Facilitator in a Box: Computer Assisted Collaboration Engineering and Process Support Systems for Rapid Development of Collaborative Applications for High-Value Tasks

Robert O. Briggs¹, Gwendolyn L. Kolfschoten², Gert-Jan de Vreede¹,², Conan C. Albrecht³, Stephan G. Lukosch²

¹Center for Collaboration Science
University of Nebraska at Omaha
rbriggs@unomaha.edu
gdevreede@unomaha.edu

²Systems Engineering Department
Delft University of Technology
g.l.kolfschoten@tudelft.nl
s.g.lukosch@tudelft.nl

³Information Systems Department
Brigham Young University
conan@warp.byu.edu

Abstract

This paper proposes Computer Assisted Collaboration Engineering (CACE) and Process Support Systems (PSS) as a new approach to move beyond limitations of the current generation of group support systems (GSS) and other collaboration technologies in supporting recurring collaborative work practices. It argues the need for certain capabilities in CACE and PSS systems, and illustrates key concepts with examples from proof of concept systems. It summarizes early results with these systems in the field.

1. Introduction

Collaboration has become increasingly important to the success of many organizations [1]. The term collaboration, as used in this paper, means joint effort toward a group goal [2]. Collaboration, however, can be a mixed blessing [3], bringing a complex set of communication, coordination, and co-production challenges[4]. Research shows that, under certain circumstances, groups can use Group Support Systems (GSS) and other collaboration technology in ways that substantially improve their efficiency and effectiveness [5-9]. A group support system is a suite of software tools optimized to support co-production processes. Teams using group support systems can attain 50-70% reductions in labor hours and can cut project cycle times from 70-90% (e.g. [10-12]).

Current GSS offer groups tools for collaborative brainstorming (ideation), reducing, clarifying, organizing, and evaluating ideas, and for building commitment among group members. To reduce barriers to participation, GSS can typically be configured so that, during certain interactions when it may be useful (e.g. idea generation, idea evaluation) users can contribute anonymously. GSS tools allow for simultaneous contributions from all participants. GSS usually offer collaborative data structuring and processing tools that enable groups to immediately discuss, vote on, and clustering results [3, 13].

While GSS offer new opportunities to teams, they have been slow to transition into the workplace [14]. People without specialized group process training often struggle to use GSS tools in ways that move their groups through a set of effective activities towards their group goals. Lacking that expertise--even with good technology at hand--groups can bog down because of misunderstandings, incompletely-shared information, differences about how shared goals can be attained, mutually-exclusive private goals, or differences of preference for approaches and conditions [15].

Some groups turn for help to professional facilitators, group-process experts who design collaborative work practices and conduct them on behalf of the group. Research shows that groups supported by an expert facilitator can be significantly more effective and efficient than groups who are left to their own devices [16, 17, 29, 30].

Unfortunately, skilled facilitators are an often unworkable solution for groups because facilitators are scarce and expensive, and because their special skills give them high professional mobility, making it difficult for an organization to maintain a stable in-house facilitation capability [18, 19]. Field studies suggest that ongoing use of a GSS in an organization may depend on a champion facilitator. When those people leave their position, the knowledge they have accumulated disappears with them, leaving nobody behind who can use the technology in ways that create value [11, 18].
2. Three directions for advancing collaboration support

The challenge, therefore, is to find a way to provide teams not only with collaboration technologies, but also with sufficient understanding of how to use those technologies to attain their goals when professional facilitation consultants or facilitation training is not available. To this end, at least three possible strategies suggest themselves: simplify the technology, improve the transferability of the facilitation skills, and build facilitation support into the technology. As we will discuss below, progress has been made on these strategies, but they do not yet provide a complete solution.

2.1. Simplify Collaboration Tools

Collaboration technology is advancing quickly with hundreds of products currently available and dozens more appearing each year [35]. Many such offerings are simple to use and widely available. Instant Messaging and Chat applications support same-time, distributed conversations. Shared editors like Google Docs provide for joint creation of digital artifacts. Streaming technologies like Adobe Connect provide audio, video, and desktop sharing. Such simple to use tools can be very effective in supporting simple collaborative tasks. However, as simple as such products are, each tends to be a silo focused on specific collaborative activities, rather than on complete collaborative work practices. All these tools are developed for the mass market and thus do not target the specific needs of a collaborating group [35]. Such needs must be satisfied by customized tool development involving the prospective end-users [31]. This leaves the challenge of tool selection, tool integration, and group process design to group members who may be experts in their own domains, but may lack the technical skills required to integrate the tools they need and the collaboration expertise to use them effectively throughout a complete collaborative work practice. In other words, current simple technologies do not provide users with a complete, well documented and fully supported work practice for completing their tasks [35].

2.2. Increase the Transferability of Facilitation Skills

To improve the transferability of facilitation skills, researchers are developing training approaches [20] and design patterns to capture collaboration techniques [21,22]. A novice facilitator typically requires about a significant apprenticeship to gain sufficient insight to design and conduct work practices on their own [3,20,23]. Collaboration Engineering researchers, however are working to distill a collection of collaboration techniques and codified them into a design pattern language called “thinkLets” [21, 22]. A thinkLet is a named, scripted procedure that invokes known patterns of collaboration [23]. ThinkLets serve as logical design elements for collaborative work practices. The essence of a thinkLet is codified in its rules. The rules of a thinkLet specify Actions that should be taken by people in certain Roles under certain Constraints using certain Capabilities [21, 22].

The documentation for each thinkLet includes a sample script for its execution. At work-practice design time, the collaboration engineer can modify the script in any way or replace it entirely. So long as the final script invokes the rules of the thinkLet, the desired patterns of collaboration should emerge among participants at execution time.

ThinkLets and other collaboration techniques are readily transferable to non-professionals. Novices can learn to conduct specific thinkLets-based work practices in a day or two instead of having to spend a year or more as an apprentice learning general facilitation skills [36]. Current collaboration technologies, however are not optimized for direct implementation of collaboration techniques like thinkLets [21,27,28,35]. They focus instead on providing general capabilities, leaving it to the users to figure out how best to use those capabilities each time to support a particular thinkLet.

2.3. Build Facilitation Support into the Technology

Group work processes often include activities like sense making, goal setting, product definition, solution generation and evaluation, negotiation, decision making, planning, co-production, and action review. Facilitators help groups by designing effective ways to such activities, by conducting workshops on behalf of groups, and by intervening throughout a collaborative process to improve issues of communication, reasoning, information access, distraction, and goal congruence. Notwithstanding some promising results in the area of multi-criteria decision making [32], computers are not yet sufficiently advanced that they can play the full role of a facilitator.

Collaboration Engineering (CE), however, suggests an alternative that may be within the capabilities of current computers. CE is an approach
to designing collaborative work practices for high-value recurring tasks and deploying them for practitioners to execute for themselves without ongoing support from professional facilitators [33]. Using the thinkLets-based CE design approach [24], collaboration engineers do not attempt to turn users of collaboration technologies into experienced professional facilitators. Rather, they embrace the fact that practitioners need only to be experts in their own domain, and so teach them just the techniques they need to conduct their own mission-critical work practice successfully (e.g. requirements negotiation, marketing focus groups, strategic planning, risk assessments). For recurring tasks, collaboration engineers can develop repeatable methodologies that produce sound results under most circumstances. These methodologies can become sufficiently honed and predictable that others can be trained to execute them successfully, even though they are not professional facilitators [24,25,26]. Research has shown that practitioners using an engineered collaborative work practice can achieve results comparable to those of professional facilitators [26] for that particular work practice.

Although simple tools and transferable collaboration techniques exist, the current generation of GSS and other collaboration technologies are not yet optimized for capturing reusable collaborative methodologies with the tools that support them as purpose-build collaborative software applications. First steps towards such a support are based on context-adaptive shared workspaces which capture the collaboration context and adapt the collaboration environment to better fit the current requirements of the group [35]. In this paper we advance Computer Assisted Collaboration Engineering (CACE) and Process Support Systems (PSS) as an approach that combines the three directions presented above to support the rapid development and deployment of collaborative applications.

3. Elements of the CACE /PSS Approach

To fulfill its purposes, a CACE tool should support at least four key challenges for collaboration engineers: a) capturing and reusing collaboration techniques; b) supporting the logical design of work practices; c) supporting the physical design of PSS applications, and d) documenting work practices and the PSS applications that support them. These four challenges are reflected in the four key elements of a CACE: a) technique editors; b) logical design editors; c) physical design editors; and d) a documentation engine. These capabilities depend, in turn on several other key technology elements. Among those are a collection of uncoupled, highly configurable collaborative components and a PSS client framework that orchestrates execution of the PSS application. In this section we describe each of these elements and discuss their utility to collaboration engineers and to practitioners. We illustrate the elements with examples from proof-of-concept prototypes of CACE / PSS systems.

3.1. Configurable Collaborative Components

The heart of a CACE / PSS concept consists of a collection of simple, decoupled, highly configurable, elementary collaborative software components. Examples could be multi-user text trees, multi-criteria polling tools, audio and video channels, voting tools, presence indicators, and shared document editors, to name but a few. The configurability of the components is important because, from a small handful of components, a collaboration engineer could configure a wide variety of collaborative applications without having to write any new software code.

Consider for example the variety of ways a collaboration engineer could configure a single shared text tree component to create what appears to the user to be a tightly-coupled, sophisticated idea generation tool for a requirements negotiation work practice. Figure 1 shows such a tool configured from several instances of a configurable shared tree component using a proof-of-concept CACE prototype called Cognito. In the left panel, an instance of the tree component is configured with a white background, black foreground, large bold font, and with bucket images as leading icons. A drag-and-drop event into this instance causes a contribution to be demoted from this top-level tree to a sub-tree that appears in a different part of the tool. Double-clicking an item on the bucket-tree causes the leading icon to change to a tipped-bucket image, and causes three child-instances of the tree component to appear on tabs in the right panel.

The child-tree instance on the first tab (labeled “Explain the category”, but not currently visible) is configured with large fonts to appear as a presentation slide explaining the parent category.

The tree instance on the second tab (labeled “Brainstorm” but not currently visible) is configured as a brainstorming tool to support an ideation technique that pushes participants for breadth rather than depth. Double-clicking an item in this component starts the brainstorming procedure.

The tree instance on the third tab (Labeled “Win Conditions,” currently visible) supports a convergence technique where the best ideas from the
brainstorming activity are added to a public list and refined by the group. It is configured with a green background, small black font, with an enumerated leading contribution type tag identifying the contributions as win conditions, and with horizontal black lines separating the contributions. A drag-and-drop event in this instance of the component moves the source contribution to the drop location. A double-click event on a contribution in this tree instance causes another child-instance of the tree component to pop up.

This instance is configured with a background that suggests lined notebook paper. Contributions appear without numbers and with a small light-bulb as a leading icon. Contributions are configured to appear with a unique enumerator tag following each contribution. The tree is configured with a header panel that displays the text of its parent contribution (in this case the text of W4). Double-clicking a contribution in this instance of the tree component has no effect.

Figure 1. A complex PSS brainstorming tool to support software requirements negotiation composed of several instances of the same configurable tree component.

Each instance of the tree in the tool structure is configured differently. Although none of the instances is aware of the others in the PSS at run time, their configurations make them behave as if they are a single tightly-coupled tool.

3.2. Computer Assisted Collaboration Engineering (CACE)

A CACE provides collaboration engineers with a rapid application development (RAD) environment in which they design collaborative work practices and configure collaborative components into applications to support those work practices.

3.2.1. Technique Editors.

Because well-tested and documented collaboration techniques can invoke predictable patterns of collaboration [21], such techniques can serve as templates for group activities. There are dozens of attributes of collaboration techniques that a collaboration engineer may want to capture (e.g. Name, patterns of collaboration it invokes, script, rules, selection guidance, training aids, caveats…) [21, 22]. A CACE should therefore provide an extensible way to record the many details of the techniques collaboration engineers want to reuse.

It would not be sufficient, however, for these codified techniques to remain in a silo to be looked up like reference documents. Many of the attributes recorded for techniques have counterparts in the hundreds of attributes of a group activity. It should, therefore, be possible in a CACE environment for a collaboration engineer to draw an item from the techniques repository and use it to instantiate automatically an activity for the group. Attributes of the newly created activity that coincide with attributes of the source technique should be populated with the generic content from the technique (e.g. script, guidelines for successful execution). The collaboration engineer would then need only to edit the content of those fields to tailor the activity to the task at hand, rather than having to start from scratch.

3.2.2. Logical Design Editors

A CACE tool should provide a set of logical design editors that allow collaboration engineers to work with practitioners to model work practices. These editors should support, for example, defining and modeling the group goals, specifying the products a group must create to achieve their goals, identifying the activities they must complete in order to create their products, defining the patterns of collaboration that should manifest during each activity, and selecting or inventing the technique(s) that will be used to invoke the desired patterns of collaboration. A conceptual design for this aspect of a CACE the tool is proposed in [27], based on a collaboration engineering design approach presented in [24].

3.2.3. Physical Design Editors

When the logical design of one or more activities has been completed, collaboration engineers may turn to physical design editors. Here, they may drag-and-drop pre-made components onto an application canvas. By organizing these components and setting their properties and events, a collaboration engineer can quickly create a customized application tailored to a group’s precise needs for each activity in a work practice. This may require at least three different
kinds of editors: An Activity-Role Editor, a Screen View Editor, and a Tool Editor

Figure 2 presents an example of an Activity-Role Editor from a prototype CACE called ActionCenters.

The editor displays five activities and three phases. It displays two roles: Analyst and Stakeholder. Icons in the cells indicate that a Screen View has been defined for each activity for each role.

The Activities column currently displays five activities for a collaborative requirements negotiation work practice. The collaboration engineer can add activities on the fly, either by creating them from scratch, by dragging a technique from a palette (not shown) onto the activity list or by dragging a previously saved activity from a palette onto the activity list.

The editor currently displays two roles for this work practice – Analyst and Stakeholder. The collaboration engineer must configure a screen view for each activity for people in each role. The presence of an icon in a cell at the intersection of a role and an activity indicates that a screen view exists at that location. A collaboration engineer may add screen views to cells either by dragging a pre-configured screen from a palette (not shown) into a cell, or by starting from scratch with a Screen View Editor.

3.2.4. Screen View Editors

Using a screen view editor, a collaboration engineer decides which tools and controls should appear where on the screens of people in a particular role for a particular activity. All practitioners in the same role would see the same screen view in the PSS at runtime. Figure 3 shows the tool from Figure 1 embedded into a screen view for the Analyst role of a collaborative requirements negotiation application.

3.2.5. Tool Editors

A collaboration engineer would use a tool editor to assemble collaborative components into more sophisticated tools, and to configure their properties and events to fit the needs of the group for the activity at hand. Figure 4 shows the structure of a brainstorming tool similar to that shown in Figure 1 as it appears in the tool editor of the Cognito prototype CACE.

Default configurations of single components appear on the palette to the left. The collaboration engineer drags them from the palette on the left into the workspace on the right, assembling them into a tree-like structure. The collaboration engineer can then configure which events in a parent component would activate the appearance of a child component. The collaboration engineer can configure dozens of attributes and behaviors of each component instance.
to create the tool the practitioners need to execute the activity at run time in the PSS.

3.2.6. Documentation Engine

Collaboration engineers must devote considerable effort to documenting collaborative work practices and the software applications that support them. Such documentation may include, for example, practitioner scripts, participant instructions, agendas, presentation slides, online tutorials, on-screen participant instructions, training guides, cue cards, and quick reference guides, to name but a few. With so many documentation artifacts, when collaboration engineers decide to change the phrasing of some key piece of text, for example, the name of an activity, they must then go through every form of documentation in the project to find and replace instances of the old term with the new term. A documentation engine allows the collaboration engineer to create an object containing any piece of text that would be used more than once in the documentation, and then to automatically generate all documentation by applying rules to these text objects. An update to any object would then be reflected in all documentation where that object appears.

3.2.7. Summary of CACE Elements

The combination of technique editors, logical design editors, physical design editors, and a documentation engine provide collaboration engineers with the tools they need to support the design of collaborative work practices and, without having to write any code, to configure task-specific collaborative software applications that a group of practitioners without professional facilitation experience can use to move themselves through a series of activities toward their goals. The inputs to the CACE environment are the logical and physical design of the collaborative work practice. The outputs of the CACE are PSS software applications that support specific work practices, and documentation for the work practices and the PSS applications that support them.

3.3. Process Support Systems

The key components of the PSS are a library of PSS applications and a PSS client.

3.3.1. PSS Library

The PSS library presents users with the collection of PSS applications installed on their server and licensed for their use. A practitioner selects an application from the library and instantiates it as a shared workspace. A PSS application provides users with a sequence of collaborative activities for achieving their goals. For each activity, the application provides users with just the tools they need, configured exactly as needed for that activity. It provides users with the communication channels and data links they need to succeed for the activity. Also embedded within a PSS are all training materials that would be required so that users can learn and remember how to execute the work practice for themselves. Each activity provides simple instructions to participants in every role. For each activity, the application provides presentation slides and a moderator’s script for instantiating the collaboration techniques selected by the collaboration engineer in concert with practitioners at design time. The script documents everything the moderator should say and do to invoke desired patterns of collaboration to move the group through the activity. Each activity has an imbedded just-in-time tutorial so that people who have not yet mastered the work practice can learn how to conduct the activity. Each application contains a training guidebook, a collection of cue cards that a novice moderator can print out and use while leading an activity.

3.3.2. PSS Client

The PSS client provides the runtime capabilities required for running the PSS applications. The key elements of the PSS client are its collection of plug-in Collaborative Components and the Client Framework that controls them. The components in the PSS are the same set that can be configured in the CACE. Each component is coded separately and independently of the others and is plugged into the framework. Each component communicates only with the framework. Based on the configurations specified in the PSS Application, the framework instantiates components, positions them on the screen, and passes each of them its data, presentation, and event rules. Any messages from a component to and from the server are mediated by the framework. Any user-initiated mouse and keyboard event that the component does not recognize is passed to the framework. The framework checks the PSS application to determine whether the events require any action. Thus, the PSS client orchestrates the
collaborative components to make them behave as if they were a tightly-integrated, custom-built application instead of the uncoupled collection of separate applications they actually are.

4. Discussion

4.1. Feasibility of the CACE / PSS approach

We interviewed the commercial vendors of three leading GSS products and compared the times required to generate such systems from scratch to the time required to configure PSS applications. The vendors reported that a) a GSS with 12 general purpose tools required 37 person years to develop; b) a GSS with 7 tools required 25 person years to develop; and c) a GSS with 15 tools required more than 50 person-years to develop. We compared these results to the time required to develop PSS applications of comparable complexity.

Working with practitioners in industry, we used the Cognito CACE prototype to create six task-specific PSS applications for real collaborative tasks being executed by the practitioners in the course of their normally assigned duties. Table 1 lists the applications, along with the number of activities in each application, the number of hours required to develop each application, and the number of hours required to document the applications. Field results suggest that the CACE approach reduced collaborative application development time. The tools required for each PSS activity were comparable in complexity and functionality to tools in commercial GSS implementations. As shown in Table 1, configuration time for the PSS application was approximately four orders of magnitude less than that required to develop comparable GSS such tools from scratch. The time required to train practitioners to lead the work practices ranged from about 2 hours for the BackLog collaborative story management application to about 2 days for the EasyWinWin requirements negotiation application. This compares favorably to the year-long apprenticeship required for conventional facilitation training.

4.2. Future Directions

Early work with CACE and PSS suggest that the approach may be technically, economically, and operationally feasible. However, much scientific and engineering research remains to be done before the approach can be considered mature. The sections below outline some key future research directions.

4.2.1. CACE / PSS Standards

It will be useful for CACE / PSS researchers and other stakeholders to negotiate technical standards for several aspects of the approach. It would be useful, for example, to publish standards for a collaboration engineering modeling language (CEML) for PSS applications. That way, applications built by one researcher could be run on platforms developed by other researchers. This would allow for side-by-side comparisons of technical approaches for PSS implementations. It would also be useful to specify standard APIs for plugging collaborative components into CACE / PSS frameworks. That way, researchers could replace less effective components with more effective components without having to rebuild their PSS applications, and could extend the capabilities of CACE / PSS systems without having to rewrite existing code.

4.2.2. PSS Application Adoption and Diffusion

While proof of concept prototypes demonstrated that the CACE / PSS approach is feasible, more research will be required to demonstrate that users in the field will accept and continue to use collaborative applications developed and deployed using this approach. This will require the development of new systems that are sufficiently robust to be left behind.

Table 1. Six PSS applications Developed and Run with Industry Partners Using the Cognito PSS. Development of PSS applications was four orders of magnitude faster than conventional GSS development practices.

<table>
<thead>
<tr>
<th>Application</th>
<th>Purpose</th>
<th># Activities</th>
<th>Development Hrs.</th>
<th>Documentation Hrs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. EasyWinWin</td>
<td>Requirements Negotiation</td>
<td>15</td>
<td>37</td>
<td>260</td>
</tr>
<tr>
<td>2. Backlog</td>
<td>Scrum Story Management</td>
<td>3</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>3. FastAnswers</td>
<td>Marketing Focus Groups</td>
<td>5</td>
<td>12</td>
<td>107</td>
</tr>
<tr>
<td>4. FastRFP</td>
<td>Joint Proposal Authoring</td>
<td>8</td>
<td>13</td>
<td>42</td>
</tr>
<tr>
<td>5. Ignition</td>
<td>Creative Problem Solving</td>
<td>9</td>
<td>17</td>
<td>237</td>
</tr>
<tr>
<td>6. TeamWork</td>
<td>General Facilitation</td>
<td>25</td>
<td>78</td>
<td>274</td>
</tr>
</tbody>
</table>
in organizational settings to support people working on mission-critical high-value tasks.

4.2.3. Group Enabled CACE

Current CACE implementations are single-user user applications. Collaboration Engineering, however, is a multi-actor endeavor. Collaboration engineers must work with practitioners, participants, and other success-critical stakeholders throughout the Collaboration Engineering lifecycle to assure that the resulting PSS application accommodates their interests and constraints. In addition, a group-enabled CACE tool will allow collaboration engineers to share PSS designs as well as discuss best practices for the design of a PSS. It will be important, therefore, for future implementations of CACE to be group enabled, and for researchers to explore how best to support collaborative PSS application design. It may be possible to implement them on the same technical platform as the PSS.

4.2.4. Intelligent Agents in CACE and PSS

There are a number of ways that collaboration engineers could be assisted by intelligent agents during the design and configuration of PSS applications. It may be, for example, that an agent could reduce the cognitive load of selecting among the many available collaboration techniques to find one that would be useful for a particular activity. A collaboration engineer must weigh many criteria when selecting techniques. These criteria include the patterns created by the technique, the nature and structure of data that will be input to the technique, the nature and structure of the outputs the technique will produce, the cognitive load imposed on the moderator of the technique, the level of skill required of the moderator to successfully execute the technique, the cognitive load imposed on participants by the technique, the technique that preceded the current activity, and the techniques to be used in the activities that follow it. Researchers have experimented with selection advisor agents ([28]), with some success. Such agents could be added to the CACE.

It may also be useful to add intelligent agents to PSS applications, for example to help a moderator decide when it is time to move on to a new activity, to track when a group may be going off task, or to discover and cue up information relevant to an ongoing discussion. First steps towards intelligent collaboration support are based on context-adaptive shared workspaces [34]. These shared workspaces capture the current context of each group member and use rule-based approaches to adapt the collaboration environment to current requirements of the group. Future research could explore these possibilities more fully and help to develop intelligent agents for PSS applications. To develop these kind of intelligent agents we envision a path from initial decision making based on expert-algorithms to eventually self-learning systems.

4.2.5. CACE for Facilitators

The CACE/PSS approach has been initiated to overcome some limitations imposed on people who need to use collaboration technology effectively without professional facilitators. As such, current research in this area is focused on optimizing system to support the practitioners rather than to support the facilitators. Yet it is likely that at some point in the future it will be possible to develop a CACE/PSS interface that would be useful to facilitators who want more flexibility and integration than current tools provide. Although current CACE/PSS focuses on configuring tools in advance so practitioners do not have to choose and configure tools at run time, there is nothing inherent in the CACE/PSS concept that would preclude the development of run-time UIs for facilitators that would allow them to reconfigure their tools on the fly at run time. Thus, this stream of research could come full circle.

4.5.6. The Challenge of Documentation

Using CACE tools, collaboration engineers can now create collaborative software applications in hours that might have taken months or years using conventional development methods. Similar gains, however, have not yet been made in documenting the PSS applications produced in a CACE. Collaboration engineers must now spend ten times or longer documenting an application than they do developing it. This suggests that the challenge of documentation would be a worthy topic for future research.

5. Conclusions

This paper advances a combination of CACE and PSS technologies to overcome several limitation that block widespread use of sophisticated collaboration tools by practitioners who do not have the support of professional facilitators. Using CACE tools, collaboration engineers have demonstrated they can create large scale collaborative software applications in a fraction of the time that conventional software development methods would require. Users can learn
to execute these applications in far less time than would be required to gain conventional collaboration skills, and can attain results equivalent to those of professional facilitators. This represents resource savings for organizations: Organizations can hire collaboration engineers to design and implement new collaborative work practices once, while organizational users can execute these work practices multiple times. In other words, the recurring requirement of hiring or retaining a professional facilitator is replaced by a one-time requirement to task a collaboration engineer to design and deploy a collaborative work practice.

However, this outcome does not mean that professional facilitators will no longer be required. CACE and PSS offer solutions only for recurring tasks. Facilitators will still be required for ad-hoc and high-risk tasks. Further, only an expert facilitator with special skills in collaboration engineering will have the requisite skills to successfully design and deploy PSS applications. Rather than limiting the role of the facilitator, therefore, the CACE / PSS approach expands the facilitator’s role, into designing, piloting, training, and maintaining work practices executed by others. With CACE / PSS, facilitators will be able to create a legacy that lasts beyond the time they spend face to face with a group. CACE and PSS will not put a facilitator in the computer. It will, however, let them package certain aspects of their skills in a form that can be reused effectively by others.

Acknowledgement

This research was funded in part by DARPA, the Air Force Office of Scientific Research, and GroupSystems Corporation. We extend special thanks to Daniel D. Mittleman, Jay F. Nunamaker, Jr., Douglas R. Dean, and Thomas Gregory for their invaluable contributions to this work.

References


