A Program for Everyone : A Grand Challenge in Software Engineering

Introduction

For the 3 decades or more since the identification of the subject of Software Engineering the interests of academic software engineering research and the needs of much of the software engineering industry seem to have steadily diverged. So much so that some key areas of the academic research are perceived by many to have little or no impact on current industrial practice. Academic concerns have largely been driven by issues of ‘correctness’ and ‘elegance’ whereas industry has focused more on ‘speed of delivery’ and ‘financial success’. These may not be completely incompatible but the intellectual division in the ranks of practising engineers and researchers is such that there is sometimes little opportunity of realistic dialogue and the identification of research themes of interest to both communities. This challenge proposal is an attempt to refocus the research agenda onto areas of greater utility and relevance to the software engineering industry without the loss of rigour and principle for academic researchers.

One fundamental fact of life that is predominant in industry but often ignored in research thinking is the volatility of requirements and the contrariness of clients and the impact this has on the progress of projects. Another important issue is the needs of the user and, although user interface design and related issues is an important research area, it does not seem to be a fundamental one in terms of the mainstream academic research directions in software engineering which appear to be more technology and theory based.

Proposal for a Grand Challenge.

“A Program for Everyone” is a Grand Challenge in Software Engineering. It seeks to provide fast and reliable software solutions to satisfy the rapidly changing information management needs of humans in society. The core of the challenge is to provide total system development capability for *everybody*. The Computer Science embodied in the novel tools will eventually be transparent and the role of Software Engineers will change from solving specific application problems to providing end users with the tools to enable them to solve their own problems.

Computers are used in almost all walks of life and the breadth of scope of software applications is truly astounding. This will increase in the future and raises a fundamental question about the direction of software engineering as a discipline.

A major bottleneck and failure point for many software projects is the communication gap between those who commission and use software artefacts and the developers who build the solutions. The complexity and depth of knowledge required to understand the environment into which an application is to be deployed is becoming more and more challenging. This will accentuate in the future as application domains become more sophisticated and critical. The possibility of training software engineers to be able to understand the subtleties of any application domain is a remote one. The radical solution is to empower the application domain experts to enable them to write their own software solutions without any intervention from professional software engineers. Thus we may see the Chief Executive or senior manager of a commercial company being able to build applications that will enable him/her to manage their company by monitoring and analysing critical activities, data and financial intelligence etc. Since they would know what the key concepts and components of the business are they would be able to specify and construct suitable software for their purposes if they were provided with suitable software support for this independent of the need to employ expensive consultants.

The small businessperson would be able to develop appropriate solutions to mange their stock

control, e-commerce business, customer relations etc. without support from programmers.
The research biologist would be able to develop virtual experiments to explore detailed models of biological systems using terminology and concepts that are relevant to biology.
The list of opportunities is endless. Their common denominator is that the requirements elicitation and program construction will be done in terms that relate directly to the application domain rather than to computing. This will lead to a wide range of different approaches based on application domains rather than on current software engineering parlance. End users won't have to worry about software architectures, class diagrams, traditional programming languages etc. they will be able to build their applications by manipulating concepts and processes that are meaningful to them.
This provides a fierce challenge to the current way that we think about software development, which is technology focused rather than solution focused. We will need to 'get into the minds' of the end users and understand how they think if we are going to succeed. This will be a good thing anyway! We will then need to map this understanding through an automated process for synthesising and validating a software solution.
Thus the challenge will be a multi-dimensional, multi-domain endeavour which will focus much more on the end users' needs, not as present where the emphasis is on finding out what they want to have as a piece of software but on how they might build their own solutions and how this could be supported and automated.
As such, it will be attractive to a wide range of researchers in software engineering, formalists developing mechanisms for defining concepts and processes and automatically reasoning about potential solutions being explored by end-users, automatic verification and testing, system modellers, HCI specialists etc. Thus it will also be, from a software engineering research perspective, a research *Programme for Everyone* (*sic*).
The tools will be validated through many test projects, multi-user, distributed information systems whose core requirements evolve critically. All information is captured in the client's own conceptual framework, with automatic consistency maintenance, model building, code generation and testing.

Relationships with other proposed challenges

The current example of a Software Engineering Challenge in Dependable Systems Evolution, a *verifying compiler*, is complementary to this proposal, with a narrower focus and yet there are a number of aspects that relate well to *programs for everyone*. As part of this proposal there is the need for the self-validation of the artefacts produced although the success of this would be judged, ultimately, by the practical success of the artefact in its application domain and the confidence that the end user has in it rather than by a mathematical property.
Some of the other Grand Challenges, for example in Ubiquitous Computing and in Artificial Intelligence will also have areas of overlap and this would be a strength of the proposal.

Criteria for a Grand Challenge.

These are outlined below:

It arises from scientific curiosity about the foundation, the nature or the limits of a scientific discipline.

It gives scope for engineering ambition to build something that has never been seen before.

It will be obvious how far and when the challenge has been met (or not).

It has enthusiastic support from (almost) the entire research community, even those who do not participate and do not benefit from it.

It has international scope: participation would increase the research profile of a nation.

It is generally comprehensible, and captures the imagination of the general public, as well as the esteem of scientists in other disciplines.

It was formulated long ago, and still stands.

It promises to go beyond what is initially possible, and requires development of

understanding, techniques and tools unknown at the start of the project.
It calls for planned co-operation among identified research teams and communities.
It encourages and benefits from competition among individuals and teams, with clear criteria on who is winning, or who has won.
It decomposes into identified intermediate research goals, whose achievement brings scientific or economic benefit, even if the project as a whole fails.
It will lead to radical paradigm shift, breaking free from the dead hand of legacy.

It is not likely to be met simply from commercially motivated evolutionary advance.

Question: Is it driven by curiosity about the foundations, nature, or limits of basic science?

At the core of this Grand Challenge lies a generally uncomputable problem: automatically generating a system that meets a given set of requirements/needs. We have the questions as to whether there are (significant) classes of real requirements/needs for which we can solve the problem and, if such classes exist, what they are. It will thus be necessary to develop a deep understanding of the nature of real software development problems. We will have to design a language that allows stakeholders to describe a wide range of requirements/needs including functional requirements and non-functional requirements (such as usability and performance) and their relative importance. It is vital that this language is suitable for people with little or no technical background, rather than just engineers and professional software developers, but is also appropriate as the basis for automatically generating a system. In order to do this effectively we will have to develop a clear understanding of the ways in which people describe their requirements/needs. Any language developed must facilitate evolution, users must be able to easily rework their description as their requirements change and must facilitate the process of resolving potential conflicts in the requirements of different stakeholders.

Question: How can it be promoted by competition between teams with diverse approaches?

Software Engineering is a relatively modern "engineering science" with respect to other engineering and science disciplines. Although software engineering has evolved since its establishments in the early 60s, it still strangles to find a comprehensive identity. Software engineering involves many diverse disciplines often advocated by competing communities of researchers and practitioners. Software engineering is continuously evolving by embracing changing technologies, methodologies and terminologies. This results into emergent features of software engineering.

Researchers and practitioners classify software engineering in order to stress specific aspects. The classification of software engineering (technologies, methodologies and terminologies) highlights diverse understanding of software engineering theory and practice. Research and practitioners often distinguish between different levels of "formalities" for specific software engineering aspects (e.g., formal methods, light-weight formal methods, quantitative, qualitative, hard software engineering, etc.). For instance, it is possible to identify two broad viewpoints: Hard and Soft (Hitchins, 1992). These viewpoints highlight some divisions in software engineering: "Hard systems viewpoints are basically those held by designers and engineers who are trying to create systems to meet an understood need in an effective and economic manner. Those in the soft camp caricature the approach as head-down, concerned with optimization, obsessed with quantitative metrics and highly pragmatic. So much so, in fact, that the term system thinking has been purloined by the soft camp as though they alone thought! The soft camp use the term engineer's philosophy, not too endearingly, to describe the hard approach, in which the requirement is stated by a customer and the engineer satisfies the requirement without question. Soft systems viewpoints are those held by behavioural, management, social anthropology, social psychology and other science students concerned with observing the living world, and in particular the human world. Human activity systems are messy, in that they do not exhibit a clear need or purpose - if they can be said to exhibit purpose at all. Indeed, so complex is the real world of people that the idea of driving towards optimal solutions may be a non-starter - perhaps we should see if we can simply understand and concern ourselves with improving the situation." (Hitchins, 1992).

Software engineering, therefore, represents a "Grand Challenge" both from the technology and the social viewpoints. The real "Grand Challenge" beyond software engineering is further to integrate diverse technologies, methodologies and work practices. Unfortunately, R&D projects provide limited resources in order to address challenging long-term objectives. Therefore, it is necessary to engage the Software Engineering Grand Challenge in the medium long-term perspective. This grand challenge involves a holistic viewpoint of software engineering. This comprehensive viewpoint involves the competition between diverse research and practice communities. However, any community has further to address the integration of diverse software engineering aspects.

Question: It will be obvious how far and when the challenge has been met (or not).

We have specified a number of intermediate targets and tests but the real challenge will be to produce systems and a theoretical foundation which will be really useful to end users in a number of application domains. The real test will be whether the end users can actually build something useful in an acceptable way and for this some rigorous evaluation techniques will have to be negotiated with end users and applied to potential solutions.

Question: It has enthusiastic support from (almost) the entire research community, even those who do not participate and do not benefit from it.

There is something here for everyone – for the theoretically minded there are many mini-challenges, trying to formalise application domains, establishing mechanisms for automated checking and analysis, etc. For the more practical there are many problems to overcome including identifying suitable software architectures, building *metacompilers* and so on. The human factors experts will also be fundamental to this project.

Question: It has international scope: participation would increase the research profile of a nation.

This is likely to appeal to researchers in theory and software engineering throughout the world. The opportunities for *localising* the application focus in terms of language, culture, region etc. are manifest. The generic issues are also attractive to all researchers.

Question: It is generally comprehensible, and captures the imagination of the general public, as well as the esteem of scientists in other disciplines.

This should be a great strength of the proposal. The public is becoming increasingly sceptical about large software development projects because of the negative publicity surrounding many high profile project failures. Changing the focus of software development to provide end users with a greater responsibility for solving their own problems fits in well with current social trends towards devolution and subsidiarity. There are great opportunities for collaboration with other branches of Computer Science – HCI, Natural Language Processing, Speech Processing and AI generally. This would be a good thing since the rapid development of computer science has led to fragmentation and a loss of common identity.

Question: It was formulated long ago, and still stands.

There have been some attempts to raise the abstraction level of programming and design notations – some 4th generation tools were motivated in this direction. A commercial software package called *The Last Program* was an attempt in this direction. End user programming has been a long term goal but is still a very long way off. Like many aspects of computing, including the promises of early AI, the results have been disappointing. The moment may have now arrived for real progress due to a juxtaposition of more sophisticated theoretical foundations, increased computational power and a much larger research community.

Question: It promises to go beyond what is initially possible, and requires development of understanding, techniques and tools unknown at the start of the project.

This is also a key aspect of the proposal. There is no way that a domain expert can produce sophisticated software without having to have a detailed knowledge of design and programming techniques. This is unrealistic at present and we need to use the power of computers to solve the problem.

Question: It calls for planned co-operation among identified research teams and communities.

The wide range of aspects of this proposal will need a lot of co-ordination. Various sub themes – both in terms of application focus and in terms of software engineering direction need to be established and must maintain contact in order to achieve the goal of the challenge. Collaboration between different parts of computer science will also be important.

Question: It encourages and benefits from competition among individuals and teams, with clear criteria on who is winning, or who has won.

There are obvious ways in which competition can be encouraged with different teams championing different approaches or working on different application domains. The evaluation of the different approaches will need to be rigorous and transparent. The inclusion of end-user communities in this process will be vital. Within the project there may be competition between, for example, formal approaches to automated testing against more AI driven approaches such as the use of evolutionary techniques. So we are not only talking about decomposition in terms of domains but also about decomposition in terms of ways to solve the same problem, so that a solution could be useful across varied application domains.

Question: It decomposes into identified intermediate research goals, whose achievement brings scientific or economic benefit, even if the project as a whole fails.

There are many different opportunities – some problem domains will be easier to deal with – perhaps they are better researched already (e.g. business information systems, web services etc.) and this will provide mechanisms whereby some progress may be made in a reasonable timeframe.

Question: It will lead to radical paradigm shift, breaking free from the dead hand of legacy.

The focus on solutions and the user context will represent a major change from current research where technology is the focus.

Most present practices of ICT system development are based on reductionist, functional engineering principles. Such practices serve few parts of society and business well. Specifically:-

- they are generally inapplicable to systems where a significant component is human beings working in harness with the technology.
- they rely on specialist technical skills which distance those who are considered incapable of developing ICT systems from those having specific needs for what such systems should provide.

As the underlying technology of information systems continues to advance, present practices will become an increasingly significant barrier to ICT`s effective use in economic and social development.

The Grand Challenge is based on very different principles of ICT system characteristics and development practices, principles in which evolution and change, driven by the ongoing, direct needs and participation of users are paramount, releasing the potential of ICT for all. These principles are non-reductionist and inclusive. A major component of the Challenge will be to embrace these principles whilst enhancing necessary, but appropriate, dependability and assurance.

Question: It is not likely to be met simply from commercially motivated evolutionary advance.

There is little prospect of commercial success in this field without the active involvement of the research community. Naturally, specialist vendors are providing powerful tools and systems for specific application domains but these are still designed by people other than those who will be using them. We wish to empower the end user to create their own systems and this is unlikely to happen if commercial companies are concerned that they will loose their traditional markets.