Report of Workshop on Problem-Solving Environments and Scientific IDEs for Knowledge, Information, and Computing (SIDEKIC 98)

On 4-5 December 1998 researchers from several universities, national laboratories, software companies, and government funding agencies met at Santa Fe, NM for the 1998 *Scientific Integrated Development Environments for Knowledge, Information, and Computing Workshop*. The purpose of this meeting was to summarize the state-of-the-art in the area of problem-solving environments (PSEs) for scientific and engineering computation, and to map out directions for future research in the area. This report presents some of the results from the meeting and recommends promising areas for further work. This report begins with a justification of the need for PSEs, which are also commonly called computational workbenches. Next a listing of characteristics that many PSEs share is presented, followed by a small sample listing of current systems. Design goals and future directions, with an emphasis on research issues, are outlined, followed by summary findings and conclusions.

1. Reasons for PSEs in Scientific and Engineering Computing

Computation and simulation have become major driving forces in modern scientific research and engineering design, and have been significant engines for national productivity and knowledge. Computational methods provide tools that take us to the quantum level of sub-nuclear forces in quantum chromodynamics, to the literally universal level of cosmology, and to a generally broader scientific understanding by allowing us to encompass and digest systems as large-scale as ocean and climate modeling.

Increasingly, however, the complexity of the scientific computing process has become a major hindrance to further progress. The computations have become more complex both in size and the amount and types of computer-assisted facilities required. Ten years ago modeling fluid flows in a system with 5000 elements was adequate to significantly advance the understanding of fluid dynamics; now scientists routinely must solve problems with 10^6 or more elements. Additional physical phenomena such as thermal effects, moving boundaries, and shock wave resolution must be modeled where previously they could be ignored or roughly approximated. Scientists are using real-time data collection instruments, accessing and using large distributed databases, and relying on sophisticated visualization systems for applications. The limits of current software methodologies are being reached, and significantly more time is being spent in debugging and validating code � in spite of the widespread availability and use of sophisticated debugging systems. This growth in computational complexity requires computational researchers to move to a higher level of abstraction in dealing with their computing systems. Problem-solving environments furnish that higher level of abstraction.

Other forces are also driving the development of computational workbenches. While even five years ago most scientists using computational methods wrote and managed all of their own computer programs, now a typical lab must use libraries and packages from a variety of sources, and those packages might be written in many different computer languages. Engineers and scientists now have a wide choice of computational modules and systems available, enough so that navigating this large design space has become its own challenge. Scientists and engineers are seeking to couple large computational systems to get more accurate simulations; prominent examples are combining ocean and climate models, or handling the interactions between fluid flows and the structures containing those flows. This means teamwork and multidisciplinary approaches are vital, since few can master all the required fields of expertise needed for a single one of those coupled computations. PSEs provide a natural platform to enable collaboration and leverage expertise from different fields. A computational workbench, for example, can provide resources for structures modeling developed by a mechanical engineer, which a scientist can then use as part of a fluid flow simulation.

In addition to this horizontal leveraging, PSEs provide a basis for vertical integration of computational knowledge. Most researchers will lack a knowledge of all the levels of techniques that are part of a major simulation: the relevant disciplinary knowledge, the best computational techniques, the use of algorithms and data structures, the associated programming techniques, the relevant GUI and HCI design principles, and methods for mapping the computations to high-performance computer architectures. Computational workbenches allow specialists at all levels of this computational hierarchy to contribute, without having to become experts in all the other levels. Even when it is feasible for a single person or research group to master the necessary details, it is a poor utilization of a researcher�s time: astronomers should be free to carry out research in astronomy instead of spending significant time developing a visualization package. At the same time, computational workbenches can give their users a choice in the level of details presented, so that an astronomer who wants to customize or even develop a visualization package is still free to do so.

In all fields of computational science, the largest cost has shifted from computer expenses to researchers� time. There is strong informal and some formal evidence that PSEs provide a significantly decreased time to solution, implying reduced overall cost for the solution as well as more timely research. For example, the PDE solving system DEQSOL was measured (compared to Fortran) in the late 1970s to provide a decrease in programming time by a factor of about 100 and a decrease in execution time by a factor of about 3. For another example, an informal survey by some workshop participants indicates that computational scientists spend about half their time building tools rather than doing the actual science. In some fields which are starting to use computational workbenches this is literally the difference between life and death � for example, better techniques for computing and visualizing the location of brain cancers. By allowing computations such as visualization or systems of equations solvers to be shared, it avoids unnecessary duplication of effort. Finally, the integrated environments provided by PSEs help manage both the temporal and spatial coherence that leads to efficient computations. This applies to the user�s mental model as well as the data locality characteristics of the computation: Computational workbenches help people manage the complexity that the scientific computing process introduces.

2. Common Characteristics of PSEs

As originally defined by John Rice,

> A PSE is a computer system that provides all the computational facilities needed to solve a target class of problems. These features include advanced solution methods, automatic and semiautomatic selection of solution methods, and ways to easily incorporate novel solution methods. Moreover, PSEs use the language of the target class of problems, so users can run them without specialized knowledge of the underlying computer hardware or software. By exploiting modern technologies such as interactive color graphics, powerful processors, and networks of specialized services, PSEs can track extended problem solving tasks and allow users to review them easily. Overall, they create a framework that is all things to all people: they solve simple or complex problems, support rapid prototyping or detailed analysis, and can be used in introductory education or at the frontiers of science.

Some synonyms are *computational workbenches*, *component frameworks*, and *webs of science solvers*. In this paper we use these terms interchangeably, but tend to emphasize the computational workbench aspects for issues dealing with human interaction. These systems share certain common characteristics:

1. A target class of science or engineering design problems.
2. Their use appears natural to people in the target application area, both in user interaction and ways of thinking.
3. Ease of use, reliability of results and low execution costs are all-important objectives.
4. Execution times and data sets are often enormous � this is often a distinguishing characteristic for scientific and engineering computing.
5. Large complicated generic science components are used. Such components may encapsulate, for example, complete PDE solvers, or computer algebra systems.
6. Large, opaque legacy components are used. An example is the use of a component that provides a sparse linear solver, which the user need not examine in detail in order to use.
7. Because few algorithms are robust or accurate for the entire range of applicable problems, multiple solution paths or algorithms are provided.
8. Algorithms are parameterized to account for problem features, solution requirements and resource availability.
9. Currently, considerable effort goes into building a typical PSE. This typically means that the effort must be justified by building features that lead to an extended software life span. This brings expectations of extensibility, architecture independence and portability.
10. The computational workbench must handle a wide range of user expertise, from undergraduate students to expert researchers.
11. The computational workbench provides multiple levels of abstraction, separating user from layers of details, and hides details on a given layer from the higher levels.

Although not every PSE shares all of these characteristics, those used in science and engineering applications will exhibit most of them.

3. Some Current Systems

The need for PSEs in scientific and engineering computations is in part evidenced by their proliferation. One of the goals of the SIDEKIC workshop was to discuss existing systems to find what commonalties they have, and what parts of those systems can either be shared, or generalized to a service that can be developed independently and then shared. The PSEs discussed below are not an exhaustive listing and were selected primarily to provide a sampling of the wide range of applications and capabilities modern computational workbenches handle. They have been organized into large categories: workbench frameworks, component composition frameworks, code composition and generation frameworks, and collaboration frameworks. However, most of these systems fit into several or all of these categories and the classification is somewhat arbitrary. The Problem Solving Environments web sites http://www-cgi.cs.purdue.edu/cgi-bin/acc/pses.cgi and http://www.cs.vt.edu/~pse/ provide more PSE examples and related information.

3.1 Workbench frameworks

These provide the user with a sense of an arena in which the work is carried out. That arena often takes the form of a computer window in which user enters commands or creates the application. Such systems include:

- *Matlab* (http://www.mathworks.com), originally a high-level interface to numerical linear algebra computations which has been extended via "toolboxes" to provide PSEs for applications including signal processing, process modeling, and image processing. Matlab has also been extended in a natural way to handle parallel computation through exploitation of its programming language�s operator overloading feature.

- *Maple* (http://www.maplesoft.com) and *Mathematica* (http://www.mathematica.com), computer algebra systems that provide symbolic computing capabilities, formatted output, some code generation capabilities (in C, C++, or Fortran) and graphics and visualization systems.

- *PELLPACK* (http://www.cs.purdue.edu/research/cse/pellpack), which targets PDE-based applications on high-performance parallel machines. Subsystems include finite element methods, foreign system solvers, parallel execution, and a graphical user interface for problem specification and solution.

- *PETSc* (http://www.mcs.anl.gov/petsc/petsc.html), a suite of data structures and routines for the scalable parallel solution of scientific application problems modeled by partial differential equations, which is being extended to wider categories of applications.

- *NetSolve* (http://www.cs.utk.edu/netsolve), allows users to access both hardware and software computational resources distributed across a network. NetSolve searches for computational resources on a network, chooses the best one available and returns the computational solution to the user. Current research is making NetSolve interoperable with a similar object-oriented system called Ninf (http://ninf.etl.go.jp).

- *The AirShed Modeler* (http://www.eng.uci.edu/mae/Faculty/dabdub/modeling.html), that provides a workbench for simulations with different air quality models and algorithms.

- *Soliton Explorer* (http://penguin.mcs.drexel.edu/Soliton/), an environment for exploring soliton geometry.

- *Cumulvs* (http://www.epm.ornl.gov/cs/cumulvs.html), a software infrastructure for the development of collaborative environments which supports interactive visualization and remote computational steering of distributed applications by multiple collaborators.

- *Webpellpack* http://webpellpack.cs.purdue.edu) allows a user to define and run PELLPACK based computations on a PC cluster located at Purdue University from any brouser. The remote user has virtual access to the PELLPACK GUI and can upload/download the results of the computations. The user can run the parallel PELLPACK solvers with limited overhead or knowledge of parallel computing.

3.2 Component composition systems

These are characterized by providing a more generalized framework in which users can wire together components to create a complete application. The components themselves are often binaries or executable objects, but can represent resources as diverse as database servers, visualization systems, or remote instruments. The components interact on a peer-to-peer basis rather than as clients and servers.

- *SCIRun* (http://www.cs.utah.edu/~sci/scirun), is a scientific programming environment that allows the interactive construction, debugging and steering of large-scale scientific computations. SCIRun can be used for interactively for example to: change 2D and 3D geometry models (meshes), control and change numerical simulation methods and parameters, and perform scalar and vector field visualization. SCIRun uses a visual programming dataflow system and is extensible to a variety of applications. SCIRun has been designed to work with third party modules written in Fortran, C, and C++.

- *Component Architecture Toolkit* (http://www.extreme.indiana.edu/cat/index.html), is a component-based software toolkit designed to facilitate the construction of high-performance scientific applications that can efficiently operate in heterogeneous distributed computing environments. CAT provides tools for the dynamic location and use of distributed hardware and software resources.

- *Workbench for Interactive Simulation of Ecosystems* (http://fedwww.gsfc.nasa.gov), is a modeling environment to design ecosystem models by coupling different energy simulation simulations, population dynamics engines, and databases.

3.3 Code composition/generation frameworks

These are distinguished by having as an end product code in a programming language, which is then compiled and executed in a standard fashion. They differ from simple integrated development environments in addressing parallel or distributed computing platforms and in conveying more of the high-level application specific information which can then be used to provide end code which is better targeted for the run-time platform. Examples include

- *POOMA/PAWS* (http://www.acl.lanl.gov/PoomaFramework), is a set of C++ class libraries for developing scientific computing code running on machines ranging up to supercomputers with hundreds of processors, and a compact easy-to-read interface.

- *ATHAPASCAN* (http://www-apache.imag.fr/software/ath1/), a high-level plug-in template library to express parallelism in a generic way.

- *Spatial Aggregation Language* (http://www.cis.ohio-state.edu/insight/sa.html), a framework for organizing computations around image-like, analogue representations of physical processes in data interpretation and control tasks.

- *SciNapse* (http://www.scicomp.com), an environment for rapidly developing and prototyping programs for solving partial differential equations by providing a high-level symbolic representation interface It can also generate C or Fortran code implementing the solution strategy.

- *Falcon* (http://www.csrd.uiuc.edu/falcon/falcon.html), an environment for the rapid prototyping, development, support, and use of high-performance numerical programs and libraries for scientific computation. Users develop programs in MATLAB and Falcon automatically generates parallel Fortran or C++ code.

3.4 Collaboration frameworks

These concentrate on providing frameworks that allow multiple users at widely separated sites to work together on a single problem. The form of collaboration can range from intelligent sharing of experimenter�s electronic workbooks to actual real-time composition or steering of an application.

- *TechTalk* (http://penguin.mcs.drexel.edu/~techtalk/), is a framework supporting shared Matlab, Maple and chat sessions between users distributed on the network.
- *Shastra* (http://www.ticam.utexas.edu/CCV/projects/shastra), is a computer supported cooperative work system for geometric modeling, simulation, interrogative visualization and design prototyping environments
- *Intelligent Archive* (http://www.llnl.gov/ia/ia.html), environment integrates custom software with commercial and public-domain software such as database systems and World Wide Web technologies to provide access to disparate types of information, including computed and experimental data, papers, reports, and notes.

In addition to these existing systems, several are under active development for applications ranging from the DOE�s ASCI project to dynamic database formatting systems. As can be seen from the brief descriptions these systems share several characteristics. More importantly, there is a tremendous amount of conceptual overlap in their subsystems: many of them provide visualization modules, or access to libraries of standard mathematical services such as eigenvalue solvers, etc. In spite of this, little code is reused among PSEs because of the lack of interface standards between subsystems, and financial disincentives to provide reusable software in research computing. One workshop recommendation in Section 6 deals with this.

4. Design Principles for PSEs

Given the wide range of application targets, system goals, and user base that existing systems address, it is surprising that there is a set of design principles to which many PSE developers and users agree. These can be grouped into three general categories: human-centered design layered, component-based architecture, and ways of interacting with science and engineering resources.

There was unanimous agreement on the need to maintain the user as the center of design for PSEs. When addressing the human-centered issues raised by development of PSEs, the metaphor of a computational workbench proves to be a helpful tool with which to think about problem solving environments and how they should be designed. For example, a laboratory notebook analog is one of the standard tools on a researcher's workbench so a notebook is a critical generic component for any computational workbench. It is a component which, if designed properly, can be reused in a number of different computational environments. Although there may be a few cases where the metaphor doesn't quite fit, there seem to be few instances where it is necessary to violate the computational workbench metaphor in thinking about the design and development of computational support for scientific problem solving.

Design principle 1: A human-centered view should drive decisions about the architecture of computational workbenches or problem solving environments. Illustrations of this fundamental design principle are detailed in additional design principles and in the discussion of future directions for PSE work. Some of the design principles described below are corollaries of this first principle rather than truly independent principles.

Design principle 2: The problem domain interaction style and interface of a computational workbench should be structured or configured around how working scientists (the end-users) think about their scientific problems and not around the underlying computational architectures required to support the computational workbench. One useful model for thinking about this principle is that of talking to a colleague and explaining what needs to be done and/or which tools are needed and how they are to be applied. The computational workbench or PSE should provide for and allow the composition of components into a functioning system. Thus the notations, tools, and components should mirror human thought patterns and ways of working on projects. This design principle does not necessarily mean that software tools must imitate "manual" practices, but it does mean that human cognitive characteristics and ergonomic factors should always be taken into account.

Design principle 3: Problem statements should be natural to the application and not limited to idealized or standardized mathematics/physics definitions. Effectively, the problem definitions required by the PSE should include or make it possible for the end-user to specify both the symbolic representation of the problem and the desired performance features of a solution. The PSE user interface might even include images of laboratory equipment or devices that can be directly manipulated by the user to define the computation. For example, it should be possible for the user to specify a cost-evaluation function to be used for resource tradeoffs and optimization. That is, a user should be able to request a quick first approximation to a solution or to specify that a very precise answer is more important than the time it takes to get that answer. Consequently, wherever possible, the user should be allowed to make declarative problem specifications, i.e., users should only have to say *what* they want done but not *how* it needs to be done.

Design principle 4: Use participatory design in developing the architecture of the entire PSE, especially those features allowing customization or tailorability. Engaging the scientific researchers (the end-users) as collaborators in the development of their own computational workbench and tool set is essential in facilitating interaction design choices and in preventing major, hard to fix design decisions which are incompatible with the researchers� needs. Also, engaging end-users as collaborators in the development of their own computational workbench and tool set is a proven and successful way of promoting the development of ideas for new tools and new applications of existing tools.

Design principle 5: Make tailoring and customization easy and optional. Given that it is important to use configuration(s) of components that are natural to people/problems, it is important to recognize that some configurations are more natural than others are and should be favored in development work. Furthermore, a computational workbench should incorporate some sort of programming language or other type of glue as a fallback when nothing that suits a present need has been built or included in the current composition or palette of tools, or when new components need to be incorporated in novel or innovative ways.

Design principle 6: It should be possible for the scientist to treat systems, tools, components and even assemblies or combinations of components as a black box if desired. On the other hand, the researcher should also be able to open up and modify various tools, components, or environments as desired. We anticipate that the primary desire will be to simply use the mature, proven technologies and, as trust of the technology grows over time, we anticipate less desire to examine the contents of a black box. Thus the user should retain the control privilege of being able to give more detailed procedural descriptions when desired.

Design principle 7: Recommender systems are tools which provide advice for navigating among several choices (problem formulation, algorithm selection, resource choices, etc.) Recommnder systems should be context sensitive in providing people with advice that is customized to their background, expertise, and current goals. It should be easy to turn on and off the advice system. It should be possible to tailor a recommender to be automatic or suppressed when dealing with what a particular user considers routine tasks. Similarly, the advice giving should be sensitive to "receiving" advice from the user at various levels ranging from general guidelines to specific decisions. In addition, the computational workbench needs to make domain-specific advice available at a variety of levels. Preferably the advice should be declarative, since it is less of a burden on the user to specify what needs to be done without saying exactly how to do it. However, a wide variety of advice should be possible, ranging from declarations in the problem definition through keyword hints about techniques to use, to the use of programming or detailed manual configuration of components.

Design principle 8: Computational workbenches or PSEs should support the tracking and recording of problem requests, of partial experiment designs, of partial data, and of intermediate and final results. Since the problem of which things should be included in the tracking and recording of information is in part a domain-dependent, problem-specific issue, the user should have tailorable control over the process or the configuration of what is being tracked, recorded or preserved for future use. At a minimum, the tracking and recording done by the PSE should include check pointing and version control.

Design principle 9: There are different categories of users involved with computational workbenches and it is desirable to have different sets of features for different categories of users. Computational workbenches have the following three classes of users: the end-user/scientist/engineer, the integrator/vertical customizer/domain expert, and the developer of PSE infrastructure and base components. People may have different roles at different times or in different contexts, and it is important to recognize is that their goals and needs can differ between roles. It is important to keep the easy stuff easy, even for developers, and recognizing these three different roles facilitates making design choices.

Design principle 10: Collaboration technologies should be an integral part of computational workbench design. For example, end-users should be able to create shared artifacts that they and their research collaborators can see, annotate, and communicate about, regardless of whether the research group are co-located. Their real-time collaboration on tasks should be possible quickly, transparently, and over diverse and distributed locations. Furthermore, their communication about a shared artifact may need to be either synchronous or asynchronous.

Design principle 11: Easy navigability and accessibility of the software architecture of the PSE helps makes the PSE more usable. The software architecture should be layered and component-based with clear separations. Layering refers to not just the usual design of software systems by separating hardware resources, communication protocols, computational libraries, and user APIs, but also to provide different levels of abstraction. Presenting separate layers of expression of the computation that range from the user down to the computing infrastructure is also necessary, so that developers and users can operate at different levels of expertise as needed. For example, a PDE solver may have a topmost symbolic interface for a user, a lower level that allows entering C or Fortran expressions defining the PDE, and a bottom layer that allows the user to directly tie together low-level numerical libraries. A naïve user need not navigate software levels of an inappropriate level.

Design principle 12: Provide users with multiple levels of information about the computation. A biologist may just need to be presented with summary results of a phylogenetic tree computation, while a developer of tree similarity algorithms may need to find details of memory usage, computation rates, and load balancing on a parallel machine for the same problem. The general goal is for a computational workbench to enable its user to manage the complexity of designing an application, but this management needs to be supplied simultaneously for users with a range of interests. At the same time the layers need to be clearly separated so that operating at one level is independent of knowledge of requirements at a lower level.

Design principle 13: Use component-based software to build PSEs. As much as possible, subsystems of a PSE need to be based on interchangeable, plug-and-play components that allow developers and users to readily swap out components like visualization, or linear solvers, etc. Such architecture allows a wider range of capabilities and updating of a PSE dynamically, without having to tear it down and rebuild it. An analogy is automobile repair � changing the air filter should not require disassembling the transmission. Component-based systems also allow experts in different fields to contribute, maintain, and improve algorithms and capabilities within a component without requiring similar expertise from the PSE builders. Furthermore, the PSE must not be restricted to one of anything: user interaction mechanism, formats, algorithms, visualization, etc. For example, the same computational workbench should be usable from a GUI, a scripting language, a programming language, or even another computational workbench. Scientific computation often involve large size of active data objects so the component design must be able to move components to the data as well as move data to the components; large data may even be immobile.

Design principle 14: Integrate the PSE with science and engineering instruments and devices. This is related to design principle 8 above, but extends to direct connections to data gathering instruments as needed � whether those instruments are physical (like remotely controlled telescopes) or virtual (such as a simulated telescope.) Modern simulations often integrate data generating tools (e.g., data mining systems, remote sensors, and lab equipment), with data presentation tools (e.g., visualization, production of papers, and electronic publication on the Web) to provide faster turnaround for the complete scientific exploration process.

Again, not every PSE will satisfy all of these design principles. The fundamental one, however, has universal agreement: keep the human explorer as the center of the system, whether that explorer is a high-school student or a senior research scientist.

5. Future Directions

Before outlining promising areas for future research and development work in computational workbenches, it is important to state the long-term goals, ones that are both desirable and technologically manageable. This leads to a non-exhaustive list of potential research directions and questions that should be addressed in the development of computational workbenches. Part of this list consists of the fundamental problems that must be solved in computer science infrastructure. Future work should target some specific criteria for success, and lines of research and development that can not meet these relatively liberal standards should probably be re-thought or abandoned.

5.1 Design goals

In keeping with the human-centric design principle, the goals are stated in terms of what the users need to have:

- It should be possible for researchers to design and/or analyze objects, systems, and experiments.
- It should be possible for researchers to model how things work.
- It should be possible for researchers to set up, execute and record "what-if?" experiments and studies.
- It should be possible to automate routine repetitive tasks and partially automate any task with user guidance.

- It should be possible to translate between notations naturally and automatically. A scientist working on a conceptually difficult problem may well need multiple notations and/or representations to think about different aspects of the problem. Although not obviously part of a physical workbench, the notations used to sketch out a computational workbench design before even starting to build something are a way to match the workbench to the end-user.

- The user should be offered a "palette" of tools and the ability to compose them. Typically a physical workbench has a nearby palette of tools which are available for movement onto and use with the workbench. The craftsperson should have ready to hand the complete set of tools needed to accomplish the work to be performed and can bring each to bear on the artifact being created as the need arises. Similarly, in scientific computing a well-designed environment has sets of tools and components that build the structures indicated by the users� notations. These toolsets include both a basic set designed to work together smoothly without gaps, and also some more specialized tools for efficiency. While the creation of good physical work benches has had the chance to evolve over time, the creation of future scientific PSEs in new domains requires careful design, domain analysis, and knowledge engineering to identify appropriate tools and new ways to facilitate the interaction between tools and between tools and users.

- These should be recommender systems that are sensitive to the overall criteria used by the user in formulating the problem and the current configuration of tool sets. The advice-giving systems should be able to provide problem-specific hints on both the computational environment and on the domain problem.

- A computational workbench should be able to communicate with other entities. These entities include other computational workbenches and desktop tools, components running on remote servers, remote control instruments and data-gatherers, and people traveling or working from home without the full workbench immediately available.

5.2 Research issues

The research questions that future work requires follow immediately from the above goals. These questions range from generic human-computer interaction issues to fundamental technological and computer systems research. Further progress in PSE technology requires significant research in all areas of computer science. As an experimental systems field, this research must be coupled with development for validation and testing, which significantly raises the cost.

Directions for Future PSE Development.

In thinking about the future research directions which need to be opened up to enhance the development of computational workbenches for scientists and engineers, there are several topics which need to be explored. Some derive from the design principles articulated above. Others derive from consideration of the current state-of-the-art in PSE development. Still others derive from general considerations related to the broader context created by current directions of computational capabilities. We do not pretend that the list described below is exhaustive.

Research issue 1: Currently a number of bottlenecks inhibit communication between humans and PSEs about goals and tasks. To remove these bottlenecks it is important to understand what they are and the appropriate realm of solutions, i.e., which functions are to be performed by human users and which by the computational workbench?

Research issue 2: As a part of the ongoing extension of computational workbenches, and in the interests of reusability of components all the way from shared code to extensible tools, better understanding is needed of problem notations and representations used by scientists and engineers. Which existing notations or problem representations work well in which domains? Which representations or notations are domain-specific? How can we best automate translation between notations?

Research issue 3: Compared with traditional modes of working, which useful communication techniques/notations are missing from computational workbenches (e.g. geometry, back of the envelope diagrams)? How can they be incorporated? Are there functionally equivalent alternatives that offer greater ease of use or more power as a way of thinking about the scientific problems being solved?

Research issue 4: Given the potential of the computer to record massive amounts of information, which information is useful to the problem solver in tracking and logging? What techniques are useful in finding out about or recapturing a particular state in the problem solving process? What information is useful to PSE developers and algorithm designers in order to assess and improve PSE performance and reliability? How can this information be collected as input to data mining systems? Can generic database systems be used here? If so, how?

Research issue 5: Recommender systems have the potential to provide advice or automatic selections for options in problem formulation, algorithms, and computer resources. Which knowledge is most useful for good recommendations in these areas? How is this knowledge collected for data mining or learning? Can ordinary production use of a PSE provide useful information for the knowledge base of a recommender system? Can generic recommender systems be constructed analogous to systems for visualization, symbolic mathematics, etc.? In what form should the advice be couched? Is it possible to anticipate the user�s needs for advice in ways that solve both the immediate goal of helping the user past a difficulty and that of facilitating the long-range goal of improving the user�s expertise with the system and the problem domain?

Research issue 6: Computational workbenches should support both individual and multi-user tracking, recording, etc. so which actions, activities and results need to be tracked or logged? Collaborative problem solving may involve people at globally distributed sites. How is an appropriate record of collaboration presented? In addition, while user control over the collaboration set up must potentially be available to multiple users? What are the mechanisms for maintaining a master control token? Collaboration technologies need to become an integral part of PSE design so that people can create shared intellectual artifacts, which can be viewed, annotated, and communicated quickly and transparently among diverse distributed locations. What mechanisms best enable this kind of scientific teamwork?

Research issue 7: One cost-mitigating approach is to leverage existing and commercial software whenever possible and reasonable. Interoperability with desktop tools enable scientific applications to increase scientific productivity and save development costs for auxiliary capabilities such as text editors, word processors, spread sheets, and graphics programs. How can high performance scientific computing best be made interoperable with desktop computing environments? Can these tools be tightly interpreted into PSEs?

Research issue 8: Because the commercial market does not fully support scientific and engineering computing *per se*, sharing software development among scientific research groups is another promising approach. Several subsystems of current and planned computational workbenches can be shared amongst groups. One example is symbolic computer algebra systems � a PSE for solving differential equations may need such a system for user input and basic preliminary manipulations - whose development cost can be shared with workbenches for performing chemical equation balancing, research into computer algebra, or numerical linear algebra. In part this code development sharing is a primary driver behind the move to component-based architectures. Preventing wheel reinvention and "not invented here" syndromes is essential to leverage expertise, particularly when scientific computing must compete economically with business and financial for technologically trained personnel. How can workbench developers best collaborate and share software creation?

Research issue 9: Shifting to a component-architecture basis for computational workbenches requires the development of high-level standards for scientific component interfaces and interoperability, so that a component can be easily plugged into multiple computational workbenches. It also implies the creation of an economic marketplace to support the design of scientific components and frameworks. Such an economy need not be completely monetarily based, but some reward mechanisms must be in place to encourage component developers to invest the additional 30-50% effort that software engineering research indicates is necessary to create reusable code. What form should interface standards take? How can the scientific computing world best encourage reusability in components?

Research issue 10: Distributed components in PSEs are required to spread the work load among compute, visualization, and remote instrument servers as well as to support collaborative problem solving. This entails its own set of difficulties. Computational workbenches need to provide quality of service functionality and fault tolerance for processors, network connections, and system information that allows objects to be moved around adaptively in the distributed environment. How should migratory behavior, with components seeking suitable hosts and hosts reinstantiating components that failed on other hosts, be achieved? Components that provide introspective and reflective capabilities need to be investigated to allow highly dynamic problem solving, with new algorithms and components hot-wired into an application in the midst of a session. Information about these dynamically introspective interfaces also need to be provided at some level to users and PSEs, so they can be notified about the availability of these new resources. More generally, practical user interfaces to distributed computing and networked resources must be provided for both developers and users. How can the advantages of distributed computing be added to computational workbenches? How can its shortcomings be mitigated?

Research issue 11: Error detection and handling are critical issues for scientific PSEs. There should be mechanisms that check or assist checking for errors in problem formulation, computational scheme used, lower level software execution, and hardware faults. Many of these mechanisms are already present in robust problem solving environments and often automatic compensation is possible. However, many errors require attention from the user. What mechanisms can provide high reliability for PSE computations? A typical validation method is to compare computed results with other supposedly identical results. The latter can come from other computations, analytic solutions or experiments. How can PSEs best facilitate the validation of answers? The error and exception handling is seriously compounded in distributed environments where one may have difficulty even identifying the software or hardware component that failed, and it needs to be made more robust even in standard single-platform PSEs. Part of error handling is checkpoint and restart capabilities, which are needed for long running problem sessions subject to system failures. While much research is needed to provide checkpointing for PSEs, their application-tailored nature provides opportunities not available in general scientific computing. For example, instead of saving gigabytes of data representing the entire state of a discretized PDE, it may be more efficient to store a few hundred bytes giving its symbolic representation and recreate the discretized system when restart is required. A computational workbench can potentially provide the level of application-specific knowledge needed to optimize error detection and handling. How should validation, error handling, exception handling, and restart/checkpointing capabilities be provided to computational workbenches?

Research issue 12: One area closely related to error and exception handling is performance information. Because computational workbenches shield the user from low levels of programming details, they also tend to hide sources of poor performance from the user�s scrutiny. A common example is with computer algebra systems: expanding and simplifying algebraic expressions sometimes causes an explosion in the size and complexity of the underlying data structures used for the computations. The user only knows that the package runs out of memory, or takes extremely long amounts of time � but has no clue of how to improve that situation. Mechanisms need to be developed to provide better performance information to users so they can ask the questions

- Is there a problem with execution?
- Where are the resource limits causing performance problems?
- When will the computation complete?
- Will the computation run faster if more resources are provided?

Research issue 13: One difficult aspect of complex computations is handling general geometric shapes. Unlike for numerical, symbolic or discrete computations, facilities are missing that reflect the common geometry operations that people use in problem solving. Examples here include "use this shape", "join these points or curves", "subdivide this curve into 5 smooth pieces", "make these 2 point corners to fit to these data", and let *x* be a point near the center of this domain". Mesh and grid generators are used to discretize geometry but this software is very complex and less than completely robust. Is it feasible to create a generic, natural geometry processing/manipulation language and system for geometry? If so, how is this done? If not, what are the best alternatives? Can we even make mesh/grid generators that are generic and robust?

Research issue 14. The final important research issue is the integration of modeling, simulation, and visualization components. While this is related to the "reusability" of components, there is a more fundamental issue � how to leverage the integrated architecture and nature of a PSE. How do we define the underlying data structures, scheduling issues, coherence issues, etc. to provide interoperability between components that address fundamentally different objectives? For very large-scale (ASCI type) problems this may require, for example, using the same mesh for both computation and visualization. How can we provide future optimization encompassing the entire computational science pipeline of components? Can we develop algorithms that optimize the integrated modeling, simulation, and visualization pipeline, instead of the current approach of locally optimizing each component separately?

In addition to these research issues, several other technological areas must be addressed before PSEs can be more widely used:

- Creation of architecture-independent PSE components
- More flexibility and adaptivity in PSEs
- Reuse of PSE frameworks for multiple application areas
- Scalability of PSEs, both upwards to advanced multiprocessor systems and downwards to off-the-shelf component PCs

- Integration with database and data mining systems
- Control of instruments as well as their access
- Security and functional guarantees for PSE systems
- Mixed symbolic-numeric-intelligent-geometric problem solving

Some of these challenges are common to computational science and engineering research, but addressing them for PSEs provides tremendous utility for multiple application in multiple areas �the potential rewards are too great to ignore or postpone the work.

5.3 Evaluation guided development of PSEs and tools

In the context of a research and development project there are many lines of research that can and should be pursued, developed, evaluated, and then either abandoned or pushed further. However, the ultimate reason for developing PSEs for science and engineering lies in their use by one or more target groups of users. Any computational workbench, PSE, or computational tool intended for "industrial strength" use ought to be able to demonstrate relative performance advantages rooted in the end-user�s needs. In other words, speed of computation is not the only performance metric that must be served. The dominant cost of most science and engineering computing projects is in software development.

At a generic and domain independent level, criteria for assessing the success of a particular computational workbench or tool seem quite straightforward. In addition to being stable and reliable and in addition to producing correct results at the desired level of precision, a successful PSE should be able to demonstrate examples of successful problem solution in some problem domain area close to the evaluator and/or user. The successful computational workbench should enable the user to do something new which could not have been done before or it should enable the user to complete old tasks much faster (3- 10 times) than before and with at least a 50-60% decrease in workload for the user.

6. Findings

Whether known as PSEs, computational workbenches, webs of science solvers, or component frameworks, there is an emerging class of high-level software systems that provide support for a wide range of scientific and engineering endeavor, and which present an opportunity to boost national productivity in areas affected by technology. Given their breadth of applicability and the diverse nature of groups developing them, it is surprising but encouraging that broad agreement holds on what needs to be done.

Computational workbenches help manage complexity, by abstracting the problem solving process to a level more natural to the application users. This entails an entire set of difficult issues of just how people interact with computer systems: what are the natural communication protocols and how can computer systems facilitate the collaborative mechanisms that users find natural? Unlike business and PC applications, scientists and engineers also need the capability to interact with their computational workbenches at multiple levels. Those levels need to be insulated from each other, but open to users who need to look "under the hood" for performance or other reasons.

The need now is for collaboration, software reuse and component sharing amongst computational workbench researchers and developers. The human capital available is too limited to allow a "not invented here" syndrome to continue in scientific software research. More support is needed for common and shared infrastructure support (such as visualization or symbolic computing components), and to support researchers who expend the additional effort to turn their work into reusable components. Collaboration between government labs, funding agencies, industry, and researchers needs to be encouraged to hasten the transfer of research PSE technology to the emerging component framework marketplace.

Computational workbenches are excellent tools to introduce students to scientific computing early, and provide opportunities for integrating research and education at universities. At the same time, students in science and engineering should be exposed to more tools and high quality software as early as possible. This implies more funding for introducing commercial quality software into the classroom.

7. Funding Recommendation

The issues of Section 5 present an enormous agenda of challenging research projects. Some issues are focused on PSEs (e.g., issues 1 and 4) and others have broad scope (e.g., issues 6 and 9). Some issues involve fairly mature methodology (e.g., issues 10 and 12) and others are in nearly virgin territory (e.g., issues 3 and 13). Rather than try to prioritize these, we observe two facts. First, substantial progress needs to be made for all these issues in order for PSEs to achieve their potential. Second, several of these issues could justifiably consume the entire budget of a typical NSF program for a decade or so.

What is the potential of PSE technology? It can revolutionize computational science and engineering by dramatically reducing software costs. Recall that software costs dominate the budgets of most computational science projects. This is true even for the Department of Energy's ASCI project where very visible $100 million machines are being purchased. The President's new information technology initiative (PITAC) emphasizes an attack on "the software problem". PSEs are a major part of the solution of the software problem for science and engineering. Further, much of the technology and infrastructure is generic and can be transferred to medicine, finance, management, manufacturing, etc.

Advances in PSE technology can be funded in two ways. First, groups with large, important projects develop a PSE for their application. The R&D cost of the PSE is folded into the project budget and largely invisible as it is primarily salaries. Second, groups can develop the infrastructure for PSEs and collaborate with application groups to create specific PSEs. The R&D cost here is quite visible as infrastructure expenditure. It is clear that the second way is more effective and economical, yet most of the PSE work is funded the first way. There is an unfortunate practice in the national science establishment (both government and industry) not to fund software infrastructure directly. Hundreds of millions, even billions, are spent on science infrastructure items like telescopes, ships, reactors, accelerators and wind tunnels. But relatively little is spent on software infrastructure; projects are expected to develop most of their software from scratch, and with money diverted from applications

The workshop's recommendation for funding is simply:

> *A major investment is required to support software infrastructure in general and, for the science and engineering communities in particular, to support PSE development*.

8. Bibliography

A comprehensive bibliography on PSEs is included in the report by E. Gallopoulos et al.; it contains 140 items up to 1991. This bibliography has been expanded by Gallopoulos to contain about 420 items up to

1999. It will appear in the book *Problem Solving Environments for Computational Science* (E. Houstis et al., eds.) to be published in 2000 by IEEE Press.

Figure 1. PSE related publications by years (left) and cumulative (right).

Appendix: List of Workshop Participants

Name	**Institution**	**Email**
Kamal Abdali	National Science Foundation	kabdali@nsf.gov
Peter Beckman	Los Alamos National Laboratory	beckman@lanl.gov
Randall Bramley	CACR, Caltech	bramley@cacr.caltech.edu
Bruce Char	Drexel University	bchar@mcs.drexel.edu
Richard Fateman	Berkely Laboratory	fateman@cs.berkeley.edu
Dennis Gannon	Indiana University	gannon@indiana.edu
Steve Hague	Numerical Algorithms Group	steve@nag.co.uk
Thomas Hewett	Drexel University	hewett@drexel.edu

Joe Hicklin	MathWorks	joe@mathworks.com
Benjamin Hinkle	Waterloo Maple Inc	bhinkle@maplesoft.com
Jan Hull	Los Alamos National Lab	jhull@lanl.gov
William Humphrey	Los Alamos National Lab	bfh@lanl.gov
Chris Johnson	University of Utah	crj@cs.utah.edu
Jeremy Johnson	Carnegie Mellon University	jjohnson@mcs.drexel.edu
Lennart Johnsson	University of Houston	johnsson@cs.uh.edu
Erich Kaltofen	North Carolina State Univ.	kaltofen@math.ncsu.edu
Elaine Kant	SciComp Inc.	kant@scicomp.com
Steve Karmesin	Los Alamos National Lab	karmesin@lanl.gov
Thomas Kitchens	Department of Energy	kitchens@er.doe.gov
Suresh Kothari	Iowa State University	kothari@cs.iastate.edu
Robert Lucas	Lawrence Berkeley Lab	rflucas@lbl.gov
David Padua	UIUC	padua@cs.uiuc.edu
Steve Parker	University of Utah	sparker@cs.utah.edu
James C. T. Pool	Caltech	jpool@cacr.caltech.edu
Naren Ramakrishnan	Virginia Tech	naren@cs.vt.edu

John Reynders	Los Alamos National Lab	reynders@lanl.gov
John Rice	Purdue University	jrr@cs.purdue.edu
Mary Anne Scott	Department of Energy	scott@er.doe.gov
Neil Soiffer	Wolfram Research	soiffer@wri.com
Alina Spectrov	Los Alamos National Lab	alina@lanl.gov
Martin Staley	Los Alamos National Lab	
Andrew Strelzoff	UC Santa Barbara	strelz@engineering.ucsb.edu
Kothari Suraj	Iowa State University	
Boleswaw Szymanski	Renesselaer Polytechnic	szymansk@rpi.edu
Marsha Valdez	Los Alamos Technical Associates	mvaldez@lata.com