CAMELOT: Technology Focused Testing of CSCW Applications

Robert F. Dugan Jr., Ephraim P. Glinert, Edwin H. Rogers
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180, U.S.A.
{dugan, glinert, rogerseh}@cs.rpi.edu

ABSTRACT
In this paper we describe CAMELOT, a novel technology-focused methodology for testing collaborative software that contrasts with existing broad-based CSCW evaluation approaches. CAMELOT is intended for use by application developers, user interface specialists, performance engineers, and quality assurance personnel. The evaluation of a CSCW application is divided into two stages: single user and multi-user. The single user stage is subdivided into general computing and human-computer interaction testing. The multi-user stage is decomposed into distributed computing and human-human interaction testing. The methodology provides a detailed, codified, checklist of testing techniques for each stage. We applied CAMELOT to a conventionally tested, mature CSCW application. Our techniques uncovered and classified over two dozen problems with the system.

Keywords
Computer Supported Cooperative Work, CSCW, Testing, Distributed Computing, Human Computer Interaction, Evaluation

1 INTRODUCTION

Evaluating CSCW software is a daunting task. In addition to a host of issues like functional and usability testing that are relevant to any software system, the CSCW evaluator must also consider problems such as scalability, synchronization, and race conditions given the application's distributed nature, as well increased usability complexity when the application becomes a vehicle for interaction between users. Industry has provided execution based testing tools for general application software. However, little guidance is offered for effective use of these tools with a CSCW application. In addition, there is no support for testing sophisticated interaction among users which lies at the heart of CSCW [6].

Research into CSCW evaluation has been broad based, advocating the examination of both social and technological aspects of an application [1, 3, 4, 15, 17]. These broad based approaches combined with the research community's preference for social evaluation have created a lack of specific techniques for the technological evaluation of CSCW software.

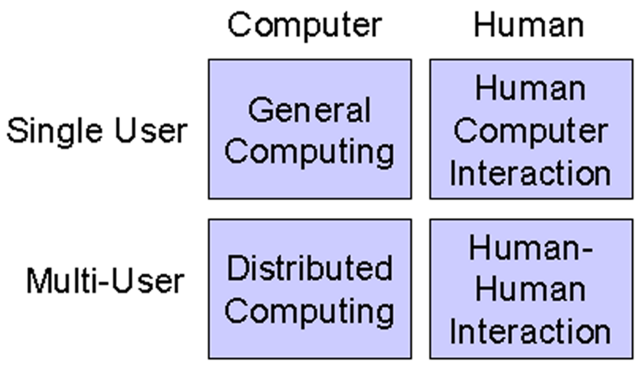

Figure 1: Intersecting CSCW Technologies

In this paper, we present CAMELOT, a technology-focused methodology for evaluating collaborative software. In contrast to existing techniques, our approach has a deliberate technological focus and can be combined with CSCW enabled testing tools, such as Rebecca-J [6], for an effective evaluation. CAMELOT is intended for use by application developers, user interface specialists, performance engineers, and quality assurance personnel.

CAMELOT decomposes a CSCW application into four intersecting software technologies (see Figure 1): General Computing, Human Computer Interaction, Distributed Computing, and Human-Human Interaction. Techniques derived from the literature are enumerated for each technology. Each technique has a unique label that can be used to classify tests and problems when using CAMELOT to evaluate an application.

- General Computing describes software components that provide general application capabilities. In its most primitive form, this describes a Turing Machine that takes input, performs operations on the input, and produces output. All software technology falls under this broad category.

- Human Computer Interaction describes components that deal with the interface between the user and the software system. These components include: processing user input from voice, mouse, joystick, and keyboard; graphical interfaces like windows, menu bars, push buttons, and text fields; processing application output like audio, video, and graphics.

- Distributed Computing describes components that are responsible for multitasking and multiprocessing in the application at the thread, process, processor, and machine levels. The main focus of distributed computing in the CSCW domain is the management of objects shared across users.

- Human-Human Interaction describes components that facilitate interaction between users during application use. Examples include floor control, session management, and shared windowing.

CAMELOT is applied in two stages: single user followed by multi-user. In the single user stage, the evaluation focuses on the single user problems in the application. For the most part, these are described by general computing and human computer interaction techniques. Distributed computing and human-human interaction techniques are used to uncover flaws in the multi-user stage. The intersecting nature of single and multi-user technologies may cause the techniques from one to trigger the development of tests or discovery of problems in another.

A unique code is associated with each test category. The code provides a classification scheme for the tests used and problems uncovered during application evaluation. We believe CAMELOT's techniques are inclusive of most of the technology tests an evaluator would want to perform on a CSCW application. As new technologies are introduced, however, we expect the list to expand.

CAMELOT provides a detailed set of techniques for detecting problems in CSCW software. Our approach is not algorithmic and cannot be fully automated. In order to guarantee that a program operates correctly, an automated test system would have to try every possible combination of input values or execution paths. Researchers have been unable to identify a computationally feasible approach to automated testing [11]. Like other intractable problems in computer science, practical testing approaches use heuristics to reduce the number of tests that must be performed. As with any heuristic, practical testing approaches like CAMELOT cannot guarantee that all application problems will be found.

2 Single User Evaluation

In contrast to previous CSCW evaluation approaches, CAMELOT has a deliberate technological focus. The first stage in our evaluation process views the CSCW application from the perspective of a single user. There are two types of single user evaluation: General Computing and Human Computer Interaction. General Computing encompasses testing techniques that can be used with any kind of software application. Human Computer Interaction techniques concentrate on identifying problems with application's user interface. Single user tests are simpler to create, execute and analyze than multi-user tests. The insights gained during this stage can be used later in the evaluation. For example, the identification of shared objects used in the application can be used for subsequent race condition and synchronization tests. As another example, the single user performance of an application function can give an indication of how that function will scale.

General Computing

Decades of research have gone into the discipline of software testing. A survey of this work was conducted prior to developing CAMELOT [6]. Testing during the software lifecycle is a process by which the behavioral properties of the software are verified. There is little evidence that testing methodologies that verify the system at the requirements, specification, or design stages are used outside academia. The extraordinary amount of effort required by these testing methods, even for small software systems, is unattractive to the commercial software community.

Taking its cue from difficulties with early life cycle testing, CAMELOT focuses on execution based testing of software. The structure of CAMELOT's general testing methodology comes from Meyer's classic work "The Art of Software Testing" [11]. The book presents a common sense approach to verification of software systems that has stood the test of time in both the commercial and academic communities. The techniques listed in table 1 are used in later stages of the software life cycle.

Performance tests are particularly critical for CSCW applications. There are two kinds of performance issues in CSCW systems: single user and multi-user. For general testing, the evaluator should focus on the response time, and resource utilization of single user scenarios. Multi-user performance will be discussed in detail in Section 3.

Although the techniques listed in Table 1 are organized by life cycle stage, the tests can be performed at any point in the cycle. For example, a security test might be performed during the implementation phase to prototype an application's security features.

Human Computer Interaction

A great deal of work by the academic and commercial communities has focused on testing human computer interaction [6]. These efforts are concentrated in two main areas: general computing and usability.

CAMELOT Code	**Development Cycle**	**Technique**
	Implementation	
GC.IM.1		Functional Test[1]
	Integration	
GC.IN.1		Bottom Up[2]
GC.IN.2		Top Down[2]
GC.IN.3		Sandwich[2]
	System Test	
GC.ST.1		Facility Test[1]
GC.ST.2		Volume Test[1]
GC.ST.3		Stress Test[1]
GC.ST.4		Security Test[1]
GC.ST.5		Performance Test[1]
GC.ST.6		Configuration Test[1]
GC.ST.7		Memory Test[1]
GC.ST.8		Compatibility Conversion Test[1]
GC.ST.9		Install Test[1]
GC.ST.10		Recovery Test[1]
GC.ST.11		Documentation Test[1]
GC.ST.12		Procedure Test[1]
GC.ST.13		Acceptance Test[1]

Table 1: General Computing Techniques from [1][11] and [2][18]

General Computing ∪ Human Computer Interaction Techniques
General computing intersects human computer interaction defining the correctness of the user interface as ”proper behavior of the graphical user interface and proper computation of the underlying application.” [14] A general computing approach to human computer interaction testing exercises the application using the techniques listed in Table 1.

Yip [24] and Schneiderman [19] provide some additional techniques. Automated record/playback tools [6, 10, 13, 20] allow the evaluator to create regression tests that ensure the stability of a new code release.

CAMELOT Code	**Technique**
GC/HCI.1	Missing,invisible, unreachable components[1] derived from: (GC.IN.1∪HCI) ⇒ GC/HCI.1
GC.HCI.2	Failure to respond to user inputs[1] derived from: (GC.IN.1∪HCI) ⇒ GC.HCI.2
GC/HCI.3	Cross-wired components (e.g. button press displays wrong component)[1] derived from: (GC.IN.1∪HCI) ⇒ GC/HCI.3
GC/HCI.4	Incompleteness (e.g. close box present in some windows, but not others)[1] derived from: (GC.ST.13∪HCI) ⇒ GC/HCI.4
GC/HCI.5	Response time[2] derived from: (GC.ST.5∪HCI) ⇒ GC/HCI.5

Table 2: General Computing ∪ Human Computer Interaction Techniques from [1][24], [2][19]

General computing tests of user interfaces suffer from a combinatorial explosion of test cases due to the number of different paths a tester can take to exercise the same application function [23]. Like black and white box tests, CAMELOT’s approach to UI path testing requires evaluator judgment. Path tests should be conducted where the evaluator believes they will be the most fruitful in uncovering application flaws.

Usability Techniques
Usability testing evaluates a software application from the user’s perspective. The correctness of an application is measured in terms of the user’s effectiveness and feelings about the application, rather than a general computing standpoint. Over the past two decades, Schneiderman has produced and revised a thorough survey of user interface development techniques [19]. CAMELOT’s usability techniques, shown in Table 3, are taken from this survey.

Usability criteria represent a general set of questions the evaluator should ask about a user’s use of the application. The *Golden Rules for Application Design* are eight guidelines for the design of any application with a user interface. *User Interface Technology Guidelines* is a list of specific techniques organized by the user interface technology. Rather than repeating the guideline specifics here the reader is referred to the original text for more detail [19].

CAMELOT Code	**Technique**
	Usability Criteria
HCI.UC.1	Time to learn system: How long does it take for a typical user to learn to use the system?
HCI.UC.2	Performance of tasks: How long does it take for a user to perform a typical set of tasks?
HCI.UC.3	User errors: How many and what kind occur while performing a typical set of tasks?
HCI.UC.4	Retention over time: Is it easy to remember how to use the system with infrequent use?
HCI.UC.5	Subjective satisfaction: Do users like the system?
	Golden Rules for Application Design
HCI.GR.1	Strive for consistency.
HCI.GR.2	Enable frequent users to use shortcuts.
HCI.GR.3	Offer informative feedback.
HCI.GR.4	Design dialogs to yield closure.
HCI.GR.5	Offer simple error handling.
HCI.GR.6	Permit easy reversal of actions.
HCI.GR.7	Support internal locus of control.
HCI.GR.8	Reduce short-term memory load.
	User Interface Technology Guidelines
HCI.UITG.1	Data Display
HCI.UITG.2	Getting the User’s Attention
HCI.UITG.3	Data Entry
HCI.UITG.4	Menu Selection
HCI.UITG.5	Form Fillin Design
HCI.UITG.6	Command Languages
HCI.UITG.7	Direct Manipulation
HCI.UITG.8	Interaction Devices
HCI.UITG.9	Error Messages
HCI.UITG.10	Color

Table 3: Usability Techniques from [19]

3 Multi-User Evaluation

The second stage in CAMELOT’s evaluation process approaches the CSCW application from a multi-user perspective. There are two types of multi-user evaluation: Distributed Computing and Human-Human Interaction. Distributed Computing focuses on the multi-thread, task, processor, and machine challenges that occur in CSCW applications. Human-Human Interaction concentrates on testing the software components that facilitate interaction between users.

Distributed Computing

Distributed computing encompasses software written for multithreaded, multitasking, multiprocessor, or multimachine architectures. The technology is concerned with communication between one or more routines executing in parallel. Communication consists primarily of requests for/updates about some form of shared data. Distributed computing software suffers from four common problems: race conditions, deadlock, temporal consistency and scalability.

Race Condition
When two or more routines executing in parallel are allowed to simultaneously manipulate the same data instance without proper control it is called a race condition. Lack of controlled access to shared data may result in data corruption. A classic illustration of this is the bank account withdrawal example from database literature [7].

Deadlock
Synchronization eliminates race conditions by restricting access to shared data in a controlled manner using synchronization primitives such as mutual exclusion, semaphores, or message passing [22]. Synchronization introduces the potential for deadlock. Deadlock can occur when two or more parallel routines share two or more synchronization primitives. Deadlock can be avoided through careful software design. Like race conditions, detecting deadlock is notoriously difficult because of subtle timing dependencies. It is also difficult to debug because of complicated dependencies between parallel routines and synchronization primitives.

Temporal Consistency
Temporal consistency is the ability to correctly order messages within the CSCW application. Temporal consistency is especially important when providing communication, feedback for the manipulation of shared objects, and user awareness. For example, consider a shared editing system with three users. userA types the word ”dessertation”. userB corrects the word by moving the cursor after the first ’e’ and changing it to ’i’. Because of a network delay between userA and userC, userB’s corrections to the word arrive at userC before the actual word arrives. Testing for temporal consistency problems involve techniques similar to those used for race conditions and deadlock. Network delay can be introduced by artificially consuming bandwidth, or by instrumenting the application to introduce artificial message delays.

Scalability
Scalability is also an important consideration in distributed computing. A system’s ability to scale as the number of users is increased measured using performance evaluation techniques. Although these techniques can be described generally, the actual evaluation is application specific. Jain’s well-known text ”The Art of Performance Evaluation” presents a general approach for most applications [9].

The key to the performance evaluation of CSCW applications is a thorough understanding of the application’s architecture and intended use. This understanding will reveal services that are candidates for scalability testing. Creating user scenarios that represent common user activity and then running these scenarios on the system using live or virtual users will place the system under a ”typical” load.

CSCW Architecture
The distributed architectures of CSCW systems fall between two extremes: centralized and decentralized. A centralized architecture concentrates the shared state in a single process on a single machine. When a process in the system manipulates shared data, it makes a request to the shared state process. Centralization simplifies access control for shared data by placing synchronization logic in a single process. Scalability problems can occur as an increasing number of users compete for the attention of the single state process.

A decentralized architecture replicates shared state within each user process. A process manipulates shared data locally and the results of the manipulation are broadcast to other processes. Decentralization has scalability advantages because the cost of data manipulation is distributed across many processes. Shared data access control, however, is more challenging because the synchronization primitives must also be decentralized.

Tightly coupled systems provide near instantaneous notification to all processes when shared data changes. Loosely coupled systems do not have strict temporal requirements. Tightly coupled systems have to be examined closely for scalability problems. The two areas to investigate are the frequency and size of the messages necessary to maintain the coupling. As the number of users in the system increases, the communication necessary for state change notification will also rise. At a certain point this communication will consume all available network bandwidth.

Another area to investigate is the impact of network delay on tightly coupled systems. In a typical development environment, there is almost no network delay because the equipment used to develop the system is on the same LAN. If the CSCW application is intended to deploy on the Internet across LANs, WANs, and backbones, then the application should be tested with network delays. Network delays can create untested timing configurations that trigger race conditions and deadlock. Network delays can be inexpensively simulated on a LAN by reducing bandwidth (downloading a large file on the LAN during a test) or by instrumenting the application with built in messaging delays.

Loosely coupled systems can also suffer from race conditions. In a loosely coupled system, a shared object is manipulated locally. Updates to the object are sent to the rest of the system intermittently, perhaps as the result of a save, refresh, or update command. The race condition occurs when two users manipulate the same object simultaneously. Typically, the system view will reflect the last user update of the shared object. An example of this is loosely coupled editing of a text document. If userA and userB are editing the same document, then one of the user's edits will be lost. The system will only retain the document state from the last user's save command overwriting previous user saves.

Distributed Computing Techniques
It is critical that the evaluator have a deep understanding of the system's architecture to test for race condition, deadlock, and scalability problems. In particular, the evaluator should understand the types of shared data in the system, the architecture that maintains the data, and user actions that trigger manipulation of the data.

Table 4 reduces this section's distributed computing discussion to a codified table.

CAMELOT Code	Technique
	Race Conditions
DC.RC.1	Race Condition
DC.RC.2	Centralized Architecture
DC.RC.3	Decentralized Architecture
DC.RC.4	Loosely Coupled
	Deadlock
DC.D.1	Deadlock
DC.D.2	Centralized Architecture
DC.D.3	Decentralized Architecture
	Temporal Consistency
DC.TC.1	Temporal Consistency
DC.TC.2	Network Delay
	Scalability
DC.S.1	Scalability
DC.S.2	User Scenario
DC.S.3	Stress User Scenario
DC.S.4	Centralized Architecture
DC.S.5	Decentralized Architecture
DC.S.6	Tightly Coupled
DC.S.7	Tightly Coupled/Network Delay
DC.S.8	Loosely Coupled
DC.S.9	Synchronization

Table 4: Distributed Computing Techniques

In addition to pure distributed computing, Table 5 introduces techniques resulting from the intersection with General Computing.

Table 6 presents techniques resulting from the intersection of Human Computer Interaction and Distributed Computing:

Human-Human Interaction
Human-Human Interaction deals with functionality supporting interaction between application users. Much of this is social, and research has focused on studying the social aspects of CSCW systems [15]. As mentioned earlier, CAMELOT does not focus on higher levels of social interaction. However, there are core CSCW technologies supporting human-human interaction that CAMELOT can be used to evaluate.

CAMELOT Code	Technique
GC/DC.1	*Stress testing*: users on joining/leaving the application Derived from: (GC.ST.3 ∪ DC) ⇒ GC/DC.1
GC/DC.2	*Stress testing*: multiuser user stress tests on shared objects Derived from: (GC.ST.3 ∪ DC) ⇒ GC/DC.2
GC/DC.3	*Volume testing*: Large shared objects consume resources Derived from: (GC.ST.2 ∪ DC) ⇒ GC/DC.3
GC/DC.4	*Compatibility testing*: incompatible versions of application Derived from: (GC.ST.8 ∪ DC) ⇒ GC/DC.4
GC/DC.5	*Subclass of distributed compatibility testing*: different versions on-line documentation. Derived from: (GC.ST.8 ∪ GC.ST.11 ∪ DC) ⇒ GC/DC.5
GC/DC.6	*Recovery testing*: unexpected joining/leaving application Derived from: (GC.ST.10 ∪ DC) ⇒GC/DC.6

Table 5: General Computing ∪ Distributed Computing Techniques

CAMELOT Code	Technique
HCI/DC.1	*Race condition testing*: for multithreaded GUIs. Derived from: (HCI ∪ DC.RC.1) ⇒ HCI/DC.1
HCI/DC.2	*Deadlock testing*: for multithreaded GUIs. Derived from: (HCI ∪ DC.D.1) ⇒ HCI/DC.2
GC/HCI/DC.1	*Response time testing*: for tightly coupled GUI components. Derived from: ((GC.ST.5∪HCI).1 ∪ DC.S.7)) ⇒ GC/HCI/DC.1

Table 6: Human Computer Interaction ∪ Distributed Computing Techniques

These technologies are the software components that facilitate communication, coordination, coupling, privacy, user awareness, and scalability.

Communication allows one user to converse with one or more users in the application. Communication can be in the form of voice, visual, text, or gesture. Unless users share the same location, the intersection between distributed computing and human-human interaction is critical. Some form of network will be responsible for transportation of user communications. In the case of high bandwidth communication such as voice or visual, the tester should ensure that there is enough network capacity as more users are added to a CSCW session This is particularly important if the application was developed in a lab with a high speed LAN but is to be deployed across multiple LANs, WANs, or the Internet. The impact of bandwidth consumption from user communication on the rest of the application should also be studied. Revealing tests will be ones that exercise tightly coupled function (such as remote cursor movement) during user communication.

Coordination of interaction focuses on how the software allows users to work together. Examples of coordination include floor control policies and social protocols. From a technology standpoint, coordination can be broken down into components that provide group control and feedback about that control within the application. Human-computer interaction evaluation of these components is necessary. Data associated with coordination can also be considered a form of shared object, thus distributed computing evaluation is also necessary. For example, the ”floor” can be considered a shared object. What happens if two users try to grab control of the floor at the same time?

Coupling defines how users see changes that others make to the shared workspace. Tight coupling provides more frequent change updates; loose coupling provides less frequent updates. There is no single ”correct” coupling for CSCW. What kind of coupling should be used varies from application to application, and even within a single application [2]. Human computer interaction response time tests and distributed computing scalability tests are useful with this technology.

Security, privacy and trust are important to cooperating users. Users should be able to work in a private area where they feel confident that their activities are protected from others. Access control for individual or group information should be available to users [21]. In situations where anonymous input is supported, users should feel assured of their anonymity [12]. General computing security tests help evaluate these issues.

Awareness of other users provides a social context in which work is conducted. Many kinds of user awareness capabilities that have been added to CSCW applications including activity graphs, telepointers and cursors, user lists, multiuser scrollbars, radar views, and fisheye views [8]. Given the information richness of some forms of user awareness, the evaluator should pay particular performance and scalability issues.

Tests that examine combinations of technologies discussed in this section may also be necessary. Are coordination mechanisms available in the application to control communication? For example, can two users talk at the same time? How tightly coupled is the act of communication to its delivery? If users have an expectation of instantaneous communication, what is the impact of network delays? If the system supports private or anonymous communication, can it be subverted? When communication occurs, can the user determine whom it came from?

4 Evaluation

Despite acknowledging a technological aspect to CSCW, existing methodologies provide little guidance to applications developers, user interface specialists, performance engineers and quality assurance personnel. CAMELOT provides this guidance by organizing a technical evaluation into two stages and four intersecting technologies and providing detailed techniques for each. In this section we discuss the steps involved in an evaluation using CAMELOT and demonstrate its applicability on a mature CSCW application.

Organizing a CAMELOT Evaluation

Application evaluation using CAMELOT should proceed in the following manner. Ordering of the techniques in each technology category is not important, but the order that the categories are used in an evaluation is critical.

CAMELOT Code	**Technique**
Communication	
HHI.CM.1	Network bandwidth sufficient to support user communication.
HHI.CM.2	Impact of user communication on other communication in the application.
HHI.CM.3	Impact of user communication on tightly coupled functions.
DC/HHI.1	Distributed computing scalability tests. Derived from: (DC.S.1 ∪ HHI.CM) ⇒ DC/HHI.1
DC/HHI.2	Distributed computing temporal consistency tests. Derived from: (DC.TC.1 ∪ HHI.CM) ⇒ DC/HHI.2
HHI.1	User communication and coordination. Derived from: (HHI.CM ∪ HHI.CD) ⇒ HHI.1
HHI.2	User communication and coupling. Derived from: (HHI.CM ∪ HHI.CP) ⇒ HHI.2
HHI.3	User communication and security. Derived from: (HHI.CM ∪ HHI.S) ⇒ HHI.3
Coordination	
HCI/HHI.2	Human computer interaction issues related to group control. Derived from: (HCI ∪ HHI.CD) ⇒ HCI/HHI.2
DC/HHI.3	Distributed computing race condition and deadlock tests for coordination shared objects. Derived from: (DC.RC.1 ∪ DC.D.1 ∪ HHI.CD) ⇒ DC/HHI.3
Coupling	
GC/HCI/HHI.1	Human computer interaction response time tests. Derived from: (GC/HCI.5 ∪ HHI.CP) ⇒ GC/HCI/HHI.1
DC/HHI.4	Distributed computing scalability tests. Derived from: (DC.S ∪ HHI.CP) ⇒ DC/HHI.1
DC/HHI.5	Distributed computing temporal consistency tests. Derived from: (DC.TC ∪ HHI.CP) ⇒ DC/HHI.2
Security	
GC/HHI.1	General computing security tests. Derived from: (GC.ST.4 ∪ HHI.S) ⇒ GC/HHI.1
Awareness	
GC/HHI.2	General computing performance tests. Derived from: (GC.ST.5 ∪ GC/HCI.5 ∪ HHI.A) ⇒ GC/HCI/HHI.2
GC/HCI/HHI.2	Human computer interaction response time tests. Derived from: GC/HCI.5 ∪ HHI.A) ⇒ GC/HCI/HHI.2
DC/HHI.5	Distributed computing scalability tests. Derived from: (DC.S ∪ HHI.CP) ⇒ DC/HHI.5
DC/HHI.6	Distributed computing temporal consistency tests. Derived from: (DC.TC ∪ HHI.CP) ⇒ DC/HHI.6

Table 7: Human-Human Interaction Techniques

The application should be examined from a single user perspective first because multiuser problems are more difficult to detect. This will familiarize the evaluator with application function, architecture, and user interface before tackling more complicated testing issues associated with distributed computing and human-human interaction. Within the single user stage, general computing tests should be performed before investigating human computer interaction. This will familiarize the evaluator with application functionality and provide a context for the user interface. The stress tests and general performance of the application during the single user tests will give the evaluator valuable insight into the application’s ability to scale with multiple users.

During the multiuser testing stage, distributed computing problems should be investigated first to provide context for later human-human interaction testing. As mentioned earlier, the intersecting nature of single and multi-user technologies may cause the techniques from one to trigger the development of tests or discovery of problems in another.

The Reconfigurable Collaboration Network

CAMELOT was applied to a conventionally tested, mature CSCW application to determine the methodology’s efficacy.

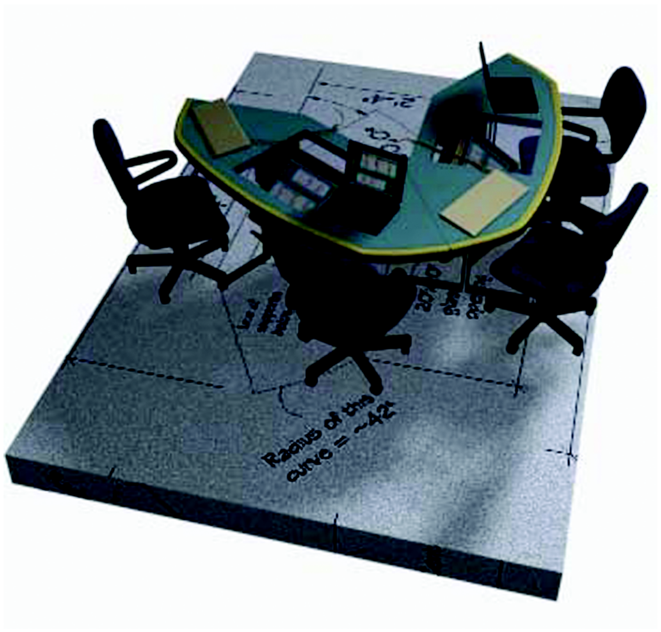

Figure 2: The Reconfigurable Collaboration Network

The Reconfigurable Collaboration Network (RCN) was developed as part of the Collaborative Classroom research effort at Rensselaer Polytechnic Institute [16]. The goal of the research was to develop a classroom where the learning came from group participation rather than lecture. The classroom design consisted of a unique combination of hardware, software, and physical architecture to promote group activity [5].

System Overview
Each RCN user has a private computer, keyboard, mouse, and display. The private keyboard and mouse are used to remotely control a public machine. Each user is required to have an additional, direct hardware connection to the public machine's display. Figure 2 depicts an RCN configuration with two private laptop systems and a shared public machine display below table's surface. Users are organized into teams. Each team has descriptive information and one or more administrators that control who is on the team. The first time a user selects a public machine a session is created. Additional users (registered or guests), may join or leave a session at any time. Multiple public machines are supported with one public machine per session. Finally the concept of a super session is supported. This allows users from different sessions to join together in a single meta-session for shared control of a public machine.

RCN differs from traditional remote windowing systems in several respects. Foremost, because users can see the public machine's display, there is no need for remote viewing capability. Second, unlike other remote windowing systems, the RCN views the public machine as a shared resource. Session and floor control functionality are included for management of the public machine during meetings.

The RCN system was implemented almost entirely in Java supporting multiple platforms including: Windows 95/98/NT, MacOS, and Linux. The architecture consists of three core components: ISServer, RCNPublicServer, and rcnClient. ISServer is responsible for session management. It keeps track of all active publics, sessions, teams, and users. It also maintains persistent store information about teams and users. RCNPublicServers register with the ISServer to advertise their availability to users. rcnClients must locate an ISServer to register as an active user and to find publics, sessions, teams and other users.

An RCNPublicServer runs on each public machine. It is responsible for receiving remote mouse and keyboard events from rcnClients and translating them to local events. If user selects ghosting, the software translates remote mouse events into move commands for a ghost icon associated with the user.

An rcnClient runs on each user's machine. The rcnClient presents the user with an array of session management functionality. Session management commands are sent from the rcnClient to a remote ISServer. When the user joins a session and selects the Interrupt button, his or her mouse and keyboard events are sent to a remote RCNPublicServer. Only one session member at a time can have this control. However, if any session member presses the ghost button, his or her mouse events are sent to the remote public machine.

CAMELOT and RCN
RCN was fairly mature at the time we evaluated it using CAMELOT. The application had been in development for over two and a half years, and there were plans to commercialize the system. During the school semester, the software was used daily by students in several courses. The development team was reasonably confident of the stability of the system. One team member suggested it might be necessary to deliberately introduce bugs into the software for the evaluation.

CAMELOT provides techniques for the tests that should be performed on a CSCW application. We used Rebecca-J, a Java-based implementation of our CSCW testing architecture to execute the tests [6]. Using these tools, two-dozen problems were discovered with the RCN system (see Table 8). Some of the problems were serious enough to jeopardize the planned commercialization of the software. This section discusses how the problems were uncovered using CAMELOT. Our detailed evaluation of RCN is beyond the scope of this paper, and the reader is referred to a separate document [6]. Instead, we summarize the problems uncovered using CAMELOT, and highlight several interesting ones.

Single User Tests
Single user testing focused on the General Computing and Human Computer Interaction aspects of RCN's three main components: ISServer, RCNPublicServer, and rcnClient.

Id	Bug Description	CAMELOT Code
A.1	Error message displayed when starting up RCNPublicServer	GC.ST.9, HCI.GR.3
A.2	Configuration of PATH shell variable necessary for NativeLibrary.dll for RCNPublicServer in Win95/98	GC.ST.9, HCI.GR.3
A.3	ISServer does not always flush terminated RCNPublicServer	GC.ST.9, DC.TC.1
A.4	Documentation Errors	GC.ST.11
A.5	Inconsistent use of Quit, Exit, Leave, Cancel	HCI.GR.1
A.6	”Pick a IS” is grammatically incorrect.	HCI.GR.1
A.7	No version number displayed in RCNPublicServer, rcnClient, ISServer	GC.ST.9, GC/DC.4
A.8	Preference Dialog Displays Invalid Colors	HCI.GR.7
A.9	Preference Dialog Displays Too Many Colors	HCI.UITG.10
A.10	Preference Dialog Allows Same Color for Two Users in Same Session	HHI.A, HCI.UITG.10
A.11	No lock mechanism for simultaneous edits of Team Information	DC.RC.4
A.12	Race Condition Joining a Session	DC.RC.2, DC.RC.3
A.13	Ghost Cursor Hidden By New Applications	HCI.UITG.7
A.14	Sticky Mouse Buttons	GC.IM.1, DC.RC.2
A.15	Multiple Client Control of Public Machine	GC.IM.1
A.16	Incorrectly Translated Keys	GC.IM.1
A.17	Sticky Shift, Alt, and Ctrl Keys	GC.IM.1, DC.RC.2
A.18	Race Condition in rcnClient’s User Interface	HCI/DC.1
A.19	Race Conditions Joining Sessions, Users, Teams, Publics	GC/HCI/DC.1
A.20	Inconsistent use of OK, Okay	HCI.GR.1
A.21	Flickering Ghost Cursor	DC.S.2, HCI.UITG.7
A.22	Confusing Display of Session Clients	HCI.UITG.1, HHI.A
A.23	Memory Leaks in Public and Client When Ghosting	DC.S.2, GC.ST.7
A.24	Can’t play Indiana Jones from rcnClient	GC.IM.1

Table 8: Bugs discovered in RCN using CAMELOT

These tests were not concerned with distributed or multiuser computing issues, although they were occasionally revealed.

General Computing Tests

The first test conducted investigated problems users might encounter installing and operating the RCN system for the first time. Several were uncovered (see entries A.1, A.2, A.3 in Table 8).

Functional testing of RCN focused on validating the system’s core capabilities. In particular, RCN’s ability to provide keyboard and mouse input to a public machine from a remote client was examined. This led to the discovery of incorrect numeric keypad translation between the client and public machine (A.16), and the discovery that if one client changed the public keyboard state (e.g. CAPS_LOCK), that state would unexpectedly carry over to other clients (A.17).

Experiences with keyboard functional testing were used to develop mouse tests. To test for mouse state problems, Rebecca-J was used to make a recording of an RCN client’s mouse movement with the left mouse button pressed (i.e. Java MOUSE_DRAGGED events) while controlling a public machine. A ”sticky” experiment with mouse events was attempted using Rebecca-J’s triggering facility [6], with results similar to the keyboard tests (A.14). In addition to verifying the state problem, as long as the mouse button from one client stayed pressed down, both additional clients could also control the public machine (A.15).

Other manual functional tests were conducted to observe how well an RCN client could control a representative set of applications on the public machine. Additional problems were detected using this technique (A.24).

During the general computing analysis, several other CAMELOT tests were conducted including: stress, documentation, compatibility, and volume. Stress testing was applied to mouse and keyboard control of the public, and interaction with RCN client’s user interface revealing a multithreaded user interface problem (A.18). Documentation testing examined RCN’s online help systems. A large number of errors were uncovered and reported (A.4). Compatibility testing considered problems that might occur between different versions of RCN’s client, public, and ISServer. With many different public machines, client machines, and ISServers it seemed likely that versions could get out of synch and there was no way for the user to determine the version of an RCN component (A.7). Volume testing investigated how the RCN application handled large data volumes. Some of the client’s user interface form fill-in fields were selected for the test and no problems were observed.

Human Computer Interaction Tests

After completing the general computing tests, a thorough examination of RCN’s human-computer interaction was conducted. User interaction with RCN takes place through a series of dialogs triggered from a central panel. Each dialog was exercised and Schneiderman’s rules for dialog design were applied uncovering several problems (A.5,A.6, A.20).

During dialog design testing, a problem with rcnClient’s user preferences dialog was detected. The dialog allowed each user to select a ghost color preference. This selection determined the color of text associated with the user (e.g. the user’s id in a team or session panel), and of the icon displayed when the user was ghosting on the public machine. Several problems were discovered with the system’s color assignment mechanism (A.8, A.10).

These color problems caused the development of a human-human interaction test: Can two users share the same or similar color? A test was created where a user in the same session selected the same color as another user from the preference dialog. RCN provided no warning that the color was already being used. Shared colors could be confusing, particularly during simultaneous ghosting (A.10).

Multiuser Tests
Multiuser testing focused on the Distributed Computing and Human-Human Interaction aspects of the RCN application. These tests were not concerned with general computing or human-computer interaction issues covered during single user testing.

Distributed Computing Tests

Multiuser race condition testing looked for problems with several users sharing access to the same data object. The first step in a race condition test was to identify shared data objects in the application. For RCN these objects were: personal information, team information, list of sessions, list of users, list of teams, and a list of public machines. The second step was to create a scenario that would likely trigger a race condition with the object. Simultaneous read/write or write/write operations on a shared object provide fruitful scenarios. The third step was to use Rebecca-J to record and instrument the scenario. Finally, Rebecca-J was used to repeatedly exercise the scenario in an attempt to trigger a race condition.

The first race condition test examined the Session List object. After logging into RCN, a user joined or created session. To join a session, the user selected a session from the Session List object displayed in the Sessions pane of the RCN client's Join A Session window. The ISServer maintained a "golden" copy of the object protected from race conditions by synchronization primitives. This eliminated the possibility of a race condition occurring in the ISServer. Unlike previous objects, however, clients maintained a local copy of the Session List that was updated from the ISServer whenever change occurred. This raised the possibility of a race condition if a client manipulated a local copy of the object after it was changed on the ISServer but before change notification was received (A.12). Similar tests were conducted on the other list objects shared by RCN clients: users, teams, and public machines resulting in race conditions (A.19).

Other race condition tests examined manipulation of the personal and team information objects. No race conditions or deadlock were uncovered. Further considration of RCN's editing model, however, uncovered a problem. If two clients were simultaneously editing an object, then one saving the object first would have his/her edits overwritten by the other client (A.11).

Scalability testing focused on the performance of the RCN application as the number of users increased. The primary scalability concern was multi-client ghosting on a single public machine. Because of tight coupling between the client mouse and the public ghost icon, users would not tolerate high response times. While other clients were ghosting, one client was used to take control of the public and the same qualitative observations were made. The test revealed scalability problems with just two ghosting clients (A.21). Memory leaks on both the public and client machines were also discovered (A.23). After some investigating, it was also determined that any application window created after a client began ghosting would hide the client's ghost cursor (A.13).

Human-Human Interaction Tests

Although RCN is a CSCW application, most human-human interaction is supported directly by the architecture of the physical space. Users can interact using voice and gesture because they are located in the same physical space. Only one user is allowed control of the public machine (and its system cursor) at any time. RCN software technology supports limited human-human interaction in the form of a ghost cursor. This cursor appears on the public machine, but cannot interact with any applications. Several ghost cursors can appear on the public display, manipulated by users that are not in control of the public machine. The human-human interaction tests performed on ghost cursors were covered under scalability testing.

Nine virtual users were configured with Rebecca-J to be used during the scalability test. The users were labeled client1, client2, etc. The User List pane didn't list these clients in any particular order. For a large number of clients, it might be difficult for a specific user to be located in the list (A.22).

Discussion
We believe the application of our methodology and testing architecture to a mature, conventionally tested CSCW application was a success. Unsolicited correspondence from the RCN team showed gratitude for the problems uncovered by the CAMELOT and Rebecca approach [6]. Two-dozen bugs were discovered in this mature CSCW application. Some of the problems were cosmetic. However, some of them were serious and should be corrected to make RCN a robust application.

5 Conclusion

Ramage believed that CSCW applications had both social and technological components. He found that most prior work in CSCW evaluation focused exclusively on the social aspects of the system. On the technological side he cautioned that:

> It may well be the case that a computer system will be designed perfectly, with all of the right sort of software engineering procedures, requirements analysis and usability testing, but that the system is introduced insensitively, or it cuts across the way people have become used to working or it changes the power relationships between workers [15].

CAMELOT's deliberate technological focus is not concerned with higher-level social aspects of a CSCW system. CAMELOT's main contribution is an organization and detailed description of the technologies that comprise CSCW software and the problems that should be tested for using these technologies. Following Ramage's concept of multiplicity, CAMELOT should be used in conjunction with other methodologies for a complete evaluation of a CSCW system.

6 ACKNOWLEDGMENTS

This research was supported in part by the National Science Foundation under awards CDA-9634485 and CCR-9527151. The authors would also like to thank software developer J.J. Johns for his support and advice during the evaluation of RCN using Rebecca-J.

REFERENCES

[1] P. Checkland. Soft systems methdology. *Human Systems Management*, 8(4):273–289, 1989.

[2] P. Dewan and R. Choudhary. A high-level and flexible framework for implementing multiuser user interfaces. *ACM Transactions on Information Systems*, 10(4):346–380, 1992.

[3] J. Drury, L. Damianos, T. Fandercla, L. Hirschman, J. Kurtz, and B. Oshika. Scenario-based evaluation of loosely-integrated collaborative systems. In *Proceedings of Conference on Human Factors in Computing Systems (CHI '00)*. ACM Press, 2000.

[4] J. Drury, L. Damianos, T. Fanderclai, L. Hirschman, J. Kurtz, and B. Oshika. Methodology for evaluation of collaborative systems, http://www.mitre.org/support/papers/tech_papers99_00/damianos_evaluating/index.shtml, 1999.

[5] R. F. Dugan, E. A. Breimer, D. T. Lim, E. P. Glinert, M. K. Goldberg, and M. V. Champagne. Exploring collaborative learning in rensselaer's classroom-in-the-round. Technical Report 98-1, Rensselaer Polytechnic Institute, April 1998.

[6] R. F. Dugan Jr. *A Testing Methodology and Architecture for Computer Supported Cooperative Work Software*. Doctoral thesis, Rensselaer Polytechnic Institute, Department of Computer Science, 2000.

[7] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. The Benjamin/Cummings Company, Inc., Redwood City, California, 1994.

[8] C. Gutwin, S. Greenberg, and M. Roseman. Supporting awareness of others in groupware: A short paper suite. In *Proceedings of Conference on Human Factors in Computing Systems (CHI '96)*, pages 205–215. ACM Press, 1996.

[9] R. Jain. *The Art of Computer Systems Performance Analysis: : Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley and Sons, New York, New York, 1991.

[10] MercuryInteractive. Winrunner. User's guide, Mercury Interactive Corporation, 2000.

[11] G. J. Meyers. *The Art of Software Testing*. John Wiley and Sons, New York, 1979.

[12] B. Myers, J. Hollan, and I. C. E. Al. Strategic directions in human-computer interaction. *ACM Computing Surveys*, 28(4):794–809, 1996.

[13] H. Okada and T. Asahi. Guitester: A log-based usability testing tool for graphical user interfaces. *IEICE Transactions on Information and Systems*, E82-D(6):1030–1041, 1999.

[14] T. Ostrand, A. Anodide, H. Foster, and T. Goradia. A visual test development environment for gui systems. In *ACM SIGSOFT Internation Symposium on Software Testing and Analysis*, volume 23, pages 82–92, Clearwater Beach, Florida, 1998. ACM Press.

[15] M. Ramage. *How to Evaluate Cooperative Systems*. Doctoral thesis, Lancaster University, Department of Computing, 1999.

[16] E. H. Rogers, C. Geisler, J. Farley, J. Johns, and C. Parker. The reconfigurable collaboration network, a demonstration of collaborative system sharing. In *Proceedings of the European Computer Supported Cooperative Work Conference 1999*, Amsterdam, Netherlands, 1999.

[17] S. Ross, M. Ramage, and Y. Rogers. Petra: Participatory evaluation through redesign and analysis. *Interacting with Computers*, 7(4):335–360, 1995.

[18] S. R. Schach. Chapter 3: Software life-cycle models. In *Software Engineering*, pages 41–68. Aksen Associates Incorporated Publishers, Homewood, Illinois, 1990.

[19] B. Schneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, Harrow, England, 3rd edition, 1997.

[20] SegueCorporation. Silkperformer user's guide. User's guide, Segue Corporation, 2000.

[21] H. Shen and P. Dewan. Access control for collaborative environments. In *Proceedings of ACM Conference on Computer-Supported Cooperative Work (CSCW'92)*, Building Real-Time Groupware, pages 51–58. ACM Press, 1992.

[22] A. S. Tannenbaum. *Modern Operating Systems: Second Edition*. Prentice-Hall, Englewood Cliffs, New Jersey, 2nd edition, 1992.

[23] L. J. White. Regression testing of gui event interactions. In *International Conference on Software Maintenance*, pages 350–358, Monterey, California, 1996. IEEE Computer Society Press.

[24] S. W. L. Yip and D. J. Robson. Applying formal specification and functional testing to graphical user interfaces. In *5th Annual European Conference on Advanced Computer Technology: Reliable Systems and Applications*, pages 557–561, Bologna, Italy, 1991. IEEE Computer Society Press.