Using Components for Rapid Distributed Software Development

Alexander Repenning , Andri Ioannidou, Michele Payton, Wenming Ye
Department of Computer Science, University of Colorado at Boulder
{ralex, andri, paytonm, Wenming.Ye}@cs.colorado.edu

Jeremy Roschelle
SRI International, Palo Alto, CA

Abstract

On paper, components would seem to be ideal building blocks for distributed software development. But how well are component-based approaches really suited for the rapid design and implementation of interactive applications? A large geographically distributed testbed consisting of domain experts, component framework coordinators, component generator tool providers, application developers, publishers, and users is producing and publishing educational applications. A rapid production pipeline process is employed to output one simple but complete application each week. The role of each stakeholder and the representations used to communicate design, and implementation issues are documented in the context of one complete application production cycle. Guidelines for effective use of component technology for distributed software development are provided.

Introduction

As a society we have grown more dependent on software development. Software is found everywhere: in our computers, TV sets, kitchen appliances, toys and even greeting cards. Unfortunately, to this day the process of software development is still highly unsatisfactory and immature. Software development has not reached the level of maturity found in other engineering disciplines to predictably produce software that works reliably, is easy to use and maintain, and is produced within the budget and on time. The situation has reached an alarming state, which has prompted the government to declare a new information technology crisis. The Information Technology research budget will significantly increase by $1.37 billion annually by the year 2004, $540 million of which is devoted to Software Engineering and Component Technologies, Human-Computer Interaction and Information Management [2, 3].

One software development approach quickly gaining support is the production of software based on components. Components that are platform-independent (e.g., JavaBeans) as well as platform-dependent (e.g., Active X) allow developers to conceptualize software as interconnectable components which act very much like software equivalents to hardware components such as integrated circuits. Components are highly reusable units of software functionality. At least in theory, building large projects out of well-defined and well-behaved building blocks can reduce the complexity of large software development, since building on stable substrates is faster [14]. The components used may either be produced and maintained by the same organization assembling components into complete applications or they may be acquired from third-party developers producing so called Components-Off-The-Shelf [5] [15].

While the component-based approach to software development is generally attractive, it appears to have exceptional appeal to distributed software development. One downfall in traditional distributed software development approaches is that software projects are often insufficiently decomposed. This results in overlapping or misunderstood responsibilities, which, in turn, can lead to significant communication breakdowns and ultimately to complete failure of a project. The nature of components forces designers and developers to better encapsulate functionality into cohesive, reasonably well-documented chunks of software.

We predict that distributed software development will rapidly gain momentum. One reason is that there is an increasing need to build relatively small

software systems for highly specific applications. This need requires a significantly different approach to software development. Unlike their large, monolithic, general-purpose software counterparts (e.g., Word produced by Microsoft), these small applications require Internet time processes. That is, these micro applications (µApps) need to be built quickly by relatively small teams of domain experts and application developers. The people needed for these tasks may be allocated on the fly and are likely to be dispersed geographically.

This paper reports on the experiences of a large National Science Foundation (NSF) supported testbed called Educational Software Components of Tomorrow (ESCOT) [http://www.escot.org]. [12] ESCOT is building a digital library containing educational software focused on Middle School mathematics. While some readers may not be intrinsically interested in educational software, the findings reported here are of a general nature. The ESCOT goals include building interactive, JavaBean-based content, and for education to explore the process of distributed software development, with the specific objective to build and deploy reliable software rapidly. The ESCOT testbed is based on a large pool of geographically distributed stakeholders in the US, and other countries. These stakeholders include domain experts, component framework coordinators, application developers, publishers and users.

An aggressive schedule to produce and publish a small but complete application is used to stress-test a component-based rapid software development process (CORD). Parallel production cycles involving different subsets of stakeholders are aligned in a pipeline fashion to output software at a weekly rate. This paper will focus on one such cycle producing four complete µApps in five weeks. The development process is described chronologically from early design all the way to the publication of the software. We show how the different stakeholders share design and implementation information over time and space. Specifically, we illustrate the incrementally increasing degree of representation formality migrating over time from mockups made out of paper and Post-It™ notes toward XML-based component connectivity representations. Following the description of the CORD process in the context of ESCOT, we share some guidelines for building component-based software with a distributed team.

The CORD Process

The **CO**mponent-based **R**apid **D**istributed (CORD) software development process is related to Extreme Programming (XP) [1]. XP replaces the four coarse sequential steps of the Waterfall model with an extremely large number of parallel analysis, design, implementation and testing cycles. CORD, like XP, employs parallelism but the granularity of the parallelism is related 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 relatively small size of components and the distributed nature of the

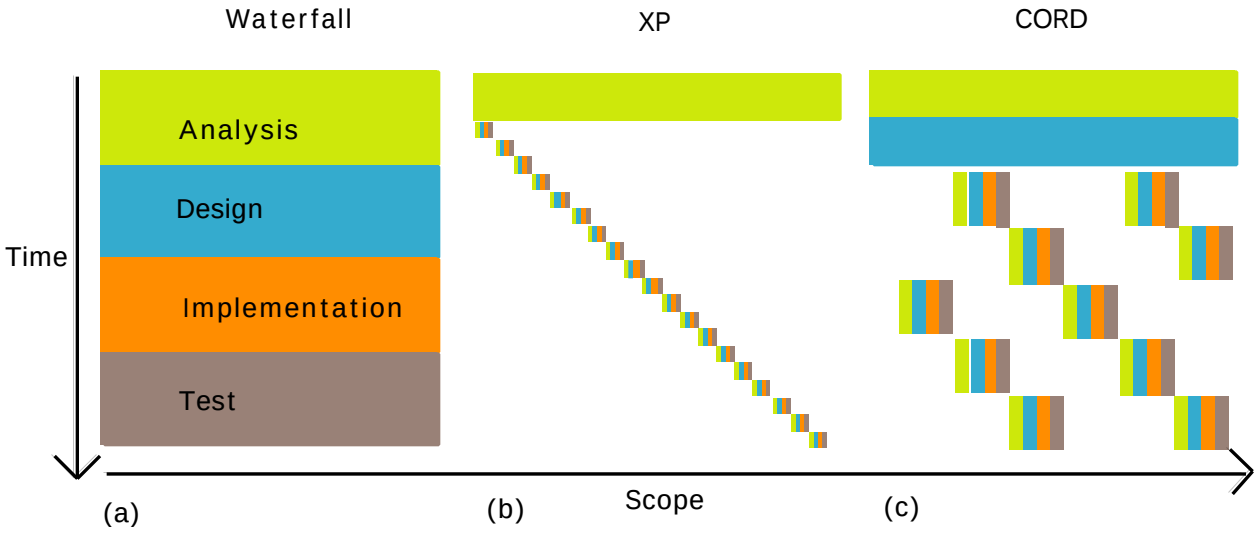

Figure 1: Models of the Software Development process: (a) The traditional Waterfall model, (b) the Extreme Programming model, and (c) the Component-based Rapid Distributed (CORD) software development model.

CORD approach suggest development activities that are relatively small in scope, highly parallel and highly iterative.

The CORD software development includes a large number of activities that unfold at two levels. At the component level, components are being implemented, generated, debugged, adapted, documented and shared. At the application level, components are assembled into small, but complete applications (µApps) that are deployed, tested and used. All activities at the component as well as at the application level are highly parallel. Multiple teams in different locations are simultaneously working on and generating different components, and evaluating the application.

Although CORD is similar to the concept of XP, there are crucial differences between them. In CORD:

- The project start includes centralized analysis and design. A large group including users, domain experts, designers, tool developers and application developers, not only analyzes project requirements, but also creates application mockups. These mockups serve as design blueprints for the further development of the project.
- Development is distributed and component-centered. After an initial centralized analysis and design has been established, development of components can be distributed to independent teams. Teams are coordinated through regular builds. Each build assembles all or some components into a testable application or applet.
- There is an even larger degree of parallelism. In contrast to XP there is not only parallelism within a development team but also parallelism resulting from multiple distributed teams.

This paper describes the CORD software development process in the context of ESCOT, where teams collaboratively produce µApps. These µApps, called ePOWs (Electronic Problems of the Week), are math activities that are published via the web for middle-school students to interact with, answer questions based on them, and get feedback from volunteer mentors. A new problem is posted each week and the problems are related to the same theme for a month, with each week's problems becoming progressively more difficult. These micro-apps are published on the Web in a format that is accessible to teachers and students. The particular µApp described in this paper centered around the derivation of the mathematical constant π (3.1415926535…) using experimental probabilities and other related concepts of probability.

Stakeholders: Collaborators and Roles

In the CORD software development process several stakeholders work together from geographically dispersed locations. These teams, called “integration teams” in the context of ESCOT, design and build

Collaborators and Roles:
1) San Diego State University, 2) Queens University ***Domain Experts*:** · Design math activities · Create HTML mockups
SRI International ***Component Framework Coordinator:*** · Design and evolve component framework · Provide component authoring tool · Organize component repository
University of Colorado ***Application Developer:*** · Produce µApp · Generate anchor-tenant component · Provide Component Generator Tool · Coordinate development among teams
The Math Forum ***Publisher:*** · Critique design · Test µApp · Publish µApp on Web
Math Teachers and Students ***Users:*** · Participate in analysis and design · Provide feedback to developer · Use µApp

Figure 2: The stakeholders, their geographic location, and the roles each played in the development of the Pi µApp.

math µApps. Team selection is based on the requirements for the core components of the µApp.

Creating these µApps is a design problem that requires more knowledge than any single person can possess, as the knowledge relevant to the problem is distributed [4]. For example, for the math µApps , the developers do not know much about the domain (namely the teaching of middle school math), whereas domain experts may not have the technical expertise to develop and assemble a component-based µApp. This "symmetry of ignorance" (or "asymmetry of knowledge") [11] can be accounted for by designing and implementing a µApp in a distributed fashion, in integration teams. Such a process allows for more interaction between the software and its intended use (in the case of πµApp, the pedagogic goals of the teachers who would use it), so that users have a better chance of getting what they need. The members of the integration team and the role each of them played in the creation of the component-based distributed πµApp are explained in Figure 1.

Next we describe the two main phases of the CORD software development process in the context of building a µApp for the ESCOT testbed.

Phase 1: Centralized Analysis and Design

In phase 1 of the CORD software development 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 breaks down when the process is insufficiently decomposed, the roles of the stakeholders are not clearly defined or overlapping, or when communication breaks down. In the CORD process, even if the largest part of the software development process is distributed, we recognize the importance of gathering all the participants together at least in the initial analysis and design phase. Despite scheduling issues, gathering the entire group is important. When the idea is first emerging and the roles are being defined and distributed among the team members, face-to-face interactions and design activities with low-fidelity media, [10] which is accessible to everyone, are crucial. The former enables issues such as role definition and task distribution to be resolved early in the process, whereas the latter allows all the stakeholders to participate in the design process.

In the case of the creation of the πµApp, low-fidelity design media (such as paper and Post-It™ notes) are used to create mockups. In a 5-day workshop held at Swarthmore College, Pennsylvania, home of the

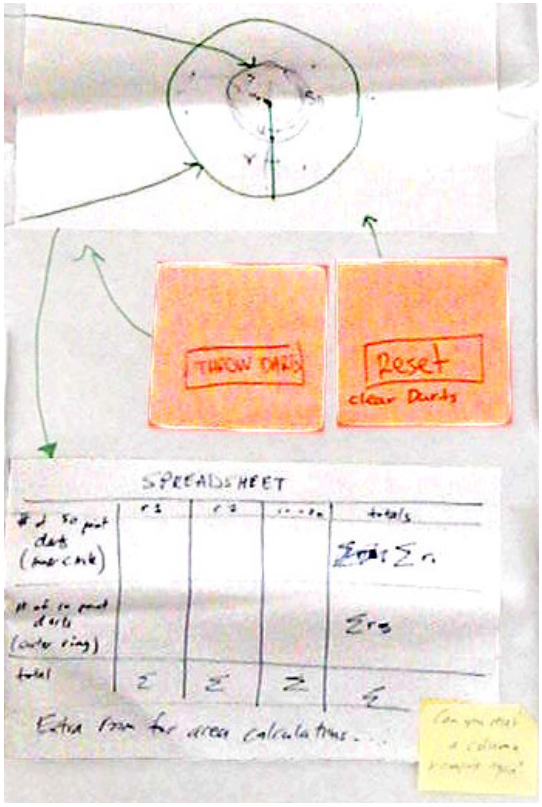

Figure 3: Small part of the Design Mockup showing simulation (top), buttons (middle), and spreadsheet (bottom) components.

Math Forum, a group of domain experts, component framework coordinators, application developers, publishers and users brainstorms ideas for µApps, one of which is the πµApp. The group analyzes a suite of about twenty such ideas and creates mockups for each one. The original mockup of the πµApp consists of poster-size paper sheets with stickies representing components. This is not only a low-fidelity design medium accessible to all the stakeholders participating in the design, but it also an ideal match for the nature of the component-based software to be built (Figure 2).

One of the domain experts in the group then takes the design and is responsible for transferring the idea from the low-fidelity mockup into a more formal

representation that the rest of the team can use to further develop the application.

Formalizing Design with HTML Mockups

Transforming the initial design from the low-fidelity mockup to a more formal medium that can be accessed by the entire team is an important next step in the CORD software development process. Documents such as HTML documents, while still fairly low-fidelity media, are more formal than paper mockups and can be used to document the design. Moreover, they are sharable among a geographically dispersed group. As such, they can be used to trigger design discussions among the stakeholders and can be good starting points for feedback solicitation not only from the internal group, but also from actual users. Finally, they are still low-fidelity enough to allow for a quick prototype even from computer end-users, not necessarily developers.

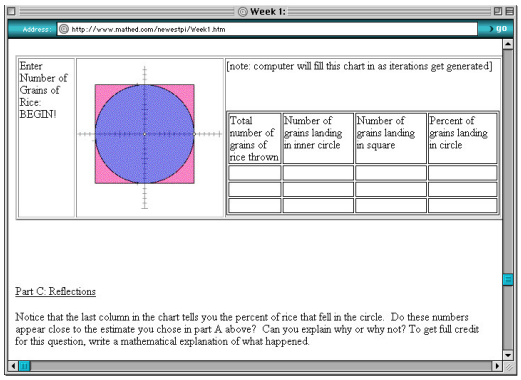

Figure 4: The HTML Mockup of a part of the Pi µApp can easily be accessed by other member of the design team and critiqued.

The domain expert for the π µApp has expertise in designing curricula for math education. She creates a preliminary design (Figure 4) and posts it on the web as a blueprint for the final application, refining the crude mockup created by the entire group.

Being able to share the design among the members of the distributed team is important, but the medium (simple HTML) does not allow manipulation or collaborative modification. The result is high email traffic leading to communication issues and the need for feedback management. Nonetheless, the initial design is evaluated both by the developers and the rest of the team (other domain experts and publishers). This evaluation / redesign / re-implementation cycle occurs in each step of the CORD software development process by the multiple teams working on the µApp (hence the parallel boxes in the CORD diagram in Figure 1).

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 the Anchor Tenant components. For the π µApp, the University of Colorado based AgentSheets® team was chosen.

Building Anchor Tenant Components

In a component-based software project, there are major components and other, peripheral or smaller-scale, components. Developers create prototypes of the major components, called “anchor tenants.” This is an important part of the process, as it provides the stakeholders involved in the CORD software development process with yet another level of formality, even closer to the final application. This allows them to provide more concrete feedback to the main developers.

Anchor tenant components in the ESCOT µApps are created using SimCalc, Geometer's Sketchpad® and AgentSheets®. The anchor tenant of the π µApp is an AgentSheets® simulation. AgentSheets®[6, 7, 9] is an authoring tool for end-users to create interactive simulations and games. End-user programmable agents created with the Visual AgenTalk language, can read web pages, play videos, play sound and MIDI music, can speak, compute formulae, and react to mouse and keyboard input. "What-if" scenarios and simulations re-packaging information gathered from the web unfold in spreadsheet-like workspaces inhabited by autonomous agents instead of just numbers and strings, Finally, Ristretto™, the Agent-to-Java-Byte-Code compiler, turns simulations directly into interactive Java applets and JavaBean components. AgentSheets is used by various people ranging from elementary school children creating EcoWorlds to explore issues of food webs and sustainability to NASA scientists to simulate the behavior of E.coli bacteria in microgravity.

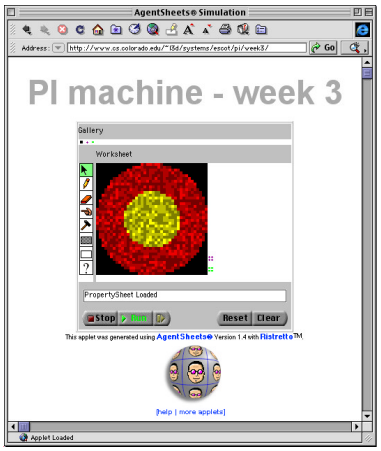

Figure 5: The central components of each project, called the anchor tenant components, are prototyped and made available to other teams as Java applets

The AgentSheets simulation [8] with thousands of agents representing randomly generated points is built by a pair of programmers in the application developer team that is, the AgentSheets team at the University of Colorado. This style of programming is compatible with the XP concept of Pair Programming [1] [16]. The results of the prototypes are shared with the rest of the group in the form of Java applets via the Web (Figure 5). Even though the applet prototype does not have the same interaction mechanisms as the finalized µApp, it serves as a good medium for initial testing. After having seen the concrete prototypes of the anchor tenant component, the distributed team argues the rationale of the µApp design and implementation. Based on the feedback, the design changes leading to multiple iterations of developing, sharing, assessing and redesigning the major component for the πµApp.

Assembling Components

In component-based software, developing the anchor tenant component is an essential part of the process, but choosing components to accompany it is also very important. In the CORD software development process, the application 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 seamless interoperability with the anchor tenant components.

For distributed component-based software development to work, it is necessary to manage component collections. This management includes the maintenance of component repositories, but also the collection of component use stories. Who has used what components in what kind of content and how? Was it used successfully, were there issues that required work around or was it even necessary to modify a component? This management is essential because in contrast to traditional software development team members building software may change all the time. As a consequence software development experience is no longer maintained as war stories shared by team members, but needs to be collected to the degree possible externally as process memory.

The developers of the πµApp have at their disposal a component repository managed by the Component Framework Coordinators who are responsible for

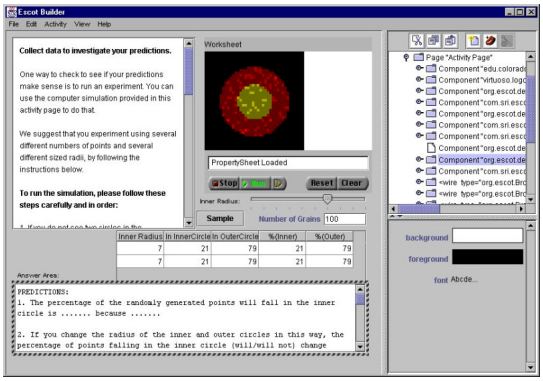

Figure 6. The ESCOT Builder tool is used to assemble simulation, spreadsheet, text, slider and button components into a complete µApp. The components used, their parameters, position and the wiring scheme between the components are specified with the builder tool and captured as XML files.

organizing the component repository and provide a level of organizational memory with component use stories. Having this organizational memory is extremely useful to the process. A large set of interoperable components by itself is not enough to support the component-based software development process. An organized repository of use stories that accompany the components is necessary to provide application developers with information on a particular component's use. This enables them to make intelligent and informed decisions in choosing components that are appropriate for use in their µApp.

Once the µApp is assembled, the producers share early prototypes with the entire group. Even if the application is not finished, sharing the prototypes is an essential part of the process as it is the closeness to

the final application that spawns more meaningful feedback. After having seen the prototypes, the stakeholders have a different and more concrete sense for the potential of the µApp than with the design mockups and therefore the feedback becomes more concrete. At this point in the process, it is necessary to have synchronous communication media in order for the stakeholders to negotiate their "asymmetry of knowledge" and evolve the design of the µApp even further.

Publishing and Using a µApp

Before a µApp in the ESCOT context gets published, it must be tested and approved by the publishers, the Math Forum. Considerations such as cross-platform compatibility, performance, and clarity of documentation need to pass through the quality control of the publishers before making the µApp available to users (teachers and students) on the web through the Math Forum electronic Problem of the Week web site.

After the µApp is 'live' students interact with the activities assigned for each week and submit their answers to Math Forum mentors that guide them through the process.

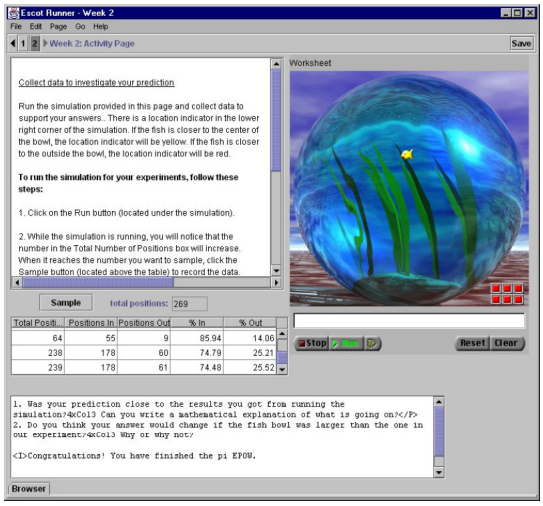

Figure 7. A finished µApp. Middle school students first predict the probability of the fish being in a certain part of the sphere and then use the Fish Tank Simulation to track the position of randomly moving fish and using the data gathered verify or reject their predictions.

Matching Representation and Communication to Task

Different phases of the CORD software development process require different communication media and different types of messages to be communicated among the members of the distributed team. Moreover, sharing increasingly formal representations is the optimal flow in the process as it allows all stakeholders to participate in the design and development process. Since not all stakeholders are at the same level of technological competency gradually introducing them to the finished product is a good idea. Everybody's feedback is needed, and to get it, it is necessary to provide concrete artifacts to critique. Taking advantage of the affordances of the tools and representations at any given level of formality allows for a smooth process in the distributed creation of the µApp. Therefore, the right level of formalization at the right phase of the process should be chosen carefully, as well as the medium chosen to communicate these.

Centralized Analysis Phase and Design Phase

The centralized analysis and design phase of the CORD process requires face-to-face communication using low-fidelity media. During the initial design, when the ideas are being shaped, the physical presence of all the stakeholders at the same place at the same time is essential. Moreover, since most of the members of the distributed software development team do not have the same technological background, to get everybody to participate and contribute to the design process, low fidelity design media, such as paper and stickies, are necessary. These media are ideal, as they are easily manipulatable and changeable.

During the design phase, more formalization is needed. The media used in this phase need to be both sharable over distant physical spaces and persistent as the stakeholders in the CORD software development process need to communicate the ideas expressed in the mock-up phase among the geographically dispersed group. The artifacts documenting the design also need to be persistent. The web provides a medium for sharing these HTML documents and distributing them to all the stakeholders.

Distributed Analysis, Design, Implementation, Testing Phase

The distributed phase of the CORD process requires both asynchronous and synchronous communication means and more formalized computational objects to be shared among the team. Communication media such as email and the web (for sharing things such as major components, in the form of applets for example, and prototypes of the µApp) are sufficient. However, major design decisions and technical emergencies are better resolved in synchronous communication. Placing conference calls, using systems such as ICQ (for 2 people synchronously communicating) and TappedIn [13] (for groups of people synchronously communicating) are some options currently available.

In this phase, developers provide accessible and simplified prototypes, at least of the anchor tenant components. Applets serve this purpose for the development of the πµApp. While they lack the interoperability with other components that the finished component would have, they can be readily distributed among the stakeholders for quick feedback. Applets have the additional advantage that they are not complicated to use. Just sending around a link to a web page and minor instructions on how to use them is enough. Soon afterwards, however, various versions of the complete application need to be distributed. Versions of the fully functional application need to be examined and tested by all the stakeholders so that they can provide comprehensive feedback. While it is important to give people prototypes in any form (paper, HTML, applets) for the design to evolve, it is not until the entire application comes together that the more meaningful feedback begins to accumulate.

Conclusions

At the surface level component-based approaches appear to be ideally suited for distributed software development. In this paper we have introduced the CORD (COmponent-based Rapid Distributed) software design approach used in the context of building educational applications. The CORD approach is related to Extreme Programming in the sense that it also replaces the traditional analyze/design/implement/test sequence with a highly parallel approach. On the positive side we have found CORD to be effective enabling extremely aggressive project scheduling. On the negative side, however, we have not found the JavaBean platform to be sufficiently mature to deliver stable, efficient interactive content cross platform.

Acknowledgements

Grant numbers, Janet Bowers, Natalie Sinclaire, Jody Underwood, …

References

1. Beck, K., "Embracing Change with Extreme Programming," *IEEE Computer*, Vol. 32, pp. 70-77, 1999.

2. Boehm, B., and V. R. Basili, "Gaining Intellectual Control of Software Development," *IEEE Computer*, Vol. 33, pp. 27-33, 2000.

3. Committee, P. s. I. T. A., "Information Technology Research: Investing in Our Future," , , 1999.

4. Fischer, G., "Symmetry of Ignorance, Social Creativity, and Meta-Design," *Submitted to Creativity & Cognition*, Vol. 3, pp. 1999.

5. Pour, G., "Moving toward Component-Based Software Development Approach," *Proceedings of the Technology of Object-Oriented Languages and Systems*, Beijing, China, IEEE, 1998, pp. .

6. Repenning, A., A. Ioannidou, and J. Ambach, "Learn to Communicate and Communicate to Learn," *Journal of Interactive Media in Education, http://www-jime.open.ac.uk/98/7*, Vol. 98, pp. 1998.

7. Repenning, A., A. Ioannidou, and J. Phillips, "Building a Simulation of the Spread of a Virus," *Learning Technology Review*, pp. 56-72, 1999.

8. Repenning, A., A. Ioannidou, and J. Phillips, "Collaborative Use & Design of Interactive Simulations," *Proceedings of Computer Support for Collaborative Learning Conference (CSCL'99)*, Stanford University, California, 1999, pp. 475-487.

9. Repenning, A., and T. Sumner, "Agentsheets: A Medium for Creating Domain-Oriented Visual Languages," *IEEE Computer*, Vol. 28, pp. 17-25, 1995.

10. Rettig, M., "Prototyping for Tiny Fingers," *Communications of the ACM*, Vol. 37, pp. 21-26, 1994.

11. Rittel, H., "Second-Generation Design Methods," in *Developments in Design Methodology*, Cross, Ed., John Wiley & Sons, New York, 1984, pp. 317-327.

12. Roschelle, J., C. DiGiano, M. Koutlis, A. Repenning, et al., "Developing Educational Software Components," *IEEE Computer*, Vol. 32, pp. 50-58, 1999.

13. Schlager, M. S., and P. K. Schank, "TAPPED IN: A New On-line Teacher Community Concept for the Next Generation of Internet Technology," *Proceedings of Computer Support for Collaborative Learning (CSCL)*, Toronto, Canada, 1997, pp. .

14. Simon, H. A., *The Sciences of the Artificial*, MIT Press, Cambridge, MA, 1981.

15. Voas, J. M., "Certifying Off-the-Shelf Software Components," *IEEE Computer*, Vol. 31, pp. 53-59, 1998.

16. Williams, L. A., and R. R. Kessler, "All I Really Need to Know about Pair Programming I learned in Kindergarten," *Communications of the ACM*, Vol. 43, pp. 108-114, 2000.