EPSRC Computing and IT Programme - systems/software engineering sector.

An analysis of some of the strengths, weakness, opportunities and threats as pertaining to parts of the programme, particularly software engineering and theoretical computer science: a personal view. Rather than present this in the classical way I have integrated strengths and weaknesses and the opportunities and threats since the views of these can differ according to one's perspective, thus activities that some might consider to be strengths could also be viewed by others as weaknesses!

Background. The CS & IT budget is a substantial one, the number of computer science departments in UK universities is large and the total of research active academic staff in the CS & IT area is also large. The IT industry is a major player in the UK economy and this is growing rapidly. There are significant skills shortages in this area. The UK Government has targeted research and development in CS & IT as a major strategic priority.

There are two further major issues:

> there is some evidence that the relationships between the publicly funded research activity in the area does not relate well with industry's interests and needs as far as these can be identified;
>
> the main driver for a lot of the commercial activity in software engineering is commercially produced technologies, tools, methods and notations (usually derived by USA companies) rather than the results of academic research projects so what should we do?

Strengths and Weaknesses.

For the past 2 decades the principal strength has been in theoretical computer science, semantics, process algebras, formal specification languages and formal verification including theorem proving. etc. These have been important in the development of the academic discipline but less influential in terms of the industrial development of software - language, tools, methods, paradigms etc. In fact, apart from some niche areas, and in spite of the large numbers of CS graduates emerging from UK universities steeped in this material, the take up in industry has been negligible.

Major reasons for this have been: the difficulty in applying these ideas, the concern that they could not scale up to industry strength problems, the conservatism of the industry and the lack of convincing practical evidence that they are worth the investment. Ironically one of the drivers of some of these techniques has been safety critical software and yet one of the main methods being proposed for this domain, the formal specification language Z, does not comfortably handle time and concurrency, two of the awkward factors that characterise most safety critical systems.

Approaches have been made to make the methods and tools more usable - for example, better interfaces for theorem provers, but generally industry is still sceptical. There may be some mileage in automatic model checking but there the key issue is finding suitable and usable modelling notations that industrial designers are comfortable and confident with.

Software engineering research - principally looking at methodologies, tools and management support have also been beset by problems of convincing evaluation which might persuade software companies to take the risk in adopting a new way of working - something that commercial organisations are rarely going to do unless they are desperate. One hope of the academic community was that the issue of quality would become paramount and this would drive companies to use better founded methods of design. Far from this happening the main driver is *time to market* and quality is sacrificed as a result. The legacy of poor software and the phenomenon of premature release and consequence users having to do much of the testing involuntarily has resulted! The demand for software is so strong and the capacity to deliver being generally insufficient has cushioned companies from the consequences of this approach. Almost everyone now delivers incrementally - with subsequent releases mainly providing *corrections* to previous versions, sometimes on a monthly basis, and increased functionality or better usability/performance being the purpose less frequently.

In the UK some potentially useful work has been done on issues relating to several major bottlenecks and problem areas in software engineering. *Requirements engineering* evolved to try to deal with the difficulties of eliciting what clients actually want and this relates to business analysis and process re-engineering. The problem with this is that often the domain chosen is general business software - databases, information systems etc. Most large companies have their own in-house techniques and these have evolved with their business and their clients. They are often weak but the house documentation is needs to be continuous - many systems are not “green field” or new builds but extensions and adaptations of existing systems driven by the need for some new functionality, or the platform technology has changed - green screen to windows to web based end user interfaces or the non-functional requirements - performance/reliability/usability etc. need to be updated. The point is that the core of the system remains and is not re-engineering very significantly - it is too risky from a business point of view.

There seems to be little work on requirements capture for other application domains - domains where the data is much more complex such as speech, language and what is traditionally referred to as AI. In fact Software Engineering has little to offer the AI community who are building large, complex, sometimes critical software systems using rather ad hoc techniques. This should be a major area of research. Intelligent agents are being proposed as potential solutions to the problems of dealing with complex data, changing environments and needs. These are not always well understood and modelling and testing this technology is still in its infancy. Nevertheless there are strong market pressures to use this technology - even in critical areas. Software engineering research has not addressed this area properly.

The developing cultural industries, involving the exploitation of digital forms of artistic content - art, literature, music, spoken word etc., are a big opportunity for Britain since we have the content - historically our resources in this area have been very rich - and of course we have the English language. These resources can be turned into sophisticated *digital objects*. We need to collaborate with our colleagues in arts and humanities to exploit this by building systems relevant to them. Software can add enormous added value to digital objects in the arts and humanities.

A further issue is that the *dynamism* of the business world means that extensive requirements elicitation processes result in the definition of target systems which are out of date before we start implementing them. Few, if any, of the software engineering methodologies can support rapid requirements change effectively. In many ways the methodologies proposed are so heavy weight and unattractive to developers that companies tend not to use them. This has a consequence when it comes to maintenance and quality assurance. New movements such as eXtreme Programming (XP) and Feature Driven Development (FDD) are attempts, originating from design practitioners in the industrial sector primarily, which attempt to address a number of issues that relate to practical design considerations rather than elegant academic principles. There is scope here for some innovative research to understand and underpin these approaches.

Another major issue is testing. This, and related activities - review, debugging, perfective maintenance - now often consume more than 50% of the budget. Testing OO systems and web-based systems, particularly, is a very difficult area - witness a number of public disasters - boo.com, egg.com etc. Significantly there is very little research going on in the U.K. relating to software testing - there are about 8 academics in this field in the UK. Providing practical and efficient methods for testing these new generation applications would make a major impact.

Opportunities and threats.

My view is that the opportunities for UK software research will focus increasingly on specialist application areas rather than general purpose hardware/software systems where the technology is dominated by the Microsofts, Oracles, Suns etc. of this world.

Although we have academic strengths in some generic technologies they are mainly academic and do not seem to match well with the interests of UK industry. The lack of a strong pull from industry seems to result in esoteric research being done which we cannot be assumed will be valuable one day because the technology is moving at such a pace and in such unpredictable directions. This distinguishes Computer Science from other, more established, subjects where there may be a chance that current research might be exploitable in a decade or so. Much of the theoretical and academic research of the 70s and 80s is now out of date because the technology has changed so rapidly. Much of the research that could be of practical use is often presented poorly - it is either too abstract and mathematical for engineers or the researchers try to persuade industry to take on board new and unproven ideas. The most effective approaches is where research is presented in forms that industry understands and relates directly to industry’s methods, tools and environements. No company is going to abandon methods and tools that have been heavily invested in so that some, possibly illusory, benefits can be achieved. It is far too risky. It is better to take existing industrial approaches and improve them simply and with little managerial cost.

There is a threat that we will continue to try to put much of our resource in the generic technologies but not be able to influence them simply because our impact is diluted by the scale of the activity in these areas in USA.

An opportunity is to build on our strengths in artificial intelligence, on specialist control applications, on new media opportunities based on our cultural capital etc. For this we need to create a clearer focus on these sectors.

As hardware capabilities increase we might consider revisiting some of our historical assumptions. For example current approaches to database technology - based around relational database management systems - is founded on the needs to organise data in a particular way to improve the efficiency of querying. Are there simpler approaches that are more maintainable? Currently if a business context changes significantly the data model has to substantially re-engineering - a costly and wasteful process. OO database technology promised to improve this with the development of supposedly more *human oriented* data models. This has not really happened. Dynamic databases such as those driving WWW search engines couldn’t function in this way. Novel approaches to databases is a potentially important area. Information extraction from free text, semantic based searching of repositories of digital objects etc. are also important areas that the UK could make a big impact in (and is doing so to some extent). However, these areas need to be driven by realistic applications with real, and usually messy, data.

Bioinformatics has turned into a hot area. However, the creation of large databases of biological information is only the start. What biologists are increasingly realising is that they need to do something useful with this data. This is tied up with building increasingly complex system models of biological processes, especially ones that integrate several subsystems into a coherent whole. This, in turn, should guide biologists towards designing better experiments. Here is an area that computer science research can make a major contribution.

Medical informatics has so far proved a problematic area for a variety of reasons. Part of this is organisational with a monolithic client and little agreement about standards etc. The rapidly changing technology coexists uncomfortably with the slow pace of decision making in the organisation. There is significant scope, however, for software advances in niche areas to develop. For example, systems to support specific clinical activities.

One area that has received little research interest is that of *end user programming*. Some prototype systems have been built that allow an intelligent end user - a business person or specialist - to build their own applications through a guided process of business process discovery and automatic code generation. These systems may not be that complex but they would provide significant and cost effective added value to a variety of computer users. It is likely that they would be fairly domain specific.

Some parts of industry are beginning to see automatic code generation as a way forward. This is true in some safety critical areas, for example the certification of software by the aerospace authorities is becoming prohibitively expensive. One commercial system has been certified as a code generator which means that only the designs are reviewed, rather than the code, with significant benefits. This will change the emphasis of design to that of simulation and the analysis of code generators. It will not, however, remove the need for *in situ* testing of software control systems in real and simulated operation environments. The areas of *smart* simulation and testing urgently need attention. Model checking, if it can be managed on an industrial scale is also important. More general state based model checking algorithms are required for this.

Some recommendations:

Domain competitions.
We should select a number of different research laboratories and provide them with funding to build a system using their different methods, then the results would be evaluated to see which produced the best, the fastest and most cost effective solution. To make the process realistic a client should be employed who would change the requirements at regular intervals to see how the different approaches coped. Some areas of AI - speech recognition, natural language processing etc. have advanced rapidly through the use of such competitions - e.g. the Message Understanding Competition. This then provides an environment where the fittest approaches survive and the weakest ones fade away. This has never happened in software engineering research where unproven and unusable technologies have survived for decades.

Software observatories.
Although some progress has been made in a more scientific and empirical approach to software engineering it is flawed for a number of reasons. The idea is that the process of building a system is examined through the use of specific metrics and measurements, the data being analysed to look for significant effects. In many cases experiments are run whereby a number of small teams develop the same system in competition using alternative methods. Usually these experiments take place in universities and the teams are made up of students. The biggest flaw is that the systems are not real ones involving a client with a real need. Thus the hard part has been removed from the experiment! We cannot expect to be able to run realistic experiments in an industrial context because of commercial confidence and risk. However it is possible to use student based teams when they are building real systems for real clients. This could be the basis for a Software Observatory which would be funded so that experienced software engineering researchers could observe closely the projects and teams in such an environment. Only a few universities in the world do projects like this but those that do provide a unique resource for this sort of research.

Application domain driven Software Engineering
It is likely that software engineering will increasingly fragment into specific application domains with their own techniques and culture for building systems. It would make sense to identify some of these important areas and encourage laboratories to focus on developing an application specific variant of software engineering. Suggested areas are: mobile communications, integrated hardware (systems on a chip), biological applications, medical applications, arts and humanities, etc. There are already existing areas where there could be more interplay with software engineering, providing that software engineering was prepared to adapt to the subject - areas such as speech. language, vision, machine learning, signal processing etc. are now building large complex software systems but generally lack a coherent and robust engineering methodology to guarantee robust and efficient solutions. In all these areas the issues of design and testing are key and little work has been done.

Collaborative programme with AHRB

As mentioned before there are many opportunities in the arts and humanities, traditionally a poorly resourced research area. I suggest a new initiative with the Arts and Humanities Research Board along the lines of the EPSRC/BBSRC Bioinformatics initiative. This could have a significant commercial impact since it will exploit the UK's considerable resources in both expertise and content in the arts.

M. Holcombe, University of Sheffield. 23/10/00