we might put it in software engineering terms), including the many accidents/incidents whose prime technical causes can be traced back to broader systemic issues, including "organizational structure and functions, resources, economic status, management policies and practices, and regulatory framework".

For example:

- in the above-cited example of incorrect attachment of pod-type jet engine mounts, the inherent risk entailed in a somewhat unforgiving pylon design was realised by ill-considered modifications to maintenance procedures introduced by the operator;
- the Air New Zealand airliner lost by collision in

1979 with Mt Erebus in Antarctica was in perfect aeronautical condition throughout, but management of the aircraft's computerised navigation system led to an unknown departure from the expected course (Job, 1998b);

- in several fatal disasters suspicion has come upon perfectly-functioning flight control systems but with ill-designed human interfaces leading to command confusion and loss of control (Job, 1998c).

4.3 Engineering Forensics Summary

In summary then, generalising from the aviation example, successful engineering forensics (including software) requires a combination of the authority to effect changes in engineering practice resulting from a basis in evidence regarding the processes and techniques that succeed vs. those that fail, combined with a complete process view that transcends the narrow scientific/technical subdomain.

5 THE SHAPE OF A SOLUTION

Consequently, faithfully following the aeronautical precedent, our nascent SEFA discipline cannot be divorced from either its technical fundamentals or the social context in which it operates. The mode of operation and structures by which we apply SEFA knowledge are just as important as the knowledge itself.

5.1 SEFA Knowledge

Three basic kinds of knowledge are required.

First obviously is basic software engineering development/technical and management knowledge, including:

- overall software process(es)
- individual stages of and artefacts from the above
- tools supporting the above.

From our experience to date, the priority software process issue is likely to concern the problems encountered in ensuring the development proceeds according to clients' actual requirements.

Second is knowledge of the legal context of software development (which may differ from jurisdiction to jurisdiction), as the fundamental basis that establishes (and *in extremis* enforces) development partners and clients' mutual obligations.

Finally are meta-level software engineering research skills: as is evident, SEFA is at an early even nascent stage, and discernment of the meta-processes by which forensic analyses are conducted will be as important as the discovery of the evidence bases for the applicability or not of the various extant software engineering tools, techniques and processes.

5.2 SEFA Operational Mode

We expect SEFA to be applicable in the following modes of operation before, during and after a software development project.

- Before: preventive assurance (i.e. avoiding the mistaken adoption of tools, techniques or processes that SEFA evidence has demonstrated to be inappropriate in a particular project's context).
- During: corrective assurance/disaster recovery (i.e. determining the causes of an impending failure and if possible identifying a rectification pathway based on SEFA evidence).
- After: providing expert witness to post-failure investigations.

In all the above, each engagement represents an opportunity to expand the SEFA knowledge base.

5.3 SEFA Operational Structure

A critical factor in the success of air accident/incident investigations may be the formally-established status of the various national investigation organisations. While an international organised counterpart to ICAO leading to such a situation may sound far-fetched, the impact of software on lives, prosperity and property would seem to justify such an ambitious goal.

In the interval however, a worthwhile enterprise will be to establish, by purely private means if necessary, an integrated organisation ("Software Forensics Institute" or some such) both to conduct SEFA operations and to nurture the growth of SEFA knowledge, as characterised above.

6 CHALLENGES

Firm establishment of SEFA as a viable contributory discipline to software engineering will require the satisfaction of many social and technical prerequisites. Among the technical prerequisites are the following:

a. how do we establish conveniently a "narrative

record" of a particular software development project from whatever evidentiary trail exists?

b. then, how do we evaluate/assess/critique such a narrative against the prevailing understanding of correct/canonical software process(es)?

c. likewise, how do we evaluate/assess/critique artefacts produced at each stage e.g. requirement specs, designs, code, test plans, etc.

Hopefully "big data" analysis solutions may be discovered that can be applied to this domain.

Chief among the social prerequisites is how to inculcate as far as possible a blame-free culture that is conducive to open self-criticism by software developers in the aftermath of a failed project, e.g. as with the UK Civil Aviation Authority Mandatory Occurrence Reporting (MOR) Scheme (CAA, 2011).

7 CONCLUSIONS

While being frank about the challenges, we propose SEFA development as a worthwhile undertaking. Generally, the goals as per our Introduction above are worthwhile, and we have established the plausibility of the precedent from aeronautics/aviation.

Specifically, SEFA raises hopes for the following:

- an evidence-based, more specific understanding of the different circumstances under which different software processes and tools are more or less appropriate;
- hopefully including a rubric when "agile" methods are appropriate (or not)!
- similarly for other variations from canonical process(es);
- meta-level tools and techniques to enable the above;
- more specific directions in software engineering education and training;
- incidentally, because software systems dominate aeronautical engineering, a formally-established "Software Forensics Institute" would discharge implicit ICAO obligations in software dimension of air accident investigations.

REFERENCES

CAA, 2011. *CAP 382 Mandatory Occurrence Reporting Scheme (9th ed.),* TSO.

Charette, R., 2005. Why Software Fails. *IEEE Spectrum.* http://spectrum.ieee.org/computing/software/why-software-fails

Chesterman, N., 2013. *Queensland Health Payroll System Commission of Inquiry Report.* http://www.healthpayrollinquiry.qld.gov.au/__data/assets/pdf_file/0014/207203/Queensland-Health-Payroll-System-Commission-of-Inquiry-Report-31-July-2013.pdf

Dromey, R.G., 2006. Formalizing the Transition from Requirements to Design. In *Mathematical Frameworks for Component Software – Models for Analysis and Synthesis,* Jifeng He, and Zhiming Liu (Eds.), World Scientific Series on Component-Based Development, pp. 156-187

Ford, P., 2013. The Obamacare Website Didn't Have to Fail. How to Do Better Next Time. *Bloomberg Businessweek.* http://www.businessweek.com/articles/2013-10-16/open-source-everything-the-moral-of-the-healthcare-dot-gov-debacle.

ICAO, 2001. Aircraft Accident and Incident Investigation. *Annex 13 to the Convention on International Civil Aviation.* International Civil Aviation Organization.

Job, M., 1998a. American 191, do you want to come back. In *Air Disaster Volume 2.* Aerospace Publications.

Job, M., 1998b. I don't like this. In *Air Disaster Volume 2.* Aerospace Publications.

Job, M., 1998c. *Air Disaster Volume 3.* Aerospace Publications.

Job, M., 2001. The Lockheed Electra Saga. In *Air Disaster Volume 4 The Propellor Era.* Aerospace Publications.

Kanaracus, C., 2012a. Air Force scraps massive ERP project after racking up $1 billion in costs. *CIO.* http://www.cio.com/article/721628/Air_Force_scraps_massive_ERP_project_after_racking_up_1_billion_in_costs

Kanaracus, C., 2012b. The scariest software project horror stories of 2012. *Computerworld.* http://www.computerworld.com/s/article/9234581/The_scariest_software_project_horror_stories_of_2012

Mathieson, S., 2011. Scrapping the National Programme for IT: a journey not a destination. *The Guardian.* http://www.theguardian.com/healthcare-network/2011/sep/22/npfit-ends-cfh-andrew-lansley-bt-csc?newsfeed=true

MTCA, 1955. *Report of the Public Inquiry into the causes and circumstances of the accident which occurred on the 10th January, 1954, to the Comet aircraft G-ALYP,* London: HMSO

NAO, 2013. *Universal Credit: early progress.* National Audit Office. http://www.nao.org.uk/report/universal-credit-early-progress/

Naur, P. and Randell, B. (Ed.) 1969. *Software Engineering: Report on a Conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968.* Brussels, Scientific Affairs Division, NATO.

OMG, 2011. *Documents Associated With Unified Modeling Language (UML), V2.4.1.* http://www.omg.org/spec/UML/2.4.1/

Peterson, J., 1977. *Petri Nets.* ACM Computing Surveys

vol. 9 no. 3 pp. 223–252.

Sauer, C. and Cuthbertson, C., 2003. The State of IT Project Management in the UK 2002-2003. *Computer Weekly Project/Programme Management Survey*. http://www.bestpracticehelp.com/The_State_of_IT_Project_Management_in_the_UK_2003_2004.pdf

Sommerville, I., 2011. *Software engineering (9th ed.)*. Pearson.

Spivey, J.M., 1992. *The Z Notation: A reference manual*. International Series in Computer Science (2nd ed.). Prentice Hall.