# A PROCESS BASED APPLICATION DEVELOPMENT SYSTEM

*Teodor Rus*

April 1, 2004

## Contents

1. **Handling system software complexity**  
   1.1 Process composition versus code composition  
   1.2 Component based software development  
   1.3 Process model for application development  
   1.4 Models of Problem Solving Environments  
   1.5 Integrating research and teaching

2. **Steps toward the new problem solving methodology**

3. **Hands-on experiments**  
   3.1 Evolving hands-on experience approach  
   3.2 Reusing educational experience

---

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Process composition versus code composition</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Component based software development</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Process model for application development</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Models of Problem Solving Environments</td>
<td>5</td>
</tr>
<tr>
<td>1.5 Integrating research and teaching</td>
<td>6</td>
</tr>
<tr>
<td>2. Steps toward the new problem solving methodology</td>
<td>8</td>
</tr>
<tr>
<td>3.1 Evolving hands-on experience approach</td>
<td>10</td>
</tr>
<tr>
<td>3.2 Reusing educational experience</td>
<td>11</td>
</tr>
</tbody>
</table>
Abstract

This proposal seeks research and education funds to support the development of a process based approach for problem solving by integrating problem solving environments (PSEs) with computing education. The objectives of the project are to:

- Develop domain ontology-based software architecture description languages and interpreters that map these languages into processes performing the algorithms represented by their expressions;
- Use this methodology to conduct problem solving experiments in such diverse problem domains as Internet Agents, Partial Differential Equations, Geography, Hydrology, and Statistics;
- Integrate this problem solving methodology within the computing education process by developing and teaching hands-on-experience courses.

These objectives are achieved by a problem solving methodology in which computer users manipulate computing processes according to the logic of the problem to be solved. Processes manipulated by computer users are high-level abstractions representing solutions to subproblems of the given problem. This is accomplished by a domain-oriented component-based application development system (ADS) where computer users develop the architectures of the problem solving systems while the ADS maps these architectures into processes that perform the solution algorithms. No programming as usual is involved. The side-effect of this approach to software development is that computer users have the ability to control the complexity of their software systems.

Problem domains are defined by ontologies where each term is associated with one or more stand-alone software artifacts that implement it. Domain specialists develop ontologies and the artifacts that implement their vocabularies, and demonstrate the use of the ADS by teaching courses using hands-on high performance computing. System software specialists develop ontology-based tools that support reasoning about user developed software architectures. Computer users manipulate ontological terms representing specification mechanisms, tools, and components.

The two fundamental ideas of this proposal are: (1) a true separation of software system architecture from software system functionality and (2) the development of component-based software systems by composing processes performed by the component codes rather than composing codes. The separation of the software architecture from its functionality opens the door to the development of machine independent software. The use of the computation process as a unit of composition during software system development allows the use of natural language as a basis for programming language design, and thus, leads to expertise reusability rather than code reusability in the problem solving process. This approach supports the evolution of current software functionality with computer technology, and the evolution of software system development with particular problem domains.

Our research aims at using domain ontology as semantic characterization for network clustering and thus to advance grid computation. The teaching approach we pursue is a component of the problem solving methodology developed in our previous research. It creates a framework to advance PSEs discovery and understanding while promoting teaching and learning. PSEs created by students during their teaching/learning period become the generators of enhanced PSEs in the domain of activity in which they are later employed. The machine independent paradigm for software development based on expertise reusability and the embedding of PSEs into network computing guarantees further impact of the results we will produce.
1 Handling system software complexity

Computer usage in science, technology, and everyday life, has exploded during the last two decades. One side-effect of this explosion is an exponential growth of software system complexity. Consequently, the cost of developing, maintaining, and using software systems for problem solving has become exorbitant. Researchers from IBM [Hor] believe that in order to sustain the current approach of problem solving with a computer, given the growth rate of computer applications, fifteen years from now the entire US population would need to work on software installation and maintenance. Among the major causes of this situation seem to be the conventional approach of computer usage for problem solving and the heterogeneity of computer systems designed to solve problems. Current methodology for computer usage requires that the human logic of problem solving be encoded into a program in computer memory whose execution is described by the loop:

\[ \text{while}((\text{PC}).\text{Opcode} \neq \text{Halt}) \text{Execute(} \text{PC} \text{)}; \text{PC} := \text{Next} (\text{PC}) \]

Here PC is the program counter register holding the memory address of the current instruction of the program, Execute() performs the operation encoded in the current instruction, and Next() determines the address of the next instruction of the program. Computer systems set this loop as the foundation for their usage in problem solving.

1.1 Process composition versus code composition

To handle software system complexity research funding agencies have promoted the creation of the science of design [Fre04]; the computer industry [IBM] suggests the development of autonomous computers, whose behavior mimics the behavior of the human body; academic research [GHR94, HRGB] suggests the development of domain-oriented Problem Solving Environments, PSEs; the software industry [BSR03, Bax01, Big98, Nei80, SK97] suggests the automation of the software system design and implementation. We believe that in order to subdue the complexity of current software, funding agencies, education, research, and industry, all need to be involved in the development of a computer supported problem solving approach based on the human logic of the problem and an appropriate problem solving methodology. This approach must be supported by a teaching methodology that uses hands-on experience to develop the intuition of software systems that solve domain-oriented classes of problems. This can be achieved by a problem solving methodology where computer users manipulate computing processes, rather than machine representations of data and operations, according to the logic of the problem, rather than following the machine-logic of program execution [Rus03]. Processes manipulated by computer users should be high-level abstractions that represent solutions to subproblems of a given problem. This implies that computer users need to be provided with programming environments populated by specification mechanisms, tools, and stand-alone software components. Specification mechanisms allow computer users to model their problems within a formal framework; tools are used to generate correct software components from correct specifications; stand-alone software components are universal algorithms over domain specific classes of problems that operate on data and function structures generated by tools from specifications. For example, a PSE dedicated to solving systems of linear equations may consist of:

- **Specification mechanism:** \( \sum_{i=1}^{n} a_{ji}x_i = b_j, j = 1, 2, \ldots, m \) where \( a_{ij} \) and \( b_j \) are real numbers and \( x_i \) are unknowns.

- **Tools:**
  1. Random number generators that generate \( n, m, a_{ij} \) and \( b_j \) for experimentation
2. Input programs that map systems $\sum_{i=1}^{n} a_{ji} x_i = b_j, j = 1, 2, \ldots, m$ into internal files
3. LU decomposition tools that map $\sum_{i=1}^{n} a_{ji} x_i = b_j, j = 1, 2, \ldots, m$ into triangular forms

- **Components:** LU decomposition solvers, say $\text{LUsolver}$

Programming languages should be used to express the architecture of the software systems that solve classes of problems in terms of the stand-alone components available within the PSE. For example, a program in the above environment would be: \textbf{Solve} $\sum_{i=1}^{n} a_{ji} x_i = b_j, j = 1, 2, \ldots, m$ by LU decomposition. Interpreters would map these architectural expressions into processes solving the problems whose solution algorithm they represent. For example, such an interpreter would map the above expression into: $\text{LUsolver}(\text{LUdecomp}(\text{Input}(\sum_{i=1}^{n} a_{ji} x_i = b_j, j = 1, 2, \ldots, m))))$. Note, the code performed by the machine while the interpreter handles $\text{Input}$, $\text{LUdecomp}$, $\text{LUsolver}$ may reside in libraries that are accessible to the interpreter and may be written in any programming language implemented on the machine, because the processes performed by these codes are composed, not the code itself.

### 1.2 Component based software development

For the vision described above to happen, problem domains need to be identified, their ontologies need to be specified in terms of software artifacts that implement their vocabularies, and their solution algorithms need be expressed in terms of more primitive solution algorithms. That is, for a given class of problems, solution algorithms need to manipulate processes represented by the more primitive solution algorithms rather than the language expression representing these processes. This is similar to mathematical thinking where mathematicians manipulate the meaning of the symbols they use by the rules provided by the meaning of these symbols, independent of the form of these symbols. That is, computers should be provided with application development tools that manipulate processes represented by software artifacts rather than the code expressing these software artifacts. Such application development tools are similar to control language interpreters (such as a shell in a Unix environment) which manipulate processes that perform computations expressed by problem solving algorithms rather than manipulating programming language expressions of these computations. Hence, the objectives of this proposal are:

1. Create the infrastructure of a process model for computer application development, ADS.
2. Develop and disseminate models of PSEs to experiment with the ADS.
3. Use a hands-on experiment approach to computing education for the integration of research and teaching within the process model for computer application development.

The experience gained by World Wide Web Consortium (W3C) within the last few years with the Semantic Web [W3C:Semantic Web] is a good model to follow in order to achieve these goals [Joh04]. Our challenge in following W3C’s experience is the recognition that the current approach for software system design hides systems complexity by encapsulating in the system design both the system architecture (the structure) and system function (the process). Hence, like W3C we will build our approach for computer application development by separating the system architecture from the system function. Moreover, the agents performing Web Services for Semantic Web[Hen01] are, in the activity we propose, computation processes defined as tuples $<\text{Agent}, \text{Expression}, \text{Status}>$ where $\text{Agent}$ is the processor that performs, $\text{Expression}$ is the expression of the computation performed by the Agent, and $\text{Status}$ is the status of the computation process. However, while W3C is concerned with data manipulation on the Internet, we are concerned with process manipulation.
by software systems where these processes solve the problems for which they were designed. In addition, while the infrastructure of the Semantic Web is developed by creating a framework for representing information in the Web [W3C:Concepts], and the creation of a Web Ontology that supports reasoning about this information [DAML01], [OWL03], we are creating an infrastructure for process manipulation by separating the two components of a software system: the architecture and the function. Our objective is to develop a problem solving methodology with a computer that allows users to manipulate symbols representing system architectures, while system software (i.e., the tools populating the problem solving environment) manipulate processes representing the functions of the symbols manipulated by computer users. Hence, we develop problem domain ontologies taking as semantics of ontological terms stand-alone software artifacts, and implement tools to reason about software systems using these ontological terms employing the set theoretical model proposed by W3C [W3C:Semantic Web].

1.3 Process model for application development

The process model for application development is a new methodology for problem solving based on the separation of the architecture of a software system from the function performed by that system. System architecture is described by an Architecture Description Language, ADL, while the system function continues to be described by any higher or lower level programming language, as currently is the case. The ADL used by a computer user is problem-domain oriented and is built on top of a vocabulary that represents the problem domain ontology. Each ontological term used by the ADL is associated with one or more stand-alone software components implementing its function.

A problem solver uses a problem-domain oriented ADL to express the architecture of the system that solves her problem. An interpreter analyzes this architectural expression, locates the functions implementing various components of the architecture, creates the processes performed by these components, and composes these processes into the process represented by the architectural expression, thus solving the problem.

We use Armani [Mon99, Mon01] as the ADL basis for the development of problem domain oriented ADLs. However, any ADL provided with a human logic to reason about system properties would accomplish our goal. We will use our own problem solving environment provided by the TICS system, Technology for Implementing Computer Software, to design and implement problem-domain oriented ADLs and their interpreters. TICS is a framework for language design, implementation, and use. Hence, we will accomplish the objective of creating the process model for application development by the following sequence of steps:

1. Expand our current problem solving environment provided by the TICS system to support domain-oriented component-based PSE by implementing tools for problem domain ontology development and reasoning, that will be provided as teaching and learning tools.

2. Develop the domain-oriented software architecture description language SELScript provided with process control-flow based on Armani and TICS. SELScript will use the ontology and the vocabulary of a problem domain to express system architectures while the code performed by the architectural components will be stored in software libraries.

3. Develop the SELScript interpreters that will map SELScript architectural expressions into processes performing the algorithms encapsulated in these expressions using software libraries in order to generate the processes composing the process expressed by a SELScript expression.

4. Provide TICS, SELScript, and SELScript interpreters as research, teaching, and learning tools to be used in the process of integrating research and teaching.
1.4 Models of Problem Solving Environments

The second objective of this proposal is to establish an open-ended list of PSEs used as test-beds for experimentation with the process model for application development. For each PSE we will create the ontology of the problem-domain it represents, specify classes of universal algorithms used by professionals to solve domain-problems, and determine the curricula employed for student education in that domain. Since we are pursuing a domain-oriented component-based approach for application development it is expected that the ontology of the domain will provide a vocabulary of specification schemes, tools, and components. This vocabulary will be used as a foundation in terms of which the architecture of domain applications will be expressed. The collection of stand-alone software artifacts representing processes used to generate problem solutions will be associated as semantics with the vocabulary terms. The following are representative PSEs:

**Internet PSE Domain:** The task of creating an ontology of the Internet problem domain is facilitated by the ontology already created by W3C. However, since our goal is to create software systems that process information available on the Internet, rather than representing resources and documents, this ontology needs to be appropriately updated. Hence, our goal is to provide a framework for the Internet service development where each service is expressed in terms of already existent services. We will use XML as a universal language that facilitates communication between the component processes of a given process. For that we will use a filter composition mechanism which consists of the following:

1. Each process is seen and an input/output black-box \( \text{XML} \rightarrow \text{Process} \rightarrow \text{XML} \) whose specificity is determined by the diagram: \( \text{XML} \leftrightarrow \text{Input} \rightarrow \text{Process} \rightarrow \text{Output} \leftrightarrow \text{XML} \).

2. Sequential and parallel compositions of processes \( P_1 \) and \( P_2 \) are specified by \( \text{XML} \xrightarrow{\text{XSLT}} \text{XML} \) filters [Rus03].

Specifications will be the BNF rules that specify XML, XML Schema, XSLT, RDF, RDFS, WOL Lite, WOL DL, and WOL Full languages. Tools will be provided by mappers of BNF notations into finite automata, pattern-matching parsers, abstract syntax trees constructors, etc. Components, such as scanners, parsers, tree-traversals and evaluators, and ontological expression evaluators will be automatically generated. The architecture of Internet agents will be expressed in SELScript and will be mapped into processes performing agent functionality by SELScript interpreters.

**Geography PSE Domain:** Many geographical problems are computationally intensive. In many cases, however, such problems can be decomposed for efficient parallel processing. To facilitate the design and the implementation of geographic algorithms we will first create a geographic ontology. This will be a challenging task because there are no models to follow other than those provided by W3C. With the help of the geographic ontology we will develop generic spatial domain decomposition tools that will employ either a regular partitioning strategy (e.g. cells or swaths), or the use of an adaptive hierarchical strategy such as a quadtree [Sam85]. The effectiveness of these approaches has already been examined in past research [CAr99, WAr03]. We will generalize and adapt them for generic PSEs. Genetic algorithms and Grid computations are the major classes of computation methods used for geographic computations. The courses we suggest for student education are Parallel Programming, Bayesian Statistics, and Linear and Dynamic Optimizations.

**Applied Mathematics PSE Domain:** Partial Differential Equations, PDE, are used as problem models and solution algorithm for many classes of problems and represent a major research objective in applied mathematics. However, there is no PDF ontology that would facilitate the development of such applications. Therefore our first objective at this problem domain is the development of
a PDF ontology which will be of a very broad interest. We will initiate the appropriate research in this PSE domain developing the methodology for solving two-dimensional linearized elasticity systems for an isotropic, homogeneous material under the action of a body force and a surface traction on a part of the boundary (the natural boundary) and with a specified displacement on the remaining part of the boundary (the essential boundary). The numerical method used is the finite element method with linear elements. Extensions are straightforward for isotropic materials and general elastic materials, as well as for three-dimensional linearized elasticity systems. The courses suggested for student education for this PSE are Parallel Programming and Differential Equations.

**Hydraulic PSE Domain:** The Computational Group in the Iowa Institute for Hydraulic Research, IIHR, uses large scale parallel computing to simulate flows in rivers and estuaries and to find solutions to protect the life of endangered fish species (e.g., salmon). Though the research in this area is of a national interest there is no hydraulic domain ontology to facilitate communication among researchers in this area. Therefore creating the hydraulic domain ontology will have a broad national impact. The experiments to be carried out by this PSE are incremental applications whose parts can be simulated on local computers before running large systems at national supercomputing centers. The focus areas are advection-diffusion and Poisson equations solvers. The architecture of these solvers should be directed toward structured and unstructured meshes. The goal to achieve is the integration of the existing experiences into the process model for developing such applications. Courses we suggest are Parallel Programming for Engineers, and Modeling of Environmental Processes.

**Markov chain Monte Carlo PSE Domain:** Markov chain Monte Carlo (MCMC) methods enable the fitting of complex, high-dimensional Bayesian statistical models by simulating draws from the joint posterior distribution of the unknown model parameters. Because an MCMC sampler typically must be run for thousands of iterations, run-times for sequential algorithms may be prohibitive if each iteration is computationally expensive. Consequently, parallelization of MCMC algorithms is a topic of active research [Wil04]. Our goal is to develop first an ontology of Monte Carlo Domain that will be of interest beyond the applications we envision here. We will use this ontology and the tools generated by this research program to extend research that has already been reported in parallelizing MCMC for certain linear models to a broader class of models. The courses we suggest for student education for this PSE are Parallel Programming, Bayesian Statistics, Linear Algebra, and Stochastic Processes.

### 1.5 Integrating research and teaching

A teaching/learning approach that supports the process model for application development has been developed by the PI as the *hands-on-machine* approach. For this to succeed, students need unrestricted and unlimited access to a powerful computer along with software tools that can simulate and carry out the activities presented in the teaching process. Using this approach students create programming environments dedicated to their fields of interest at the beginning of the class. These environments are then improved and extended by hands on machine and problem. At the end of the teaching period these environments become major tools for solving target classes of problems in the domain of interest.

Among many other things, student contributions to education through the development and use of educational tools provide a beneficial feedback on the software technology itself. With the TICS project, the Department of Computer Science at The University of Iowa, has a long history of achievements in this area. The TICS research group consists mainly of students who used our parallel machines during their classes following the *hands-on-machine* approach and succeeded
in designing and implementing teaching tools that allowed us to successfully use this approach for various classes across several generations of students. Thus, the TICS research group has created a problem solving environment populated by specification rules, tools, and components that allow students to develop processes that solve their problems using subprocesses that represent solutions to subproblems provided by components available in TICS environment. This methodology was successfully applied in such courses as Parallel Programming, Genetic Algorithms, Compiler Construction, and System Software. Currently, we are expanding the area of such courses by adding Internet Processing Environments and System Architecture Specification Languages. That is, the research aspect of this project concerns the integration of our experience with the hands-on machine approach with research on software system design, implementation and use, thus contributing to the development of a new problem solving methodology with the computer and checking the viability of this methodology with the students themselves. The educational components we envision here will come from all PSEs suggested in this proposal.

Computing artifacts handled in this proposal are tuples \( \langle \text{syntax, semantics, ontology} \rangle \) where syntax is the architectural expression handled by the user, semantics are the universal resource identifiers, URIs, of the stand-alone software artifacts implementing the tuple, and ontology are the URIs of the terms in the ontologies where they belong. This representation allows us to see the computing objects we manipulate three-ways: architectural, as blue prints represented by syntax, machine operational, represented by semantics URIs, and human logical, represented by ontological URIs. The separation of architecture (structure) from the function (the code implementing the tuple) and from the logic (ontology of the universe of discourse) characterizes all other engineering disciplines where objects manipulated are similar representations, \( \langle \text{blue – print, sensorial, logic} \rangle \). Therefore, we expect that the computing artifacts we manipulate will advance knowledge on software engineering across all field of endeavor where computers are used as problem solving tools. Specifically, this provides a logic foundation of the component-based domain-oriented software development approach by blue-print composition at the architecture level, process composition at the implementation levels, and logical proof at the ontological level. Consequently, this approach allows us to control software complexity by reusing expertise encapsulated in such artifacts. In addition, blue-print composition supports the advancement of natural language as a machine-communication tool and opens many possibilities for reproducing software artifacts and thus teaching them by experimentation rather than by textbook. The experience we gained with the TICS system is a guarantee of success.

Application development by computer process composition is the natural way for computation distribution in computer networks. The process management system we discuss here has the potential to embed PSEs within computer networks by locating and sharing resources. PSE domain ontology can be used as the semantics for network clusters thus creating the mathematical infrastructure for reasoning about the computation activity performed by a computer network. Therefore this project has the potential to provided the theoretical foundation for grid computing and network design. In addition, this proposal presents a new methodology to foster integration of research and teaching. The benefits of this integration go beyond the academic institution promoting it. Following the research, teaching, and learning approach discussed in this proposal students develop their own problem solving environments during their learning process. These environments are then further ported into the students’ work-areas in the institutions where they are hired. Our previous experience shows that students’ PSEs grow into problem solving environments in the respective institutions. This ensures an evolutionary process of discovery and understanding fostered by the process of problem solving itself. Moreover, since the process is grass-roots oriented it provides an equal opportunity for advancement.
2 Steps toward the new problem solving methodology

The four step methodology followed by a human while solving a problem, as explained by Polya [Pol73], implies: (1) understand the problem, (2) make a plan to develop a solution, (3) carry out the plan, and (4) look back at the completed solution. This methodology is not much different when software systems are used as tools for problem solving [RR95]. It still requires a user to understand the problem. But since software systems are used as tools to automate problem solving, the issue now is the development of a methodology for the generation of such tools. Software tools are, however, among the most complex artifacts produced by humans so far and their complexity induces more complexity in the methodology of their automatic generation. The current experience with software tool generation pertains mainly to the generation of software tools for program generation and execution not for problem manipulation (i.e, application development). Therefore we are still assisting to an expandable gap between applications and tools supporting computer applications. Hence, since there is little experience with tools for computer application generation, the intuition of a human-oriented methodology for problem solving with such tools needs to be developed. The work we currently perform may be seen as an experimental framework for the development of this intuition.

The term “problem solving environment”, PSE, was coined early in computer history, and is illustrated by very simple PSEs developed as libraries of functions associated with various software systems. Examples of such software systems are operating systems and compilers. This idea has emerged again and the new meaning of this term has become “a computer system that provides all necessary computational facilities to solve a target class of problems” [GHR94, HRGB]. The meaning of the PSEs illustrated by the libraries of functions mentioned above is no different. The target classes of problems in these cases are the management of program execution by operating systems and the management of program development by compilers. In other words, in building our experimental framework for the new problem solving methodology we observe that in the early PSEs as well as in their new instantiations, a PSE refers to a problem domain identified by: (1) a collection of specification mechanisms, (2) tools that operate on specification mechanisms mapping them into data and operations appropriate for components, and (3) components that solve classes of problems by universal algorithms expressed in terms of data and operations generated by tools. There is a dynamic relationship between specifications, tools, and components, where tools may be used as components and components may be used as tools. The net effect is a hierarchical methodology where problem solving algorithms are incrementally and interactively developed in terms of more primitive algorithms which are previously implemented. This is magnificently illustrated by current operating systems and compilers and is now being illustrated in other problem domains as well. Examples of such PSEs in computational science [GHR94, HRGB] are: PSEs for partial differential equations, PSEs for linear system analysis, PSEs for scientific computing and visualization, PSEs for interactive simulation of ecosystems, PSEs for network computing. All of these PSEs are collections of specifications, tools, and components. However, these PSEs also have drawbacks which make them more complex than necessary and thus difficult to learn, use, and reproduce. These drawbacks are:

1. These PSEs are based on conventional programming language constructs, not on the natural language understanding of the specifics of the problems they solve.

2. The final product is a program on a given platform that is composed of components previously developed, perhaps on different platforms. Therefore, the usage of these systems requires users to master the programming objects they use rather than the problem-domain they approach.

3. There is no clear methodology for teaching, learning, and reproducing these systems.
4. These PSEs appear as packages added to the conventional programming environment rather than providing a problem solving methodology in the domain of interest independent of the platform on which they operate.

Our idea is to embed such PSEs as vocabularies in the ontology of a system architectural language, such as Armani [Mon01], and to extend the language thus obtained with an interpreter that maps software architectures into processes that perform the functions the user desires to attach to such architectures. Consequently, while constructing a system from components, processes performed by the system components are composed, not the code that represents these processes. Since these processes are abstractions that represent problem domains, problem solving is based on domain-expertise not on platform peculiarities. This is illustrated by the control language of a time-sharing system and by the extension of the TICS system reported in [Rus03].

3 Hands-on experiments

Recent developments in computer technology have led to a society in which computing artifacts invade all aspects of human life. This phenomenon, characterized by researchers as ubiquitous computing [LY02], raises new teaching and learning challenges. Traditional textbook-centered teaching methodologies are inadequate for educating students about concepts behind the relentless development of new computer-based “gadgets” such as PDAs, GPS receivers, and cell phones. The time needed to develop a textbook to teach the design, implementation, and usage of new computing gadgets can be longer than the time needed to develop even newer gadgets, and therefore, a textbook may become obsolete before even being printed. This situation leads to the requirement of a new educational methodology based on the availability of computing tools that are able to represent and implement the concepts and methods presented in the teaching process. With such tools, students have the hands-on means to repeatedly simulate and experiment with phenomena presented in the classroom. This phenomenon has been the object of the PI’s observation during the last 20 years of teaching operating systems, compiler construction, system software, parallel programming, algorithm design, implementation, and use, and various computer applications, in the Computer Science Department at The University of Iowa. Our conclusion is that a major hurdle in teaching fast-changing computing technologies is the lack of tools that facilitate the transformation of repetition into intuition and of intuition into knowledge. This can be alleviated by the hands-on-experience approach to teaching, based on computer simulation, that consists of:

1. Unrestricted access to a computer with capabilities appropriate for the particular subject matter.

2. Software tools that can simulate various aspects of the teaching and learning process. In some cases, these tools can be generated by students during their own learning processes.

3. Hands-on instruction, in which the course material is illustrated using appropriate tools installed on the computer, combined with hands-on learning, in which students use the computer and installed software tools to create and experiment with new tools that implement course concepts.

4. Creation, by students, of personal portfolios (which are customized personal PSEs) based on their hands-on educational experiences. The software artifacts and tools created in hands-on courses have long-term value, in both later classes and ultimately, their professional careers.
By necessity, but perhaps not always consciously, many computer science instructors employ hands-on teaching methods. The PI first consciously used this approach in 1987 during the development of the course 22C:132, Parallel Programming, with the specific goal of developing student intuition for parallel computations. The relevant computer available at that time was an Encore parallel processor provided to us to experiment with parallel program development. The tools available for this task were a version of the UNIX operating system and the usual programming languages (e.g. Pascal, Fortran, and C) and their compilers. The most important was the C language because it provides the programmer with a process creation capability during program development. Thus, process manipulation by students becomes feasible, and allows students to develop the intuition of a process abstraction that represents a schedulable and composable unit of computation that interacts with other processes.

3.1 Evolving hands-on experience approach

In 1992, the Committee on Physical, Mathematical, and Engineering Sciences of the U.S. Federal Coordinating Council for Science, Engineering, and Technology, assessed the major problems faced by new developments in US science and technology, and published the report *Grand Challenges: High Performance Computing and Communications* [GC92]. This report identifies parallel computing as the major tool to solve grand challenge problems. This situation led to the national requirement to create “a culture” of parallel computing. Our experience with the parallel programming course allowed the Department of Computer Science to respond quickly to this requirement by establishing a consortium for student education in high performance computing. This consortium consisted of four Iowa education institutions: The University of Iowa and three outstanding private colleges, Cornell, Graceland, and Luther. The goal of this consortium was to enable education in parallel computing by the *hands-on-machine* approach at both graduate and undergraduate levels. Consequently, a proposal to acquire an appropriate computer was submitted to NSF by Prof. T. Rus (Computer Science), PI and CoPIs Prof. F. Potra (Computer Science), Dr. J. Brown (Weeg Computer Center), Prof. J. Jones (Graceland College), Prof. T. deLaubenfels (Cornell College), and Prof. W. Will (Luther College). This proposal was funded by the NSF grant DUE–9551183 ($215,369.60) with a match from The University of Iowa. Thus, in 1996, the by-then-obsolete Encore computer was replaced by an SGI Power Challenge equipped with 16 R10000 processors. To achieve our goal we developed new courses, reshaped older courses in this area, and today we can report that hundreds of Iowa students benefited from the use of this machine. In addition, we generated new research activities on high performance computing and enhanced our curriculum in both computer science and applied mathematics.

We are now facing a new grand challenge problem, which may be characterized as the *source of all grand challenge problems*. It is the challenge to develop a problem solving methodology that is able to simplify the use of current and future computer technologies to solve other grand challenge problems and to control the complexity of software system development, installation and use. This problem, once again requires us to unify forces on the University of Iowa campus by creating an Interdisciplinary Computing Laboratory to play the role of a test-bed for the process-oriented component-based problem solving methodology we are currently creating. At the basis of this Laboratory sits our experience with problem solving environments, as reported in the literature mentioned above, and the results we obtained with the consortium for the development of a critical mass of parallel computer users. The Interdisciplinary Computing Laboratory will be established by joining forces of major users of high performance computer technology on campus, namely:

1. Marc Armstrong, Department of Geography, developing mathematical models of geographic domains and using high performance computing to solve such problems.
2. George Constantinescu, Department of Civil and Environmental Engineering, using mathematical modeling for the development of tools to predict hydrological processes and large scale parallel computing to predict multi-scale environmental processes.

3. Mary Kathryn Cowles, Department of Statistics and Actuarial Sciences and Department of Biostatistics, Markov chain Monte Carlo algorithms for Bayesian modeling.

4. Weimin Han, Department of Mathematics, developing algorithms for large-scale numerical solution of differential equations with applications in engineering, physical and biological sciences.

Experiments developed by students attending the courses on parallel programming over the last 15 years cover all these areas and illustrate the new problem solving methodology we are developing with PSEs provided as final projects for the parallel programming class. Therefore, we expect that the funding of this project will expand further the results obtained so far, leading to both improved education in science, technology, and engineering through adaptation and implementation of the educational practices that we have developed before and to the concrete use of the new problem solving methodology in the research areas listed in Section 1. Our belief is based on the fact that the hands-on approach to education in high-performance computing gained in popularity in the University over the last 10 years at both graduate and undergraduate levels, thus the necessity of a new problem solving methodology became obvious.

3.2 Reusing educational experience

The SGI Power Challenge, purchased in 1996, is obsolete and can no longer be maintained. Meanwhile, high performance computing has expanded substantially, and greater challenges have been created for student education in this area. Therefore, replacement of the SGI Power Challenge is important to the continued health and further enrichment of the high performance computation curriculum in two key ways:

1. It will directly support existing and planned courses on high performance computing education. We will expand our courses on parallel programming and parallel algorithms with new application-oriented courses in all areas of research mentioned in Section 1.

2. It will facilitate building on our educational tool development experience to improve software engineering education through component based software development methodologies.

To achieve these goals we need to acquire a new parallel machine as a tool to facilitate expanding the hands-on-machine approach to a larger educational context. This context is provided by new developments in high performance computing that are mainly generated by interdisciplinary research. This machine will be used by the Interdisciplinary Computing Laboratory as the main computing facility to experiment with the new problem solving methodology we develop. It will also be offered to all instructors who would like to teach their students high performance computing using the hands-on approach. Based on our past experience, we expect an average enrollment of 30 students per class taught by one instructor, in each of these classes. Hence, approximately 300 students per year would benefit from the services offered by this machine and will experiment with the new problem solving methodology we create.

Adding the software engineering component to our goals is a natural consequence of our experience which shows that the biggest barrier to machine-independent thinking in software engineering is the deep intertwining in today’s software of two things of a different nature: system architecture
and system function. Due to new accomplishments in software development achieved by us and by others [All02, LS02, RKS+], the hands-on experience approach, which was successfully used for teaching parallel programming, now can be successfully used to teach process-oriented component-based application software development as well. This allows teachers and students to develop the intuition of the software system as an abstraction independent of the function it performs. Moreover, in the long term, this provides a foundation for the use of natural language as a man-machine communication tool.

References


[DAML01] DAML+OIL Reference, DAML+OIL Reference Description, Dan Connolly, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein (editors), W3C Note 18 December 2001.


Mapping the right web application development process flow is a key to success for a project of any size. Despite the fact that the development of the web-based app is pretty similar to the regular desktop application development it has some differences that may impact the overall process and make it more complex in some aspects. Existek is a custom software development company helping our customers to solve their business challenges with best in its class software. Contact us to get an instant and free expert consultation about your project. development processes for mobile applications used by software development companies, as well as by independent developers. The sure include short and frequent development cycles, frequent technological changes (platforms, operating systems, sensors, etc.), limited documentation, specific requirements and resources of the development team and the client, among others. But software development isn't one of them. Instead, as Benjamin Franklin so famously put it: if you fail to plan, you are planning to fail. Benjamin Franklin Every great piece..._– Benjamin Franklin_ Every great piece... However, in most practical applications the phases overlap slightly, with feedback and information being passed between them. Some people also like to call this a plan-driven process as in order to complete a project, you first need to know everything that needs to be done and in what order. Hence the name Waterfall. The V-shaped software development process is a take on the classic Waterfall method that makes up for its biggest downfall: A lack of testing. Fall in with Project Management. Try Planio. In software engineering, a software development process is the process of dividing software development work into distinct phases to improve design, product management, and project management. It is also known as a software development life cycle (SDLC). The methodology may include the pre-definition of specific deliverables and artifacts that are created and completed by a project team to develop or maintain an application. Find out here what the application development lifecycle consists of and how to achieve success with the creation of advanced software, based on the best practices shared by professional developers. SDLC, or software development life cycle, describes the process of software creation. The stages are all the same as for the application development life cycle (ADLC), which explains how any professional IT company makes apps: discovery, design, development, release and maintenance. Now let's take a closer look. Six stages of the application development life cycle. Here are the phases that no digital project can go without: Discovery phase.