Sustainable Software Solutions Supporting Society (S^5)

A journey towards a new Software Engineering Science for the 21st Century

This is a position paper for the Grand Challenge exercise prompted by the Dependable Systems Evolution Challenge discussions.

A number of colleagues from the software engineering community have commented that the current proposal does not fully address the issues of *dependability* and *evolution*, and that the focus of the current work is a little narrow; here we try to open up the discussion and propose a more holistic approach which is, necessarily, both multi-disciplinary and inter-disciplinary in its approach.

The phrase '*correct systems*' was used from time to time during the meeting but what does it mean? Here is a quote from Holcombe & Ipate, "*Correct systems – building a business process solution*" (Springer Applied Computing, 1998):

"*What is a correct system?*

> *A correct system is one that works, one that I understand how to use, one that does not keep going wrong and one that solves the problems that I face – a user;*
>
> *A correct system is one that has been formally proven to satisfy the mathematical formula that defines it – an academic computer scientist.* "

Whatever system is considered, it has to exist within some context, the economic or business motivation for it, the sociological, environmental or operational environment it has to perform within and the managerial and psychological framework within which it is produced, maintained and evolved. We cannot remove these issues from the equation if we are to understand, in a fundamental way, how software systems are made and used.

In brief, our challenge should be to provide a fundamental framework for the understanding of a system within its *complete context*: economic, human, and social as well as the internal logic of its own existence.

In order to explain these issues and to provide some practical examples of the issues and possible ways to deal with them we will examine the component terms in the title individually.

We start with the term *software*; this is the focus of most of the contributions to the panel. Here the emphasis is on the formal specification and the validation of a clearly articulated *requirement*. There is a need to focus on the issues of requirements change and technology change and the consequences of these. However, without trying to understand the causes of and potential need for change there will be a limit to the practical value of this very narrow approach. This raises a very serious problem for those who base their approach on the development and analysis of a formal mathematical model. Even if this is feasible for some systems it leaves us with the problem of understanding how to transform formal models and manage the validation process when the requirements, and thus the model are not *stable* entities.

Although the rationalist approach to software development, whereby the use of mathematical formalisms takes precedence over all other approaches, has contributed greatly to academic software engineering it is important to recognise that other, less formal approaches or even hybrid approaches have much validity. There are, for example, more *agile* approaches to software development, such as Extreme Programming (XP), which have tried to take a fresh perspective on the software development process. Such approaches should not be dismissed on the basis of prejudice but explored in a scientific way in order to establish their capabilities.

What I am suggesting is that we reconsider some of our core attitudes in the light of the experiences of the past 30 years and not assume that the answer is to change the way industry does things without providing real and tangible evidence that what we propose is best! If we are going to understand the real issues of software production we need to try to to handle them, even though it is often difficult, in a realistic way, whether this is by developing suitable formalisms or by adopting other approaches.

I will now look at a number of these issues which are very important and could be the focus of novel and important research.

Solutions: What drives the development of a piece of software is some perceived need and at the most basic level a business case. However it is not quite as simple as this since there are many business models for software companies and it is impossible to generalise. But it should be possible to derive mathematical models and analysis, even if crude and fuzzy, which could inform the fundamental decisions needed to define and implement a possible solution. The principles behind this need to be explored; they are not always obvious to the development team and many (academics) seem to make assumptions about the basic realities of business that may not be valid.

Software may be correct in the technical sense but may not be a solution to the 'business' problem. This can be for many reasons. One in particular is that the business context has changed - business is a very dynamic activity - and the original requirements have altered. This is extremely common and one of the major reasons why systems - even if delivered - are not used. The model in which the requirements are frozen so that a formal model can be built and analysed is not credible. A much more *agile* approach is needed and, unless this is realised, nothing of any great value will result from this challenge. Another problem may be that the non-functional requirements are not satisfied. The formal verification of the software (as currently perceived) - if fully achieved - may not address all of these issues.

Example: Early commercial Internet banking
Soon after the emergence of the Internet a number of business banks started to explore the use of this new technology in business banking. Unfortunately, the new technology was incompatible with the legacy back end transaction systems. Most banks thus started to re-engineer these transaction systems to make them compatible with Internet technology. One, however, built a quick web based front end and went live with the service. They employed many hundreds of data entry clerks who printed out the Internet transactions and typed the details into the back end system—an inefficient and, from an engineering point of view, a bad model. No academic software engineer, let alone a formalist would recommend this strategy! However, this bank grabbed the market, thus producing income to fund the re-engineering of the legacy system. They achieved this and moved seamlessly to it without any break in service. This bank still has 75% of the market share because they got into the market first, provided an adequate service and have prospered. This is a very common business strategy and one that we should ignore at our peril. If the purpose is to succeed in business doing things in a set way may not deliver. What this tells us is that there is a need to match development strategy with market and economic information. This should be possible to do.

Society: the purpose of the challenge should be to aim for a fundamental step change in the quality and appropriateness of the solutions that are to be produced for the benefit of society. This is, of course, a very general ambition but one that should be approached through a scientific and empirical

approach to measuring genuine benefit and utility. Ignoring the social and political realities of an application environment can lead to the software being used inappropriately or not at all. It is vital that enough empirical analysis of the operational environment is carried out so that the designers of the system can understand the likely realities of the world in which the software runs. Whether this can be done in a sufficiently formal way to allow for some kind of simulation of the social context of operation is an open question. It might be possible to provide some sort of modelling and automated analysis of the interactions of the system with its human environment – we should try to find out!

Example: Hospital patient records system – due to Ian Sommerville
In this case software was produced to enable ward nurses to enter patient details and the results of treatments into an integrated information system. It was soon discovered that the records were rarely complete despite the careful design of the user interface. It was only when sociologists/ethnographic researchers studied the ward environment carefully was it realised that the nurses kept on getting interrupted by emergencies when they were using the system. This system thus was neither safe nor dependable even though the software may have been proved correct (in fact it probably wasn't but the point here is that *correctness of the software does not guarantee correctness of the system*).

Supporting: providing a dependable foundation to the societal activity that the solution is benefiting. As we rely more and more on software, the needs of developers to provide a much more dependable product increases. This may not be achieved by formal analysis of the specification, the code or the use of verified platforms alone. It is important to model the key apsects of the operational environment and this is a serious challenge in itself. Most safety critical systems operate in a real time, concurrent, stochastic world dominated by real not discrete valued parameters. Just understanding a simplified discrete model cannot be sufficient.

Example: Behavioural analysis of automotive code
The world's most prestigious mass-market car maker spent much time and money trying to use a variety of analytical techniques and tools in an attempt to calculate important data such as maximal execution time. They were unable to get any sensible analytical answers. Using evolutionary algorithms on the source code, however, provided excellent results. The use of evolutionary testing among other approaches, to solve hard problems in code analysis is now a major area of activity. Introducing AI techniques into software engineering is proving extraordinarily successful and offers a new approach to solving difficult problems.

Sustaining: the process of providing an on-going benefit that can adapt to the needs and changing requirements of society and technology - the evolutionary part.

It has only recently been a realisation that requirements may change and thus the traditional software models involving a fixed requirement, a detailed formal model and the refinement of this into some sort of implementation or design no longer fits with most of the real world.

What may be needed is the development of much more lightweight formal notations which have the ability to model change as a first class consideration. Then it will be possible to examine how any benefits from the formal model, such as test sets, can be exploited despite the existence of change in the model. Test set transformations should be automatically triggered by formal requirements change.

We need to identify areas where change is likely and try to predict it by developing a greater knowledge of that application environment, the politics of the client organisations and the behaviour of the market place. These are all challenging things to identify, characterise and model but there will be benefits in doing so.

Example: Domestic violence information system
We have building a system to record information about domestic violence incidents for 9 different agencies. These include the police, hospitals, housing services, social services and a number of charities: Victim Support, Rape Crisis, etc. This is a very challenging agenda since the various agencies have different objectives, and are very protective of their own data. It was hoped, following high profile tragedies such as the Victoria Climbié case, to develop a system that would provide the means for communicating critical information about people at risk to appropriate agencies. This has not yet been possible since the very basic alerting mechanism first implemented was too much for some of the agencies. The situation is changing, however, and as national policies on data sharing change we will have to change the system, this means that we have to be prepared for this and so have used a lightweight formal method embedded in an XP like process to support this development. Much more work is needed in this area.

So many large projects hit difficulties because of what are essentially *political* problems in organisations – either with the clients or the developers. No amount of technical and formal analysis will overcome such problems. Perhaps there are ways of modelling and analysing these political dimensions that would make a practical impact on the problem.

Finally we should consider the situation of the people who will be building these software systems.

Example: Well-being and design processes
Recent research into the behaviour of design teams, in various areas of engineering by neurophysiologists and work psychologists has highlighted a number of key issues relating to how design teams work, on group cohesion, on individual motivation, anxiety, etc. This is what is generally termed *well being* and its impact on the quality of the artefacts produced is important. Despite the vast amount of effort developing formal notations and tools, on training hundreds of thousands of graduates in their use, the take up of the technology in industry can only be described as abysmal. There may be many reasons for this and until we investigate this issue scientifically it is unlikely that we will understand what to do about it. One area that has been considered is the impact of the generally poor quality of the user interfaces of formal methods tools. It is likely, however, that the fundamental problem is rather more serious than that.

There is now overwhelming evidence that moderate fluctuations in feelings can systematically affect cognitive processing (Ashby, Isen, & Turken, 1999). Isen and others (1987) have shown that mild *positive affect*, of the sort that people could experience every day, *improves creative problem solving* (Mertz, & Robinson, 1985), *facilitates recall of neutral and positive material* (Isen, Shalker, Clark, & Karp, 1978) *and systematically changes strategies in decision-making tasks* (Carnevale & Isen, 1986). Ashby has shown (1999) that many of the cognitive effects of positive mood are due to increases in brain *dopamine* levels that co-occur with mild elevations in mood. The neurophysiological basis of the effect is clear. Research is now looking at how software development processes affect the *positive affect* of the engineers through analysis of their well-being, etc. When this research is applied to the context of software development there are clear differences in the well being levels measured (using Warr's measure) that demonstrate that teams

using lightweight or agile methodologies experience much higher levels of well being.

It has long been known that programmer capability varies greatly; some industrialists have identified a 100 - fold difference in programmer productivity between individuals on the same project. Individual personalities are also a major factor in teamwork situations and there is evidence that the mix of personalities of the team members has a larger impact on the quality of the product than the methodology used. We need to find out much more about these issues.

At the heart of most formal approaches is the construction and analysis of detailed mathematical models at an early stage of the project life cycle. This has been a somewhat bureaucratic approach to project management that has many benefits but may not fit as well with human needs and the encouragement of creativity and problem solving as it could be. The inevitable dynamics of the requirements of most projects is an important issue and one that needs to be addreessed using techniques, notations and tools that are flexible, powerful and user-friendly.

Summary

The purpose of the challenge should be one of *integrating* many of these issues into a coherent, holistic framework that allows for the rigorous representation and analysis of all aspects of a Sustainable Software Solution Supporting Society. This is a major challenge because it is both multi-disciplinary as well as being highly subject to dynamic and evolving technologies and environmental pressures. It may be possible for existing languages and formal techniques to adapt to this new challenge but we need to address these issues seriously if we are going make a real difference in the world of software engineering.

Mike Holcombe, 27/07/04