DRAFT

The Verified Software Repository: a step towards the Verifying Compiler

GC6 Steering Committee*

December 20, 2004

Abstract

The Verified Software Repository is dedicated to a long-term vision of a future in which all computer systems justify the trust which Society increasingly places in them. This will be accompanied by a substantial reduction in the current high costs of programming error, incurred during the design, development, testing, installation, maintenance, evolution and retirement of computer software. An important technical contribution to this vision will be a Program Verifier: a tool which automatically proves that a program will always meet its requirements, insofar as these have been formalised, without even needing to run it. This has been for over thirty years a challenge for computing research, but the current state of the art gives grounds for hope that it may be implemented in the foreseeable future. Achievement of the overall vision will depend also on continued progress of research into dependability and software evolution, as envisaged by the UKCRC Grand Challenge project in Dependable Systems Evolution.

The Verified Software Repository is a first step towards the realisation of this long-term vision. It will maintain and develop an evolving collection of state-of-the art tools, together with a representative portfolio of real programs and specifications on which to test, evaluate and develop the tools. It will contribute initially to the interworking of tools, and eventually to their integration. It will promote transfer of the relevant technology to industrial tools and into software engineering practice in UK. The Repository will build on the recognised achievements of UK in practical formal development of safety-critical computer applications, and it will establish UK as the leading nation in any future international initiative in Verified Software, covering theory, tools and experimental validation.

1 Introduction

A Scientific Repository is a selected collection of scientific data constantly accumulating from experimental or observational sources, together with an evolving library of programs that process the data. This document presents the steps towards setting up a scientific repository serving the advancement of Computer Science, particularly in the area of strong software engineering and the mechanical certification of computer programs. The long-term goal of the repository is to develop a toolset of software engineering aids for use in the construction, deployment, and continuous evolution of dependable computer systems such that when the tools have proved their maturity, they can be integrated into a standard "verifying" compiler [18], and so meet an outstanding Grand Challenge in Computer Science that goes back more than thirty years. The verifying compiler itself being a substantial component of the programme towards the Grand Challenge project in Dependable Systems Evolution [60].

*This draft has been prepared by Juan Bicarregui, Tony Hoare, and Jim Woodcock. The members of the Steering Committee are: Keith Bennett, Juan Bicarregui, Jonathan Bowen, Tony Hoare, Cliff Jones, John McDermid, Colin O'Halloran, Peter O'Hearn, Brian Randell, Martyn Thomas, and Jim Woodcock

2 Background

2.1 Grand Challenges

The idea of grand challenges has a long history. From the problem of longitude in the 18th Century, through Hilbert's programme for 20th Century mathematics, to the "space race" of the 1960s and the human genome project of the 1990s, grand challenges have led to considerable innovation and have accelerated engineering and technological advancement towards their goal.

An interesting example of the "chain reaction" that can be created by a well articulated grand challenge is the DARPA 2004 grand challenge [41] which offered a $1M prize for a race of autonomous robotic vehicles from Los Angeles to Las Vegas. Although none of the entrants came anywhere close to completing the 150 mile "obstacle course" set by DARPA, and so the prize money was not required, the challenge did result in instigating a huge effort in engineering and fostered significant innovation which might have otherwise been delayed.

In computing, since the early 1990s, Nasa has run a series of "science teams" investigating computational grand challenges covering Geophysics, Astrophysics, and Cosmology [42]. Similarly, computational grand challenges have been identified in Astrophysics [43], Molecular Science [44], Aerospace [47], Energy [48], Environmental Monitoring and Prediction [49], Molecular Biology and Biomedical Imaging [50], Product Design and Process Optimization [51], and Space Science [52]. Grand Challenges in Artificial Intelligence have also been suggested, famously, the Turing test and its more focused versions [46]; and more recently, under the semantic web initiative, the ambitious vision of extracting an encyclopaedic knowledge base from information available on the World-Wide-Web [53].

In the UK, a programme of Grand Challenges in Computer Science has identified seven themes; they are:

GC-1 In Vivo $\Leftrightarrow$ In Silico [55].

GC-2 Science for Global Ubiquitous Computing [56].

GC-3 Memories for Life [57].

GC-4 Scalable Ubiquitous Computing Systems [58].

GC-5 The Architecture of Brain and Mind [59].

GC-6 Dependable Systems Evolution [60].

GC-7 Journeys in Non-Classical Computation [61].

Alongside these runs an underlying principle of establishing a computational model to include, biological and quantum computing and massive distribution over space and time. The whole programme is coordinated through a programme committee and master discussion list, GC-0 [62], under the chairmanship of Tony Hoare.

This paper describes an initiative under the Verifying Compiler strand of work in the Dependable Systems Evolution challenge, GC-6.

2.2 The Verifying Compiler

Ideas behind the Verifying Compiler have a long history. In 1950, Turing first proposed using assertions in reasoning about program correctness; in 1967, Floyd proposed the idea of a verifying compiler [15]; and in 1968, Dijkstra [10] proposed writing the assertions even before writing the program. A verifying compiler is a tool that proves automatically that a program is correct before allowing it to run. Program correctness is defined by placing assertions at strategic points in the program text, particularly at the interfaces between its components. These assertions are truth-valued expressions that are evaluated when control reaches a specific point in a program. If the assertion evaluates to false, then the program is incorrect; if it evaluates to true, then no error

has been detected. If it can be proved that an assertion will always evaluate to true, then the program is correct, with respect to this assertion, for all possible executions.

Early attempts at constructing a verifying compiler were frustrated by the inherent difficulties of automatic theorem proving. These difficulties have inspired productive research on a number of projects, and with the massive increase in computer power and capacity, considerable progress has now been made. A second problem is that useful assertions are notoriously difficult to write. This means that work on a verifying compiler must be matched by a systematic attempt to attach assertions to the great body of existing software. The ultimate goal is that the major interfaces in software components should be fully documented by assertions approaching in expressive power a full specification of its intended functionality.

The Verifying Compiler has been described in some detail in [18].

2.3 Scientific Repositories

A first step in a community effort to develop a verifying compiler would be to set up a *scientific repository* in the area of strong software engineering and the mechanical certification of computer programs. The repository would consist of software analysis tools and example developments to act as challenges for those tools. To apply a tool to a challenge is like a scientific experiment. Its results may be reported in the literature and added to the repository for use in subsequent experiments.

There are many examples of scientific repositories in the UK, some going back several decades. For example, in molecular chemistry, the *Community Computational Projects* [65] assist with the provision of hardware and software and the definition of standards for inter-operability of programs and the Cambridge Structure Database (CSD) [7] is a repository of crystallographic information for organic molecules. The NIST Chemistry web-book [40] contains data on thermal and spectral properties. Notably the CSD System also includes analysis software for 3-D structure visualisation and statistics and numerical analysis.

Repositories of tools also exist in gene sequencing [33], mathematics [16], and many other disciplines. The NIST scientific and technical databases [27] provide repositories of Standard Reference Data for a broad range of scientific and engineering applications.

In Software Engineering, the Open Middleware Infrastructure Institute [30] provides a co-ordinated repository of middleware components for which they have produced a roadmap [28]. In Formal Methods, the Community Z Tools project [9] has produced open-source libraries for building and integrating Z tools. The Software Engineering Institute at CMU provides a forum for the contribution and exchange of information concerning software engineering improvement activities [34].

2.4 Digital Curation and Open Access

The provision of scientific repositories is one example of increased access to research results enabled by the emergence of ubiquitous access to global computing. Indeed, the scientific community is at a watershed in its approach to the management of research outputs. Curation of scientific data is seen as increasingly important and the rapidly increasing capacity of data storage technology means that it is now feasible to preserve far more than previously possible. Furthermore, many years of debate into access to scholarly publishing are coming to a conclusion in favour of increasingly open access to research outputs.

The recent Parliamentary Science and Technology Committee debate into *Scientific Publications: Free for all?* [45] reported support in principle for open access stating that publicly funded research outputs should be freely available. Numerous publishers have recently made declarations

releasing significant copyright to authors and institutions to place papers in institutional repositories. Recent Government prescriptions in Electronic Records Management [75], Freedom of Information [73], Data Protection [74] and Open Source Software [77] also require greater openness.

Institutional repositories are becoming a significant force in the Open Access arena. Both the number of institutions with repositories and the number of records in these repositories are growing rapidly [54]. However, a challenge in moving to self archiving is the provision of metadata in standard formats to allow cross searching of distributed heterogeneous resources. The Dublin Core ISO standard [64] from the library and information management community is notable for its wide applicability but, as the name suggests, only provides basic *core* information. The ISO standard for Open Archival Information System (OAIS) [63] is a reference model for digital archives and is widely adopted in many areas from digital libraries to science archives. The Common European Research Information Format standard (CERIF) [67] is an EU recommendation to member states to allow interoperation of scientific information for the European Research Area.

Technological support for open access includes the DSPACE programme at MIT which provides open-source metadata collection software [68]; the Collection of Open Digital Archives (CODA) at Caltech [69]; and the Lots of Copies Keeps Stuff Safe initiative at Stanford [70]. Technical developments include W3Cs URIs (Uniform Resource Indicators) and DOIs (Digital Object Identifiers) which provide globally unique identification of resources. The UK National Digital Curation Centre [71] and the Digital Preservation Coalition [72] both provide policy advice on digital curation.

Together, Digital Curation and Open Access provide key technologies for underpinning the long-term sustainability of research outputs and will foster a step-change in the scientific process by providing end-to-end integration from application to publication. Openness, transparency and immediate availability of scientific outputs will yield improved auditability both for soundness and value for money, greater opportunities for cross-disciplinary research, and will significantly facilitate progress towards a great many scientific challenges. The verified software repository will take advantage of these initiatives and apply the results in the area of software engineering.

2.5 Ownership and IPR

A fundamental issue in providing an open scientific repository is clearly the ownership of the resources it contains. The current situation with respect to ownership of research outputs is unclear: levels of ownership and IPR rest variously with the investigators, the universities, the research councils and the public. The Open Access argument states that public funding implies public ownership. However, some IP clearly also rests with the investigator and/or their Institution. Overall co-ordination of policy towards ownership of public sector research has been the subject of increased attention since the Baker Report [66].

One scenario depicts public ownership with time-limited exclusive access rights for the researchers. A curation plan is proposed, costed and assessed as part of the normal research proposal review process. Any value added to the research through the wider access to its results can then be assessed alongside the benefit to those directly involved and evaluated against the cost of the research itself and the cost of supporting that wider access. The curation plan would define how and when the data was to be deposited and this would be audited as part of the final review process. For example, it might be proposed that the investigators have exclusive access to the research outputs for an initial period beyond the end of the project to enable publication and follow on research, after which the outputs would be made public for a further specified period. The cost-benefit of curation and provision of public access could be assessed alongside the case for the primary research as part of the proposal. Such models realise a principle where the funding bodies *commission* research on behalf of the public rather than *grant* funds to the researchers.

2.6 The Process of Science

But will the results of these related initiatives ever be exploited in the production of software products used throughout the world? The answer depends on a societal change to a culture of open access and therefore uptake is likely to be incremental.

With the World-Wide-Web and Open Access initiatives, has come an increasing expectation of instant and free access to all forms of data, information and knowledge. From the academic literature and other technical information including scientific data in both its raw and analysed form, to open source and packaged software that can be put together simply to produce new functionality on demand, the world-wide-web has changed everything and is having a radical influence on the behaviour of scientists working in all fields. It is becoming normal practice to simply search for the nearest "experiment" to the one hand and exploit knowledge gained from that as a baseline for the current task.

In software development, the constituency served by the repository is likely to initially include those who have submitted the challenge codes and the analysis programs. But as the scope and maturity of the repository grows, so will the community it serves. Computer scientists from around the world will develop the repository by running the programs on selected data, by interpreting and reporting the results, and by submitting the results to the repository for subsequent use in other investigations. As the project progresses and the utility of the repository grows, the constituency will expand to include leading-edge software development organisations wishing to understand and take advantage of its results to improve the quality of their product. From there, usage will spread into the mainstream of software development in companies looking for assurances of reliability that can be offered by a verifying compiler in order to address the widespread and well-justified reluctance in society to place trust in software [76].

3 The Verified Software Repository

The verified software repository will assist in the development of software by facilitating access to a managed collection of *analysis tools* and a repository of case studies or *challenge codes* to exercise these tools. The emphasis will be on flexibility: the potential to encompass a broad range of analysis techniques, such as verifiers, provers, model checkers, static analysers, test case generators, *etc* and a broad range of design artifacts, such as documents, codes, test suites, safety cases, *etc*.

As a result, the repository will make it easier for software engineers to learn how to use analysis tools effectively. It will also provide examples of good practice for software engineers and facilitate further development and improvement of both the tools and the examples, by bringing them together with common standards.

The long-term goal of the repository, which may take more than a decade to achieve, would be to develop a toolset of software engineering aids for use in the construction, deployment, and continuous evolution of dependable computer systems. It would provide an initial step and continuing technical support for a proposed Grand Challenge project in Dependable Systems Evolution [38], that exploits and develops the particular strengths of UK computing research. One can speculate that such a toolset would bring about a worthwhile reduction of the high costs of errors in programs, both errors detected before deployment and those detected after. When the tools have proved their maturity, they may be integrated into a standard 'verifying' compiler, and so meet an outstanding Grand Challenge in Computing that goes back more than thirty years.

3.1 The Tools

The analysis tools admitted to the repository will include tools that analyse the code for conformity to coding standards, for data flow, for control flow, for alias checking, for type consistency with familiar and novel type systems, for verification condition generation, *etc.* For each language there will be a compiler that makes available an agreed internal interface for inspection and modification by other tools. There will be programs that transform the code, optimise it, and that compare it with previous versions. There will be programs that infer assertions, guess and check conformity to design patterns, generate test data, construct test harnesses, analyse the results of tests, *etc.* Any of these may call on the services of a range of theorem provers, model checkers, constraint solvers, decision procedures, *etc.* In suitable cases there will be competing programs to provide these services, provided that they conform to interfaces that promote interoperability and facilitate direct comparisons of performance.

Our initial list of tools for consideration includes the following: the Daikon dynamic invariant detector [11]; SLAM [3]; the Splint static analyser [12]; the Alloy model checker [21]; the FDR [14] and SPIN [19] model checkers; the ACL2 [23], HOL [17], ProofPower [31], PVS [35], and Z/Eves [32] theorem provers; ESC [25]; the Soot Java optimisation framework [39]; the B-toolkit [2]; JML [24]; SPARK Ada [4]; and the Cyclone safe dialect of the C programming language [22].

3.2 The Challenges

The scientific data held in the repository, the *challenge codes*, will be of varying size and expressed in varying languages. They will be selected because they have proved their significance by actual use in past or present applications. Early examples may be selected from embedded applications (*e.g.*, smartcards, sensors, *etc.*); from critical control systems (reactors, flood gates); and from open sources of off-the-shelf software. The codes will be accompanied by all available forms of machine-readable documentation—specification, design trajectory, development history, internal interface specifications, simulators, test case generators, regression tests, and even conjectures, theorems, and proofs. Selection of challenge codes will be influenced by availability of this material, and an early task of the analysis programs will be, under human guidance, to regenerate parts of it that are missing (sometimes perhaps, the code itself). It is expected that the target levels of certification will be appropriate to the criticality of the application, from basic soundness (crash-proofing), through levels of security, immunity to attack, continuity of service, observance of safety constraints, and ultimately, total conformance, to functional specification.

The following well-known experimental applications may be included: the steam boiler [1], the storm-surge barrier control [8], the Mondex smart card [36], railway signalling problems [13], the Paris Metro [6], the IBM CICS Z specification [20], the inmos floating-point transputer [5], the DeCCo high integrity compiler [37], and ammunition control systems [26]. More ambitious examples may include verifying selected open-software web-services for their essential functionality, and for crash-freedom and deadlock/livelock freedom. This might lead to verifying the fundamental software building blocks of the whole Internet, with respect to their key properties for their serviceability.

3.3 Systems evolution

Many of the challenge codes will be undergoing evolution at the hands of their owners and users, and it will be part of the challenge to track the evolution. If the challenge is met, the owners will be pleased at some stage to take over the annotated code, and use the tools to assist in its further evolution. Of course, the tools will only address aspects of the process of evolution that

can be formally expressed. The Challenge will be to widen as far as possible that range, and to collaborate with investigations into the wider aspects of evolution that would not benefit from formalisation. Similarly, the concept of dependability has many important aspects that cannot yet be formalised, and many that never will be. The project will aim to extend the range of formalisation, and collaborate with centres researching the wider problem, using a wider range of scientific techniques than mathematical modelling and formalisation.

4 Current Plans

We now describe the specific objectives, proposed activities and workplan for a coordinated community project to establish the repository described above. We begin by giving the goals of the project.

4.1 Goals

1. To establish and maintain UK leadership in a long-term world-wide Grand Challenge project, specifically: Verified Software – Theories, Tools and Experiments.
2. To promote scientific collaboration and constructive rivalry among researchers engaged in the project.
3. To provide educational, advisory, administrative, public relations and programming support for UK and European participants.
4. To communicate and collaborate with similar repositories in other parts of the world.
5. To assist in the transfer of maturing technology to industrial and commercial exploitation.

4.2 Objectives:

To accelerate the advancement of technology contributing to the development of the *Verifying Compiler* by providing a repository of tools and challenges for software analysis and hence facilitate progress towards the Grand Challenge in *Dependable Systems Evolution* . Specifically,

1. To facilitate the widespread and effective use of software analysis tools by providing access to a managed collection of tools such as verifiers, provers, model checkers, test case generators, test rigs, *etc*).
2. To improve understanding of design issues in enabling software verification through provision of a managed repository of case studies demonstrating good practice.
3. To encourage further development and improvement of both the tools and the challenges by bringing them together with common standards to enable the example developments to be used to improve the design of the software analysis tools
4. To provide resources to support the repository users in order to encourage its widespread use and hence maximise the benefits arising from its provision.

4.3 Key outcomes over one, five and ten years

Year 1

Milestones

1. Establish repository and supporting resources and services: recruit and train staff, install hardware and software, design and implement website, set up helpdesk.
2. Recruit and assemble an initial constituency of users.
3. Establish and agree terms of reference for the *User Forum*.

4. Set up and agree terms of reference for the *Committee of Guidance*.
5. Populate the repository with case studies contributed by the user community.
6. Populate the repository with at least three research tools selected from existing UK and European sources.
7. Populate the repository with at least three major challenge codes selected from existing critical applications.

Result

- An established repository of tools and challenges serving and an identified community of software engineering researchers wishing to use it.

Year 5

Milestones

1. Support provided for full range of discrete tools, say fifteen, of which six are in their second version.
2. Support provided for a full range of challenge codes, say fifteen, of which six have been subjected to experimental annotation and proof, with the results recorded in the repository.
3. Substantial interoperability between tools and challenges: each code being compatible with several tools, and each tool applicable to several codes.
4. Agreed standards published for universal interworking of tools and challenge codes.
5. Trial integration of a selected group of tools.

Results

- An active community of academic and industrial users of the repository depositing and using the repository.
- A significant library of software development examples to assist with education and training in software engineering techniques
- Some examples of community development of case studies by interested parties applying their tools and techniques to "library" challenges
- Significantly increased interoperability of tools through standardisation of interfaces.

Year 10

Milestones

1. An integrated toolset of mature tools in general use by researchers.
2. Almost full interworking of tools and challenge codes.
3. Fifty challenge codes, up to a million lines in length.
4. Industrial and commercial trials underway.
5. Some parts of the technology transferred to commercial toolsets.
6. Established library of twenty software systems, mechanically verified to an appropriate and formally defined level of correctness, and beyond.

Results

- Availability of "one-click" analysis tools offering some assurance for minimal investment
- Regular examples of community development of case studies by interested parties applying their tools and techniques to "library" challenges.

- A self serving community of repository users depositing and exploiting the tools and challenges in the repository.

4.4 Activities

The project will undertake a number of activities and manage a suite of resources on behalf of the strong software engineering community. Co-ordinated management of these activities and resources will improve overall efficiency by enabling some standardisation and economies of scale. Examples of activities which would be appropriate are listed below. The activities break down into the five workpackages described below.

4.4.1 Tools

Each year, we will select a nuymber of tools for inclusion in the repository. We will negotiate with their authors for their acquisition, and then install them in the repository. We will provide support for dissemination updates from the originators of the tools in order to facilitate their uptake by example challenges.

- Negotiation with providers of analysis tools for the requisite releases and IPR, together with material to support an advisory role, and a feedback path for correction and improvement of the tool.
- License coordination and negotiation where appropriate.

4.4.2 Challenges

Each year, we will select a number of challenge codes for inclusion in the repository. We will negotiate with their authors for their acquisition, and then install them in the repository. We will provide support for dissemination updates from the owners of the challenges in order to facilitate their uptake by tool providers.

- Negotiation with providers of challenge codes for requisite releases and IPR, together with specifications, test sets, development trajectories, etc.
- Copyright coordination and negotiation where appropriate.

4.4.3 Repository

This work-package will set up, manage and develop the repository, which will be organised around a web-based system for depositing and accessing tools, challenges, training materials, publications, and project data. It comprises two parts. In Task 1, we will gather requirements, design, build and maintain the repository and web-based access to it. In Task 2, we will develop standards and supporting software enabling interoperability between tools and challenges.

- Provision of a repository for use by the community to deposit and access tools and challenges for software analysis.
- Design and implementation of interface converters to ensure that the same (selected) analysis tools will apply to the same (selected) challenge codes. (Some of this work could be contracted out or commissioned).
- Co-ordination of the development of standards to enable interworking of tools and challenges.
- Negotiation with providers of tools and codes for standards that will facilitate interworking without specialised converters.
- Provision of aids to standards compliance, monitoring and enforcement.

4.4.4 User Support

This work-package will also be responsible for providing support to the constiuency of potential users of the repository. It will provide supporting material, events and a helpdesk for users of the repository.

- populate, maintain, refine, and extend a web-site framework including:
  - guidance on installation and/or web use of tools
  - guidance of the case studies software
  - training material on use of the repository including that arising from the organised events.
- Helpdesk providing first level support for repository users.
- Support for the publication of newsletters and a journal devoted to publication of the results of verification experiments, before the results are incorporated in the repository.
- Planning and conduct of educational courses and updates for participants from the community.
- Organisation of user forums and events including an annual Marktoberdorf-style summer school devoted to training in the use of that year's tools and in the understanding of that year's challenge codes.

4.4.5 Management and Dissemination

This workpackage will coordinate the management of the project and will oversee the publicity and public relations surrounding it.

- Establishment and administration of a management structure to pursue the scientific goals of the project and to act as guarantors of its scientific integrity. (See Section 4.5.)
- Assistance in establishment and maintenance of scientific relations with similar repositories in other countries.
- Organisation and hosting for meetings, workshops, conferences, summer schools in relavent topics.
- Establishment and organisation of newsletters, websites, and publications, including an international Journal of record for the results of the research.

4.5 Management

We will establish an oversight body, the *Committee of Guidance*, to steer the project and to monitor its use of resources. The Committee of Guidance will review the programme of work and set priorities annually, and review progress and performance metrics against that programme quarterly. The initial Committee of guidance will review the programme of work for the first year and will be to establish a user forum and a mechanism for ongoing electing the future Committee of Guidance. The Committee will consist originally of the promoters of the project[1], and subsequently of those elected by the constituency.

We will also establish a *Stakeholder Forum* for users of the facility (tool providers, case study providers, users, etc) to give feedback on the operation of the facilities provided and advise the Committee of Guidance. Initially membership of the user forum will be open although it may eventuially become necessary to limit numbers by election of representatives from classes of user[2].

[1] Suggested initial members of the Committee of Guidance are: Tony Hoare (Microsoft), Cliff Jones (Newcastle), Colin O'Halloran (QinetiQ), Peter O'Hearn (Queen Mary), and Jim Woodcock (York), Juan Bicarregui (RAL),

[2] Suggested initial members of the Stakeholder Forum are: those who have contributed content to the repository and any other interested parties such as representatives of their institutions, representives of the funders *etc.*

4.5.1 Committee of Guidance

The role of the Committee of Guidance is:

1. To pursue single-mindedly the scientific goals of the project, and act as guarantors of its scientific integrity.
2. To direct the policies of the repository and monitor performance.
3. To lay down appropriate procedures for the selection and classification of material for acceptance into the repository.
4. To assess annually the current status of the repository, and draw up prioritised plans for its future progress.
5. To negotiate with its constituency a division of labour to implement the plans.
6. To negotiate issues of inter-working standards and workplans with its constituency and with related centres in other nations.
7. To lay down and administer fair procedures for entry of new candidates to the constituency.
8. To make commendations of teams who have made exceptional contributions.
9. To arrange for conferences, workshops, seminars, summer schools, meetings, and for publication of newsletters and journals in furtherance of the aims of the project.
10. To promote wider education in the understanding of new tools and their potential use.
11. To promote general public awareness of the nature of the science and the goals and achievement of the project.

4.5.2 Stakeholder Forum

The Stakeholder Forum will discuss options for the content and function of the repository and advise the Committee of Guidance (CoG) on its development.

The role of the Stakeholder Forum is:

1. To provide cannels for coordination of the community of potential users of the repository
2. To provide feedback to the CoG on the repository and its associated activities
3. To advise the CoG on the content and functionality of the repository
4. To contribute to the organisation events, publicity, and other materials concerning the repository
5. To make recommendations to the CoG on the strategic direction for the repository
6. To liaise with other related user fora, such as Z users, BUG, etc.

4.6 Resources

Due to limited resources, the project will necessarily have to start relatively small. However, a certain scale of perhaps a dozen tools and a dozen case studies would be the minimum required to establish the facility as a useful addition to the status quo. Once established, we imagine that gathering momentum through self-service contributions from facility users (*c.f.*, Wikipedia) would provide significant gearing to the resources provided. Ultimately we can expect a fairly large scale operation being maintained through relatively little dedicated resource.

5 Discussion: Towards Dependable Systems Evolution

Systems are not static. Throughout their lifecycle, systems are subject to modifications, upgrades, reconfigurations, and extensions. In short, systems evolve. They evolve from the moment they are conceived, through all the stages of their development and throughout their service lifespan. Their

decommissioning, if this should ever happen, can be seen as the final stage of evolution: replacement by a fitter variety. Dependability, if it is an issue at all, is just as important at every stage of this lifecycle. From commissioning, through upgrades and reconfigurations, to decommissioning, today's systems rarely operate in isolation but must remain compatible with their environment as they change themselves and must adapt to environmental changes it without compromising their dependability.

Compatibility of system components in a compound system is today a major hurdle to the dependability of such systems. Even for shrink-wrapped off-the-shelf packages, compatibility with versions of operating systems and other applciations can be an issue. Compatibility of modules in an even slightly more complex system configuration, such as a server running an operating system, database and variety applications on the database, is even more difficult and the dependability of each particular configuration is often based on testing rather than any designed-in features. On the other hand, incompatibility of components is the accepted norm and is tolerated as an inevitable consequence of complexity. Today's version control and configuration management techniques can be successfully employed within a controlled environment of a particular software suite, but do not easily extend to heterogeneous systems or systems of systems, where demonstration of compositionality of system components essentially relies on testing *in situ*.

Formal methods have for many years advocated compositionality as a way to manage complexity in software and systems design and development through decomposition of the design and verification of it. This approach to structuring, if extended to more aspects of the lifecycle, for example, to requirements and testing, can go a long way to achieving interoperability between systems components. However, because of the wide variety of possible ways to configure software, and in particular the large number of potential interfaces to essentially the same piece of functionality, such structuring needs further features if it is to help with maintenance and reconfiguration. If software is to have the same success in reuse and reconfiguration as other engineering disciplines, we must also achieve flexibility through standardisation and reuse of commonly adopted components. What is required is the ability to put components together with confidence that the dependibility of the whole will be predictable from the dependibility of the parts. A verifying complier, able to demonstrate at compile time that the software it produces will perform as required would be a significant step forward in the ability to make such predictions.

But standardisation of software functions and interfaces is not enough. To enable dynamically reconfigurable systems requires dynamic assessment of new components when they are brought into the system such as is done, for example, in the USB standard for peripherals which enables different peripheral devices to be added whilst dynamic interaction between platform and peripheral is undertaken to establish the correct interface for communication and to reconfigure the platform to exploit the new hardware. To establish comparable standards enabling dynamic reconfiguration of software would require dynamic assessment of the interface and function of software and is certainly way beyond the current state of the art of software engineering. A verifying compiler, able to assess dynamically the properties of software in order to determine automatically whether a component is applicable to a given purpose would provide a very significant step towards realising this vision.

Diminishing cost of hardware and increasing numbers of installations open the potential for ubiquitous, highly-functioned assisting devices for very many aspects of everyday life. However major benefit arises only if these components can communicate in a dependable manner ultimately enabling trustworthiness to be placed on them. The emergence of grid infrastructures and in particular the potential for mobility from the next generation grids will further open the possibilities for such applications to facilitate geographically independent access to resources on a global scale, provided the significant integration and interoperability aspects can be addressed. Interoperability is therefore one of the greatest challenges to enabling dependable systems evolution, itself an

essential constituent for global ubiquitous computing. Again, the verifying compiler will be a major facilitator in this respect.

The existence of the repository proposed here is expected to have a significant effect on the behaviour of scientists working in the field and so ensure that the results of their collaborative research are complementary and cumulative. If successful, this will have a radical influence on the speed of scientific progress in the area and will bring foward the development of the verifying complier and with it the more widespread application of software analysis tools into mainstreeam software engineering practice.

References

[1] J.-R. Abrial, E. Börger, and Hans Langmaack (editors) 1996. *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control. Lecture Notes in Computer Science* **1165**. Springer-Verlag.

[2] J.-R. Abrial 1996. *The B-Book: Assignbing Programs to Meanings*. CUP.

[3] Thomas Ball *et al.* 2001. Automatic predicate abstraction of C programs. *ACM SIGPLAN Conference on Programming Language Design and Implementation* 203–213.

[4] J. Barnes 2003. *High Integrity Software: the* SPARK *Approach to Safety and Security*. Addison-Wesley.

[5] G. Barrett 1989. Formal methods applied to a floating-point number system. *IEEE Transactions on Software Engineering* **15**(5):611–621.

[6] M. Carnot, C. Da Silva, B. Dehbonei, and F. Mejia 1992. Error-free software development for critical systems using the B-methodology. *Third IEEE International Conference on Software Reliability*: 274–281.

[7] `www.ccdc.cam.ac.uk/products/csd/`

[8] M. Chaudron, J. Tretmans, and K. Wijbrans 1999. Lessons from the application of formal methods to the design of a storm surge barrier control system. *FM'99: World Congress on Formal Methods in the Development of Computing Systems. Volume II*. Editors: J. M. Wing, J. Woodcock, and J. Davies. *Lecture Notes in Computer Science* **1709**:1511–1526 Springer-Verlag.

[9] `czt.sourceforge.net/`

[10] E. W. Dijkstra, A Constructive Approach to the Problem of Program Correctness. BIT 8, No. 3, pp.174-186, 1968.

[11] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin 2001. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering* **27**(2):1–25.

[12] D. Evans and D. Larochelle 2002. Improving Security Using Extensible Lightweight Static Analysis, *IEEE Software*.

[13] *FMERail: Workshops on Formal Methods in the Railway Industry*. `www.ifad.dk/-Projects/fmerail.htm`.

[14] *Failures-Divergence Refinement: FDR2 User Manual*. Formal Systems (Europe) Ltd 2003. `www.fsel.com`.

[15] R. W. Floyd, Assigning meanings to programs, Proc. Amer. Soc. Symp. Appl. Math. 19, (1967) pp.19–31.

[16] `gams.nist.gov/`

[17] M. J. C. Gordon 1988. HOL: A proof generating system for Higher-Order Logic, *VLSI Specification, Verification and Synthesis*, Kluwer pp.73–128.

[18] C. A. R. Hoare, The Verifying Compiler: a Grand Challenge for Computer Research, JACM(50) 1, pp.63–69 (2003).

[19] Gerard J. Holzman 2004. *The SPIN Model Checker: Primer and Reference Manual*, Addison Wesley.
[20] I. Houston and S. King 1991. CICS project report: Experiences and results from using Z. *VDM'91: Formal Development Methods. Lecture Notes in Computer Science* **551** Springer-Verlag.
[21] D. Jackson 2002. Alloy: A lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology* ***11***(2):256–290.
[22] T. Jim, G. Morrisett, D. Grossman, M. Hicks, J. Cheney, and Y. Wang 2002. Cyclone: A safe dialect of C. In *USENIX Annual Technical Conference*, Monterey.
[23] M. Kaufmann and J S. Moore 1996. ACL2: An industrial strength version of Nqthm. *Compass'96: Eleventh Annual Conference on Computer Assurance.*
[24] Gary T. Leavens *et al.* 2003. An overview of JML tools and applications. ***in*** Arts & Fokkink (editors) *Eighth International Workshop on Formal Methods for Industrial Critical Systems (FMICS'03). Electronic Notes in Theoretical Computer Science* volume ***80***:73–89. Elsevier.
[25] K. R. M. Leino, G. Nelson, and J. B. Saxe 2000. *ESC/Java User's Manual.* Technical Note 2000–002. Compaq Systems Research Center.
[26] P. Mukherjee and V. Stavridou 1993. The formal specification of safety requirements for storing explosives. *Formal Aspects of Computing* **5**(4):299-336.
[27] `www.nist.gov/srd/`
[28] `www.omii.ac.uk/OMIIRepositorySubmissionStandardsV1.pdf`
[29] `www.omii.ac.uk/OMII_Roadmap_Feb04.pdf`
[30] `www.omii.ac.uk/`
[31] `www.lemma-one.com/ProofPower/index/`.
[32] Mark Saaltink 1997. The Z/Eves System. *Proccedings of the 10th International Conference of Z Users.* Lecture Notes in Computer Science ***1212***:72–85.
[33] `www.sanger.ac.uk/Software/`
[34] `seir.sei.cmu.edu/seir/frames/frmset.help.asp`
[35] U. Shankar, K. Talwar, J. S. Foster, and D Wagner 2001. Detecting format string vulnerabilities with type qualifiers, *Proceedings of the 10th USENIX Security Symposium.*
[36] S. Stepney, D. Cooper and J. C. P. Woodcock 2000. *An Electronic Purse: Specification, Refinement, and Proof*, **PRG-126**, Oxford University Computing Laboratory.
[37] S. Stepney. Incremental development of a high integrity compiler: Experience from an industrial development 1998. *HASE'98: Third IEEE High-Assurance Systems Engineering Symposium.*
[38] UKCRC 2004. *Dependable Systems Evolution.* `www.nesc.ac.uk/esi/events/-Grand_Challenges/proposals/dse.pdf`.
[39] Raja Vallée-Rai, *et al.* 1999. Soot—a Java optimization framework. *Proceedings of CASCON 1999* 125–135.
[40] `webbook.nist.gov/`
[41] The DARPA Grand Challenge for Autonomous Robotic Ground Vehicles, `www.darpa.mil/-grandchallenge04/sponsor_toolkit/brochure.pdf`
[42] Nasa, Science Team III Grand Challenge Investigators, Computational Technologies for Earth and Space Sciences, `ct.gsfc.nasa.gov/grand.st3.html`.
[43] Institute for Pure and Applied Mathematics, Grand Challenge Problems in Computational Astrophysics, `www.ipam.ucla.edu/programs/pca2005/`.
[44] The Computational Grand Challenge Applications (CGCA) projects in environmental molecular science `mscf.emsl.pnl.gov/research/intro_cgca.shtml`
[45] Parliamentary Science and Technology committee report into scientific publications (20/7/2004) `www.publications.parliament.uk/pa/cm200304/cmselect/cmsctech/-`

399/399.pdf.
[46] Stanford Encyclopedia of Philosophy, The Turing Test, plato.stanford.edu/entries/-turing-test/.
[47] Aerospace GC ftp://avalon.caltech.edu/pub/murray/preprints/mur03-ejc.pdf.
[48] Energy GC
[49] Grand Challenges in Earth and Environmental Sciences: Science, Stewardship, and Service for the Twenty-First Century, Mary Lou Zoback, U.S. Geological Survey, www.environmentalsafeguards.com/zoback.pdf.
[50] dcegod.nci.nih.gov/epi/BCBRWG.htm.
[51] Product Design and Process Optimization GC
[52] Space Science GC
[53] Semantic Web Road map www.w3.org/DesignIssues/Semantic.html (See also for example www.w3.org/DesignIssues/RDFnot.html www.cs.vu.nl/ frankh/postscript/-IJCAI99-III.html www.ht03.org/papers/pdfs/7.pdf).
[54] The ePrints-UK project, Statistics eprints-uk.rdn.ac.uk/stats/?action=table.
[55] In Vivo ↔ In Silico, www.nesc.ac.uk/esi/events/Grand_Challenges/proposals/-ViSoGCWebv2.pdf, Contact: Ronan Sleep.
[56] Science for Global Ubiquitous Computing, www.nesc.ac.uk/esi/events/-Grand_Challenges/proposals/Ubiq.pdf, Contact: Marta Kwiatkowska and Vladimiro Sassone.
[57] Memories for Life, www.nesc.ac.uk/esi/events/Grand_Challenges/proposals/-Memories.pdf, Contact: Andrew Fitzgibbon and Ehud Reiter.
[58] Scalable Ubiquitous Computing Systems, www.nesc.ac.uk/esi/events/-Grand_Challenges/proposals/Memories.pdf, Contact: Jon Crowcroft.
[59] The Architecture of Brain and Mind, www.nesc.ac.uk/esi/events/Grand_Challenges/-proposals/ArchitectureOfBrainAndMind.pdf, Contact: Aaron Sloman.
[60] Dependable Systems Evolution, www.nesc.ac.uk/esi/events/Grand_Challenges/-proposals/dse.pdf, Contact: Jim Woodcock.
[61] Journeys in Non-Classical Computation, www.nesc.ac.uk/esi/events/-Grand_Challenges/proposals/stepney.pdf, Contact: Susan Stepney.
[62] The Grand Challenges Exercise of the UKCRC; Tony Hoare, Malcolm Atkinson, Alan Bundy, Jon Crowcroft, John McDermid, Robin Milner, Johanna Moore, Tom Rodden, and Martyn Thomas,
www.nesc.ac.uk/esi/events/Grand_Challenges/PC-report.pdf
[63] The Open Archival Information System (OAIS) reference model for digital archives, ISO standard, ISO 14721.
[64] The Dublin Core Metadata Initiative dublincore.org/
[65] The Collaborative Computational Projects (CCPs) www.ccp.ac.uk/
[66] The Baker Report. "Creating knowledge creating wealth". Realising the economic potential of public sector research establishments. A report by John Baker to the Minister for Science and the Financial Secretary to the Treasury. August 1999, www.hm-treasury.gov.uk/Documents/Enterprise_and_Productivity/ Research_-and_Enterprise/ent_sme_baker.cfm
[67] CERIF: the Common European Research Information Format, www.cordis.lu/cerif/
[68] dspace.mit.edu/index.jsp.
[69] Caltech Collection of Open Digital Archives (CODA) library.caltech.edu/digital/.
[70] Lots of Copies Keeps Stuff Safe lockss-docs.stanford.edu/.
[71] The UK National Digital Curation Centre www.dcc.ac.uk/news.html.
[72] the Digital Preservation Coalition www.dpconline.org/graphics/index.html.

[73] Freedom of Information Act 2000, HMSO, www.hmso.gov.uk/acts/acts2000/-20000036.htm.
[74] Data Protection Act 1998 , HMSO, www.hmso.gov.uk/acts/acts1998/19980029.htm.
[75] e-Government Policy Framework for Electronic Records Management , Cabinet Office, e-Government Unit, www.nationalarchives.gov.uk/electronicrecords/pdf/-egov_framework.pdf.
[76] Cybertrust and Crime Prevention Project, Department of Trade and Industry, Foresight, June 2004. www.dti.gov.uk, DTI/Pub7186/5K/NP www.foresight.gov.uk/-cybertrust.html.
[77] Open source software use within UK Government. Open Source Software Policy Document, The Office of Government Commerce www.ogc.gov.uk/oss/OSS-policy.pdf.