# Using Components for Rapid Distributed Software Development

**Alexander Repenning, Andri Ioannidou, and Michele Payton,**
*University of Colorado, Boulder*

**Wenming Ye and Jeremy Roschelle,** *SRI International*

A large, geographically distributed testbed consisting of domain experts, component framework coordinators, developers, publishers, and users produces educational applications using a rapid production pipeline process.

Software development has not reached the maturity of other engineering disciplines; it is still challenging to produce software that works reliably, is easy to use and maintain, and arrives within budget and on time.[1] In addition, relatively small software systems for highly specific applications are in increasing demand. This need requires a significantly different approach to software development from that used by their large, monolithic, general-purpose software counterparts such as Microsoft Word. These microapplications (μApps) require very fast cycle time; that is, relatively small teams of domain experts and developers must build them quickly and iteratively. The organization might allocate these people on the fly, and they are likely to be dispersed geographically.

One software development solution that has a long tradition of advocates,[2] is recommended by leading experts,[3] and is quickly gaining support is component-based development. Components (for example, platform-independent JavaBeans and Windows-only ActiveX controls) are highly reusable units of software functionality[4]; they let developers conceptualize software as interconnectable building blocks.[5] These software components support modular engineering practices, just as integrated circuits support modular design of hardware.[6] At least in theory, building large projects out of well-defined and well-behaved building blocks can reduce the complexity of software development, because building on stable substrates is faster than building from scratch.[7] The same organization assembling the components might produce and maintain them in complete applications, or acquire them from third-party developers producing so-called commercial-off-the-shelf software packages.[8]

## Distributed software development

The component-based approach to software development is generally attractive and has exceptional appeal in distributed software development. One downfall of traditional distributed software development approaches is that software projects are often insufficiently decomposed. This results in overlapping or misunderstood responsi-


0740-7459/01/$10.00 © 2001 IEEE

**Table 1**

**The Stakeholders, Their Roles, Responsibilities, and Products in µApp Development**

| Stakeholders | Roles | Responsibilities | Products |
|---|---|---|---|
| San Diego State University and Queens University, Kingston, Ontario, Canada | Domain experts | Design math activities | HTML activity mockups including explanatory text and pictures |
| SRI International, Menlo Park, CA | Integration team workshop organizers | Organize workshop bringing together all the stakeholders to analyze and design activities | Set of paper-based low-fidelity activity mockups<br>Implementation schedule for all activities |
| | Component framework coordinator | Design and evolve component framework<br>Provide component authoring tools<br>Organize component repository, including components and use stories<br>Accumulate and delegate change requests to component authors | Component repository<br>Design guidelines<br>Design and implementation services |
| University of Colorado, Boulder | Developer | Develop simulation components<br>Combine library, hand-coded and generated components into complete µApp | Simulation component prototypes<br>Component generator tools<br>µApp |
| The MathForum, Swarthmore, PA | Publisher | Critique µApp design in terms of pedagogy<br>Test µApp for cross-platform compatibility, performance, and clarity of documentation<br>Publish and support µApp users through mentoring service | Final µApp<br>Support materials for users |
| | Producer | Hold teams accountable for their responsibilities, including content, adherence to guidelines, and scheduling | Schedules<br>Reminders<br>Feedback |
| Math teachers, students | Users | Participate in analysis and design<br>Provide feedback to developer<br>Use µApp | Ideas leading to µApps<br>Feedback |

bilities, which can lead to significant communication breakdowns and complete project failure. The nature of components forces designers and developers to better encapsulate functionality into cohesive, reasonably well-documented chunks of software.

This article reports on the experience of a large testbed called Educational Software Components of Tomorrow (www.escot.org), supported by the US National Science Foundation. Escot is building a digital library containing educational software focused on middle-school mathematics.[9] The ESCOT goals include building interactive JavaBean-based content for educational purposes and exploring the distributed software development process with the specific objective of building and deploying reliable software rapidly. A large pool of geographically distributed stakeholders in the US and Canada participate in the project.

## The CORD process

The Component-Oriented Rapid Development process happens in parallel prediction cycles involving different subsets of stakeholders aligned in a pipeline fashion to output completed software at a weekly rate. It resembles extreme programming.[10] XP replaces the four coarse sequential steps of the waterfall model found in most object-oriented software engineering approaches—including object-oriented design, object-oriented system analysis, the object modeling technique, hierarchical object-oriented design, object-oriented structured design, and responsibility-driven design—with an extremely large number of parallel steps, including analysis, design, implementation, and testing.[11] CORD, like XP, employs parallelism, but the granularity of its parallelism relates not only to the process but also to the software components used and the number of distributed teams involved in the development process. The CORD approach involves a number of parallel threads representing distributed teams working on sets of components. The components' relatively small size and CORD's distributed nature suggest development activities that are relatively small in scope, highly parallel, and highly iterative.

CORD differs from XP in several crucial ways. In CORD,

- The project start includes centralized analysis and design. A large group of users, domain experts, designers, and developers analyzes project requirements and creates application mockups and interoperability specifications.

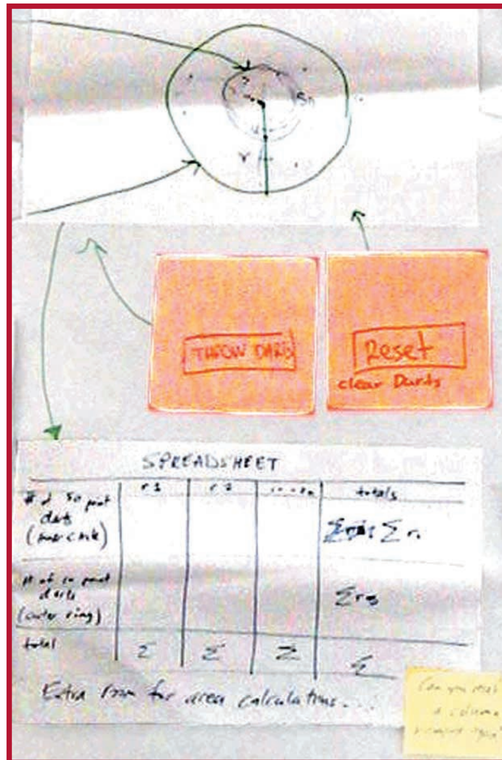



**Figure 1. A small part of the design mockup showing simulation (top), buttons (middle), and spreadsheet (bottom) components. If you randomly throw darts at two concentric circles, what is the chance of hitting the smaller circle?**

These mockups serve as design blueprints for the project's further development. The interoperability specifications attempt to anticipate connectivity issues that will arise during later component-based development and create a robust framework to handle them.

- Development is distributed. After the initial centralized analysis and design, the group distributes development to independent teams, coordinated through regular builds. Each build assembles all or some components into a testable application or applet.
- Development is component-centered. This approach is well suited for distributed teams using heterogeneous sets of tools and platforms. In the ESCOT project, we find teams producing components using component generators, retrofitting off-the-shelf components, or programming them from scratch. Some teams use low-end programming environments such as Sun's Java JDK, and others use more sophisticated integrated development environments such as CodeWarrior.
- Development and delivery are cross-platform. Individual developers and users can use the platform of their choice.

This article describes the CORD process in the context of ESCOT, in which teams collaboratively produce µApps that they publish on the Web. Middle-school students explore them interactively, solve mathematical puzzles, and submit answers to a mentoring service. The Web site posts a new problem each week, and the problems relate to the same theme for a month, with each week's problems becoming progressively more difficult. The particular µApp we describe focuses on the geometry of circles. The purpose of this µApp was to develop a sense of spatial relationships and probability and explain the derivation of the mathematical constant π using experimental possibilities.

## Stakeholders: Collaborators and roles

In the CORD process, several stakeholders from geographically dispersed locations work together. These integration teams, which are formed based on the requirements for the µApp's core components, design and build math µApps.

Creating these µApps requires more knowledge than any single person possesses. For instance, developers are technologically savvy but might not sufficiently understand the requirements of end users. End users, on the other hand, are experts in their application domains but might not adequately comprehend technological limitations or opportunities. We can accommodate this "symmetry of ignorance" by combining collaborative design with distributed development in the integration teams.[12] Table 1 lists the integration team members and the roles they play in the creation of the component-based distributed π µApp.

## Phase 1: Centralized analysis and design

In phase 1 of the CORD process, a large subset of the stakeholders meet to hash out analysis and design issues.

***Brainstorming through low-fidelity design media.*** Distributed software development often disintegrates when the process is insufficiently decomposed, the stakeholders' roles are overlapping or poorly defined, or communication breaks down. In the CORD process, even if we distribute the software development process, we gather all the participants together, at least in the initial analysis and design phase. Despite scheduling issues, gathering the entire group is important. When group ideas are first emerging, face-to-face

interaction is essential, enabling us to resolve issues such as role definition and task distribution early in the process. Another important part of the process in the initial meeting is designing activities with low-fidelity media such as paper and Post-It notes, which are accessible to everyone and let all the stakeholders participate in the design process.[13] The result of this step is a storyboard, a list of components, and a sense of interaction between the components. The component list feeds a stepwise refinement procedure gradually leading from the identification of the necessary components to their implementation and integration.

In a five-day integration team workshop held at Swarthmore College, Pennsylvania, in August 1999, a group of domain experts, component framework coordinators, developers, publishers, and users brainstormed ideas for µApps, one of which was the π µApp. We analyzed a suite of about 20 such ideas and created mockups for each one. The original mockup of the π µApp consisted of poster-size paper sheets with Post-It notes representing components (see Figure 1). The design became clearer when participants returned to their home organization. It was, therefore, absolutely essential to capture design representation (the poster boards) and additional discussions through a dedicated scribe.

***Formalizing design with HTML mockups.*** Transforming the initial design from the low-fidelity mockup to a more formal medium is an important next step. We turn rough sketches of text and images into more explicit representation, forcing designers to fill in conceptual gaps. A geographically dispersed group of stakeholders can easily share HTML documents to trigger design discussions. These documents can also be good starting points for soliciting feedback—not only from the internal group, but also from actual users.

The publishers and the component framework coordinator scheduled the development of this µApp to begin in February 2000. The domain expert had expertise in designing curricula for math education. She created a preliminary design (see Figure 2) based on the paper mock-up and posted it on the Web as a blueprint for the final application. The Web page prompted a lot of feedback from developers.

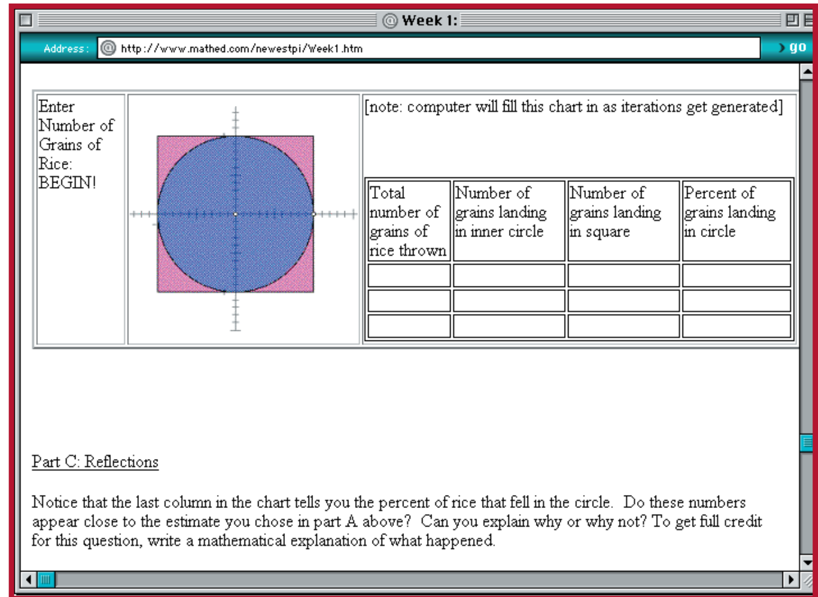



**Figure 2. Other members of the design team can easily access and critique the HTML mockup of a part of the µApp.**

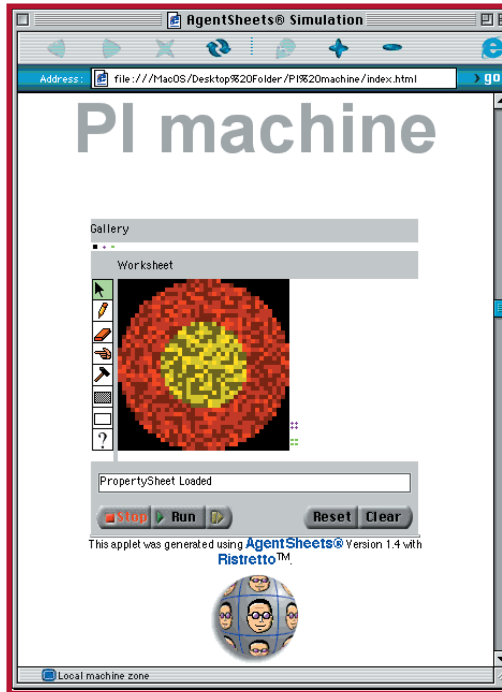
The simulation part of the activity (the blue circle in the red box) consisted of throwing darts at a target and counting the darts hitting the blue versus the red part as a means to approximate the value of π. This clarified what kind of information would go into the tables, but the static Web pages shed little light on the way in which the simulation would work.

***Specifying interoperability design patterns.*** The component framework coordinator analyzes the set of low-fidelity requirements, identifying interoperability requirements that all component-based µApps will need to solve. In our specific case, important common requirements include

- synchronizing data values across components,
- dynamic publishing and subscribing to data,
- writing the state of an assemblage of components to persistent storage (for example, XML files),
- overcoming component mismatches, and
- submitting user responses to the educational activity (for example, answers to challenges) to a server.

To address these requirements, the component framework coordinator extends standards already well supported by the individual component vendors. In our case, ESCOT extended a de facto standard, the design patterns underlying JavaBeans, with additional design patterns and conventions. Furthermore, ESCOT provided utility classes that im-

**Figure 3. We prototyped the central component of each project, called the anchor tenant component, and makes available to other teams as Java applets.**




plement these patterns, off-loading common interoperability tasks to a centralized software development while leaving the details of µApp design and implementation to the integration teams.

The developers use these patterns to instrument component generator tools. In spirit, these tools are similar to the rapid application development systems, such as JBuilder, commonplace in integrated development environments. However, unlike RAD systems, which focus on general-purpose GUI assembly, component generator tools focus on a narrow vertical application area. ESCOT uses two component generators, AgentSheets and the Geometer's Sketchpad. AgentSheets (described in more detail later) enables authors to design multiagent simulations quickly. The Geometer's Sketchpad lets them design animated sketches that obey Euclidean geometric constraints. Once an author creates a specific design, either tool can output a new JavaBean that conforms to the interoperability design patterns. This facilitates just-in-time production of new components that are highly targeted to a particular domain problem.

In addition, the component framework coordinator maintains a set of end-user (scripting) programming languages, which fill the gaps among existing components in the repository. ESCOT allows pluggable scripting languages and presently supports JavaScript and Logo (a common educational programming language). By having scripting languages, we avoided the need to generate new components that only one specific µApp required, and we implemented the behavior quickly in a script without resorting to the slower cycle time required by our primary programming language, Java.

Thus, by using component generators and scripting languages, CORD circumvents the common component-based development problem in which existing components are "never quite right" for the task at hand.

### Phase 2: Distributed analysis, design, implementation, and testing

Phase 2 represents a fundamental shift from a centralized mode of operation to a distributed one. Independent, geographically separate teams now work in parallel at the project and local team level. Team selection is based on the requirements for the core components, called *anchor tenant* components. We use the term anchor tenant in analogy to the large department stores that create the primary organization of shopping malls. We typically design each µApp around one major component, such as a simulation, with many supporting components such as control widgets, data displays, and so forth.

***Building anchor tenant components.*** Developers create prototypes of the anchor tenant component. This is an important part of the process, as it provides the stakeholders involved in the CORD process with concrete representations close to the final application.

In February 2000, we selected AgentSheets as our component generator tool to build the anchor tenant for the π µApp. AgentSheets is an agent-based simulation component authoring tool. It applies to many kinds of applications, including mathematics, sociology, physics, chemistry, and art.[14] We authored the simulation in the Visual AgenTalk end-user programming language and rendered it into JavaBeans with the Ristretto component generator built into AgentSheets.[15]

We made the simulation available to the other team members as a Ristretto-generated, Java-enabled Web page for critique (see Figure 3). With the availability of an executable prototype, email discussion increased sharply. Based on the feedback, the design changed significantly through multiple iterations.

In other cases in which it was impossible or too costly to build interactive prototypes, we sometimes built animations instead.

While some consider the cost of prototyping generally too high, we found that building executable prototypes was essential for stakeholders' discussions of crucial design and implementation issues.

***Assembling components.*** Developing the anchor tenant component is an essential part of the development process, but a complete application also requires choosing the peripheral components such as databases, charting tools, and buttons. In the CORD process, the developers evaluate the initial choices of components made at the analysis and design stages and accept or reject the choices based on component functionality, usability, and interoperability with the anchor tenant component.

In February 2000, we assembled the complete μApp, including the simulation, buttons to control the simulation, a data table to collect output from the simulation, and an input text field to define the number of samples.

Components are tightly coupled through events and values. For instance, the simulation component simultaneously writes statistical information to a table component and reads simulation control parameters from text field components (see Figure 4). Developers assemble components visually by laying them out in a work area and connect components semantically by a wiring metaphor. In simple cases, component outputs are directly connected to other component inputs. More complex arrangements require adapter components or scripts. Guidelines and patterns stipulated by the component framework coordinator support the compatibility of components. Furthermore, we note compatibility problems and use them to guide framework revision.

For distributed component-based software development to work, it is necessary to manage component collections. This management includes maintaining component repositories and collecting component use stories. Who has used which components in what kind of context and how? Did they use them successfully, or did they have to work around issues or even modify a component? In our case, an ethnographer (from the μApps' publishers) captured use stories, which we shared among the extended group as text. We passed the use stories on to the component framework coordinator who archived them. The plan is for the component framework coordinator to maintain these use stories by connecting them to the component catalog. This management should help create a cumulative organizational memory that informs the distributed and changing teams participating in the overall testbed.

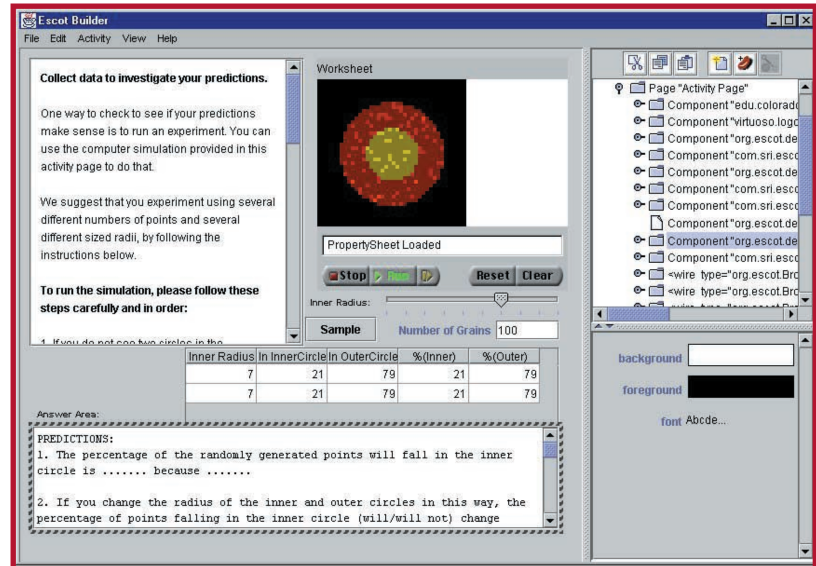



**Figure 4. We use the ESCOT builder tool to assemble simulation, spreadsheet, text, slider, and button components into a complete μApp. We specify the components used, their parameters and position, and the wiring scheme between them with the builder tool and capture them as XML files.**

In addition, the component framework coordinator helps guide the improvement of components in the repository based on specific μApp needs. This is a delicate matter of balancing generality and functionality, as well as granularity, of components. Developers often want highly specific functionality, but components can quickly become unwieldy if they implement behaviors that are only infrequently used.

Furthermore, if we add too many variant versions of components to the repository, the collection becomes hard to comprehend. Inheritance hierarchies might seem to be a solution to this problem. However, in our experience, ad hoc growth of an inheritance tree (for example, driven by needs of individual μApps) leads to clutter. We might use inheritance, but it should be driven by decisions at a product line level, where we can design branches to conform to stable, recurrent niches in market requirements.

Hence, to partly resolve tensions arising in a single μApp cycle, the component framework coordinator often advises the μApp team to use a component generator tool or scripting language to solve nongeneral implementation problems, so as not to clutter the repository. The repository stays focused on highly general and fairly large-grain components, as these are considerably easier to comprehend and use.

In February 2000, for the μApp's final

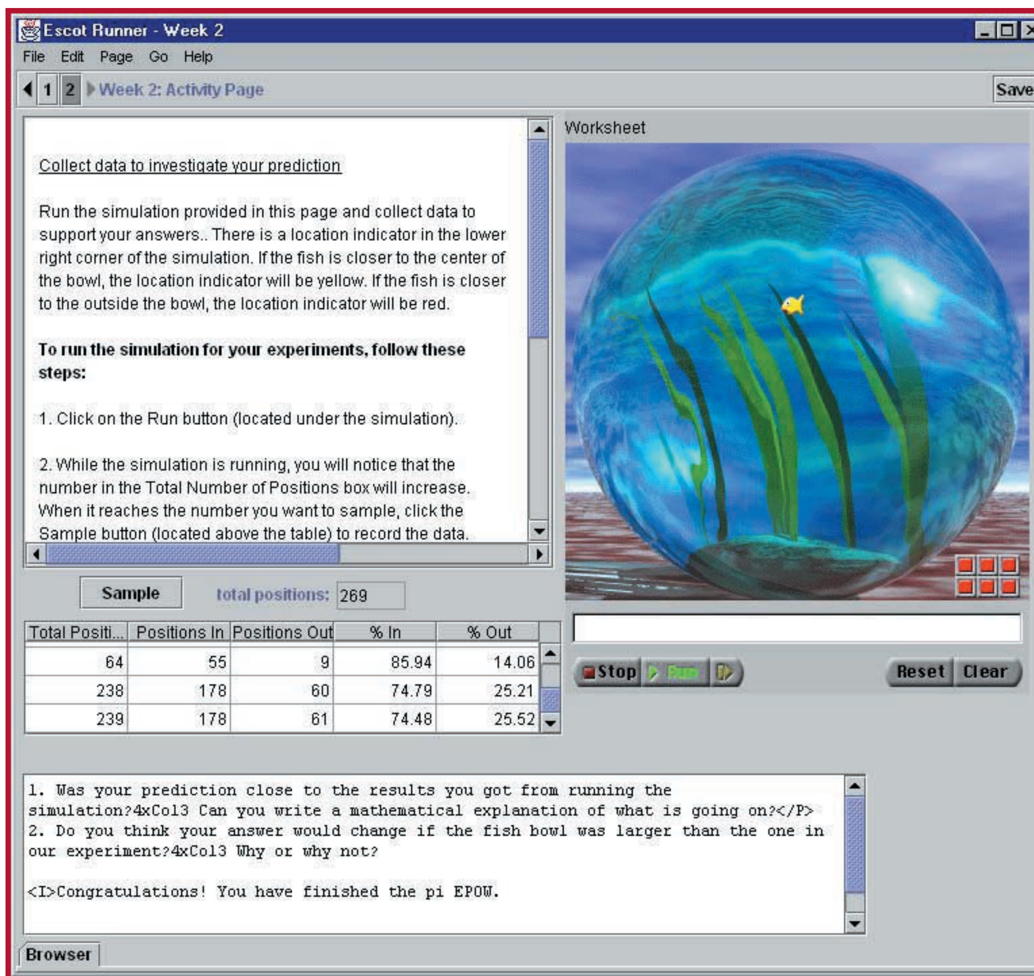



**Figure 5. A finished µApp. Middle-school students first predict the probability of the fish's being in a certain part of the sphere. They then use the fish bowl simulation to track the position of randomly moving fish and, using the data gathered, verify or reject their predictions.**

version, we replaced the idea of throwing darts at a board (see Figure 4) with a fish swimming around randomly in a fish bowl. We modified the layout a number of times in reaction to users' tests and replaced the crude artwork previously showing in the simulation component with a more sophisticated rendered image (see Figure 5).

## Publishing and using a µApp

Before we publish a µApp in the ESCOT context, the publisher, the MathForum, must test and approve it. Testing the µApp and its components consists of a mix of formal test cases and less formal, functional tests. For the formal portion, the component framework coordinator has been unit-testing individual components in the repository. We maintain unit tests in the repository and update them as we discover bugs, so they become more complete over time.

The publisher also tests the candidate µApp at a functional level by asking a representative sample of users to put it through the steps needed to solve the educational problem that the µApp will pose. Considerations such as cross-platform compatibility, performance, and clarity of documentation must pass through the publisher's quality control before making the µApp available to users (teachers and students, in our case) through the MathForum Electronic Problem of the Week Web site.

After the µApp went "live" in March 2000, students interacted with the activities assigned for each week and submitted their answers to MathForum mentors, who guided them through the process. A significant number of users experienced problems loading the µApp. Part of the problem was the use of an ESCOT-specific runner software (a tool developed specifically for running these µApps), which we later replaced with a browser-only solution.

At the surface level, component-based approaches appear to be ideally suited for distributed software development. We have found CORD to be effective in building educational applications, enabling aggressive project scheduling. CORD's component-based nature enables a high degree of parallelism involving distributed teams of domain experts, component framework coordinators, developers, publishers, and users in the software's development process. We advocate the use of increasingly formal design representations progressing from informal Post-It notes toward working applications. Increasingly formal design representations enable essential

communication between the distributed team members and avoid premature design commitments by allowing the right degree of design elasticity at different points in the development process.

The use of JavaBeans enabled our distributed, heterogeneous developer community to build components using different tools (basic JDKs, IDEs, and generators) and platforms (Windows, Mac, and Unix). However, to make sure that we properly integrated components, we needed a component framework coordinator. The CORD approach's scalability is bound by the number of components involved in a design. Typical CORD μApps have fewer than 20 components. However, a component's external complexity (that is, the complexity of its API) does not indicate its internal complexity. Interactive simulations, componentized legacy code, and database components often have simple interfaces but complex implementations.

On the negative side, we have a number of issues with the JavaBean platform. While JavaBeans have enabled true cross-platform development, we frequently found platform and virtual-machine-dependent implementation discrepancies that required a significant number of additional development cycles for debugging and work-around implementation. We recommend that project managers prepare themselves for extremely low development- versus testing-time ratios.

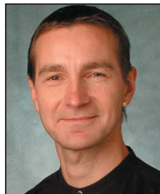
## Acknowledgments

**NSF grants REC 9804930 and SBIR DMI 9761360 supported this work. We would like to thank Janet Bowers, Natalie Sinclair, Chris DiGiano, and Jody Underwood for their valuable contributions to this article.**

## References

1. B. Boehm and V.R. Basili, "Gaining Intellectual Control of Software Development," *Computer*, vol. 33, no. 5, May 2000, pp. 27–33.
2. M.D. McIlroy, "Mass Produced Software Components," *Software Engineering*, P. Naur and B. Randell, eds., NATO Scientific Affairs Division, Garmisch, Germany, 1968, pp. 138–155.
3. PITAC Report, "Information Technology Research: Investing in Our Future," 24 Feb. 1999, www.ccic.gov/ac/report (current 2 Feb. 2001).
4. I. Jacobson, M. Griss, and P. Jonsson, *Software Reuse: Architecture, Process, and Organization for Business Success*, ACM Press, New York, 1997.
5. J. Udell, "Componentware," *Byte*, May 1994, pp. 46–56.
6. B.J. Cox, "Planning the Software Industrial Revolution," *IEEE Software*, vol. 23, no. 11, Nov. 1990, pp. 25–33.
7. H.A. Simon, *The Sciences of the Artificial*, 2nd edition, MIT Press, Cambridge, Mass., 1981.
8. J.M. Voas, "Certifying Off-the-Shelf Software Components," *Computer*, vol. 31, no. 6, June 1998, pp. 53–59.
9. J. Roschelle et al., "Developing Educational Software Components," *Computer*, vol. 32, no. 9, Sept.1999, pp. 50–58.
10. K. Beck, "Embracing Change with Extreme Programming," *Computer*, vol. 32, no. 10, Oct. 1999, pp. 70–77.
11. I. Jacobsen, *Object Oriented Software Engineering: A Use Case Driven Approach*, Addison Wesley, Wokingham, UK, 1992.
12. H. Rittel, "Second-Generation Design Methods," *Developments in Design Methodology*, John Wiley, New York, 1984, pp. 317–27.
13. M. Rettig, "Prototyping for Tiny Fingers," *Comm. ACM*, vol. 37, no. 4, Apr. 1994, pp. 21–26.
14. A. Repenning and T. Sumner, "AgentSheets: A Medium for Creating Domain-Oriented Visual Languages," *Computer*, vol. 28, no. 3, Mar. 1995, pp. 17–25.
15. A. Repenning and A. Ioannidou, "Behavior Processors: Layers between End-Users and Java Virtual Machines," *Proc. 1997 IEEE Symp. Visual Languages*, IEEE CS Press, Piscataway, N.J., 1997, pp. 402–409.

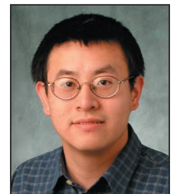
## About the Authors



**Alexander Repenning** is CEO and president of AgentSheets and an assistant professor and member of the Center of Lifelong Learning & Design at the University of Colorado at Boulder. His research interests include end-user programming, visual programming, computers and education, human–computer interaction, and artificial intelligence. He has a PhD in computer science from the University of Colorado at Boulder. He is a member of the IEEE and ACM and the creator of AgentSheets. Contact him at the Dept. of Computer Science and Center of LifeLong Learning & Design, Univ. of Colorado at Boulder, Campus Box 430, Boulder, CO 80309-0430; ralex@cs.colorado.edu.

**Andri Ioannidou** is a PhD candidate in computer science and a member of the Center of LifeLong Learning & Design at the University of Colorado, Boulder and a senior project manager at AgentSheets. Her research interests include educational uses of technology, interactive simulations, end-user programming, and end-reuse. She has a BS and an MS in Computer Science from the University of Colorado. She is a member of the ACM. Contact her at the Department of Computer Science and Center of LifeLong Learning & Design, University of Colorado at Boulder, Campus Box 430, Boulder, CO 80309-0430; andri@cs.colorado.edu.

**Michele Payton** is a database administrator at IBM and a former graduate research assistant in the computer science department at the University of Colorado, Boulder. She is finishing her master's degree in computer science at the University of Colorado, Boulder. She is a member of Tau Beta Pi and the ACM. Contact her at the Department of Computer Science, University of Colorado at Boulder, Campus Box 430, Boulder, CO 80309-0430; paytonm@cs.colorado.edu.



**Wenming Ye** is a software engineer at the Center for Technology in Learning at SRI International. His interests include components-based software design, open source software, and Java development. He has BS and MS degrees in computer science from the University of Colorado at Boulder. Contact him at the Center for Technology in Learning, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025-3493; yew@cs.colorado.edu.

**Jeremy Roschelle** is a senior cognitive scientist in the Center for Technology in Learning at SRI International. His research interests include interoperable and reusable components for math learning; the design of digital libraries for mathematical applets; and networked, handheld learning devices. He has a PhD in education from the University of California, Berkeley, and a BS in computer science and electrical engineering from the Massachusetts Institute of Technology. Contact him at the Center for Technology in Learning, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025-3493; Jeremy.Roschelle@sri.com.