

Innovation and Practise in Software and Systems Engineering

Preface

Details

- Author - Dr Daniel M. Berinson BE PhD MIEAust MIEEE AFAIM GAICD
- Organisation - Systec Engineering Pty Ltd
- Type of organisation - industry consultant and contractor in software and systems engineering
- Address - PO Box 556 Dianella WA 6059
- State or Territory (or country if not Australia) - Western Australia
- Email and phone contact - daniel@systec.com.au and 040 888 0278

Declaration of Interest

- Member of Engineers Australia ITEE (WA) Committee.
- Board member of ICT Industry Collaboration Centre (WA).
- Member of Industry Advisory Committee for Computer Science and Software Engineering at University of Western Australia.
- Member of Engineering Industry Advisory Committee for Murdoch University.
- Sessional lecturer in Strategic Management in Engineering at Murdoch University.
- Consultant and contractor in defence, resources, finance and utilities industries.

Summary

The purpose of this submission is to focus attention on software and systems engineering as it relates to defence, telecommunications, utilities and other industries that rely on safety, integrity and reliable systems; however similar issues arise in enterprise computing that I mention in an addendum. The poor understanding among many engineers, computer scientists and specialists in software and systems engineering makes for many difficulties in building a case for innovation and improving the state of practise in these closely-related fields.

The first step needed to remedy the situation involves redefining software engineering as a subclass or speciality of systems engineering which is itself reasonably well understood by most engineering practitioners and poorly understood by others not working intimately in the field; or at the very least clarifying the relationship between these two disciplines. The second step is to construct an integrated, elite software and systems engineering programme to address the critical shortage of practitioners in these fields, including requirements engineering, test, verification and validation, software and systems architecture, detailed design, and issues of construction, configuration, packaging, deployment and maintenance throughout the whole software and systems development life-cycle.

I also draw attention to the need to focus on fundamental programming, technologies and practices that are essential to software engineering, especially safety-critical and reliable systems, in addition to those aspects that characterise the field as systems engineering.

Barriers to innovation in software and systems engineering

The primary reason to emphasise the barriers to innovation is that the same issues affect the daily and evolutionary practise in the field of software and systems engineering. The reasons for poor on-time delivery, poorly-met requirements and unmet customer expectations for software projects involve the same factors that detract from proper performance during project execution. Project, programme and strategic management are relevant but are on the periphery of the issues that I am highlighting in this submission: Aside from management issues per se, the relevant issues are those in commercial practise and academic presentation of the subject matter.

Issues in defence, industry and academia

The critical shortage of experienced software and systems engineers is in defence where the worsening shortage of specialist skills in requirements and test specification, system design and construction will continue to have an increasingly negative impact on project delivery. The general absence of undergraduate and postgraduate engineering courses that specialise in defence systems is perhaps understandable because the sector, while influential, is small relative to the pool of potential employers for university graduates.

The objectives of defence, industry in general and the academic community are intertwined when we consider the need to reasonably match the supply of graduates to the demand for qualified personnel in industry. The demand for graduates in the defence sector is fed by government and procurement policy while the demand by industry in general is cyclical with the economy. Notably the resources boom has skewed talent entering and leaving university towards the mining and resources industry arguably to the detriment of other industries.

An obvious part of the solution is an upwards adjustment in wages to encourage school leavers to enter the relevant university courses and for graduates to join defence and other industries where there is a shortage of graduates to fill the places. Over time, the bigger problem is the combination of the decline in experienced personnel in these industries as well as the tendency for short-term goals to steer many school leavers towards vocational training and employment when they are well suited towards a university education.

The flow-on effect of experienced practitioners leaving the field is that the level and quality of postgraduate mentoring and coaching in the workplace will decline. Anecdotally, there is evidence to suggest that computing and engineering graduates already have fewer career pathways and mentoring opportunities than their peers in other professions, for example, lawyers serving as articulated clerks expect their work to be closely reviewed and marked with red ink by senior associates and partners; and doctors serving their internship and residency are supervised and driven by registrars to improve their clinical knowledge and to excel in its application.

One aspect of these professions currently lacking and that is being developed in software engineering is that of licensing beyond the automatic membership of professional societies granted by virtue of graduating from an accredited institution. Beyond licensing and registration of professionals, a shift in mindset and approach is needed to elevate the level of on-the-job postgraduate professional development of graduates in software and systems engineering.

Supporting R&D and commercial innovation

Among the wide variety of research and innovation undertaken by research institutions and other companies, much of it intellectually or commercially valuable, there is

significant R&D undertaken by industry that tends towards being conservative, mundane and uninformed; matched by university-based research that is irrelevant, derivative and uninformed. In itself this claim is irrefutable and we should accept that a few gems of research outcomes are the valuable result of a large amount of money, time and effort invested in research and associated infrastructure.

However we must make greater effort to redirect some resources towards research programmes that are relevant and industrial R&D that addresses known and real problems. In either case, the opportunity for academic research to be informed by outstanding problems in industry and commercial realms is a challenge to take up because of the strict demarcation lines between industry practitioners and academics. It is equally true that many academics will not be welcomed with open arms into the world of commerce just as industry practitioners (with or without a PhD) are looked upon with suspicion when trying to access research funding channels.

As General co-Chair of ASWEC 2008 (Australian Software Engineering Conference), held recently in Perth for the first time in its twenty year history, the relevance and importance of communication between academia and industry was highlighted by the intense participation of both sectors. ASWEC is very rare conference insofar as it encourages mixing between academics, policy makers, teachers and researchers, and practicing engineers, programmers, managers and others in industry. Software engineering is a practice-based discipline and as such it is difficult to imagine software engineering as an academic discipline divorced from its commercial application. This sort of communication, other such forums and cooperative research and development centres need to be encouraged, nourished and supported in order for this field to flourish.

The way software engineering is taught

The various computer science and software engineering degrees taught at our Australian universities appear to be solidly based and founded on a sound scientific and engineering basis. However, some of the content and manner in which these courses are taught, particularly at the postgraduate level where I have heard several disturbing anecdotes of extremely poor course structure and content, needs to be addressed immediately in order to arrest the anecdotal decline in quality of graduates and the very real long-term decline in the number of school leavers entering computer science, engineering and science courses in general.

Part of the attraction of professions such as accounting, law and medicine are the high standards of quality expected of their members, for example, the respect attributed to holders of Chartered Accounting and Certified Practising Accountant qualifications. In many undergraduate, and more so at postgraduate, schools the restricted intake of elite students entering into a period of high-quality education matched by personal challenges to meet the high standards that are expected increases the real and perceived value of the qualification, for example, MBA. I recommend the creation of elite schools that can meet the expectations of quality and guarantees of employment and career development expected of the best schools in the other professions and some engineering disciplines.

State of practise in software and systems engineering

Systems engineering as a discipline

The discipline of systems engineering includes requirements engineering, architecture development and evolution, systems decomposition, modelling and synthesis of the problem domain and selecting one of many viable solutions as a pragmatic approach to solving the problem and meeting the customer's requirements. Verification of the build

and construction process, including traceability of design and implementation artifacts (eg. software modules, packages and applications) to the requirements specification, is a precursor to validation of the solution against the customer's need.

Sometimes a solution is constructed from scratch however in general a number of existing, reusable components and third-party, commercial off-the-shelf (COTS) components need to be integrated into cohesive system as part of the solution. The process of decomposing and assigning requirements to subsystems and later integrating each of those subsystems is the essence of systems engineering. Essentially the same description applies to the software engineering processes involving the analysis and design of a software solution for nontrivial systems. The distinguishing feature of nontrivial systems (colloquially known as "complex systems") is that the project team is necessarily composed of people with skills in various disciplines, design and construction takes a significant amount of time, and the delivered system needs to be supported for a significant period of time, often measured in decades for defence and some commercial systems.

The general absence of specialist undergraduate and postgraduate engineering courses that specialise in safety and reliable systems continues to surprise however courses tend to be specialised by discipline or industry. Focused delivery of material as elective units to a core degree in engineering or computing would seem suitable at an undergraduate level, or postgraduate courses that build on the generalist and formative knowledge base that is the foundation of an undergraduate degree. In academia and industry it is rare to find individuals with sufficient knowledge in control systems, digital signal processing or systems analysis as well as a strong enough foundation in computer science or software engineering that they can competently and confidently deliver systems that require this sort of mix of skill sets.

Platforms, frameworks and languages are important

A firm grasp of language-based implementation idioms by software engineers is crucial for the development of high-quality software systems in the sense of the -ilities, for example, portability, maintainability, understandability and reuseability. In addition to these qualities the non-functional requirements to be met often include performance and reliability that require an intimacy between the implementation language and the deployment platform.

For the development of distributed, real-time, mission-critical software and computing infrastructure it is essential that the underlying platform and programming language are considerations during analysis and design phases through to in-service deployment. Less-demanding problem domains have weaker constraints where the business requirements can probably be realised by any number of possible implementations however this is not the case with the subset of critical applications we are referring to here.

Importance of C++, STL, Corba/IIOP, ACE, Boost for teaching and systems development

Well beyond C with classes, C++ is the quintessential system-programming language, offering a rich toolkit of implementation idioms that can be exploited by experienced programmers in addition to a simple programming model that can be used by novices. The facilities of object-oriented programming, generic programming and functional programming enable C++ to be a first-class language for building Domain-Specific Languages (DSLs) and unsurpassed in this role by any other language. The C++ features of extension by derivation coupled with operator overloading enables new types to be defined that operate in the same way as built-in types.

The Standard Template Library (STL) is a rich, contract-based set of iterators, containers and algorithms that enables well-defined behaviour on template (generic) types and through extension achieved via traits (template specifications) without adhoc language extensions for run-time type introspection. As powerful and general-purpose as are languages like Ada, Java and C# they cannot replace C++ in defence, telecoms and control systems, on scales from embedded systems to supercomputing.

While C++ has a solid standard-based definition since 1998 and has facilities provided by ACE and Boost libraries, for example, other languages that are its competitors (and some allege its superior) have required numerous language extensions in order to add basic facilities that mirror C++ features, usually weakly, which are part of the core language. For example, generics and templates in Ada, Java and C# are less powerful than C++ templates; and containers provided by STL have poor cousins in Ada generics, Java and C#.NET container libraries.

C++ and Corba/IIOP (Internet Inter-ORB Protocol) are the premier software and integration standards in use across the majority of systems today even though EJB/IIOP (Enterprise Java Beans) and various .NET Channel bindings to Corba/IIOP are not so well known. The increasingly popular, though limited, Java and C# programming languages, while they are eminently suitable for business programming, are unsuitable for many application domains in embedded, real-time, control, safety-critical and reliable systems.

Importance of Spark and Ada in teaching and developing reliable systems

Ada is widely used in defence, NASA and avionics where mission-critical systems are deployed, for example, embedded systems in the Boeing 777 and 787 run Ada code because the systems require highly-reliable, mission-critical software. Ada allows developers to prove security properties about programs, for instance, the ability to prove that a variable is not altered while it is being used through the program; and Ada supports comprehensive static analysis for dissecting the program flow to ensure correct behaviour.

SPARK is a high-level programming language and toolset designed by Praxis High Integrity Systems for writing software for high integrity applications. In their words, "SPARK gives confidence in the correctness of code – it is verifiable. Early detection and prevention of defects reduces the cost of verification and validation. SPARK is a very portable language and has minimal run-time system requirements. SPARK is used on safety-critical systems, primarily in the Aerospace, Defence, Rail and Security industries. It is suitable for use wherever there is a desire to produce high-quality software which behaves in a predictable manner."

As a teaching aid in software engineering, a research tool in universities and other institutions, and as an implementation language for many applications in the Australian context, Ada and SPARK are suitable languages for safety and high reliability and their use should be encouraged where appropriate.

Research and teaching recommendation

I strongly recommend the continued teaching of C++ and the allied technologies of STL, ACE, Boost and Corba at our universities. The Adaptive Communication Environment (ACE) is a highly-regarded object-oriented network programming toolkit used for writing sophisticated concurrent, parallel, and distributed applications. Systems built upon ACE are widely deployed across many problem domains and industries including telecommunications, defence, aerospace and finance. In addition, I recommend that we create specific programmes for research and teaching of Spark and Ada.

We should have programmes of research supporting the development of advanced systems built on these technologies otherwise we continue to run the risk of falling further behind, both technologically and economically, as a nation in this area. A concerted effort is desperately needed to create and fund one or more centres of expertise to develop the required skills and drive the research programme in addition to the broad-scale evolution in the content and structure of teaching and practice.

Recognising that Java and C# are already in widespread use and it is likely their continued growth will lead them to dominate wide-scale software development in the future it is important to study these languages. However, C++ in particular must continue to be taught to undergraduates so it can be satisfactorily deployed in industry.

Enterprise Systems Development

There are many reasons to extend good practise to Java/J2EE, EJB/IIOP, C#.NET and SOA systems in the enterprise systems space. The same lessons that apply to good software and systems engineering in general naturally apply also to enterprise systems development however there is much less attention paid to rigour this field by academics and industry participants. Most enterprise software development involves integration of COTS systems using various integration technologies, including EJB/IIOP and service-oriented architecture (SOA), with custom applications written in a range of technologies including Java 2 Enterprise Edition (J2EE) and C# on the .NET platform.

Prof. Ian Sommerville, one of the worlds eminent leaders in software engineering (author of the leading text book and himself known as the "Father of Software Engineering") emphasised in his keynote talk at ASWEC 2008 that enterprise software includes health and medical systems, defence and logistics systems, aerospace systems and airspace management, in addition to the mission-critical applications for corporations including enterprise resource planning (ERP) and many other systems. He said that the direction in which enterprises are heading includes construction by configuration and integration of COTS components so it is important for research programmes to be created in these areas.

One problem is that the time frames of enterprise systems development span many months or years for a strategic programme of work and these time frame do not fit the normal academic time frame for publication. As already pointed out, it is essential that such efforts include experienced industry practitioners however their involvement depends upon commercial and R&D funding that is very difficult to access, in addition to the need to break down the notional barriers between academia and industry.

The report entitled The Challenges of Complex IT Projects by the Royal Academy of Engineering and the British Computer Society states, "there is a broad reluctance to accept that complex IT projects have many similarities with major engineering projects and would benefit from greater application of well established engineering and project management procedures. For example, the importance of risk management is poorly understood and the significance of systems architecture is not appreciated."

The report goes on to paint the general direction that needs to be taken in order to remedy the situation facing organisations which undertake complex IT projects. Two key recommendations are: 1) There is an urgent need to promote the adoption of best practice amongst IT practitioners and their customers; and 2) Basic research into complexity and associated issues is required to enable the effective development of complex, globally distributed systems. The discussion and recommendations in this report are strongly relevant and applicable in the Australian context.