D R A F T

The Grand Challenges Exercise of the UKCRC.

Report to the UKCRC from the Programme Committee. May 13, 2003

Background

The UK Computing Research Committee (UKCRC) is a joint expert panel of the British Computer Society and the Institute of Electrical Engineers. It is seeking an appropriate status with the Committee of Professors and Heads of UK Computing Departments (CPHC). It has undertaken as its primary goal that of promoting the good health and high international standing of UK research in Computer Science.

Inspired by the recently completed Human Genome Project, the Committee has noted that the progress of a mature branch of science can occasionally be accelerated by the promotion of a Grand Challenge Project. It has therefore embarked on an Exercise to explore the views of UK academic research scientists on possible topics for a Grand Challenge Project, and on the means for addressing it.

From the beginning, the Committee set severe criteria for judging the maturity of a project proposal for promulgation as a Grand Challenge: the aim was to distinguish a Grand Challenge Project from other kinds of computing research initiative that are promoted currently, or have been in the recent or more distant past. Such a project would be distinctive, but entirely complementary to the more familiar modes of conducting and organising research. A Grand Challenge should enjoy the widest support from the whole scientific community, though only a minority of the community will be actively collaborating in it.

Criteria of maturity of a proposal for a Grand Challenge Project

A Grand Challenge Project is a long-term, large-scale international research project, with clearly defined deliverables, mile-stones, and plans for development, evaluation, and validation of its research results. As a ball-park figure, we took a fifteen year time-scale, involving (say) fifteen leading research laboratories spread over several different countries of the world.

A Grand Challenge Project is a significant commitment of scientific resources, and its promotion needs justification by a strong case that the project, which has been infeasible in the past, can now succeed. This case must be based on a survey of the current state of the art, and its predictable development using known research methods and available research skills.

A Grand Challenge Project has as its primary goal the advancement of scientific understanding or engineering accomplishment in a particular branch of research. It may be specific to that single branch of science, examining its essential nature, its foundations and its limitations; the results of the research may be applicable only within that single branch of science or engineering.

The case for promotion of the Challenge may be strengthened by speculation about the relevance of the eventual research results to the welfare of human society. Any promise of achievement of these benefits should apply to the period which follows completion of the scientific research.

The adoption and promotion of a Grand Challenge Project is pointless unless it leads to a beneficial change in the attitudes and behaviour of scientists, including those not engaged in the project.

The strictness of these criteria, and their novelty in the context of UK computing research, explain why progress towards the emergence and general approval of a good Grand Challenge proposal will be slow. It is quite likely that no suitable Challenge will emerge in the early stages of the current UKCRC Exercise; such a deferred outcome would be far better than the waste of scarce scientific resource that would result from embarking on an immature Grand Challenge Project.

Progress to date

The UKCRC began the Grand Challenges Exercise by appointing a Programme Committee to organise and conduct it, beginning with a Grand Challenges Workshop. The Programme Committee consists of Malcolm Atkinson, Alan Bundy, Jon Crowcroft, Tony Hoare (chair), John McDermid, Robin Milner, Johanna Moore, Tom Rodden and Martyn Thomas. The Workshop was held in Edinburgh on November 2002, and discussed 109 submissions from the UK computing research community.

The details of the call for submissions, together with the planning and conduct of the Workshop and what should follow it, are all reported in detail on the website of the Grand Challenges Exercise:
http://umbriel.dcs.gla.ac.uk/NeSC/general/esi/events/Grand_Challenges/ .
In summary, a set of up to ten possible topics for Grand Challenges was identified at the Workshop for further development, and a champion for each chosen to carry the development forward. A drafting phase followed the Workshop, and in January 2003 several draft proposals were mounted on the website, each to be discussed publicly by email, moderated by the champion, with the discussion archived on the website. The Discussion, to continue until May 26 2003, was advertised to the research community via the CPHC mailing list.

A particular feature of the Exercise is that no submission from the community is ever rejected by the committee; thus, all 109 original submissions (except those withdrawn by authors) are still accessible on the website. Indeed further submissions may be made at any time.

At the date of this report, there are seven discussion group reports, each of which has been subject to considerable discussion both publicly and in private among its drafters. These proposals, with their moderators, are as follows:

In Vivo <=> In Silico: High fidelity reactive modelling of development and behaviour in plants and animals	(GC1)	Ronan Sleep
Science for Global Ubiquitous Computing	(GC2)	Robin Milner

"Memories for life" – Managing information over a human lifetime	(GC3)	Andrew FitzGibbon Ehud Reiter
Scalable Ubiquitous Computing Systems or just Ubiquitous Systems	(GC4)	Jon Crowcroft
The Architecture Of Brain and Mind	(GC5)	Mike Denham Aaron Sloman
Dependable systems evolution	(GC6)	Jim Woodcock
Journeys in Non-Classical Computation	(GC7)	Susan Stepney

The Programme Committee has considered these reports, and their degree of maturity as judged by the criteria, and makes the following recommendations.

Conclusion

In summary, the results of the Exercise to date have fulfilled our reasonable expectations of progress. We have discovered considerable enthusiasm for the concept of a Grand Challenge, and we have identified certain groups of researchers who are keen to work together towards the clearer definition of a common Grand Challenge Project.

In general, the reports of the Discussion Groups have addressed most of the points that distinguish a Grand Challenge Project from other kinds of research initiative. However, there is considerable variation in the depth and detail with which the various points have been addressed. In the following table, we assess very roughly the degree of coverage of each of the reports under the headings of:

Advancement of science or engineering
Planning for the project itself
Feasibility, current state of the art
Benefit to society expected from application

(GC7 is omitted from the table, since its originators have agreed the its research topic, though challenging and important, does not conform to criteria for a Grand Challenge we have adopted.)

	Advancement of science or engineering	Planning for the project itself	Feasibility, current state of the art	Benefit to society expected from application
GC1				
GC2				
GC3				
GC4				
GC5				
GC6				

(This paragraph to be adjusted according to what the PC think.) Most of the reports give a good account of challenging research problems and the potential of

advancement of science and/or engineering in the relevant area. They give a fair account of the long-term potential benefits to society. They vary in the depth and breadth of the survey of the state of the art. And most of them would need more work on a detailed project plan to justify promotion as a GC project.

Next steps

The Programme Committee recommends that the Grand Challenge Exercise should now be split into separate streams, one for each topic of a discussion group report (though some topics may decide to merge). For each topic, we should recruit a small Topic Committee, to take its proposal further.

We recommend that each Topic Committee should work towards producing, in due course, a revised and extended report that would include a comprehensive survey of the state of the art, a more detailed prediction of the phases and subdivisions to structure the work of the project, and an assessment of the expressed support and willingness to participate of the relevant research community, both in UK and outside. The example of the Foresight exercise in Cognitive Systems may be a useful model.

This recommendation allows maximum benefit to be obtained from the work of each of the discussion groups. Even if the format of the Grand Challenge Project is found inappropriate (as for example in GC7), a revised and extended report may advise other ways in which its particular research topic should develop, and may recommend how current research practices, policies, priorities and funding arrangements should be adapted if necessary to promote the topic for the long-term benefit to UK computing research and its international standing.

To address its task, each Topic Committee should be invited to organise a call for contributions and a workshop along the same lines as the Edinburgh Workshop last November, but taking into account the lessons learnt, and the progress made so far, and bearing in mind the alternatives to a Grand Challenge. To this end, the Topic Committee should be encouraged to solicit funds from EPSRC to support this Workshop, and perhaps subsequent ones, with the express aim of defining a long-term research initiative in its Topic.

Historical. The idea of using assertions to check a large routine is due to Turing [12]. The idea of the computer checking the correctness of its own programs was put forward by McCarthy [13]. The two ideas were brought together in the verifying compiler by Floyd [14]. Early attempts to implement the idea [15] were severely inhibited by the difficulty of proof support with the machines of that day. At that time, the source code of widely used software was usually kept secret. It was generally written in assembler for a proprietary computer architecture, which was often withdrawn after a short interval on the market. The ephemeral nature and limited distribution for software written by hardware manufacturers reduced motivation for a major verification effort.
Since those days, further difficulties have arisen from the complexities of modern software practice and modern programming languages [16]. Features such as concurrent programming, object orientation and inheritance, have not been designed with the care needed to facilitate program verification. However, the relevant concepts of concurrency and objects have been explored by theoreticians in the 'clean room' conditions of new experimental programming languages [17,18]. In the implementation of a verifying compiler, the results of such pure research will have to be adapted, extended and combined; they must then be implemented and tested by application on a broad scale to legacy code expressed in legacy languages.

Feasible. Most of the factors which have inhibited progress on practical program verification are no longer as severe as they were.

1. Experience has been gained in specification and verification of moderately scaled systems, chiefly in the area of safety-critical and mission-critical software; but so far the proofs have been mainly manual [20,21].
2. The corpus of Open Source Software [http://sourceforge.net] is now universally available and used by millions, so justifying almost any effort expended on improvement of its quality and robustness. Although it is subject to continuous improvement, the pace of change is reasonably predictable. It is an important part of this challenge to cater for software evolution.
3. Advances in unifying theories of programming [28] suggest that many aspects of correctness of concurrent and object-oriented programs can be expressed by assertions, supplemented by automatic or machine-assisted insertion of instrumentation in the form of ghost (model) variables and assignments to them.
4. Many of the global program analyses which are needed to underpin correctness proofs for systems involving concurrency and pointer manipulation have now been developed for use in optimising compilers [29].
5. Theorem proving technology has made great strides in many directions. Model checking [30-33] is widely understood and used, particularly in hardware design. Decision procedures [34] are beginning to be applied to software. Proof search engines [35] are now well populated with libraries of application-dependent theorems and tactics. Finally, SAT checking [36] promises a step-function increase in the power of proof tools. A major remaining challenge is to find effective ways of combining this wide range of component technologies into a small number of tools, to meet the needs of program verification.
6. Program analysis tools are now available which use a variety of techniques to discover relevant invariants and abstractions [37-39]. It is hoped that that these will formalize at least the program properties relevant to its structural integrity, with a minimum of human intervention.

7. Theories relevant for the correctness of concurrency are well established [40-42]; and theories for object orientation and pointer manipulation are under development [43,44].

Cooperative. The work can be delegated to teams working independently on the annotation of code, on verification condition generation, and on the proof tools.

1. The existing corpus of Open Source Software can easily be parcelled out to different teams for analysis and annotation; and the assertions can be checked by massive testing in advance of availability of adequate proof tools.
2. It is now standard for a compiler to produce an abstract syntax tree from the source code, together with a data base of program properties. A compiler that exposes the syntax tree would enable many researchers to collaborate on program analysis algorithms, test harnesses, test case generators, verification condition generators, and other verification and validation tools.
3. Modern proof tools permit extension by libraries of specialized theories [34]; these can be developed by many hands to meet the needs of each application. In particular, proof procedures can be developed that are specific to commonly used standard application programmer interfaces for legacy code [45].

Effective. The promulgation of this challenge is intended to cause a shift in the motivations and activities of scientists and engineers in all the relevant research communities. They will be pioneers in the collaborative implementation and use of a single large experimental device, following a tradition that is well established in Astronomy and Physics but not yet in Computer science.

1. Researchers in programming theory will accept the challenge of extending proof technology for programs written in complex and uncongenial legacy languages. They will need to design program analysis algorithms to test whether actual legacy programs observe the constraints that make each theoretical proof technique valid.
2. Builders of programming tools will carry out experimental implementation of the hypotheses originated by theorists; following practice in experimental branches of science, their goal is to explore the range of application of the theory to real code.
3. Sympathetic software users will allow newly inserted assertions to be checked dynamically in production runs, even before the tools are available to verify them.
4. Empirical Computer Scientists will apply tools developed by others to the analysis and verification of representative large-scale examples of open code.
5. Compiler writers will support the proof goals by adapting and extending the program analyses currently used for optimisation of code; later they may even exploit for purposes of further optimization the additional redundant information provided with a verified program.
6. Providers of proof tools will regard the project as a fruitful source of low-level conjectures needing verification, and will evolve their algorithms and libraries of theories to meet the needs of actual legacy software and its users.
7. Teachers and students of the foundations of software engineering will be enthused to set student projects that annotate and verify a small part of a large code base, so contributing to the success of a world-wide project.

Incremental. The progress of the project can be assessed by the number of lines of legacy code that have been verified, and the level of annotation and verification that

has been achieved. The relevant levels of annotation are: structural integrity, partial functional specification, specification of total correctness. The relevant levels of verification are: by testing, by human proof, with machine assistance, and fully automatic. Most software is now at the lowest level – structural integrity verified by massive testing. It will be interesting to record the incremental achievement of higher levels by individual modules of code, and to find out how widely the higher levels are reasonably achievable; few modules are likely to reach the highest level of full verification.