Politecnico di Milano

Facoltà di Ingegneria dell'Informazione

Master of Science Degree in Computer Engineering

Dipartimento di Elettronica e Informazione

EXTENDING CLOUDSTUDIO WITH COLLABORATIVE REMOTE DEBUGGER

Supervisor: Elisabetta DI NITTO

Co-supervisor: Martin NORDIO

Graduation thesis of: Rand NEZHA matr. 764284

Mert TÜFEKÇİ matr. 763076

Academic Year 2011-2012

Sommario

Il soggetto di questa tesi è estendere il CloudStudio con un JavaScript debugger il quale è incapsulato in un debugger collaborativo e remoto. Il debugger incapsulente è stato disegnato a poter funzionare con altre implementazioni dei debugger. L'aspetto più importante del soggetto di questa tesi è che il Collaborative Remote debugger abilita gli sviluppatori, che sono distribuiti geograficamente, a fare il debugging insieme di un'applicazione in un modo collaborativo tramite un web browser. Molti debugger oggi sono integrati negli Integrate Development Environments (IDE) che girano sul locale. Dato che i debugger vengono usati ampiamente e visto la loro necessità, c'è il problema della cooperazione fra gli sviluppatori. Sempre più sviluppatori distribuiti geograficamente sviluppano diverse parti degli stessi moduli che interagiscono fra di loro. Un blocco di codice che è stato scritto da uno sviluppatore potrebbe causare errori in un altro blocco di codice che è stato scritto da un altro sviluppatore. In questo caso gli sviluppatori devono fare il debugging insieme per trovare il problema e risolverlo. Tradizionalment queste sessioni di collaborative debugging vengono effettuate tramite la condivisione dello schermo dello sviluppatore che ha il controllo del debugger. La comunicazione viene di solito effettuato tramite le applicazioni VoIP o IM. Questo riduce la collaborazione fra gli sviluppatori dato che soltanto uno di loro ha il controllo del debugger e l'altro invece guarda lo schermo. Il CloudStudio Collaborative Remote JavaScript Debugger è stato disegnato e implementato per abilitare sempre di più la collaborazione fra gli sviluppatori distribuiti geograficamente.

E' stato disegnato un case study per valutare la prestazione e l'efficienza del tool implementato. Il case study è stato condotto con alcuni gruppi d'utenti e i dati sono stati raccolti dai partecipanti e dai system log. Questi dati sono stati analizzati per riassumere gli andamenti e per arrivare a una conclusione.

Table of Contents

List of Figures

Chapter 1

Introduction

The subject of this thesis is extending the CloudStudio with a JavaScript debugger which is encapsulated in a Collaborative Remote debugger. The encapsulating debugger is designed to work with other debugger implementations. The most important aspect of the subject of this thesis is that the Collaborative Remote debugger enables developers, that are in a geographically distributed environment, to debug an application together in a collaborative way through any web browser. Most of the debuggers today are inside the integrated to Integrated Development Environments (IDE) that run locally. Because of widely usage of debuggers and the necessity, it becomes becomes a co-operation problem between developers. More and more geographically distributed developers are writing different parts of the same modules that interact with each other. A block of code that a developer wrote may cause an error in another block of code that another developer wrote. In such a case both developers might need to debug together to find the problem and resolve it. Traditionally these debugging sessions are made through sharing the screen of the developer in control of the debugger with other developers and comunicating through some VoIP or IM applications. This reduces the collaboration between debuggers since only one of them has the actual control over the debugger and the other is only watching the debugging session or dictating through some application to the other developer without having any actual control over the debugger. The CloudStudio Collaborative Remote JavaScript Debugger is designed and implemented to enable more and more the debugging collaboration between geographically distributed developers.

A case study is designed to evaluate the performance and the efficiency of

the implemented tool. The case study is driven with several user groups and data are collected from the participants and the system logs. These data are then analyzed to get the trends and to come to a conclusion.

Contributions

This thesis is distinguished in these original contributions:

- We searched the current implementations and/or designs of collaborative programming/debugging environments. We then compared these tools with our ideas of a collaborative remote debugger.
- We searched and investigated existing technologies that we can use to implement our collaborative remote debugger.
- We studied the requirements of a collaborative remote debugger and pointed the features of this debugger.
- We implemented the collaborative remote debugger according to the requirements and design by using the tecnologies that we selected.
- We designed a case study to experiment the new innovative aspects of our collaborative remote debugger and we drived it with people that come from the computer engineering background.
- We studied the possible future works and extensions of the collaborative remote debugger that we implemented.

Organization

This thesis is organized as described below.

- In chapter 2 already existing tools that aim to increase the debugging collaboration between geographicaly distributed users are introduced.
- In chapter 3 technologies that are chosen in the implementation of the CloudStudio Collaborative Remote JavaScript debugger are introduced. The requirement analysis, the design and the complexity of the solution is introduced.

- In chapter 4 the actual implementation is introduced.
- In chapter 5 the case study which is made with the implemented application with the participation of many geographically distributed developers is introduced.

The thesis concludes with chapter 6. Some possible future works on the tool are presented.

Chapter 2

State of art

2.1 Introduction

Nowadays the world became more united using technologies, therefore locations of the workers is not the first important thing anymore. Many companies use many different type of computer-supported cooperative tools to manage the work flow among the workers that involve in different projects. Actually there are many tools designed to realize this concept, to let it be easier working in international projects where the workers are spread over the borders of countries. Cloud computing play a main rule in development CSCW (Computer-Support collaborative work) tools. It is evident that the software development world is going towards the cloud solution in a cooperative way, and with the enhancement of cloud computing year after year, and specially PaaS (platform as a service), it becomes very necessary to search for very advanced solutions that make cloud computing more usable.

2.2 Computer-Supported Cooperative Work

CSCW is a term coined by Irene Greif and Paul M. Cashman in 1984 [46], they organized a workshop contains twenty people from different fields, but in standard interest in how people work, gathered to explore the technology's role in the work environment. CSCW started as an effort by technologists to learn from economists, social physiologists, anthropologists, organizational theorists, educators, and anyone else who could shed light on group activity. CSCW

focuses on the study of tools and techniques of collaborative software (also called groupware) and their psychological, social, and organizational effects. The most common definition of CSCW The definition of Wilson [62]: CSCW is a generic term, which combines the understanding of the way people work in groups with the enabling technologies of computer networking, and associated hardware, software, services and techniques. In the last years it was evident that there is a strong immigration to new forms of teleworking which include the following main features [44]:

- Working with groups rather than individuals.
- Induction, synthesis and dialog rather than deduction analyze and one-way transmissions.
- Not co-located with peers, management or factory.
- Mean rules of telecommunication technologies in supporting the works.
- More collaboration in working, as the knowledge becomes more specialized.

The ACM digital library organizes [3] conference every year in different cities of the world for discussing CSCW [4], these conferences are a leading forum for presenting and discussing research and development achievements concerning the use of computer technologies to support collaborative activities, as well as the impact of digital collaboration on users, groups, organizations and society. In these conferences there are presented research in the design and the use of technologies that affect groups, organizations, and communities. The conferences include both the technical approaches of CSCW and social challenges presented when supporting collaboration. The conference brings together top researchers and practitioners from academia and industry who are interested in both technical and social aspects of collaboration. The last conference took place in February 2012 in Bellevue, CSCW 2012 was the fifteenth CSCW conference. where were included 15 workshops and a doctoral colloquium.

One interesting publication presented in these conference a paper that study one of the most important challenge in CSCW which is the different time zone among the collaborators, and how they use the technologies to schedule

their work [58]. They actually conducted interviews with sixteen members of teams that worked across global time zone differences. Despite time zone differences of about eight hours, collaborators still found time to synchronously meet, The interviews identified the diverse strategies teams used to find time windows to interact, which often included times outside the normal workday and connecting from home to participate. Recent rends in increased work connectivity from home and blurred boundaries between work and home enabled more scheduling flexibility. The tools Supported time-shifted collaboration in this case study: while the teams used a variety of tools to support their cross-site collaboration, there was a heavy reliance on email, a range of reactions to using video, and a number of other tools used to support their collaboration such us social networking and other modern collaboration tools. As a conclusion of reviewing the observations from the interviews of globally distributed teams, they identified several design implications to better support working across global time zone differences. Here in the following a brief description of the main improvements can be done for supporting more the working across different time zones.

Remove firewall access constraints for collaboration tools: Firewall often prevent the usage of collaboration tools from home (video, data access).

Integrated tools: Several of the participants mentioned the desire to integrate the tools needed to collaborate together. Several teams had to manage tools for video conferencing, phone, IM, email, scheduling, computer window sharing, shared file repositories, etc. Totally separately. The use of those tools had to be coordinated together, often in the context of a synchronous cross-time zone meeting.

Pervasive difficult of video capacity: There are still obstacles to routinely using the video. Usability and reliability of video conferencing remain as obstacles to use, especially outside the typical workday when technical support resources may not be readily available.

Increased calendaring support: A rather simple design implication of shared on-line calendars is to allow more subtle and fine-grained support for finding time windows to interact with time zone-shifted collaborators

Supporting temporal isolates: Many of the teams leveraged temporal isolates either as an isolated participant at a site or as a temporal ambassador for the site. this pattern may be prevalent among time zone-shifted teams, and such temporal isolates may benefits from tools tailored to their roles as individuals at a remote site.

2.3 Collaborative software (Groupware)

The collaborative software is a computer software designed to help group of workers who involve in a common task to reach defined goals. Basically this software facilitates the organization between the members of a group working together spread over the geographic distances, by providing tools which offer way to communication , collaboration and managing the rules. The groupware is part of CSCW. It addresses how collaborative activities and their coordination can be supported by means of computer systems.

Levels of collaboration

The collaborative software can be divided into three categories depending on the level of collaboration [43]:

1. **Communication:** Interchanging information among the workers. examples : phone calls, voice mail, Im Chat, e-mail, revision control, etc.
2. **Conferencing:** Interactive work to achieve a shared goal, example: internet forums, video conferencing, application sharing, etc.
3. **Co-ordination:** Collaborative management tools. examples : electronic calendars, project management systems , online proofing, social software, etc.

Problems and challenges for designing groupware software

Because of the social and political factors at work in group settings, achieving groupware acceptance is much trickier than single-user product acceptance. In addition to technical challenges, groupware poses this fundamental problem for product developers: Because individuals interact with a groupware application, it has all the interface design challenges of single-user applications, supplemented by a host of new challenges arising from its direct involvement in group processes. Here there are the main challenges of designing groupware software [47]:

The disparity between who does the work and who gets the benefit: A groupware application never provides precise the same benefit to every group member. Costs and benefits depend on preferences, prior expressions, roles, and assignments. Most groupware requires some people to do additional work to enter or process information that the application requires or produces.

Critical mass and prisoner's dilemma problems: Most groupware is only useful if a high percentage of group members use it. Different individuals may choose to use different work tools, but two co-authors must agree to use the same co-authoring tool. Achieving a critical mass of users is essential of communication system. Even one or two defection may cause problems for meeting scheduling, decision support, or project management applications.

Social, political and motivational factors: Groupware may be resisted if it interferes with the subtle and complex social dynamics that are common to groups. Often unconsciously, persons actions are guided by social conventions and by his awareness of the personalities and priorities of people around them. Tacitly understood personal priorities are tactfully left unspoken, yet unless such information is made explicit, groupware will be insensitive to it. Developers need sophisticated understandings of prospective users workplaces. Working with representative users whenever possible is standard advice for developing interactive systems. It is particularly good advice for groupware developers.

Exception handling in workgroups: In general the work processes can usually be described in two ways: the way things are supposed to work and the way they do work. Software designed to support standard procedures can be too fragile. A wide range of error handling, exception handling and improvisation are characteristic of human activity [57]. Designing software that help the worker in their work should start from base specifications done by interviewing the working themselves, but it is not an easy task to determine the actual standard procedure for the work, since the actual work of the worker is not standardized. However, no standard procedure done by a groupware can be complete specially because of unpredictable situations.

Designing for infrequently used features: Work has important social elements that can use support, but groupware features will be used less frequently than many features supporting individual activity. This has two important implication: the first groupware features will be more effective if integrated with features that support individual activity. Second, design to be inconspicuous yet accessible. Infrequently used groupware features must not obstruct more frequently used features, yet they must be known and accessible to users. However, This is a difficult balancing act

The underestimated difficulty of evaluating groupware: Task analysis, design, and evaluation are more difficulty for multi-user applications than for single-user applications. A worker's success with a particular process is not affected by the backgrounds or personality of the group members, while groupware is affected by them, often groupware collect worker with different backgrounds, personalities , roles, and preference. There are many other factors make designing groupware more complex than what it seems to be.

2.4 Collaborative integrated development environments

There is a wide set of collaborative software in the market, specially those which use cloud computing solutions. Some of them are free and others are licensed, some of these tools are open source projects. Examples of collabora-

tive IDE on the cloud: web2project [39], codeBeamer [12], GForge Advanced Server [23],Kodingen [33], Amy Editor [7], EXO IDE [22], Codiad [14], Shift Edit [37], Ecoo [21], Compilr [18], Collide [17], ideone [29], Collabode [15, 16] , and many other more, each one have different level of collaboration and cloud activities. in the following section are described three of collaborative software that offer collaborative development platform for the developers to implement software and applications.

2.4.1 IBM Jazz- Rational Team Concert

IBM Jazz [28] is a collaborative technology platform for software delivery. It is an extensible framework, IBM Jazz can integrate and synchronize people, processes, and assets related to software projects. IBM Jazz offers the following main features:

- Team awareness through automatic notifications (team members can see the work of the other members).
- Collaboration features such as project integrated presence and messaging.
- Extensible infrastructure through open Web standards for including new plug-ins.
- Process awareness guiding team workflow and automating process steps.
- Flexible process definition by specifying and modifying process rules for different subjects or projects.
- Connector framework for integrating other source control and change management systems, such as subversion.

IBM Rational Team Concert [36] is one of the IBM Jazz based products that provides planning, source code, work item, building, and project health management. To carry out these tasks, this tool integrates modules to support process and team awareness, work item tracking, and agile planning, providing tools to create plans for teams and continuous build integration. Here in the following are describe the main features of IBM Rational Team Concert :

Item tracking: IBM RTC offers Web based tools for tracking work items (events, errors, exceptions, any time of team activities), as create new work item from zero or from predefined template print, import, query, and events logs for the work item to track the changes. RTC has a way to address the members of the team in a work item comment in social way to be notified the addressed one. And a member can capture a part of the screen, annotate the image with a line drawings, arrows and text, and attach it to a new or existing work item.

Source control: The Source Control component of Rational Team Concert is a component-based version control system built on the Jazz platform. Its focus is on supporting geographically distributed teams, to track and manage changes in the source files.

Planning: IBM provide tools to create products, release and sprint backlogs for teams, to create individual plans for developers, and to track the progress during an iteration and to balance the work load of developers. It offers ways for Planning a Sprint, Managing a Release Backlog, Planning the Daily Work, and tracking the progress of the release and its sprints.

Team organization: Rational team concert has tools to manage team-related information or performing team-related operations. For example, Set up a team, Join a team, Presence and chat in context, configure the Team Artifacts, build events for team members, and others

Builds awareness: RTC provides build awareness, control, and tractability to the team. Team members can track build progress, view build alerts and results, request builds, and trace builds to other artifacts like change sets and work items.

Problems tracking: RTC has an automatic way to detect violations of the team's process and the instant they happen, and report this information for further uses.

2.4.2 Cloud9

Cloud9 [10] is a commercial open source development as a service (DaaS) platform, It is web based IDE (integrated development environment), founded in 2010, supports several programming languages for editing, and a subset of languages for executing and smaller subset for debugging and testing is written in JavaScript. It provides a free version and a version with costs.

Here in the following are described the main Cloud9 features:

Management and version control functionality: Cloud9 is designed to integrate tightly with source doe repository sites GitHub [25] and BitBucket [8]. But it also supports other repositories like mercurial [35], Git [24], and FTP servers offers the developer a way to clone, edit, push code.

Support for deployment: Cloud9 supports application deployment to Heroku [27], joyent [30], or Windows Azure [40].

Interactive code editor: Which include basically syntax highlighting, optional support for Vim key bindings, code completion (for JavaScript/Node.js). The editor of cloud9 can be used also for offline editing, which allows the developers to continue working without internet connection. To use offline editing in cloud 9 the developer has to install a special client application on their computer that can handle synchronization between the local file and the cloud-exciting one. Cloud9 editor offers real-time collaborative editing in the browser allowing multiple developers to edit the same file at the same time so the developers can pair the program and review code, chat windows to let the developer to chat while developing.

Run-time environment: Cloud9 offers run-time environment for Javascipt/Node.js, Python, Ruby, and Apache+php applications, Cloud9 also offers the possibility to execute basic linux commands by its console. Cloud9 provides for every workspace an apache server. With it a web project can be tested.

Debugging in Cloud9 Actually Cloud9 offers a debugger to the developer of javascript/Node.js applications with all the debugger functionalities (debugging navigation, callstack, expressions evaluation, variables inspection,

breakpoint management). The debugger of Cloud9 can be used individually by each developer, and a debugging session can't be shared among the members of a specific project. So it is not included in the collaborative solutions which cloud9 offers, like shared code reviews and others. It can be seen as a debugger that runs in the browser and lives in the cloud, which is a part of web based IDE.

2.4.3 CodeRun

CodeRun Studio [13] is a cross-platform open source integrated development Environment(IDE) designed for the cloud. Here in the following are described the main features of CodeRun tool:

Projects creations: CodeRun offers ways for developers to create a project from the existing one (Visual Studio projects) or start from one of the built-in templates [2]. It supports C# /.NET (3.5), PHP (5.1), JavaScript, HTML and CSS. C# support includes ASP.NET, WCF, Silverlight and WPF browser application development and deployment. Database support includes SQL Server 2005 and Amazon SimpleDB [6].

Code editing: Code Run offers real-time syntax-coloring and automatic code-completion for C# / ASP.NET, C# / WPF, C# / Silverlight.

Compile code: CodeRun offers server-side compilation for Microsoft .NET managed code applications. The output console and error list window reports the standard messages.

Project collaboration: The developers can share their codes since each project is assigned with a unique URL which can be sent to others. The project can be shared also on some social networking websites. The invited developers can receive a separate, modifiable, and runnable version of the code. The owner of the project can, using the team collaboration mode, set read/write permissions for different users. There is also integrated Meebo chat rooms [34] for group chatting.

Run and debug code: CodeRun offers a debugger which contains the main functionalities, navigating the code, adding breakpoints, inspecting the call stack, and adding watch items. Since the developers can choose to share their code, so they can allow the code to be debugged by others. but still the debugging activity is performed individually and separately by each developer.

2.5 Collaborative debugger

Most of the described or mentioned Collaborative IDE in the previous sections have very limited features for offering a collaborative debugging. Most of them offer the collaborativity as a consequence of the possibility of sharing the code and the project, so each developer can debug the same shared code, but they still can't share the debugging sessions itself. However, there are some solutions in the market that tries to offer a collaborative way for debugging codes and projects. Some of these solutions are similar in how they let the developers to share the debugging activity, and others introduced new types of debugging. However, by collecting feedback experience from developers who use most of these tools plus our own experiences of using them as well, none of these solutions offer a complete collaborative debugger that help efficiently the developers to find the bugs together in a real-time collaborative way. In the following sections are described the most important solutions we have found as collaborative debuggers, about their features and limitations.

2.5.1 Microsoft Visual Studio 2010

Visual Studio [38] is an integrated development environment from Microsoft. It has different versions. Its 2010 version offers the possibility of sharing the debugging session information among the developers. It can be useful when a developer who starts the debugger gets stuck and decides to ask help from another developer exporting his/her session information. So the other user can just pick up where the first user left off. In other words, the the whole process can be described as switching the debugging session between the developers. The best way to describe all the collaborative functionalities that this tool offers, is to provide a scenario where it can be usable. Here in the

following example is described a scenario with each step shows when and how the collaborative approach of Visual Studio can be used.

1. Developer A starts the debugger and then end up at point where he thinks the bug exist, but he gets stuck and wants to ask help from developer B.

2. Developer A exports breakpoint information as an XML file, which contains the current debugger code line and all the breakpoints on developer A code. Developer A can pin some variables (commenting of a variable status) which he thinks that are causing the bug and exports this information also as an XML file.

3. Developer A sends the files by any communication channel to developer B.

4. Developer B after receiving the generated files can save them into his machine, and import them and then can start at the break point that developer A exported, and can read the comments of the pined variables.

Limitations

- The developers should have the same code version. So the breakpoint will end up on the code of B in the same position as A exported them.

- The developers still have to use the traditional connection way to share the files which save the state of the debugging sessions.

- The control is still single, meaning that the collaborativity is just presented in sharing debugging related information. None of other developers can influence the debugger flow when it is controlled by one developer at a time.

- The whole operation is asynchronous, so it depends on the channel which the developers communicate. A real time collaborative debugging is provided just in case the developers organized that, in other word the synchronization is not provided in an automatic way.

- VisualStudio is a desktop platform, and it doesn't offer online versions. So if a developer want to use it just for collaborative debugging he and all the involved developers have to install and configure it in their machines. In other words it is not platform/operating system/language independent.
- It is not free.

2.5.2 IBM rational debugger

IBM Rational Team Concert, described in the previous section 2.4.1, is a software development team collaboration tool based on Jazz platform. It is developed by the Rational brand of IBM. It offers a collaborative environment which the developers can organize their work in.

For collaborative debugging feature, IBM offers an extension for the RTC server. The extension allows *Team debug* by sharing the debugging session between Rational team concert team members. it has more or less the same approach of Visual Studio of transferring the debugging session information, but with more advanced features.

As a general idea IBM RTC debugger extension works as the following: When a new team debug session is launched by client developer, the remote system which hosts the debugger establishes the communication to the RTC server which connects back to the client. So all the communication between the client and the remote debugger is now done by the RTC server.

There are two basic aspects the IBM Rational debug extension for RTC offers for debugging collaboratively, here in the following are described briefly

Transferring the debugging session: A developer as a member of a team can transfer the debugging session to another team member. For example, the developer A which is a team member of a specific repository can start debugging an application and insert some breakpoints. But at some point the developer A decides to ask help or review from the developer B which is also a team member of the same application. The developer A sends to the developer B a transfer request which goes through the RTC server. The transferred debugging session holds the current state, breakpoints and monitors. And further comments can be added by developer A to the request. Developer B

gets notified to respond the developer A's request. Once accepted the request, the communication between the developer A and the RTC server is terminated and another communication between the developer B and RTC server is established. He will become the owner of the debugging session and can continue from the same point where the developer A left off. The developer B can transfer the debug session to another developer as well by performing the same operations. There is just one debugging session owner at a time, and the debug session can be switched among the members of the team at anytime.

Parking the debug sessions: A developer can decide to suspend the debugging session temporarily for further user. For example, if the developer A of the previous example wants to ask for help or review but he doesn't know yet who actually could help. Developer A can 'park'the debugging session to the team repository which will be suspended at its current state. Unlike transferring the park, the debugging session does not belong to any developer in the team, and it remains in its current state until someone of the team retrieve it from the team repository. Then the developer B who is interested in continuing debugging the developer A's debugging session can search for it in a particular area of the team repository, by performing a query for all the parked debugging sessions in this repository. He should make sure that no one is currently using this debugging session, then the developer B can continue debugging at the point where it is lifted off, and can also add his own breakpoints.

Limitations

The limitations of this tool are almost the same of the previous tool Visual studio described in section 2.5.1, since they offer nearly the same solution approach. Here also the control is still single. A developer can't see the changes made by the others real time, he can't neither influence their debugging flow. It is actually like dividing the debugging tasks by developer, each one debugs the piece of code which is familiar with by continuing to debug at the point where the last developer left off.

2.5.3 JS Bin

JS Bin [32] is a web-based tool for collaboratively testing, executing, and debugging JavaScript, CSS, and HTML applications. This type is a new way for collaborative testing and debugging. It is a live output rendered in the browser of the testers and users who are sharing the same URL. It is a free and open source project. The idea of JS Bin is to create test and debug cases and then collaboratively debug these cases. JS Bin allows the developer to edit and test JavaScript and HTML reloading the URL and also maintains the state of the code.

To describe well this tool it is better to provide a simple scenario where it can be used: The developer A is working on a project (JS/HTML/css) and at some point he needs help or reviews from some of his colleagues. He can send them either a URL to see the code being updated by him, or a URL to see the live output rendered of his code in real time as he types. The sent URL will be his same URL but instead of ending with /edit will end with /watch. Then all the developers who have the link can see in real-time the code and application output as well. They also can edit the code but the further changes will be saved in a new milestone, ie new version and not in the origin one.

How JS Bin works in general

As typing the code, the tool sends an Ajax save request on the server side. This triggers an event which says 'the bin with this URL has just changed ', now when viewing the code or the output URLs, each one is connected to the Spike application. It listens for the event that says that a bin has changed and when this happens it just finds all the users watching this particular URL and sends the updated panel an event. Source maintains a persistent connection with the server and it is listening for those events in the browser. When the event is received, depending on the content type , it'll either inject the content live, or refresh the page providing user the latest version of the code [31].

Limitations

- Based on personal experience sometimes the changes doesn't receive updates automatically. There is the need to refresh the page.

- There are some delays in the automatically reloading the pages, since it reloads the page every time the developer changes the code.
- It is just for HTML/CSS/JavaScript applications. The project which the developer can test, using this tool, is just a HTML/CSS/JavaScript application. However this solution of debugging is suitable just on these types of applications since they are already web languages.
- It doesn't provide the traditional way for debugging (navigating the code, breakpoints, expressions evaluation, etc.)
- Two developers can't edit the same version of the project. So if some developer wants to show some changes' effects, the changes will be saved in a new version and will not be visible to the original code owner.
- The developers still need to use the traditional communication tools to share the link of the code.

2.5.4 DebugLive

DebugLive [19] is a web based, collaborative remote debugging solution developed by DebugLive.com Inc. It offers a way to the developers to debug their applications remotely in real time. It also offers the possibility of collecting information to continue debugging offline. DebugLive basically helps to developers to solve the software troubleshooting on remote sites .

DebugLive offers the following features:

1. Shared platform for debugging information.
2. DebugLive debugger can be configured for email notifications when certain events occur, and maybe package files created by the debugger and uploaded to the DebugLive server. This allows the developer to monitor his/her applications.
3. DebugLive can be configured to collect automatically specific type of information as an action at a specific type of events.
4. Remote debugging session between the developer and a target machine which host the target application.

5. Debugging windows applications.

6. Creating memory dumps for local or remote machine.

To initialize the debugging session the developer should navigate to the start debugger page. Then an executable is downloaded to the system and saved in the browser's cache which contains the DebugLive debugger components. It is stored in a compressed form. Then when the developer launches the executable it creates a DebugLive directory, uncompresses the component into the directory and launches the main DebugLive executable. The main executable, in return, opens the DebugLive debugger webpage. It loads the necessary debugger components, and makes them available to the debugger web page as a set of ActiveX control

The basic idea of DebugLive is to help the specialists to solve collaboratively reliability and performance issues in the following kind of applications:

- native windows applications(c++/win32)
- .NET applications
- ASP.NET applications

To describe how and where DebugLive can be used we describe a scenario which uses DebugLive as the solution: An application is crashed on the customer site, so instead if asking to the customer to create the crash dump and send it back to the specialists, Debuglive offers a way that lets the customer to login in the browser (or by DebugLive add-in of Visual studio) on a specific DebugLive account who is known also by the specialist who is trying to solve the problem. From this point on a remote debugging session is started. Now the specialist by navigating a specific DebugLive URL with a browser, and then joining to the debugging session, can take the control. Then he can list the processes, debug them in real time, create crash dump, and create different kinds of troubleshooting information like log files, registry settings event log entries. DebugLive offers also the possibility to upload the information collected to this specific account for further analysis

The participant can belong to the following categories: organizer, observer or leader. Every remote debugging session has exactly one organizer, one leader, and unlimited number of observers. Only the current leader can send commands to the debugging session.

Organizer

The Organizer is the person who starts the remote debugging session. The organizer is in complete control of the remote debugging session. Then he can specify the list of the users that are allowed to participate in the session and can at any time lock the session to prevent all the other participants from accessing it. The organizer can perform the following tasks :

1. **Starting a new debugging session**: Using the session windows he should specify the debugging session name, which will be (in addition to session ID) used by the observers as identifier to connect to this session. He can specify if someone can participate in this session or not. Making the session private so only this current user is able to connect to it . After starting the session a session ID will be assigned. It contains the unique ID of the remote debugging session. It can be sent to a specific developer to be able to precisely identify the session and connect to it.

2. **Controlling who is allowed to access the session:** The organizer can at anytime add or remove users which can access the debugging session.

3. **Close the session:** Observer can uniquely close the remote debugging session. Then after the remote debugging session has been closed by the organizer, all the other participants of the session are immediately notified.

4. **Locking and unlocking the session**: The observer uniquely can at anytime lock the debugging session. Locking the session prevents the observers from sending commands. The organizer can as well unlock the session at anytime. After locking or unlocking the remote debugging session by the organizer all the other participants of the session are immediately notified about it.

Observer

The observer is a specialist who joins to an existing debugging session. The observers observe the whole process, receive outputs of commands sent by the

debugging session leader, and can ask to be the leader of the debugging session. The Observer can perform the following tasks :

1. **Joining to a session:** To join to the session, it is necessary to know either the name of the remote debugging session, its ID, or both in order to be able to identify it.

2. **Leaving the session:** The observer can leave the remote debugging session at anytime, and after a participant has left the debugging session all other participants are immediately notified about it.

3. **Leading the session:** Since just an observer can be a leader at a time, so the observer can ask to be the leader of the remote debugging session. After a participant becomes the leader of the remote debugging session, all other participants of the session are immediately notified about it

Leader

The leader is an observer registered himself as the leader of the session, the leader is the observer who can uniquely send commands to the Organizer's instance of DebugLive debugger, which processes the commands and returns the output back to the observers. The organizer and the others observers who participate in the debugging session will also be able to see the commands and their outputs. The leadership can be moved from one observer to another at anytime.

The leaders can send commands to the DebugLive server then, if the remote debugging session is not locked by the organizer, the server stores the command in the command message queue associated with this remote debugging session. The organizer's instance of DebugLive debugger contacts the server periodically to fetch the commands from the message queue. Once the command is fetched from the server the debugger executes it, collects its output and sends it back to DebugLive server. When the DebugLive server receives the command output messages, stores them in the output message queue associated with this remote debugging session. The observer instances of DebugLive debugger (including the leader) contact the server periodically to fetch the command output from the message queue, to be displayed to the observers.

DebugLive offers the following collaborative features to the participants to collaborate among each other:

- Sending message to all the participant or to just the organizer.
- Checking the session status: When the observer checks the session status, the reported information will be the session id and the list of users which are currently participating in the session (with their rules such as organizer, leader, etc.). In addition to this information the organizer will be informed about the list of the users which are allowed to participate in the session.
- Data exchange: A participant can upload some files from the remote machines to Debuglive server and share them with the other participants.
- Watching what is going on the remote machine by taking a snapshot of the remote machine screen, or by remote video recording.
- Sending files to the remote machines: A participant can send files to the remote machine .

Here are some of the commands that can be sent by the particpants are: ProcessList command, PrccessDetach, GO , Break, stepIn, stepOver, stepOut, breakpointList, eventSetting, dumpOpen, fileUpload, GetValue, Memory, registerList, callStack, disAssemply, moduleList, Threads. Each command has a rule in the debugging activity.

Limitations

- **Single control debugging:** Even though it is a collaborative debugger, it is still single control at a time, since just the leader of remote debugging session can send commands at a time.
- **Operating system dependent:** Even though DebugLive is a web-based tool, it is not operating system independent. DebugLive can be used just on a machine which has Windows as the operating systems. There is no support for Linux or MacOS.

- **Command-based communication mode:** The communication between the observer's and the organizer's session is based on the commands and command output. Then all accessing debugger information performing operation are done by sending command by the participants and receiving back output, so sending commands/receiving output is not a transparent process to the participant, indeed, this is the unique way to perform the debugging and collaborative operations.

- **Asynchronous communication:** The communication between the organizer and observers is completely asynchronous [20], and there is no reliable way to determine the debugger state at any moment of time from the observer's machine, so observer can send inappropriate command with respect to debugger state and then will be refused to avoid harms.

- DebugLive is designed just for debugging 32-bit Windows application code [20].

- DebugLive is not for free.

2.5.5 IBM patent: collaborative software debugging in a distributed system with symbol locking

There is a set of theoretical patents [42] released by IBM by the inventor Cary L. Bates [9] who works at IBM . The patents introduce together a collaborative debugger, but this invention doesn't have any corresponding real tool that realize the introduced features. These patents introduce the distributed system as the compositions of a debug server, a plurality of debug clients, and a data communication network. The debug servers is coupled for data communications to the plurality of debug clients through the data communication network and the debug server includes a debug administrator, a message router, a back-end debugger, and a debuggee.

From debugger server prospective

From the perspective of the debug server, collaborative software debugging in the distributed system includes:

- Receiving, by the debug server from the debug clients asynchronously during a debugging session of the debuggee, a number of application-level messages and a request to the desirable command/operation.

- Routing, by the message router in accordance with an application-level message passing protocol, the application-level messages among the debug clients, the debug administrator, and the back-end debugger, thereby providing distributed control of the back-end debugger to the debug clients with application-level messages routed to the back-end debugger.

- Returning, by the debug server to the debug clients in response to the application-level messages routed to the back-end debugger, client-specific debug results including, returning, responsive to a request o the desirable command/operation.

From debugger client prospective

From the perspective of the debug clients, collaborative software debugging in accordance with embodiments of this patent invention includes:

- Presenting, by each debug client to a user of the debug client, a client-specific graphical user interface.

- Detecting, by each debug client, user input through the client-specific GUI, including detecting, by requesting debug client, user input specifying the desired command/operation and one or more events.

- Generating, by each debug client in dependence upon the detected user input, one or more application-level messages, including generating one or more application-level messages that form the request to the desirable command/operation.

- Sending, by each debug client, the application-level messages to the debug server, including sending to the debug server the request to the desirable command/operation.

- Receiving, by each debug client responsive to the application-level messages, client-specific debug results, including receiving by a requesting debug client responsive to a request of the desired command/operation.

- Displaying, by each debug client in the client-specific GUI, the client-specific debug results.

2.6 Summary

Most of the collaborative debugging softwares, described in this chapters or already exist in the market, suffer from wasting time problems. The extra time is actually wasted on activities that are not, in fact, central to solving the core problem. These wasteful and costly activities include emailing others in the team, spending time on phone calls and conference calls, sending files, organizing information from various sources, and other distractions that do not help to solve the task in the hand. CloudStudio collaborative debugger removes these bottlenecks and allows team members to focus entirely on solving the problem in the hand in a collaborative team-focused real-time environment.

With the most of the existent collaborative debuggers the debugging sessions can be **divided** by the developers but in fact it cannot be **shared** in real collaborative way. Indeed, the control of debugger is still single as well as browsing the debugger status, navigating the variables, and adding expressions. Two developers in the same time cannot control the debugger or even watch the debugger status at the same time. While with CloudStudio collaborative debugger all the developers can browse the debugger status in the real time, and can influence its flow as well. Also CloudStudio collaborative debugger offers the possibility of the single control mode if a specific developer need to be the master of the debugger session at a specific time, and the others can involve as watchers of the debugger status navigating the local variables and adding expression. However, the mastership of the debugger session can be moved form a developer to another at anytime.

In most cases, the collaborative debuggers are parts of a larger development environment which, in most cases, embrace the old desktop paradigm and provide only a limited capacity for remoting and collaboration. So the developer who needs to use the debugger should install the entire environment, all his team's members do so as well. It would be convenience if the developers already use this environment in their project, but what if they don't! While with CloudStudio collaborative debugger, since it is a web-based tool and lives on the cloud, there is no need for any installation or extra work. All the developer

needs is internet connection and then the tool can be accessed in any part of the world by computer, smart phones, or tablets.

Most of the existing tools are language dependent, since most of them are a part of IDE, so the collaborative debugger is limited to the languages offered from this IDE. While CloudStudio collaborative debugger can be implemented for any language, it is actually language independent. Currently CloudStudio collaborative debugger is implemented for JavaScript, but it can be easily implemented for other languages with minimum efforts.

The most of the existing tools still need the use of the traditional communication tools for sending files, sending URLs, communicating among the members to synchronize the operations. While with CloudStudio collaborative debugger everything is already provided to the developers, and there is no need to send any file of URL, and the developers are automatically synchronized and they don't need to use other tools to help them in synchronizing the tasks.

Actually analyzing the state of art of existing collaborative debuggers, almost no products or services on the market that compete directly with CloudStudio collaborative debugger. We have leveraged the Internet in a pioneering way that achieves a completely new approach to enable teams in real collaborative way to find software bugs.

Chapter 3

Analysis of the problem and the solution

3.1 CloudStudio

This thesis is a part of CloudStudio project [50]. CloudStudio is an Integrated software Development Environment (IDE) developed by a team of developers from different universities supervised by ETH university of Zurich. The motivation of CloudStudio kays in the challenges introduced in distributed projects [54], where teams collaborate in a geographically distributed setting. Such distribution introduces new challenges [51, 54]. For example how to design an API, how to write requirement documents or how to manage a project. The CloudStudio project is a result of the experiences from a course on distributed software development, DOSE [52, 50], taught at ETH Zurich. During the course, software projects are implemented in a collaborative fashion by students from several universities in Asia, South America, and Europe. A lack of integrated tools that support distributed software engineering has been identified during the course, enabling distributed projects to produce software on the cloud.

CloudStudio enables every developer to work on a common project repository [11], shared on the cloud. One of the main differences with traditional IDEs is that configuration management becomes unobtrusive; instead of the explicit update-modify-commit cycle, CloudStudio keeps track of successive versions and maintains the history automatically. Direct modification of a

shared repository avoids raising spurious conflict notifications even when two developers are working on the same module; actual conflicts are detected early, and resolved through prevention rather painful post-hoc reconciliation of changes.

CloudSudio includes many collaborative features needed for IDE used teams, and in the features :

- Compilation tools: plus a way lets the developers to display and compile the changes introduced by other developers, actually the tool also lets the developers to compile the code using or ignoring the changes of other developers.

- Code explorer: the developer can see the changes introduced by other developers, actually the tool lets the user to display or hide the changes introduced by other developers

- Testing on the Cloud: CloudStudio supports testing on the cloud. it integrates AutoTest on the cloud. AutoTest is an automatic contract-based testing tool developed by the Chair of SE - ETH.

- Proofs on the Cloud CloudStudio also supports formal verification on the cloud. CloudStudio integrates a tool called Eve Proofs. This tool translates Eiffel to Boogie and uses the Boogie verifier.

Integration of verification tools in CloudStudio CloudStudio IDE offers verification tools which are static as well as dynamic to verify the CloudStudio projects, here there are the verification tools CloudStudio offers:

- A static auto proof tool which offers the possibility of automatically proofing Eiffel code [49, 60, 59] . The idea is basically testing the post conditions against preconditions to determine the cases where satisfactory preconditions yield unsatisfactory post conditions.

- An automated unit-testing suite which concludes tests based off contracts [61, 48]. This Auto Test test the bound of the contracts of a given class and reports failures.

- Auto Fix, working on this tools is in progress. Auto fix tries to generate fixes for the errors found [55]. Fixes are generated analyzing both statically and dynamically, then the generated fixes are tested by regression tests to see if they are suitable fixes to the error found.

3.2 Rhino JavaScript engine

3.2.1 Introduction

Rhino is an open source implementation of JavaScript written entirely in Java. It is managed by Mozilla Foundation. Rhino can be embedded into any Java application.
Rhino works in two modes : compiled mode and interpretive mode. In compiled mode Rhino compiles all JavaScript code to Java bytecode in general Java class files, but this mode suffer from two limitations. First, long compilation time since generating/loading Java bytecode is a very resource intensive process. Second, leaked memory since most Java Virtual Machine (JVM) don't collect unused classes or the strings that are interned as a result of loading a class file.

Rhino offers the following main features:

1. Direct scripting of Java
2. Features of JavaScript 1.7
3. A JavaScript shell for executing JavaScript scripts
4. A JavaScript compiler to transform JavaScript source files into Java class files
5. A JavaScript debugger: for scripts executed with rhino
6. Java Adapter: allows JavaScript to implement any Java interface or extend any Java class with a JavaScript object.

The debugger which Rhino offers, is a stand alone application and can be used only locally. So, for our purpose it was necessary to build our debugger which is remote and collaborative, getting the benefit from the classes and interfaces of Rhino core. In chapter 4 will be discussed with more details how

we implemented Rhino interfaces.

3.2.2 Advantages and disadvantages

Rhino comparing with other options

Before starting our implementation we did some research to be sure that we will choose the most suitable JavaScript Engine that fully meets our needs and will work efficiently
Other JavaScript engine options in addition to Rhino:

- Spidermonky: is also open source JavaScript interpreter managed by Mozilla and has also embedding APIs. Written in C.
- V8(JavaScript Engine) : is an open source JavaScript Engine managed by Google. It is written in C++ and have embedding APIs. It offers a special protocol for client-server debugging. It is needed to implement a bridge to be used in Java applications.

Why Rhino?

- It is written in Java so we can embed it simply to our application which is also written entirely in Java.
- It has embedding API that meets all our needs.
- It has JavaScript Debugging interfaces that can be freely implemented depends on our needs.
- It has an elegant solution for multithreading (a specific object called context used to store thread-specific information about the execution environment).
- It is open source.
- Mozilla license is compatible with closed-source projects.

Rhino limitations

- Performance issues compared with other JavaScript engines.
- Very few documentations.

3.3 GWT - Google Web Toolkit

Google Web Toolkit is a development toolkit for building and optimizing complex browser-based applications. GWT is used by many products at *Google*, including *Google AdWords* and *Orkut*. It's open source, completely free, and used by thousands of developers around the world [26].

With GWT SDK, developers write AJAX front-end in Java which is then compiled by GWT cross-compiler into JavaScript that automatically works across different web browsers. Developing with GWT, developers doesn't have to think about the low level details of client-server communication. It is possible to use other languages than Java to write the server code since GWT works with many standard communication protocols. GWT creates a separate compiled version of the application, so a Firefox browser displaying an application in German doesn't have to download extra code for other browsers or languages. It is possible to create widgets by composting other widgets and to lay them automatically on the panels. It is possible to pack the widgets into JAR packages.

Client server communication

Whether using GWT RPC or getting JSON over HTTP, all the calls from the HTML page to the server are asynchronous. This means they do not block while waiting for the call to return. The code following the call executes immediately. When the call completes, the callback method that is specified when making the call will execute.

Asynchronous calls are a core principle of AJAX (Asynchronous JavaScript And XML) development. The benefits of making asynchronous calls rather than simpler (for the developer) synchronous calls are in the improved end-user experience:

- The user interface remains responsive.

- It is possible to perform other work while waiting on a pending server call.
- It is possible to make multiple server calls at the same time.

Remote Procedure Calls (GWT RPC)

The very important difference between AJAX and HTML web applications is that AJAX applications doesn't need to fetch new HTML pages while they execute. Since AJAX runs more like application within the browser, it doesn't require to request new HTML from the server to update the user interface. The mechanism for interacting with a server across a network is called making a Remote Procedure Call (RPC). GWT RPC makes it easy for client and server to pass Java objects back and forth over HTTP.

Retrieving JSON data via HTTP

If the application talks to a server that cannot host Java servlets, or one that already uses another data format like JSON or XML, it is possible make HTTP requests to retrieve the data. GWT provides generic HTTP classes that can be used to build the request, and JSON and XML client classes that can be used to process the response. It is possible also to use overlay types to convert JavaScript objects into Java objects that can interact with in IDE while developing.

Cross-Site requests for JSONP

While creating a mashup application that needs to use data from one or more remote web servers, there is a need to work around SOP (Same Origin Policy) access restrictions.

3.4 Requirement analysis

In a distributed and collaborative software development environment like CloudStudio, it is not enough implementing a debugger with basic debugger functionalities such as step into, step over, etc. More functionalities for a

collaborative remote debugger are studied to facilitate the collaboration between users and the remote communication between client and server during a debugging session.

The extension consists of two different implementations. The first one is the implementation of a **CloudStudio JavaScript Debugger** with basic debugging functionalities using Rhino JavaScript implementation. The second implementation is **CloudStudio Debugger** which will orchestrate the underlying debuggers of CloudStudio with collaboration and remoteness properties.

3.4.1 CloudStudio JavaScript debugger

CloudStudio JavaScript Debugger is implemented as a debugger for JavaScript which implements the `Debugger` interface of Rhino JavaScript implementation. It's a no-collaborative, no-remote, pure debugger with the basic debugging functionalities. This debugger is one of the debuggers of CloudStudio IDE. It is designed to run on the server-side and have no direct communication with the client side nor any direct request can come to it from the client side.

Requirements

- Basic debugging functionalities should be implemented. These functionalities are start, step into, step over, step out, resume, terminate.
- The debugger should return all updated variables on each line change.
- It should be possible to send expressions to be executed. Execution of these expressions should not change the state of the debugger and should not have effect on the flow of debugging.

3.4.2 CloudStudio debugger

CloudStudioDebugger is the orchestration of the pure debugger that lays under it. Since all of the underlying pure debuggers have the same unique interface, this debugger makes no operational distinction between them. CloudStudio Debugger enriches the underlying debuggers with collaborative and remote properties.

Collaboration requirements

- More than one user should be able to share the same debugging session. The meaning of this is that users can see real-time the changes of the state of the debugger.
- Users should be able to control the flow of the debugging together.
- Users should see the variables of the script, and these variables should be updated on each execution of line.
- Users should be able to execute expressions during a debugging session. Execution of an expression should have no effect on the flow of the debugger and should not break the current debugging session.
- It should be possible to add users to an active debugging session.
- A user which is not a developer of a project should not be added to a debugging session. It should not be even possible to send a debugging invitation to it.
- Only the users who are logged in the project can share a debugging session. Even though a user is a developer of the project, if it is not online in that project, it cannot join to the debugging session of that project. Though, it is possible to send an invitation to the user to inform it to log in the project.
- In an application wide sense it should be possible to have more than one debugging sessions at the same time. Different debugging sessions should have no effect neither on the work of other users nor on the different debugging sessions of other users.
- A user should be active only in one debugging session.

Remote requirements

Since the technology used for the development of the CloudStudio, CloudStudio Collaborative Remote debugger should follow the design keys that GWT imposes. First of all GWT converts the client side code into JavaScript which has no threading support.

- Since the client side cannot be multi threaded and because of the collaboration requirements the CloudStudio JavaScript Debugger should be placed on the server side.
- Client side should send the commands to the debugger and receive information about the state of the debugging session.
- Server should inform client via push notifications since because of the design of GWT RPC communication, right after a request is sent from client to server, client doesn't wait for the reply and continues its execution.
- Server should inform all of the users that participate in a debugging session about the state of a it.

3.5 Analysis of the application: goals and characteristics

For a distributed and collaborative environment, Rhino is integrated into the CloudStudio and a JavaScript debugger is implemented as the underlying JavaScript debugger. However the overall design of the debugger is generic to CloudStudio. The underlying JavaScript debugger is in charge of executing the script and providing the basic controls that are explained in *debugger flow controls* and *common debugger tools* sections.

3.5.1 CloudStudio JavaScript debugger

Flow controls

CloudStudio JavaScript Debugger has all the basic debugger functionalities. These functionalities are:

- **Step Into:** If the current line contains a function, the debugger enters in the function. In the lack of a function, the debugger steps on the next line in the current function. On the last executable line of the program, the debugger terminates.

- **Step Over:** Execution will resume and the debugger steps on the next line of the script. On the last executable line of the program, the debugger terminates.
- **Step Out:** Execution will resume until the current function returns or a breakpoint hits.
- **Resume:** Execution will resume until a breakpoint hits or the script completes.
- **Terminate:** Execution of the debugger will be terminated.

Tools

Some tools are designed to give information on variables and expressions within the context of the current active debugging session. They don't have effect neither on the state of the debugger nor on the flow of debugging. These functionalities are:

- **Variables:** On each line change the CloudStudio JavaScript Debugger returns back all of the variables of the entire script with the updated values. Variables that are not defined in the function where the debugger waits in are shown as undefined. It is not possible to control the variables from outside, it's a read only structure.
- **Expressions:** It is possible to provide any expression to the debugger. The expression provided will be executed within the current context of the current active debugging session.

3.5.2 CloudStudio debugger

Collaboration roles

The need of collaboration between users of the CoudStudio IDE brings along with itself the distinction between them. Users that participate in the debugging session will be separated into two roles. Later on, these roles are used to design the debugging modes which is discussed in this section. These roles are:

- **Normal user:** This user doesn't have control over the commands that affect the flow of the debugger. These roles are explained under the *debugger flow controls* section. This user has the functionalities that has no effect over the flow of the debugger such as visualizing variables, adding breakpoints or executing expressions. In a debugging session there can be zero or more normal users.

- **Master user:** This user has full control over the debugger, including the commands that affect the flow of the debugger. Furthermore, this user can add other users to the current active debug session or set another user which participates in the current active debugging session as the *master user*. In a debugging session there can be one or more master users.

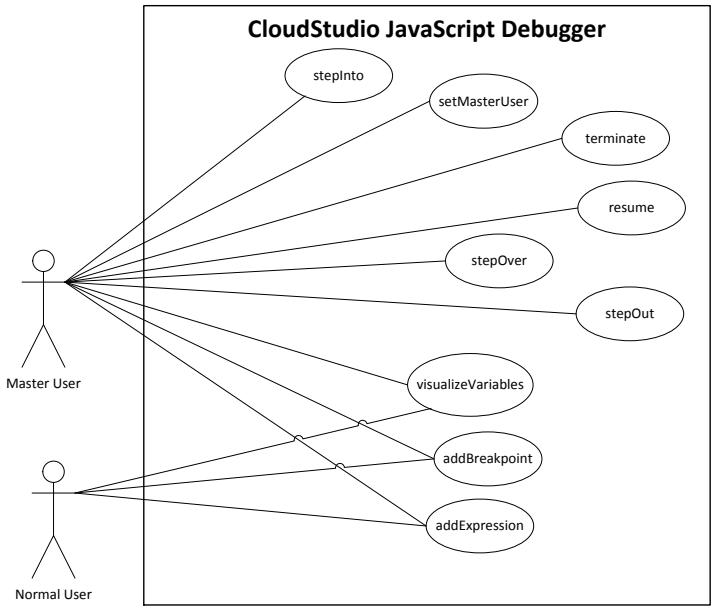


Figure 3.1: User roles and matching functionalities

Debugging modes

To maximize the collaboration of debugging between users, the CloudStudio Collaborative Remote Debugger is designed in a way to alow users to work in three different modes as illustrated in Figure 3.3. These modes are:

- **Single user mode:** The only user of the debugging session is the user which runs the debugger. It is the master user by default, since then it has full control over the debugger. This user can send invitation to other users and eventually add them to the current active debugging session. Doing so, the debug mode becomes *multi user single control*.
- **Multi user single control mode:** In this mode there are more than one user in the debugging session and only one of them is the master user. All other users are normal users. Either the debugging session starts in this mode or a *single user* debugging session enters into this mode by adding a second user to the debugging session.
- **Multi user single control mode:** In this mode all of the users in the debugging session are master users, since then all of them have full control over the debugger.

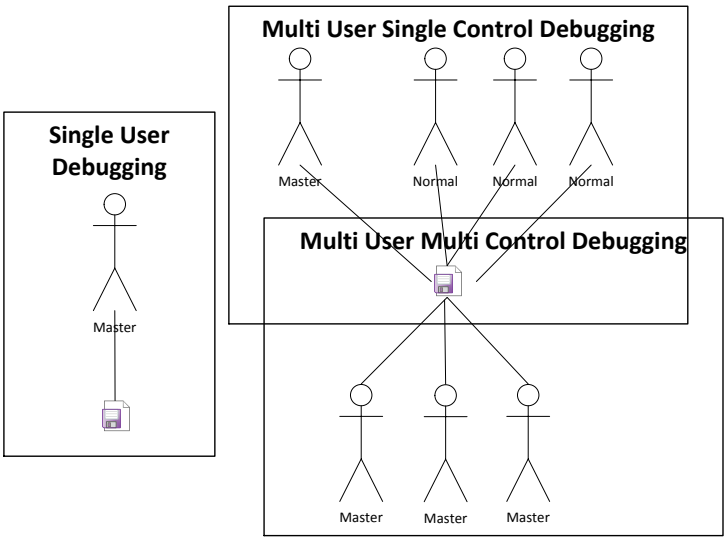


Figure 3.2: Debugging modes and matching user roles

Collaboration controls

Collaboration controls give users the possibility of managing the collaborative debugging session. These functionalities are:

- **Add User:** Users that are logged into CloudStudio IDE and not participating in the current active debugging session can be selected and an invitation to join to the current active debugging session will be sent to them. Though, the user can join a debugging session if and only if it is logged in the project of the debugging session. Otherwise both the inviter and the invitee receive a message which informs them that it is not possible to add the user to the current active debugging session. This option is enabled in every debugging mode. In case of a multi user single control debugging mode, only the master user can invite other users. In case of a multi user multi control debugging mode, since all users will have the full control over the debugger, all of them can invite other users.

- **Set Master User:** This functionality is enable only in the multi user single control debugging mode where there exists only one master user. The master user of the debugging session can assign another user as the master user. Doing this it will loose its role and become a normal user.

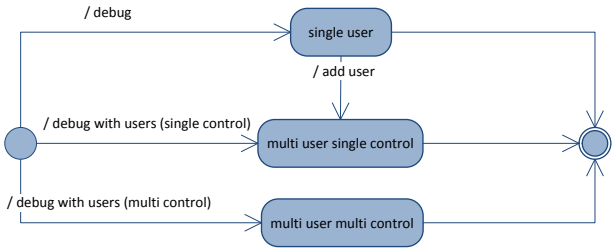


Figure 3.3: Debugging modes

Remoteness

- GWT defines an asynchronous communication between client and server.

- GWT defines a client side and another server side service should be defined. The client side service is in charge of sending asynchronous messages to the corresponding server side service.

- The communication of the client to the server is made only through these services described above.

- The communication of the server to the client is made through push notifications. GWT imposes that the server cannot call methods on the client side.

3.6 Complexity of implementation cycle

Extending the CloudStudio with a debugger which is collaborative and remote and for JavaScript comes along with many difficulties. The solution has different construction steps. First of all a JavScript debugger should be implemented. Then over this debugger, the collaborative and remote debugger should be implemented. Since the CloudStudio IDE supports several languages, the collaborative and remote debugger should be designed to work also with a future implementation of the debugger of another language. The JavaScript debugger should be exposed as if it was an API to the Collaborative Remote debugger. Doing so, another debugger can be implemented exposing the same methods to the Collaborative Remote debugger.

CloudStudio uses GWT RPC mechanism to make calls from client to server. GWT RPC is an asynchronous method to make the calls. Because of the the client requests are made through calling the methods on the server side asynchronously. The server replies by a callback to the user who made the request. However in a debugging session with many clients, a command comes only from one user and the server has to inform all other users about the state of the debugging session. Because of this requirement server has to use long-polling push to inform each user. This is enabled by using *events* that are described in chapter 4. Clients can handles the events that are pushed by the server and each event is handled in its block of code. Events should be designed in a very careful way to make it enable to update correctly the state of the debugging session on the client side. Each event has a different task to perform. For example a breakpoint hit should inform the clients about the line where the breakpoint is hit, so that the client can update the editor. Differently, an event about a user joining to a debugging session has to pop an information page which will disappear automatically.

GWT converts the client Java code into JavaScript and within the browser of the users there is only JavaScript code running for the client side. GWT RPC enables highly responsive user interfaces. While on the server side the

debugger is waiting on a breakpoint, actually a thread is waiting a notify from outside. However the same thing cannot be done on the client side since JavaScript has no support for threading and client side code cannot be suspended. Because of this, non debugger logic should be implemented on the client side. All the actual debugger mechanism should be on the server side. Otherwise it is not possible to implement a correct working debugger.

3.7 Summary

A JavaScript debugger extension which is encapsulated in a Collaborative Remote debugger is described in this chapter. The design is made to make possible extending the CloudStudio with another debugger without changed the Collaborative Remote debugger design. According to this design the implementation of the pure underlying debugger will be enough to integrate it as a collaborative remote debugger. To extend the CloudStudio, technologies such as *GWT* and *Rhino* are studied for the integration.

Chapter 4

Project and the implementation of the solution

CloudStudio is an IDE for several languages. These languages are currently Eiffel, JavaScript, Java and C#. So the IDE has to have a debugger of each language, and the encapsulating debugger which enriches them with collaboration and remote properties. The subject of this thesis is implementing a JavaScipt debugger within the CloudStudio and implementing a collaborative remote debugger which is supposed to orchestrate the underlying debuggers enriching them with collaboration and remoteness properties. The UML diagrams displayed in this chapter may be simplified compared to the actual implementation to improve the comprehensibility.

In section 4.2 the implementation of the JavaScript debugger is described. In sections 4.3 and 4.4 the implementation which enriches the JavaScript debugger with collaboration and remote properties are described. In section 4.5 the processing sequence of the requests are described.

4.1 The implementation

Since the solution described in section 3.5 divides the entire solution into two different components as **CloudStudio JavaScript Debugger** and **CloudStudio Collaborative Remote Debugger**, a bottom up strategy is followed for the implementation of the CloudStudio JavaScript debugger.

`JavaScriptDebugger` class is created to live on the server side and to serve as the JavaScript debugger. This debugger is tested alone for its functionalities.

For the implementation of the CloudStudio Collaborative Remote JavaScript debugger a top down strategy is followed. The key points of the collaboration and remote requirements, the integration with the CloudStudio components are studied carefully. The CloudStudio Collaborative Remote Debugger is composed of several classes that are created to live on the server. CloudStudio GUI and the controllers are changed to be enriched with functionalities to enable the CloudStudio Collaborative Remote Debugger.

4.2 CloudStudio JavaScript debugger

Since there will be other debuggers within the CloudStudio IDE, the debugger class is designed polymorphically. `CSDebugger` is the class which will be extended by each debugger later on. `JavaScriptDebugger` is the JavaScript debugger implementation which extends `CSDebugger`. This class has the basic debugger features described in 3.5.1. The classes which extend this class has to implement these methods in order to implement a complete debugger.

The implementation of the JavaScript debugger is realized by implementing the `Debugger`, `DebugFrame`, `ContextAction`, and `ContextFactory.Listener` interfaces of Rhino by `JavaScriptDebugger` class. `JavaScriptDebugger` receives *start*, *step into*, *step over*, *step out*, *resume*, *terminate*, *evaluate expression*, *set breakpoint* and *get variables* from outside and elaborates the operations.

Each `JavaScriptDebugger` instance runs in its own thread. `RunProxy` class which implements `Runnable` interface is in charge of starting a new thread and calling `JavaScriptDebugger.start()` method. For each debugging session a debugger is created, and for each debugger a thread is started. Because of this property, there occurs no interference problem between different debugging threads.

The default behaviour of the debugger is to start, execute the entire script and terminate if no breakpoint is set. When a debugger object is created with valid breakpoints, a `monitor` object is used to `wait`.

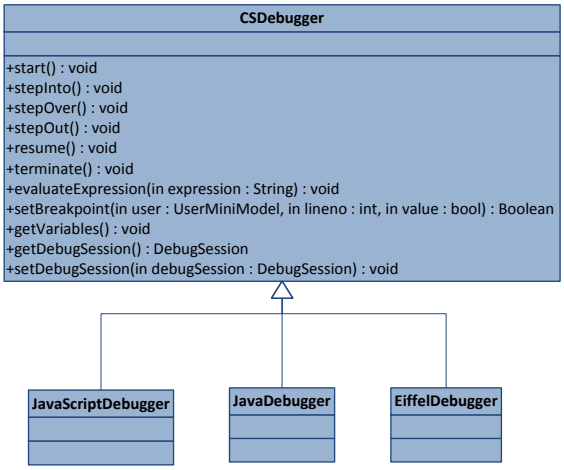


Figure 4.1: JavaScriptDebugger class

4.3 GWT RPC communication services

The server-side code that gets invoked from the client is often referred to as a service. The implementation of a GWT RPC service is based on the well-known Java servlet architecture. Within the client code, the developer uses an automatically-generated proxy class to make calls to the service. GWT handles serialization of the Java objects passing back and forth the arguments in the method calls and the return value. [26].

There are three steps of setting up a GWT RPC. The service that runs on the server whose methods are called by the clients, the client code which invokes the services, the Java data objects that pass between the client and server. Both the server and the client can serialize and deserialize objects.

In order to defined the RPC interface, the three components below should be written:

1. Defining the `DebuggerService` interface for the service which extends `com.google.gwt.user.client.rpc.RemoteService` and lists all of the RPC methods. This should be defined under the client package.

2. Defining the `DebuggerServiceImpl` class which extends `com.google.gwt.user.server.rpc.RemoteServiceServlet` and implements the interface created above.

3. Defining the asynchronous `DebuggerServiceAsynch` interface to the service to be called from the client-side code.

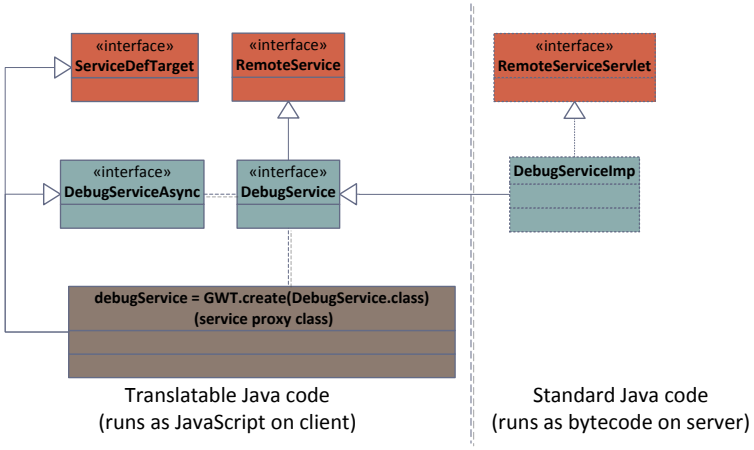


Figure 4.2: GWT RPC Class Diagram

The client side defines synchronous and asynchronous codes which are toghether in charge of sending asynchronous requests to the server side. The corresponding server side service which is an extendion of `com.google.gwt.user.server.rpc.RemoteServiceServlet` receives these requests and elaborates them. For this purpose `DebugService` and `DebugServiceAsynch` classes are created on the client side, and `DebugServiceImpl` class is created on the server side. `DebugServiceImpl` is the entry point of the server. The client sends its requests to this service on the server. The comunication back to the clients are made via long-polling server push.

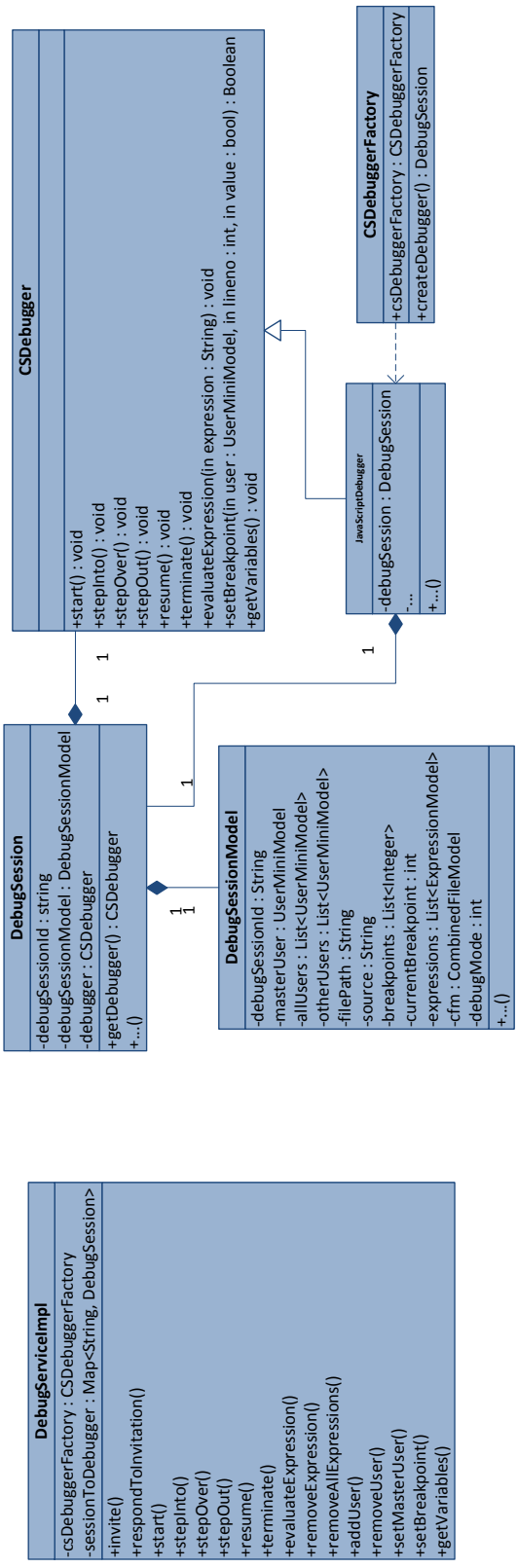


Figure 4.3: Class Diagrams

4.3.1 Models and events

In GWT the data is represented as models and they are serialized for transmission. Two models are created for transmitting the useful debugging data to the clients.

DebugSessionModel

`DebugSessionModel` is a serializable object and keeps the state of a debugging session. A `DebugSessionModel` object cannot be shared among different debugging sessions. It is created during the initialization of the debugging session and updated continuously during the lifetime of the debugging session. It keeps information about what is the *unique ID* of the debugging session, who is the *master user*, who are the *other users*, what is the *file path* of the debugging script, what is the *source* script, which lines contain *breakpoints*, which *line* the debugger currently waits on, what are the *expressions* that the debugger has to evaluate, what is the *debugging mode*.

ExpressionModel

`ExpressionModel` is a serializable object which contains information about each JavaScript expression added by the users. It holds a string of the expression and the results. It is updated continuously during the lifetime of the debugging session. Indeed, it is updated whenever the line of JavaScript code being debugged is changed. `ExpressionModel` is not specific to the JavaScript debugger. It can be used as a model for the expressions for any future implementation of another debugger.

JavaScriptVariableModel

`JavaScriptVariableModel` is a serializable object which contains information about each JavaScript variable in the JavaScript code being debugged. It holds the name of the JavaScript variable, the value, and a boolean that determines if it is an object or a primitive variable. `JavaScriptVariableModel` is encapsulated in the `JavaScriptVariableTreeModel`.

JavaScriptVariableTreeModel

`JavaScriptVariableTreeModel` is a serializable object which implements N-tree data structure. This tree represents the JavaScript variable as a tree, where the primitive variable is represented as a tree with just a root, and the object variables are represented as N-tree where the children are the field of the JavaScript object variable, which can have children as well if they are objects. `JavaScriptVariableTreeModel` is encapsulated in the `JavaScriptVariablesCollectionModel`

JavaScriptVariablesCollectionModel

`JavaScriptVariablesCollectionModel` is a serializable object which contains the list of all the JavaScript variables of the JavaScript code being debugged. It actually contains list of list, where the first one is the list of all the frames of the JavaScript code and each element of this list (which represent a frame) carries a list of the local variables of this frame of the JavaScript code. `JavaScriptVariablesCollectionModel` object is created during the initialization of the debugging session, actually it is belong to `DebugSessionModel` object, and is updated continuously during the lifetime of the debugging session. Indeed, it is updated whenever the line of JavaScript code being debugged is changed.

`JavaScriptExpressionModel` implements `VariablesCollectionModel` interface. For any future implementation of another debugger it will be enough implementing this interface and the rest of the code won't need any modification by this polymorphic design.

Events

Clients invoke the methods of the services on the server to make a request. In a traditional web application, mostly only the user who makes the request gets the response. However in the CloudStudio Collaborative Remote Debugger the clients form a debugging session and all of the users who participate in the same debugging session should receive the response of the request of a single client of the same debugging session. Server informs the CloudStudio clients about the updates via long-polling push. The clients open up a request which the server will not answer immediately but wait until there are events to push.

- `BreakpointHitEvent`: Whenever the debugger hits a breakpoint, all the clients receive the updated `DebugSessionModel` so that they can update their screens to show the current state of the debugging session. All users of the debugging session get informed of this event.

- `BreakpointSetOtherUsersEvent`: During and active debugging session, if a user sets or resets a breakpoint on some line the client sends this request to the server and after handling the request the server fires this event to inform other clients. All other users except the user who performed the action get informed of this event.

- `DebugCompletedEvent`: When a debugging session terminates, all users of the debugging session get informed of this event.

- `DebugInvitationEvent`: When a user invites some other users to join a debugging session, the invited users get informed of this event.

- `DebugInvitationResponseEvent`: When the invited users respond to a debugging session invitation, the user who invited gets informed of this event.

- `DebugMasterUserSetEvent`: When the master user of the debugging session is changed, all users get informed of this event.

- `DebugRequestedResponseEvent`: When a user starts a debugging session all users of the debugging session, or when a user adds other users to an active debugging session the added users get informed of this event.

- `ExceptionThrownEvent`: If the JavaScript script that is being debugged throws an exception, all users of the debugging session get informed of this event. The exception thrown is not an exception of CloudStudio but an exception of the JavaScript script that users debug.

- `ExpressionRemovedEvent`: When a user removes an expression from its view, since all users share the same view all other users get informed of this event.

- `UserLeftDebugSessionEvent`: When a user leaves the debugging session, other users get informed of this event.

4.3.2 Java components

Several methods that the clients need to invoke to control the debugging session and the debugger are defined in the `DebugService` interface. The implementation within the server side class `DebugServiceImpl` does not contain the actual implementation but they pass the request to the objects (which are described in section 4.4) that are in charge of handling the requests. The methods are:

- `invite(String projectId, String debugSessionId, String filePath, UserMiniModel masterUser, List<UserMiniModel> users)`: When *master user* tries to start a debugging session with multiple users or tries to add a user to an existing debugging session, this method is invoked to send an invitation to the invited user.

- `respontToInvitation(String debugSessionId, UserMiniModel user, String response, UserMiniModel masterUser)`: When a user receives an invitation to join a debugging session it has the possibility to `ACCEPT` or `REJECT` it. This is the case if it was logged in to the same project which it was invited to. If it wasn't logged in to the project which it was invited to, it is not added to the debugging session and it receives a message that it was invited to a debugging session and in order to join to the session it has to log in to the project of the invitation. Also the inviting user receives a similar message.

- `start(ProjectModel project, CombinedFileModel cfm, UserMiniModel currentUser, List<UserMiniModel> selectedUsers, String filePath, String source, List<Integer> breakpoints, int debugMode)`: This method is invoked in two cases, when a user requests a single user debugging session, or when it requests one of the multiple users debugging sessions and receives a positive or negative response from all of the users. This method fires the starting of the actual debugging session. The file, source script, master user, other users, file path, breakpoints and debugging mode information are transmitted.

- `stepInto(String debugSessioId)`: Requests from `DebugSession` a step into command.

- `stepOver(String debugSessioId)`: Requests from `DebugSession` a step over command.
- `stepOut(String debugSessioId)`: Requests from `DebugSession` a step out command.
- `resume(String debugSessioId)`: Requests from `DebugSession` a resume command.
- `terminate(String debugSessioId)`: Requests from `DebugSession` a terminate command.
- `evaluateExpression(String debugSessioId, String expression)`: Requests from `DebugSession` to execute the expression on the debugger.
- `removeExpression(String debugSessionId, int index)`: Requests from `DebugSession` to remove the expression at *index* on the debuger.
- `removeAllExpressions(String debugSessionId)`: Requests from `DebugSession` to remove all the expressions on the debugger.
- `addUser(String debugSessioId, UserMiniModel user)`: This method is invoked when a user accepts an invitation to an active debugging session. Requests from `DebugSession` to add the user to the debugging session.
- `removeUser(String debugSessioId, UserMiniModel user, boolean hideDebugToolbar)`: This method is invoked when a user leaves the active debugging session. Requests from `DebugSession` to remove the user from the debugging session.
- `setMasterUser(String debugSessioId, UserMiniModel masterUser, boolean masterUserLeft)`: This method is invoked either when the *master user* leaves an active debugging session or passes the role to a user it chose. Requests from `DebugSession` to set the new master user.
- `setBreakpoint(UserMiniModel user, String debugSessioId, int lineno, boolean value)`: A breakpoint is set or an existing breakpoint is reset on the line. Requests from `DebugSession` to perform the operation on the debugger.

- `getVariables(String debugSessioId)`: Returns all of the variables of the debugging session. Requests from `DebugSession` to get all the variable from the debugger.

4.4 Backend

CloudStudio Collaborative Remote Debugger is implemented on top of the debuggers to enrich them with collaboration and remote properties. Clients see only one debugger with more functionalities than a traditional debugger. However in the implementation the *CloudStudio JavaScript Debugger* is encapsulated with the *CloudStudio Collaborative Remote Debugger*.

Entry point for the collaborative remote debugger is the services described in section 4.3. There can be either a single user debugging or multiple user debugging. In the latter case the initialization of the debugging session starts immediately. In the multiple user debugging once the invitations are sent to the users, and each user responded to the invitation, the initialization of the debugging session can start.

4.5 Processing sequence

Starting a debugging session

As described in section 3.5.2 the solution for the CloudStudio Collaborative Remote Debugger foresees the implementation of a debugging session in three different modes. They are *single user mode*, *multi user single control mode* and *multi user multi control mode*. Starting a debugging session in each mode has a different workflow between client or clients and the server.

Prerequisites

A user cannot request a second debugging session or join to a second debugging session before terminating the active debugging session. However a user can receive a debugging session request while being is an active debugging session. In the last case, if the user accepts joining to the debugging session three scenarios can happen:

- If the active debugging session is a single user mode debugging session then the client automatically sends a termination request of the session.
- If the active debugging session is a multi user single/multi control mode debugging session and if the user is the master user of the debugging session then the client automatically sends a master user set request. The next user in the users list becomes the new master user of the debugging session.
- If the active debugging session is a multi user single/multi control mode debugging session and if the user is not the master user of the debugging session then the client automatically sends a remove user request.

Phases

Debugging session starting has 3 phases.

- **Invitation phase:** Debugging session invitations and responses take place in this phase. In a single user debugging mode this is not the case since there is only one user for the debugging session.
- **Object creation phase:** Objects that are required to start a debugging session are created in this phase. These objects are `DebugSession`, `DebugSessionModel` and `JavaScriptDebugger`
- **Thread creation and starting the debugger phase:** In that phase a new thread is created for the debugging session and the debugger's start method is invoked.

Invitation phase

This phase takes place if and only if the user requests a *multi user single/multi control* debugging. The user can invite only online users. The steps of this phase are:

- The client invokes `DebugServiceImpl.invite(...)` method to send an invitation to the selected users.
- `DebugServceImpl` sends an invitation to each user by a push notification through `DebugInvitationEvent` event.

- Here there are two scenarios:
  - **The invited user is logged in to the project it was invited to:** The invited user can *ACCEPT* or *REJECT* the invitation. The client invokes `DebugServiceImpl.respondToInvitation(...)` method to send the response back to the user who invited by a push notification through `DebugInvitationResponseEvent`.
  - **The invited user is not logged in to the project it was invited to:** When the client receives the invitation, it show to the invited user the message below and Automatically invokes `DebugServiceImpl.respondToInvitation(...)` passing *NOT_ONLINE* argument. A push notification is sent to the user who invited by a push notification through `DebugInvitationResponseEvent`

    "*You are invited to debug session of project: Project X of file: /src/Main.js by userA. In order to join the debugging session you have to log in the project and ask master user to add you to the debugging session.*"

    After that `DebugServiceImpl` sends the information to the user who invited by a push notification and on it's screen the message below is shown:

    "*userB cannot be added to the debug session because the user is not logged in this project* "

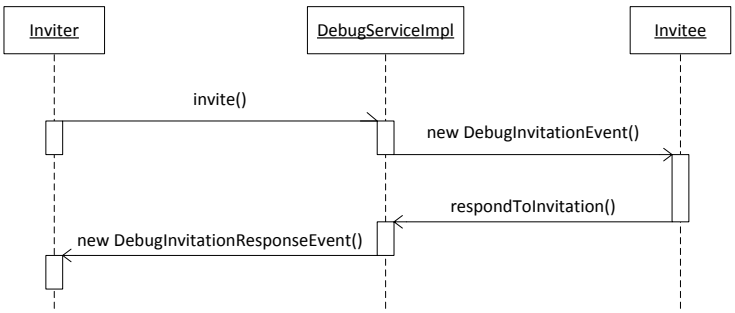


Figure 4.4: Invitation phase sequence diagram

When for each user a response comes back from the server, the object creation phase starts. The object creation phase starts by the client's invoking `DebugServiceImpl.start(...)` method.

Object creation phase

`DebugServiceImpl` calls `CSDebuggerFactory.createDebugger(...)` method. `CSDebuggerFactory` is a *singleton*. For the object creation phase, `CSDebuggerFactory` does these steps:

- An instance of `JavaScriptDebugger` is created and assigned to a `CSDebugger` reference.
- An instance of `DebugSession` is created and its debugger is set as the previously created debugger. `DebugSession` then creates the `DebugSessionModel` object which will be returned to the client then.

`DebugSession` and `CSDebugger::JavaScriptDebugger` contain references to each other. The reason is that the debugging session object controls the debugger object and the debugger object asks to the debugging session object to fire events on the server to the clients.

`CSDebuggerFactory` keeps in a map all of the alive `DebugSession` objects and handles them as the requests come from `DebugServiceImpl`. The alive `DebugSession` ojbect is removed from the map when the `DebugSession` tells `CSDebuggerFactory` that the debugging session has terminated and there is no need to keep the object.

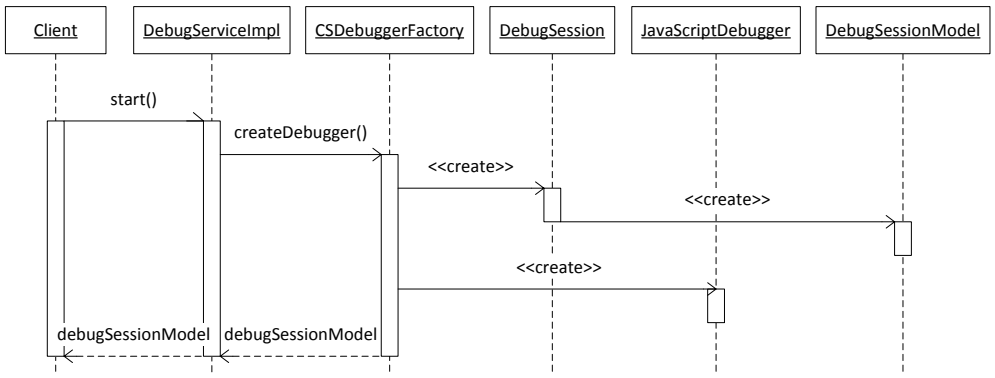


Figure 4.5: Object creation phase sequence diagram

Thread creation and running the debugger phase

After the creation of the objects that are needed, the creation of the thread and running the debugger takes place. These steps are followed:

- `CSDebuggerFactory` creates a new instance of `RunProxy`. `RunProxy` implements `Runnable` interface.
- A new `Thread` object is created by passing the `RunProxy` object as argument.
- Within `RunProxy.run()` method `JavaScriptDebugger.start()` is invoked.

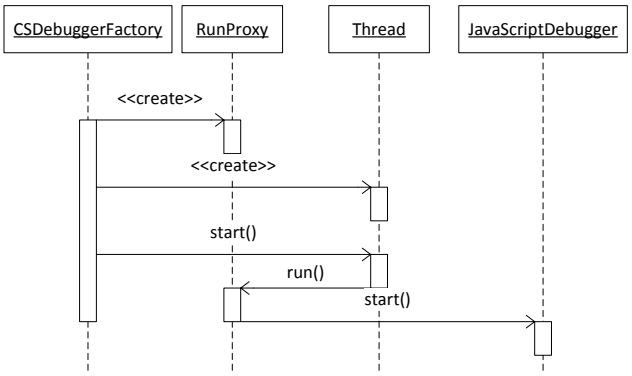


Figure 4.6: Thread creation phase sequence diagram

`DebugSession` has references to `DebugSessionModel` and `CSDebugger`. It receives the request from `DebugServiceImpl`. For the basic debugger operations like step into or resume it controls the debugger. `DebugSession` handles alone the collaboration related operations like adding a new user to the debugging session. All of the events directed to the clients are fired by `DebugSession`. Other objects can request `DebugSession` to fire the events.

Listing 4.1: Java Pseudo Code: Initialization of the debugging session

```
1 String debugSessionId =
2       project.getPid() + "_" + users.hashCode();
3
4 CSDebugger debugger = null;
5 DebugSession debugSession = new DebugSession(debugSessionId,
      debugger, masterUser, users, filePath, source,
     breakpoints, -1, cfm, debugMode);
```

```
6
7  if(project.getLanguage().equals("JavaScript")) {
8    debugger = new JavaScriptDebugger(debugSession, source,
         breakpoints);
9  }
10 debugSession.setDebugger(debugger);
11 sessionToDebugger.put(debugSessionId, debugSession);
12
13 RunProxy proxy = new RunProxy(debugger);
14 new Thread(proxy, debugSessionId).start();
15
16 return debugSession.getDebugSessionModel();
```

When a `JavaScriptDebugger` starts running the debugger within its own thread, it requests from `DebugSession` to send a `DebugRequestedResponseEvent` event which transmits an instance of `DebugSessionModel` to the users of the debugging session. Doing so, all users that joined to the debugging session update their state of debugging.

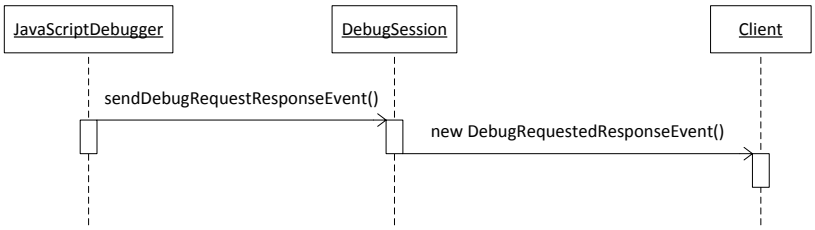


Figure 4.7: Debugging response sequence diagram

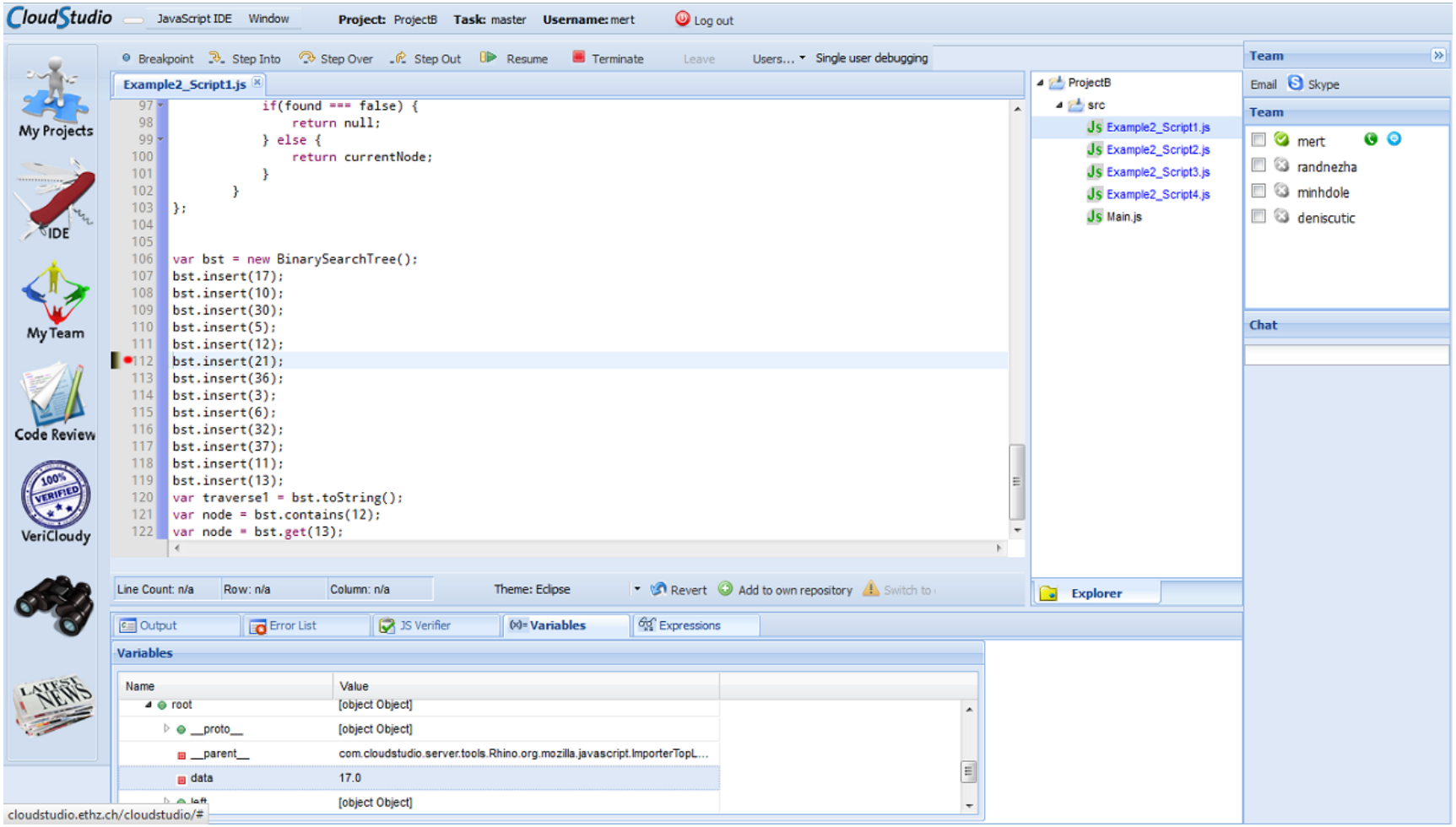


Figure 4.8: CloudStudio IDE debugging session

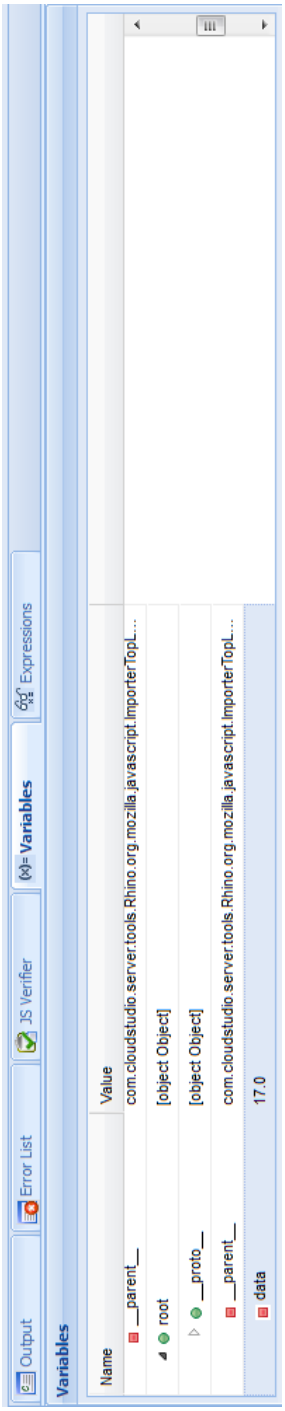


Figure 4.9: CloudStudio IDE variables tab

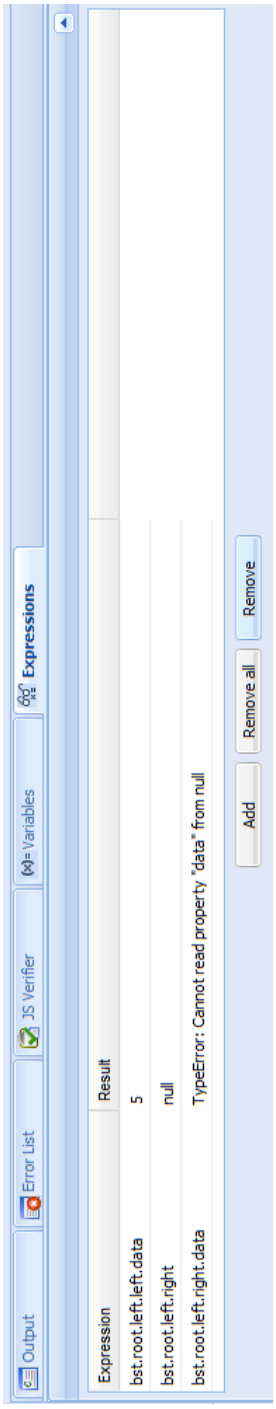


Figure 4.10: CloudStudio IDE expressions tab

4.6 Summary

The implementation of the CloudStudio Collaborative Remote JavaScript debugger is composed of several sub-implementations. First of all a pure underlying JavaScript debugger is implemented. The encapsulating Collaborative Remote debugger has the responsibility of passing the data from client to server and vice versa. Client requests are made through asyncronoush RPC calls to the server and the server responses are made via long-poll push mechanism.

Chapter 5

Case study

5.1 Introduction

The main aspect of the CloudStudio Collaborative Remote Debugger is its enabling the collaboration between users on a geographically distributed software development environment. Traditionally the collaborative debugging, which is a session in a distributed environment, is realized through a remote connection program such as TeamViewer. The user who has the IDE shares his screen with other users and they all together drive the debugging session. To analyze the effects and the usability of the CloudStudio Collaborative Remote Debugger over the traditional collaborative debugging methods a case study is designed.

The case study is basically designed to measure how innovative and efficient CloudStudio Collaborative Remote Debugger is. This case study is not for measuring the performance of this tool, since the collaborative debugger is a part of the CloudStudio IDE, so it is very difficult to measure the performance of this tool independently from the whole CloudStudio project.

Basically the idea is to be able to respond to the following main questions:

1. What is the advantage to use CloudStudio Collaborative Remote Debugger over using the existing technologies which can help the collaborative worker in their shared work? For example video conference, chatting, remote desktop sharing.

2. A platform that offers debugging in a collaborative way in real time, is it really what the developers need in their real projects?

3. Does the Cloudstudio Collaborative Remote Debugger respond efficiently to the needs of the collaborative workers and help them to collaboratively find the bugs in their codes?

4. How can a collaborative tool, like the CloudStudio Collaborative Remote Debugger, affect the activities of team's members when they try to reach together a shared goal?

The case study began in a cool way. We organized an event on Facebook to invite the developers to join us and try the tool. The event was called "let's try debugging collaboratively" to make the persons more curios about the tool. Then the participants were grouped into several groups to perform two assignments (one using CloudStudio multi user collaborative debugger, and the other using CloudStudio single user and remote shared screen), each assignment contains several debugging tasks.

5.2 JavaScript projects

In order to perform a valid case study the JavaScript projects, which are used in the assignments, should be chosen carefully. The projects should have the same level of the complexity. So, for that reason the choice of JavaScript projects is directed into known algorithms that implement famous data structures. In this way the developers will be familiar a bit with the code which they should found the bugs in.

The JavaScript projects implement: **linked list** and **binary search tree** data structures. Each one contains many functions such as add, remove, get, and traverse. The JavaScript project is organized in 4 scripts, and in each script a bug is inserted. The bugs don't have extreme difficulty level, that's to avoid the developers to get stuck in the code rather than trying the tool.

5.3 Setting of assignments

The assignments use the JavaScript projects described above, there are two type of assignment. The matrix types of the assignments depends on:

JavaScript project: The project used as code sample which contains bugs to be recognized by the group's members: *Linked list* or *binary search tree* projects.

Collaborative tools: The tool used to collaborate among the group's members to find the bugs in the codes:

- **CloudStudio Collaborative Remote Debugger:** The group's members can use all the collaborative functionalities of the CloudStudio Collaborative Remote Debugger.
- **CloudStudio Collaborative Remote Debugger single user and shared screen:** One developer from the group logs in to the CloudStudio to use the single user debugger, the other developers connect remotely to the desktop of the developer who is logged in to CloudStudio (using Team Viewer).

The duration of the assignments is determined to be 90 minutes, but in any case a timeout is determined for each assignment. The timeout for the first assignment is 1:15h and the timeout for the second assignment is 1:00h. So when the group reaches the timeout while performing an assignment, the group's members have to finish the current task which they are working on, for then moving to the next step of the case study (second assignment or the questionnaire). The difference in the duration between the first assignment and the second one is for the expectation that the participants will spend more time in the first assignment where in the next one will be more familiar with the tool, the tasks style, and the whole process in general.

The assignment can be one of the following types:

- Assignment uses *linked list* project and CloudStudio Collaborative Remote Debugger.
- Assignment uses *Linked list* project and CloudStudio Collaborative Remote Debugger single user and shared remote screen.
- Assignment uses *Binary search tree* project and CloudStudio Collaborative Remote Debugger.

- Assignment uses *Binary search tree* project and CloudStudio Collaborative Remote Debugger single user and shared remote screen.

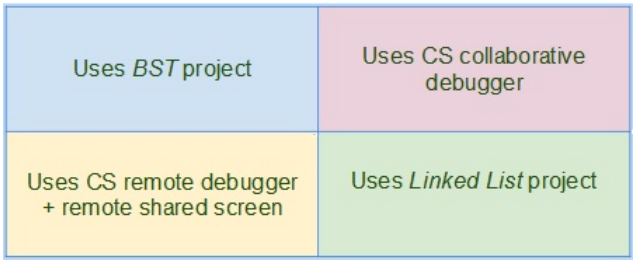


Figure 5.1: Assignments types matrix

Assignment's tasks

Each assignment has 5 tasks to be solved by the team members. The tasks are organized in a gradual level of complexity. In this way the members become more familiar with the tool during the assignments. The first task of each assignment is designed to be primitive, where are asked exercises to let the participants play with the features of the tool, and to guarantee that the participants know where they can find what they need, and they have been introduced well to all the parts of the tool.

The first task of each assignment contains exercises such as: navigate in the code until the line L, search for the variable V, evaluate the expression E, pass the control to the next user, and so on.

The later tasks are more complex, is asked to evaluate some variables at a specific lines, and see if they contain the correct provided values, if not they have to search for the cause of the bug by using the debugger, and report it briefly by specifying the line of the bug in the code. Other tasks ask the participants to search for the cause of an exception thrown by a piece of code, like null pointer exceptions.

5.4 Setting of groups

After collecting developers distributed in different countries to join us, groups are created to begin the case study. Actually the groups are designed in efficient way that produce valid results. A group can be one of the following types:

Group A: This group performs the assignments in the following order. First: Assignment uses *Linked list* project and CloudStudio Collaborative Remote Debugger single user and shared remote screen. Next: Assignment uses *Binary search tree* project and CloudStudio Collaborative Remote Debugger.

Group B: This group performs the assignments in the following order.

First: Assignment uses *linked list* project and CloudStudio remote collaborative debugger. Next: Assignment uses *Binary search tree* project and CloudStudio remote debugger single user and shared remote screen.

The matrix of group types depends on: The number of the members and the type of the group (A or B).

Any group should be one of the following categories:

- Group type A and contains 2 members.
- Group type A and contains 3 members.
- Group type B and contains 2 members.
- Group type B and contains 3 members.

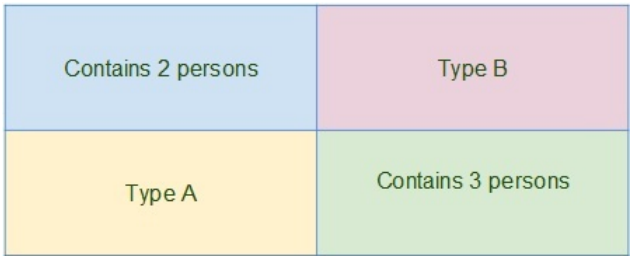


Figure 5.2: Groups types matrix

5.5 Logs

A logging mechanism is integrated to the CloudStudio to monitor the behavior of the users during the case study. The reason of keeping the logs is analyzing the user behavior trends during shared screen and collaborative assignments. For each debugging session a log file is created. Every communication from client to server is inserted to log files. This is made to monitor the commands of

the clients. However the communication from server to clients are not logged. In the log file these information can be found:

- Starting time of the debugging session
- Debugging mode
- Initial master user
- Initial other users
- Script path
- Initial breakpoints
- Command time
- Command name
- User who performed the command
- End time

Each log file is inserted in the corresponding folders. The folders are structured as:

GroupName

— Assignment1
— Assignment2

An analyzing tool is implemented to parse all of the log files and give the aggregated data for each group, assignment, and task. The analyzer produces a CSV file. Doing so, the data that is produced during the case study became easy to analyze. After having inserted all of the log files to their corresponding folders, the analyzer program is run and the CSV file is produced.

5.6 Questionnaire

Besides analyzing the logs during the assignments, we decided to analyze the group opinions as well. A questionnaire is designed to be filled in by the participants after finishing the assignments. The questionnaire focuses on the difference between using CloudStudio Collaborative Remote Debugger and CloudStudio Collaborative Remote Debugger single user plus shared remote screen. Many questions are asked to receive feedback of the efficiency of the collaborative aspect of CloudStudio Collaborative Remote Debugger, and how the participants feel the difference between the two experiences of the both tools. The questionnaire is divided into 4 sections, in the following they are described briefly:

General information: This section contains questions asked to get general information about the participant, for example the participant is asked for his name, his email, his group letter (A, B) which will be provided to the participants in the preparing emails (see the section .6), and the number of the members in the participant's group.

Experiences: This section contains questions asked to get information about the experience level of the participant. We believe that the experience level of the participants can effect how they involve in the whole process, for example if the participant has no experience in debugging software, so he will prefer being observer rather than controller of the debugger. In addition, a participant who has a higher Javascript experience level will encourage him to involve more than the others who don't have such a high level, even if the projects implement general algorithms and don't contain specific JavaScript code.

In this section the participant is also asked if he participate in any type of collaborative debugging session before, and if so he is asked to provide the kind of this collaborative debugging used (together in front of the same computer, software that allows to watch another person's screen, remote access to a machine, and other)

Debugging using a shared screen: This section contains questions about the participant's experience in the collaborative debugging using shared remote

screen during the assignments. It focuses mainly on how the participants missed collaborative tools that let them work in a real collaborative way.

The questions asked in this section are for example: The role of the participant, if he was the controller or the observer, how often the participant told the person who controlled the debugger what to do by using chatting tools, how much the participant actively involved in the debugging session. There are similar questions asked to see how much the participant missed the presence of actual collaborative tool like: everybody can add breakpoints, everybody can see variables, everybody can browse the code individually, everybody can control the debugger, etc. At the end of this section the participant is asked a question about how much efficient the collaborative debugging using remote shared screen is.

Debugging using CloudStudio Collaborative Remote Debugger: This section contains questions about the participant's experience in the collaborative debugging using the CloudStudio Collaborative Remote Debugger during the assignments. it focuses on how the participants feel the efficiency of the collaborative tools that let them work in real collaborative way.

The questions asked in this sections are for example: how much the participant actively involved in the debugging session, how often the participant controlled the debugger, e.g. add breakpoints, investigate expressions, etc. There are similar questions asked to see how efficient the collaborative tool is, for example: how much useful that everybody can add breakpoints, everybody can see variables, everybody can browse the code individually, everybody can control the debugger, etc. The participant is also asked about the efficiency of the several CloudStudio collaborative debugger modes (single control, multi control). At the end of this section the participant is asked a question about how much efficient the collaborative debugging using remote shared screen is.

Other information: In this section the participant is asked simply to choose which tool he preferred during the assignments between shared screen debugging and CloudStudio Collaborative Remote Debugger, and if he has another comments to provide.

The questionnaire can be found the appendix .5.

5.7 Preparing the groups

To get valid results from this case study, we prepared the groups for the participating in a very careful way. We were very careful about making sure that the participants don't know the subject of the case study before the starting date, making sure that the participants don't know about the second assignment before finishing the first one, making sure that the participants don't know about the questionnaire before finishing the second assignment, making sure that the participants have already installed all the required tools before the starting time, and others standards to make sure about the validity of the case study results. The members of each group received an email one day before their scheduled time for participation. The email contains the following main information:

- The time and the date of the participation.
- Group member's names to be able to contact each other before the starting time.
- Required tools, and an encouragement to prepare them before the starting time.
- CloudStudio Collaborative Remote Debugger guide. This guide is designed just for this purpose which introduces the main technical aspect of the tool. It explains how to use the buttons, the tool bars, and the tabs in the CloudStudio Collaborative Remote Debugger. (the guide can be found in the appendix .5)
- Some important notes that guarantee the validity of the whole process, for example to not to change the JavaScript code, since by changing it they will utilize another tools of the CloudStudio which is not an aspect of this case study.

Then Members of the group received 5 minutes before the starting time another email, to guide them how to start the process. This email contains the following main information:

- How to proceed: explaining in general the whole process, that they have to perform two assignments, and when they finish the first one they will

receive the second assignment and when they finish it they will receive the questionnaire.

- Information about the duration and timeout of the assignments.
- Their group letter (A or B) they don't know what this letter mean, they just use it in the questionnaire.
- The link to the first assignment and how they can access it (all the case study documents were hosted on Google Docs)
- Reminder for the required tools.
- Our contact information to contact us if they need any help.

The templates of the emails can be found in the appendix .6.

5.8 Case study challenges and problems

This case study is designed in a very careful way that can we can obtain valid and efficient results and analysis. We were very careful in preparing every single step, writing the assignments and tasks, choosing the JavaScript projects, writing the guide tool, how to prepare the groups, designing the questionnaire, and each detail. During the preparation of this case study we faced some of problems and challenges, some of them depended on the nature of the tool and others on the nature of the assignments, and others depended even on the participants. In the following are described some of the main problems we faced during the case study design, and the actual performing by the participants.

5.8.1 Organizing the groups

The main problem was collecting the participants and scheduling them in groups. This case study is designed in very standard way, that can be applied on any number of participants without any additional efforts. We believe that more the participants are, more the result are real and valid. So we managed to collect at least 20 persons (for the thesis case study) to participate and organize them in the following way:

2 groups of 2 members of the Type *Group A*

2 groups of 2 members of the Type *Group B*

2 groups of 3 members of the Type *Group A*

2 groups of 3 members of the Type *Group B*

But unfortunately, it was very difficult to collect 20 persons who can give us 2 hours easily, so the final participants number in this case study is 13. In addition to the difficulty of collecting persons, there was the difficultly of arranging them in suitable time schedules which should be based on their preferences. To resolve this problem we created a time table for determined days and let the participants fill it in as for their availability. Then we organized the groups based on the matching in time stamps. But as it is expected it was impossible to get the groups as we wanted, so we were forced to organize the groups like following:

2 groups of 2 members of the Type *Group A*

3 groups of 2 members of the Type *Group B*

0 groups of 3 members of the Type *Group A*

1 groups of 3 members of the Type *Group B*

5.8.2 Duration of the assignment

Each assignment is designed to be performed in 45 minutes, but actually this duration can take place just in perfect cases, where all the participants start in a synchronized way, the participants don't lose time in external stuff like connection problems, the participants take a reasonable time in finding the bugs since they are developers, the participants had already installed all the required tools before starting, the participants have already read the guide of the tool so they don't lose time in discovering how the buttons work, and other unexpected behavior can effect the duration of the assignments. Since the participants were volunteers, so we couldn't expect that they will perform everything in the perfect cases. for that reason we expected that they will take more than the determined time for finishing the assignments, and for not to "steal" persons' time we decided to determine a timeout as an initial solution for this problem. Even if the timeout resolve the problem of stealing persons' time, but it creates another type of problems, with timeout some groups will finish more tasks than others, and it depends on many factors.

Actually we resolved this problem by tracking the logs which allow us to see when they start/finish each task, and we took this information into account when evaluating the case study results.

5.9 Results

5.9.1 Analysis of the logs

Logs that were recorded automatically during the case study are analyzed to get some trends on how the CloudStudio Collaborative Remote Debugger effects the behavior of the users.

- Total time spent for the shared screen debugging is more than the collaborative debugging.
- Number of actions performed in the collaborative debugger is evenly distributed. It is possible to say that the CloudStudio Collaborative Remote Debugger enables the collaboration between geographically distributed developers while debugging the same code.
- There are more actions performed during the collaborative debugging than the shared screen debugging.
- There are more breakpoints for the shared screen debugging.
- During the shared screen debugging there were initialized more debugging sessions than the collaborative debugging mode.

5.9.2 Analysis of the questionnaire responses

The results say that the participants in the shared screen debugging missed the collaborative features that the CloudStudio Collaborative Remote Debugger offers, which help them debugging the code in a real collaborative way like everyone can add breakpoints, every one can see the code individually, everyone can add expressions, and others. Most of the participants felt the efficiency of the collaborative features that CloudStudio collaborative debugger offers. Here in the following the responses of the participant are analyzed, where in

some of questions they are asked to give a votes for several aspects, all the votes are from 1 to 5 (*'not at all'* to *'very much'*).

Seeing the variables individually: With CloudStudio collaborative debugging The 46% of the participants gave 5/5 for the efficiency of the fact that everyone can see the variable individually, and other 46 % gave 4/5.

While with shared screen debugging 23% of the participants gave 1/5 for how much they missed that every body can see variables individually, 23% gave 2/5, 15% gave 3/5, 23% gave 4/5, and 15% gave 5/5

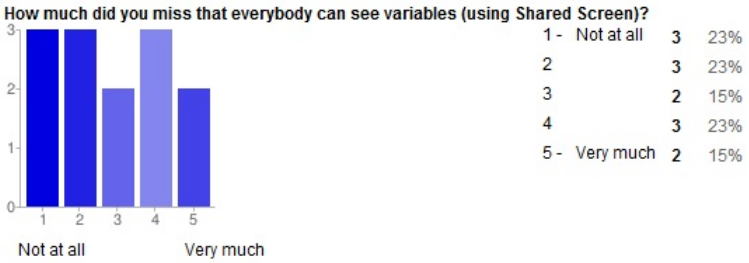


Figure 5.3: Missing Collaborative variable evaluation

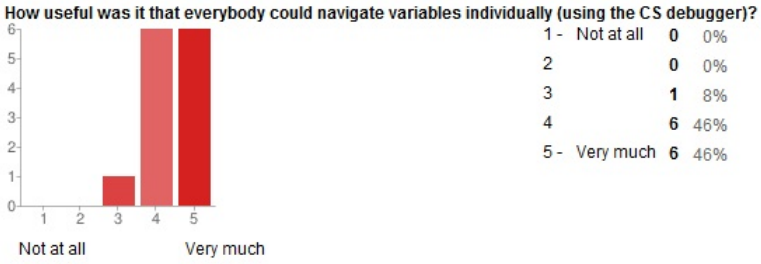


Figure 5.4: Collaborative variable evaluation efficiency

Seeing the code individually: With CloudStudio collaborative debugging the 69 % of the participants gave 5/5 for the efficiency of the fact that everyone can see the code individually.

While with shared screen debugging, 15% gave 2/5 for how much they missed that everybody can browse the code individually, 15% gave 3/5, 8% gave 4/5, and 23% gave 5/5.

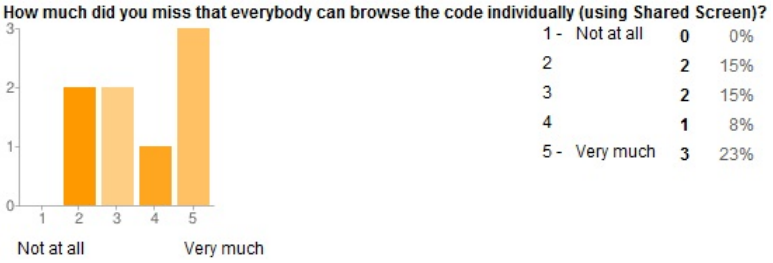


Figure 5.5: Missing Individual code visualization

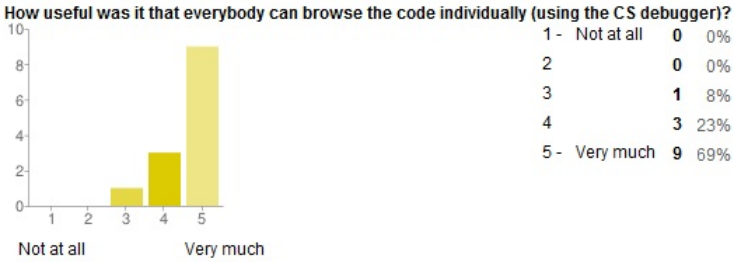


Figure 5.6: Individual code visualization efficiency

Adding/sharing breakpoints: With CloudStudio collaborative debugging the 15 % of the participants gave 2/5 for the efficiency of the fact that everyone can add/share breakpoints, 23% gave 3/5, 23% gave 4/5, 38% gave 5/5.

While with shared screen debugging, 23% gave 1/5 for how much they missed that everyone can add/share breakpoints, 23% gave 2/5, 15% gave 3/5, 23% gave 4/5, and 15% gave 5/5.

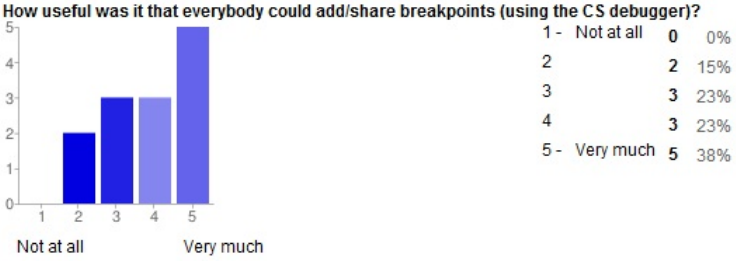


Figure 5.7: Collaborative breakpoint efficiency

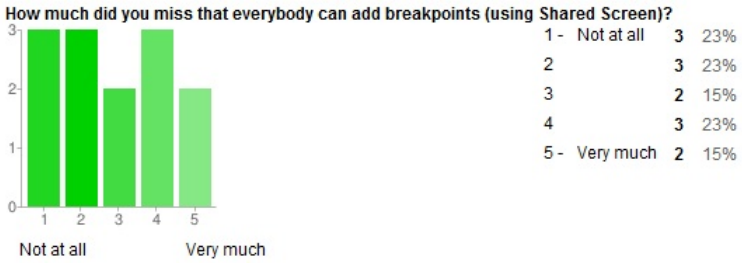


Figure 5.8: Missing Collaborative breakpoint

Adding/sharing expressions: With CloudStudio collaborative debugging the 15% of the participants gave 3/5 for the efficiency of the fact that everyone can add/share expressions, 54% gave 4/5, and 31% gave 5/5.

While with shared screen debugging, 15% gave 1/5 for how much they missed that everyone can add/share expressions, 8% gave 2/5, 31% gave 3/5, 31% gave 4/5, and 15% gave 5/5.

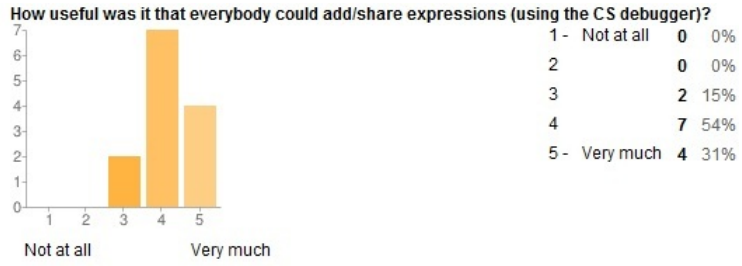


Figure 5.9: Collaborative expression evaluation efficiency

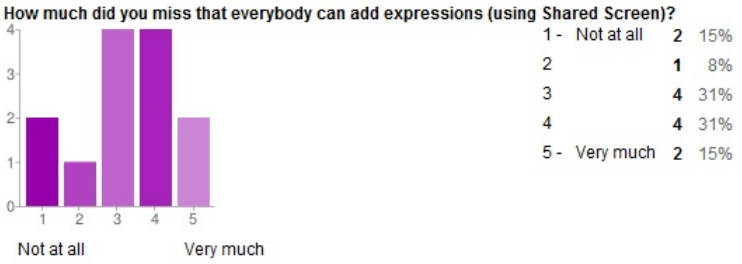


Figure 5.10: Missing collaborative expression evaluation

With users multi control debugger mode: With CloudStudio collaborative debugging the 8% of the participants gave 2/5 for the efficient of the debugger mode *with users multi control*, 15% gave 3/5, 46% gave 4/5, and 31% gave 5/5.

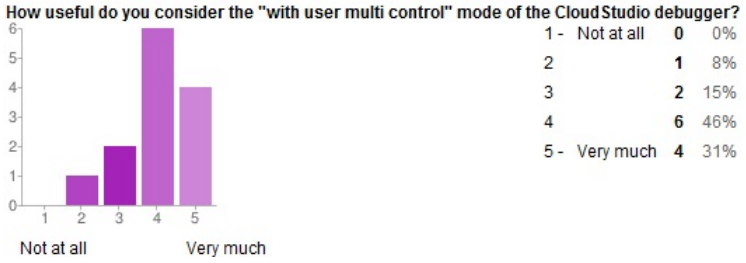


Figure 5.11: Debugger mode with users multi control

With users, single control debugger mode: With CloudStudio collaborative debugging the 23% of the participants gave 2/5 for the efficient of the debugger mode *with users, single control*, 23% gave 3/5, 46% gave 4/5, and 8% gave 5/5.

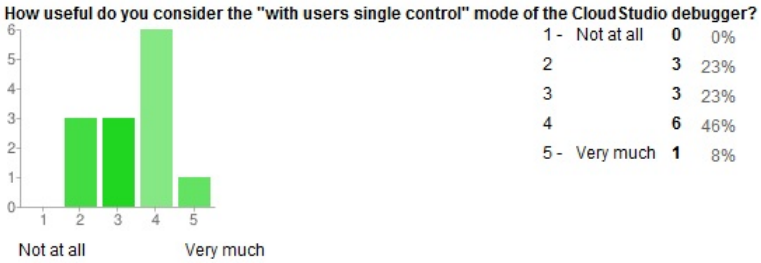


Figure 5.12: Debugger mode with users single control

Controlling the debugger: During the collaborative debugging using CloudStudio collaborative debugger 46% controlled the debugger 4-7 times, 15% controlled the debugger 8-12 times, and 38% controlled the debugger more than 12 times.

While during the Shared screen debugging 23% of the participants were the controller of the debugger(the participant who loged in to CloudStudio from his own machine), 31% of the participants tell the controller what to do 1-3 time, 23% 4-7 times, 15% 8-12 times, and 8% more than 12 times.

So the all participants during collaborative debugging, using CloudStudio Collaborative debugger, involved in controlling the debugger in collaborative way, where in the shared screen debugger were busy in telling the controller what to do rather than being able to intervene in the work by themselves.

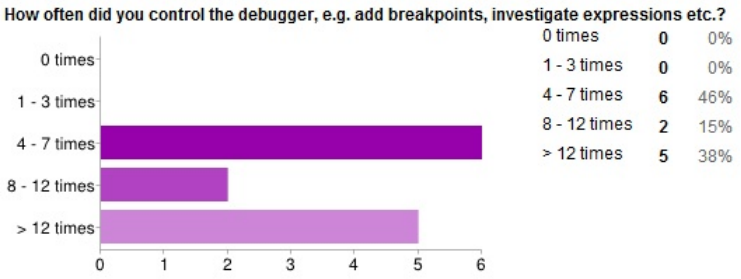


Figure 5.13: How much the participants control the debugger

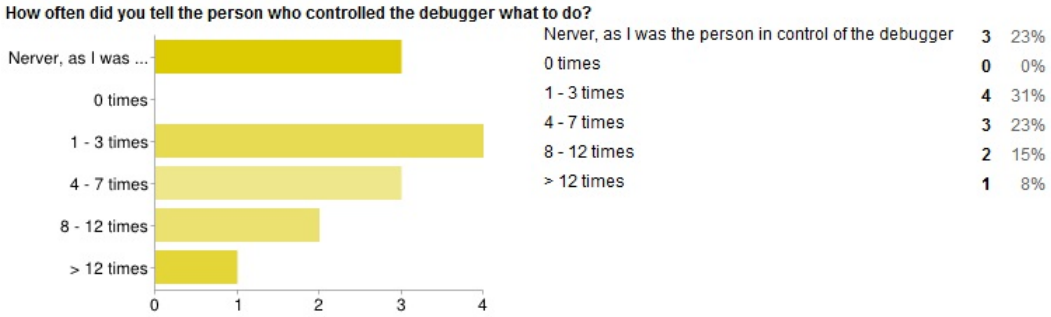


Figure 5.14: How much the participants tell the controller what to do

Involving in the debugging session: As a vote form 1 to 5 (*'not at all'* to *'very much'*) for how much they were involving in the debugging session. During the debugging using CloudStudio collaborative debugger 15% gave the vote 3, 38% gave the vote 4, and 46% gave the vote 5.

While during the shared screen debugging 8% gave the vote 2, 46% gave the vote 3, 23% gave the vote 4, and 23% gave the vote 5

It is evident that the participants during CloudStudio collaborative debugging have involved more than during the shared screen debugging. As an important note the percentage of participants (23%) who gave 5/5 - which mean that they involved very much- during shared screen debugging is the same of the percentage of participants who were controlling the debugging during shared screen debugging (who has the debugger session active uniquely on his machine)

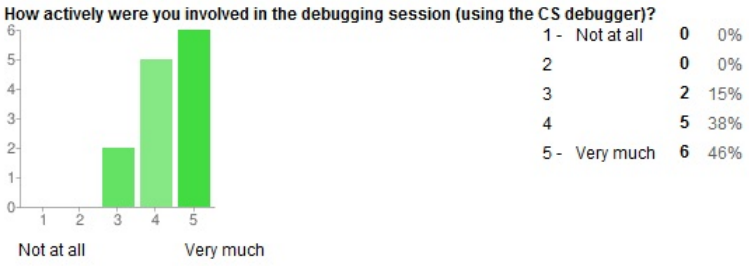


Figure 5.15: Involving in the debugger during CloudStudio collaborative debugging

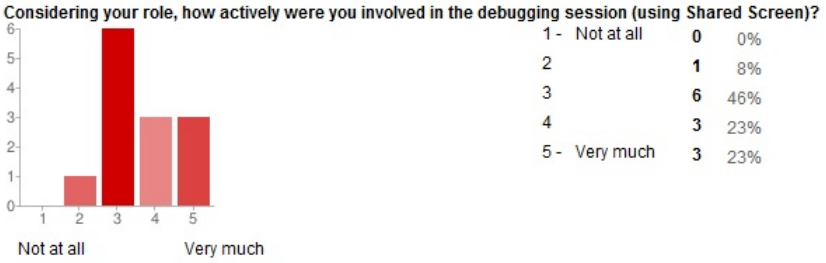


Figure 5.16: Involving in the debugger during shared screen debugging

Efficiency of the collaborative tools: As a vote form 1 to 5 (*'not at all'* to *'very much'*) for the efficiency of each collaborative tool used during the assignments. For CloudStudio collaborative debugging 8% of the participants gave 2/5, 8% gave 3/5, and 85% gave 4/5.

While for shared screen debugging 8% of the participants gave 1/5, 46% gave 2/5, 15% gave 2/5, and 31% gave 4/5. Evidently we can see how the participants saw the difference of the efficiency between CloudStudio collaborative debugging and shared screen debugging.

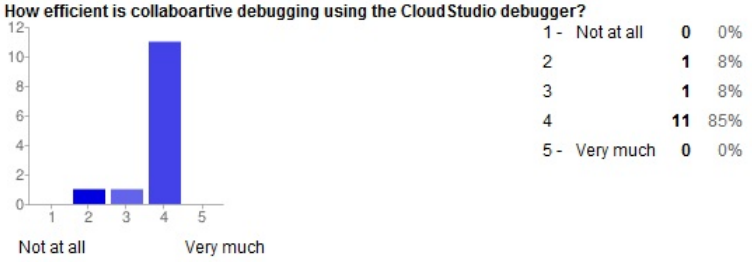


Figure 5.17: CloudStudio collaborative debugger efficiency

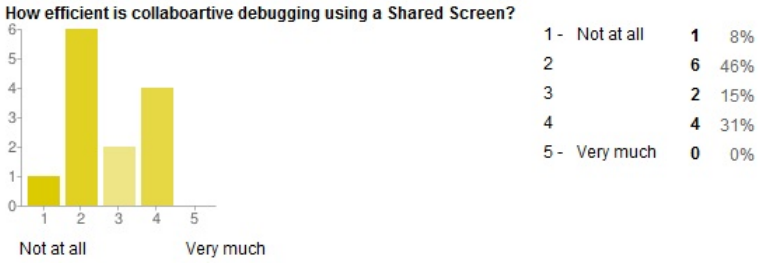


Figure 5.18: Debugging used shared remote screen efficiency

Collaborative debugging preference: 85% of the participants preferred CloudStudio collaborative debugging over Shared screen Debugging.

Figure 5.19: Users preference between the two collaborative debugging experience

Users comments: During the shared screen debugging session, the users complained about some restrictions of this kind of debugging session. Some users complained the fact that while watching the debugging session during the shared screen debugging, the screen sharing was interfering with the work, meaning that the screen sharing had a negative effect on the productivity of the work done.

Another complaint was about having to use an IM for communication. Since the tools for screen sharing do not provide a messaging tool they had to use other tools for communication. They had to switch the IM and shared screen windows back and forth to communicate and to work.

Every programmer has its own way to work. During the case study we noticed that some of our participants are not very collaborative and since then they were not quite happy using a collaboration tool where the participants can control the debugger and perform actions, it depends strongly on the personality of the participants.

5.10 Threats to validity

Threats to the validity of this case study are internal and external. Internal validity is whether the case study supports the outcome. External validity is whether the outcomes can be applied to different domains.

5.10.1 Internal validity

The outcome showed us that the amount of data is not sufficient to answer the questions of the case study. However we found some significant trends from the data that we collected during the case study. There are two sources that the data comes from. The first one is the logs which cannot be manipulated by the participant users, so these log data cannot present a threat to the validity. However the second source, which is the questionnaire which each participant has to fill, it can be manipulated by the participants. This data presents a threat to the validity.

We asked participants to use some *instant messaging* program to communicate between them. They are then asked to send the chat logs to us. Doing so we could see their behavior. At this point we cannot be sure about whether they used some other communication tools during the case study other than the logs they sent. These logs present a threat to the validity.

This case study is driven by a group of non pre-selected computer engineering students. Not all the students come from the same background and they have different levels of experience in debugging. Some are familiar with debugging, and on the other hand some don't use usually debugging tools. We overcame this difference by asking them in the questionnaire their experience of debugging.

This case study is driven to see how the CloudStudio Collaborative Remote Debugger helps developers to debug in a collaborative way in a geographically distributed environment. As for the human nature, not all participants were willing to collaborate and this might prevent us to see the real efficiency of the tool.

5.10.2 External validity

The setting of the case study is sufficient to monitor the behavior of the participants in a shared screen and CloudStudio collaborative environment. However, this case study is made with university students. Thus, this case study should be driven with developers that are more experienced with debugging and programming.

5.11 Summary

After all, we can say that the participants feel the efficiency of the features that CloudStudio collaborative debugger offers. They took the advantage of these features to collaboratively find the bugs inserted in the scripts. The participants did more actions during the assignment which uses the CloudStudio Collaborative Remote Debuger since they were all able to control the debugger from his own machine and browser.

Chapter 6

Conclusions and Future Works

6.1 Conclusion

This thesis extends CloudStudio a web based IDE with a collaborative remote debugger. Using collaborative debugger the users who share the same projects can share a debugging session as well. by sharing the debugging session each involved user can individually see the code and how the debugger is navigating the code in real time, browse the local variables and search for specific one, see the others expressions in real time and add new expressions as well which will be seen by the others. The collaborative debugger has 3 modes of debugging, the first mode is single user debugger, in this mode the user can debug the code individually on the cloud without any collaborative actions, but in any time can move it to one of the other modes. The second mode is debugging *with users single control*, here just one user (master user) at a time can do debugging actions (step into, step out, step over, resume, terminate), but still all the others can individually add breakpoints, view the variables, evaluate expressions, see the how the master is navigating in the code in real time. The master can in anytime pass the control to other user(who should be online and share the same project). The third debugger mode is *with users multi control*, here every user is a master. So, all the users can control the flow of the debugger and do the other actions. Any user, who is sharing the debugger session, can invite in anytime users to the debugger session (the invited user should be online and loged to the shared project).

The collaborative debugger is designed in a standard way, that can be used

to implement any debugger for other languages. In this thesis the Collaborative debugger is implemented for JavaScript.

After finishing the implementation of this tool a case study is designed to see how efficient the collaborative debugger is, and what is the advantage of using it instead of using the exiting tools that help the users to collaborate as chatting, remote shared screen, and others.

6.2 Future works

The work presented in this thesis offers several opportunities for further research and enhancement. Here in the following are introduced the main possibilities.

6.2.1 Implementations for more languages

CloudStudio collaborative remote debugger is designed in standard way, so it can be easily implemented for another languages without additional efforts. Currently CloudStudio collaborative remote debugger is allowed just for JavaScript, so just the users of JavaScript projects can use this tool. High recommended extension is to implement CloudStudio collaborative remote debugger for another language, that Cloudstudio IDE includes, like Java or Eiffel. The efforts of developing this extension are not complicated, rather it is very forward since it should just adapt the language of the debugger with the CloudStudio collaborative remote debugger.

6.2.2 Improvement for tablets

One of the advantage of cloud applications is that the cloud unite the users of pc with those with mobiles. So, another high recommended enhancement is to improve cloudStudio collaborative remote debugger to be adapted to tablets. It would be a great that a developer can perform collaborative debugging when he is going to his work from his tablet or ever his smart phone. Currently Cloudstudio collaborative remote debugger is working on tablets and even on smart phones since it is web based application, but it misses the enhancement to be adapted to the tablets screens and other stuffs related to the tablets and smart phones.

6.2.3 Extend the case study

As mention in the before in the section which describes the problems and challenges of the case study 5.8, we had problems with the case study in collecting persons and creating the groups. Actually to reach more realistic results the case study should contain about 100 persons or even more, and in e perfect cases described in this section 5.8.1, in this way the results will reflect more the efficiency of CloudStudio collaborative remote debugger. Currently the case study contains 13 persons organized in 6 groups. The goal is to arrive to at least 100 persons grouped in well organized groups (the same number of each type of group matrix shown in this section 5.4) The idea is: in future the case study can be insert in a course in which the student can participates as a part of course project or something similar. in this way we can guarantee large number of participants, standard background of the participants, and specially motivated participants. Then the results will be more realistic.

Appendices

In this appendix the assignments that are sent to users are attached. These assignments are used to drive the case study.

.1 Assignment 1 - Debugging using a shared screen

In this assignment, you and your colleagues have to debug a set of JavaScript programs using a Shared Screen. One user should connect to CloudStudio, log in and use the debugger. This user is called the masterUser in this document, the other users (called otherUsers in this document) should connect to the masterUser's computer (using Skype-Shared-Screen or TeamViewer, etc).

Settings Instructions

masterUser should open cloudstudio (`http://cloudstudio.ethz.ch/`), then login using the following user name and password:

username: username

password: password

Then, select the project "ProjectA". In this project there are several files: Example1_Script1.js Example1_Script2.js Example1_Script3.js Example1_Script4.js

The examples Example1_Script1.js ... Example1_Script4.js are similar. However, they have different issues to solve and they have some minor modifications.

Goal of the assignment

The assignment consists of debugging different programs in collaboration with the other users. In these tasks, you do NOT need to fix the problem of the program, i.e. you do not need to modify the program. To avoid errors, please do not write in the editor.

Different tasks will use different files. We will mention explicitly which file to use in each task.

Tasks

In the following tasks, you will perform some simple debugging. This will help you to get familiar with the debugging tool. We recommend that the *masterUser* performs the tasks, but if you use a Remote Desktop program (i.e. everybody has control over *masterUser*âĂŹs mouse and keyboard) the *otherUsers* can also perform the operations. This is a **collaborative debugging** session and the *masterUser* and the *otherUsers* should work together to solve the tasks. When you start the tasks, please write down your **starting time**. When you finish the task, also write down the **finish time**. When you finish each task or sub-task, fill in the reports section at the end of each task with the solution of the tasks and the times it took you to solve them. You all have a space for yourselves.

Task 1: Open the file "*Example1_Script1.js*". Put breakpoint at line 73, and start the debugger. Debug -> Debug (i.e. in the normal mode not with other users):

Apply "step over" until the debugger reaches line 77.

- **Task1.1:** Take a look to the variable list from the variables tab. What is the value of the field "*_length*" of the variable "*list*". **Hint:** you find this information in the variables tab.
- **Task1.2:** Using the expression tab, evaluate the expression "*list.item(1)*", "*list.item(2)*", "*list.item(3)*" and "*list.item(4)*" (still at line 77).
- **Task1.3:** At line 77, step into the the function "*add*". Then, evaluate the variable "*current*" at line 40.
- **Task1.4:** Finally, step out, to return to the main program (line 78). What is the value of the field "*_length*" of the variable "*list*"at line 81? Then terminate the debugger by terminate button.

The following tasks are of increasing complexity. In general, the tasks will request you to find some problem or perform some action. You can decide whether the masterUser executes the tasks of the otherUsers.

Task 2: Open the file "*Example1_Script1.js*". Evaluate the variables red, yellow and st at Line 81. Their values should be:

red = "red"

yellow = "yellow"

st = "red,orange, yellow,green,black,brown,pink"

After doing the evaluation, you will find that there is a bug in this program. Find out which function has the bug, and locate the lines that might produce this bug. In a short sentence, report what is wrong (you do not need to fix the problem).

Task 3: Open the file "*Example1_Script2.js*". Insert breakpoints in places that let you see the following variables *orange*, *st*, *st2* and *yellow*. The values should be:

orange = "orange"

st = "red,orange,yellow"

st2 = "yellow, orange, red"

yellow = "yellow"

After doing the evaluation, you will find that there is a bug in this program. Find out which function has the bug, and locate the lines that might produce this bug. In a short sentence, report what is wrong (you do not need to fix the problem).

Task 4: Open the file "*Example1_Script3*". Start debugging the program and the program will give you an exception with the type of the exception and the line-number where it is thrown. The debugger will terminate when the exception is thrown so you can start the debugger again to find the cause of the problems. Find out which function produces the exception and which line produces the bug. Briefly mention how it could be fixed (you do not need to fix the problem).

Task 5: Open the file "*Example1_Scrip4*". Start the debugger after putting breakpoints in suitable places that let you see the following variables *tostring*, *tostring1*, *yellow*, *orange*, *pink* and *pink1*. The values should be:

tostring = “red,orange,yellow”
tostring1 = “pink,pink,black”
yellow = “yellow”
orange = “orange”
pink = “pink”
pink1 = “pink”

After doing the evaluation, you will find that there is a bug in this program. Find out which function has the bug, and locate the lines that might produce this bug. In a short sentence, report what is wrong (you do not need to fix the problem).

.2 Assignment 1 - Debugging using CloudStudio Collaborative Remote Debugger

In this assignment, you and your colleagues have to debug a set of JavaScript programs using the collaborative debugger. Using this tool, several users can share the debugging session, add expressions, break points, etc. Take a look to Introduction to Collaborative Remote Debugger .

The debugger has two options: *debug with user single control* and *debug with users multi control*.

Note: You should NOT use a shared desktop. You will share the debugging session using the CloudStudio tool. You have to solve the tasks using the collaborative debugger using *singleControl* and *multiControl*, and you are NOT allowed to use the debugger alone.

Settings Instructions

All the users, from now on *User1*, *User2* (and for some cases *User3* and *User4*) should open cloudstudio. Then login using the following user names and passwords:

For User 1: username password
For User 2: username password
For User 3: username password
For User 4: username password

Then, select the project "*ProjectA*". All the users will see the same project. In this project there are several files:

Example1_Script1.js
Example1_Script2.js
Example1_Script3.js
Example1_Script4.js

The examples *Example1_Script1.js ... Example1_Script4.js* are similar, however, they have different issues to solve and they have some minor modifications.

Goal of the assignment

The assignment consist of debugging different programs in **collaboration with the other users**. The tool letâĂŹs you see the expressions of other users and share the debugging session. In these tasks, you do NOT need to fix the problem of the program, thus you do not need to modify the program. To avoid errors, please do not write in the editor.

Different tasks will use different files. We will explicitly mention which file to use in each task.

Tasks

In the following tasks, you will perform simple assignments, this will help you to get familiar with this new tool. We will first suggest who performs which tasks (if there are more than 2 users, the tasks can be performed by the other users too). When you start the tasks, please write down your **starting time**. When you finish the task, also write down the **finish time**. When you finish each task or sub-task, fill in the reports section at the end of each task with the solution of the tasks and the times it took you to solve them. You all have a space for yourselves.

Task 1: Open the file "*Example1_Script1.js*". User1 puts a breakpoint at line 73. User1 should start the debugger. Debug -> debug with users single control :

User1 will see a dialog and can add the other users (user2, ...) to the debugging session. The other users should accept the invitation. Once everybody is sharing the debugging session, apply "step over" until the debugger reaches line 77.

- **Task1.1:** ALL USERS: Take a look to the variable list from the variables tab. What is the value of the field "*_length*" of the variable "*list*". Hint: you find this information in the variables tab.

- **Task1.2:** Using the expression tab, you need evaluate the expressions "*list.item(1)*", "*list.item(2)*" "*list.item(3)*" and "*list.item(4)*" (still at line 77). To do it, each user should go to the expression tab, and each user should add at least one expression.

- **Task1.3:** Now, user1 should pass the control of the debugger to user2. (users ->Set master user) a dialog will open, then select user2 from the list. At line 77, the new master (user2) steps into the the function "*add*". Then, evaluate the variable "*current*" at line 40.

- **Task1.4:** Finally, step out, to return to the main program (line 78). What is the value of the field "*_length*" of the variable "*list*" at line 81? Then terminate the debugger by terminate button.

The following tasks are of increasing complexity. In general, the tasks will request you to find some problem or perform some action.

For Task2 and Task3 use Debug -> debug with users single control.

For Task4 and Task5 use Debug -> debug with users multiple control.

Task 2: User1: Open the file "*Example1_Script1.js*". User1 should start the debugger. (Debug -> Debug with users (single control)) :

User1 will see a dialog and there you can add the other users (user2, ...) to the debugging session. The other users should accept the invitation. Once everybody is sharing the debugging session, evaluate the variables *red*, *yellow* and *st* at Line 81. Their values should be:

red = "red"yellow = "yellow"st = "red,orange, yellow,green,black,brown,pink"

After doing the evaluation, you will find that there is a bug in this program. Find out which function has the bug, and locate the lines that might produce this bug. In a short sentence, report what is wrong (you do not need to fix the problem).

Do it in a collaborative way with the other users. Remember you can always pass the control to other users; all the users can add expressions, breakpoints, etc.

Task 3: User2: Open the file "*Example1_Script2.js*". User2 should start the debugger. (Debug -> Debug with users (single control)) :

User2 will see a dialog and there you can add the other users (user2, ...) to the debugging session. The other users should accept the invitation. Once everybody is sharing the debugging session, insert breakpoints in places that let you see the following variables *orange*, *st*, *st2* and *yellow*. The values should be:

orange = "orange"
st = "red,orange,yellow"
st1 = "yellow, orange, red"
yellow = "yellow"

After doing the evaluation, you will find that there is a bug in this program. Find out which function has the bug, and locate the lines that might produce this bug. In a short sentence, report what is wrong (you do not need to fix the problem).

Do it in a collaborative way with the other users. Remember you can always pass the control to other users; all the users can add expressions, breakpoints, etc.

Task 4: User2: Open the file "*Example1_Script3.js*". User2 should start the debugger. (Debug -> Debug with users (multiple control)) :

User2 will see a dialog and there you can add the other users (user1, ...) to the debugging session. The other users should accept the invitation. Once everybody is sharing the debugging session, the program will give you an ex-

ception with the type of the exception and the number of line where it is thrown. The debugger will terminate when the exception is thrown so you can start the debugger again to find the cause of the problems.

Do it in a collaborative way with the other users. Remember you can always pass the control to other users; all the users can add expressions, breakpoints, etc.

Task 5: User1: Open the file "*Example1_Script4.js*". User2 should start the debugger. (Debug -> Debug with users (multiple control)) :

User2 will see a dialog and there you can add the other users (user2, ...) to the debugging session. The other users should accept the invitation. Once everybody is sharing the debugging session, insert breakpoints in places that let you see the following variables *orange*, *st*, *st2* and *yellow*. The values should be:

tostring = "red,orange,yellow"
tostring1 = "pink,pink,black"
yellow = "yellow"
orange = "orange"
pink = "pink"
pink2 = "pink"

After doing the evaluation, you will find that there is a bug in this program. Find out which function has the bug, and locate the lines that might produce this bug. In a short sentence, report what is wrong (you do not need to fix the problem).

Do it in a collaborative way with the other users. Remember you can always pass the control to other users; all the users can add expressions, breakpoints, etc.

.3 Assignment 2 - Debugging using a shared screen

In this assignment, you and your colleagues have to debug a set of JavaScript programs using a Shared Screen. One user should connect to CloudStudio, log in and use the debugger. This user is called the masterUser in this document, the other users (called otherUsers in this document) should connect to the masterUser's computer (using Skype-Shared-Screen or TeamViewer, etc).

Settings Instructions

masterUser should open cloudstudio (`http://cloudstudio.ethz.ch/`), then login using the following user name and password:

username: username

password: password

Then, select the project "ProjectB". In this project there are several files: Example2_Script1.js Example2_Script2.js Example2_Script3.js Example2_Script4.js

The examples Example2_Script1.js ... Example2_Script4.js are similar. However, they have different issues to solve and they have some minor modifications.

Goal of the assignment

The assignment consists of debugging different programs in collaboration with the other users. In these tasks, you do NOT need to fix the problem of the program, i.e. you do not need to modify the program. To avoid errors, please do not write in the editor.

Different tasks will use different files. We will mention explicitly which file to use in each task.

Tasks

In the following tasks, you will perform some simple debugging. This will help you to get familiar with the debugging tool. We recommend that the

masterUser performs the tasks, but if you use a Remote Desktop program (i.e. everybody has control over *masterUser*âĂŹs mouse and keyboard) the *otherUsers* can also perform the operations. This is a **collaborative debugging** session and the *masterUser* and the *otherUsers* should work together to solve the tasks. When you start the tasks, please write down your **starting time**. When you finish the task, also write down the **finish time**. When you finish each task or sub-task, fill in the reports section at the end of each task with the solution of the tasks and the times it took you to solve them. You all have a space for yourselves.

Task 1: Open the file "*Example2_Script1.js*". Put breakpoint at line *106*, and start the debugger. Debug -> Debug (i.e. in the normal mode not with other users):

Apply "step over" until the debugger reaches line *110*.

- **Task1.1:** Take a look to the variable list from the variables tab. What is the value of the field "data"of the field "root" of the variable "bst". Hint: you find this information in the variables tab.
- **Task1.2:** Using the expression tab, evaluate the expression "bst.root.right.data", "bst.root.left.data", "bst.contains(17)", "bst.contains(106)" (still at line 110).
- **Task1.3:** At line 110, step into the the function 'insert '. Then, evaluate the variable "currentNode.data" at line 26.
- **Task1.4:** Finally, step out, to return to the main program (line 111). What is the value of the field "datarq' of the variable "bst.root.left.left" at line 111? Then terminate the debugger by terminate button.

The following tasks are of increasing complexity. In general, the tasks will request you to find some problem or perform some action. You can decide whether the masterUser executes the tasks of the *otherUsers*.

Task 2: Open the file "*Example2_Script1.js*". Add a breakpoint on line 122 and evaluate the variable *traverse1* at Line 120. Its value should be:

traverse1 = "3, 5, 6, 10, 11, 12, 13, 17, 21, 30, 32, 36, 37"

After doing the evaluation, you will find that there is a bug in this program. Find out which function has the bug, and locate the lines that might produce this bug. In a short sentence, report what is wrong (you do not need to fix the problem).

Task 3: Open the file "*Example2_Script2.js*". Insert breakpoints in places that let you see the following variables *before*, *after1* and *after2* . The values should be:

before = "3, 5, 6, 10, 11, 12, 13, 17, 21, 30, 32, 36, 37"

after1 = "5, 6, 10, 11, 12, 13, 17, 21, 30, 32, 36, 37"

after2 = "5, 6, 10, 11, 13, 17, 21, 30, 32, 36, 37"

After doing the evaluation, you will find that there is a bug in this program. Find out which function has the bug, and locate the lines that might produce this bug. In a short sentence, report what is wrong (you do not need to fix the problem).

Task 4: Open the file "*Example2_Script3*". Evaluate the variable before at Line 121. Its value should be:

before = "3, 5, 6, 10, 11, 12, 13, 17, 21, 30, 32, 36, 37"

After doing the evaluation, you will find that there is a bug in this program. Find out which function has the bug, and locate the lines that might produce this bug. In a short sentence, report what is wrong (you do not need to fix the problem).

Task 5: Open the file "*Example2_Scrip4*". Start the debugger after putting breakpoints in suitable places that let you see the following variables *before*. The values should be:

before = "3, 5, 6, 10, 11, 12, 13, 17, 21, 30, 32, 36, 37"

After doing the evaluation, you will find that there is a bug in this program. Find out which function has the bug, and locate the lines that might produce this bug. In a short sentence, report what is wrong (you do not need to fix the problem).

.4 Assignment 2 - Debugging using CloudStudio Collaborative Remote Debugger

In this assignment, you and your colleagues have to debug a set of JavaScript programs using the collaborative debugger. Using this tool, several users can share the debugging session, add expressions, break points, etc. Take a look to Introduction to Collaborative Remote Debugger .

The debugger has two options: *debug with user single control* and *debug with users multi control*.

Note: You should NOT use a shared desktop. You will share the debugging session using the CloudStudio tool. You have to solve the tasks using the collaborative debugger using *singleControl* and *multiControl*, and you are NOT allowed to use the debugger alone.

Settings Instructions

All the users, from now on *User1*, *User2* (and for some cases *User3* and *User4*) should open cloudstudio. Then login using the following user names and passwords:

For User 1: username password

For User 2: username password

For User 3: username password

For User 4: username password

Then, select the project "*ProjectB*". All the users will see the same project. In this project there are several files:

Example2_Script1.js
Example2_Script2.js
Example2_Script3.js
Example2_Script4.js

The examples *Example2_Script1.js* ... *Example2_Script4.js* are similar, however, they have different issues to solve and they have some minor modifications.

Goal of the assignment

The assignment consist of debugging different programs in **collaboration with the other users**. The tool letâĂŹs you see the expressions of other users and share the debugging session. In these tasks, you do NOT need to fix the problem of the program, thus you do not need to modify the program. To avoid errors, please do not write in the editor.

Different tasks will use different files. We will explicitly mention which file to use in each task.

Tasks

In the following tasks, you will perform simple assignments, this will help you to get familiar with this new tool. We will first suggest who performs which tasks (if there are more than 2 users, the tasks can be performed by the other users too). When you start the tasks, please write down your **starting time**. When you finish the task, also write down the **finish time**. When you finish each task or sub-task, fill in the reports section at the end of each task with the solution of the tasks and the times it took you to solve them. You all have a space for yourselves.

Task 1: Open the file "*Example2_Script1.js*". User1 puts a breakpoint at line *106*. User1 should start the debugger. Debug -> debug with users single control :

User1 will see a dialog and can add the other users (user2, ...) to the debugging session. The other users should accept the invitation. Once everybody is sharing the debugging session, apply "step over"until the debugger reaches line *110*.

- **Task1.1:** ALL USERS: Take a look to the variable list from the variables tab. What is the value of the field "data" of the field "root" of the variable "bst". Hint: you find this information in the variables tab.
- **Task1.2:** Using the expression tab, evaluate the expression "bst.root.right.data", "bst.root.left.data", "bst.contains(17)", "bst.contains(106)" (still at line 110). To do it, each user should go to the expression tab, and each user should add at least one expression.

- **Task1.3:** Now, user1 should pass the control of the debugger to *user2*. (users ->Set master user) a dialog will open, then select *user2* from the list. At line 110, step into the the function "insert". Then, evaluate the variable "currentNode.data" at line 26.

- **Task1.4:** Finally, step out, to return to the main program (line 111). What is the value of the field "data" of the variable "bst.root.left.left" at line 111? Then terminate the debugger by terminate button.

The following tasks are of increasing complexity. In general, the tasks will request you to find some problem or perform some action.

For Task2 and Task3 use Debug -> debug with users single control.

For Task4 and Task5 use Debug -> debug with users multiple control.

Task 2: *User1:* Open the file "*Example2_Script1.js*". *User1* adds a breakpoint on line 122 and should start the debugger. (Debug -> Debug with users (single control)) :

User1 will see a dialog and there you can add the other users (user2, ...) to the debugging session. The other users should accept the invitation. Once everybody is sharing the debugging session, evaluate the variable *traverse1* at Line 120. Its value should be:

traverse1 = "3, 5, 6, 10, 11, 12, 13, 17, 21, 30, 32, 36, 37"

After doing the evaluation, you will find that there is a bug in this program. Find out which function has the bug, and locate the lines that might produce this bug. In a short sentence, report what is wrong (you do not need to fix the problem).

Do it in a collaborative way with the other users. Remember you can always pass the control to other users; all the users can add expressions, breakpoints, etc.

Task 3: *User2:* Open the file "*Example2_Script2.js*". *User2* adds a breakpoint on line 222 and should start the debugger. (Debug -> Debug with users (single control)) :

User2 will see a dialog and there you can add the other users (user2, ...) to the debugging session. The other users should accept the invitation. Once everybody is sharing the debugging session, insert breakpoints in places that let you see the following variables *before*, *after1* and *after2*. The values should be:

before = "3, 5, 6, 10, 11, 12, 13, 17, 21, 30, 32, 36, 37"
after1 = "5, 6, 10, 11, 12, 13, 17, 21, 30, 32, 36, 37"
after2 = "5, 6, 10, 11, 13, 17, 21, 30, 32, 36, 37"

After doing the evaluation, you will find that there is a bug in this program. Find out which function has the bug, and locate the lines that might produce this bug. In a short sentence, report what is wrong (you do not need to fix the problem).

Do it in a collaborative way with the other users. Remember you can always pass the control to other users; all the users can add expressions, breakpoints, etc.

Task 4: *User1:* Open the file "*Example2_Script3.js*". *User1* adds a breakpoint on line 122 and should start the debugger. (Debug -> Debug with users (multiple control)):

User1 will see a dialog and there you can add the other users (user2, ...) to the debugging session. The other users should accept the invitation. Once everybody is sharing the debugging session, evaluate the variable before at Line 121. Its value should be:

before = "3, 5, 6, 10, 11, 12, 13, 17, 21, 30, 32, 36, 37"

After doing the evaluation, you will find that there is a bug in this program. Find out which function has the bug, and locate the lines that might produce this bug. In a short sentence, report what is wrong (you do not need to fix the problem).

Do it in a collaborative way with the other users. Remember you can always pass the control to other users; all the users can add expressions, breakpoints, etc.

Task 5: *User2:* Open the file "*Example2_Script4.js*". *User2* adds a breakpoint on line 123 and should start the debugger. Debug -> (Debug with users (multiple control)):

User1 will see a dialog and there you can add the other users (user2, ...) to the debugging session. The other users should accept the invitation. Once everybody is sharing the debugging session, start the debugger after putting breakpoints in suitable places that let you see the following variables before. The values should be:

before = "3, 5, 6, 10, 11, 12, 13, 17, 21, 30, 32, 36, 37"

After doing the evaluation, you will find that there is a bug in this program. Find out which function has the bug, and locate the lines that might produce this bug. In a short sentence, report what is wrong (you do not need to fix the problem).

Do it in a collaborative way with the other users. Remember you can always pass the control to other users; all the users can add expressions, breakpoints, etc.

.5 Cloudstudio collaborative remote debugger guide

In this appendix the guide which is sent to the participants of the case study is attached.

Introduction into JavaScript Cooperative Remote Debugger

JavaScript debugger is a part of the web-based IDE Cloudstudio.
is a cooperative debugger among multi users that can share debugging session remotely.

Debugging modes

You can start JavaScript debugger in three different modes. in the following there are brief description of each one of them:

1.Single user debugging

After putting a number of break point on the code, you can choose the option debug in the debug menu.The code that is going to be debugged is the code that you are visualizing at that moment. And you will be the only user who is controlling and observing the debugging session. It is possible to add users to the debug session later on.

2.Debugging with users single control (Master logic)

Master logic: in this debugging mode. at a time one user can have the control of the debugging flow (step into , over, out, resume and terminate) However, other users can still add breakpoints and expressions, they can also see variables and observe the flow of the debugger. the master of the debugger session can pass the control at anytime to anther user. and so on

how-to: when you choose to start debugging with users single control , you will be asked to select from the online users, that are already logged in for the same project, to share with you the debugging session. Users will receive an invitation for the debug session. After the positive/negative responses of all invited users, if the user has accepted the invitation a new-non editable tab will be opened with the code that master user visualizes at that moment. and from now all the invited users are connected to the debugger session.

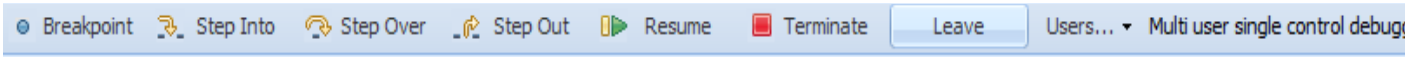
note: even if you started with single user debugging mode you can transform it to “with users single control ” mode by adding users in the add menu at anytime.

Changing the master user: master user can decide at any point of the debugging to pass the control to another user who is in the debug session. You can do this by choosing “set Master user” from Users menu and then by choosing a user from the list. the new user will be notifed about the fact that s/he become the master of the debugger session.

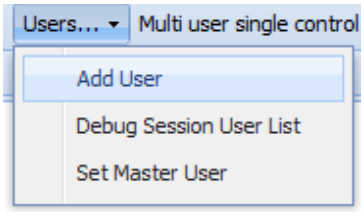
3. Debugging with users multi control.

This debugging mode is actually the same of above but the difference here that there is no master. There is just the user who start the debugging with him/here code version and invite the initial users, but whenever the debugger is started, anyone of users, that is sharing the debugger session ,is able to act any debugger command.

JavaScript debugger tool bar

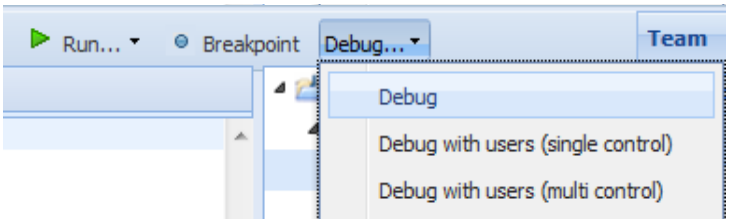


Breakpoint : to set a break point at the row at which there is the cursor currently .
Step Into: to single step interring any function call
Step Over: to single step to the next line in the current function.
Step Out: to continue execution until the current function returns.
Resume : to resume execution of a script.
Terminate: to terminate the debugging session.
Leave: to leave the session, if the debugger mode is Single control , and the user who is leaving the session is the master , so the control will be moved automatically to the next user, who will be notified of that.
all other users who is sharing the same debug session will be notified of that the user has leaved the session.



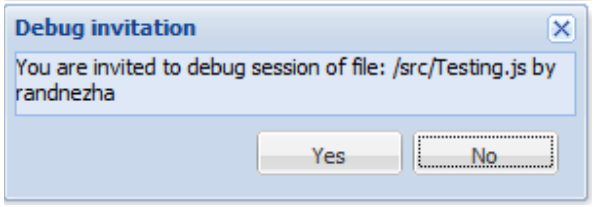
Users-Add User : To invite users to the debugging session. any user who is sharing the debugger session in any mode can do that. the invited user should be online and already share the project.
Users-Debug Session User List : to get the list of the users who is sharing the current debugger session.
Users-Set master User : to passing the debugger control to another user who is sharing the debugger session

Debug-Debug : to start the debugger in local mode(without users)
Debug-Debug with users (single control) : to start the debugger with users with master logic mode

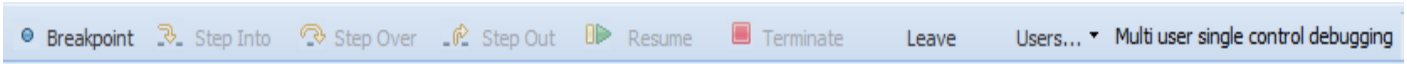


Debug-Debug with users(multi control): to start the debugger with users without master logic mode

Notes:
1.When add new user, this user will receive an invitation to accept :

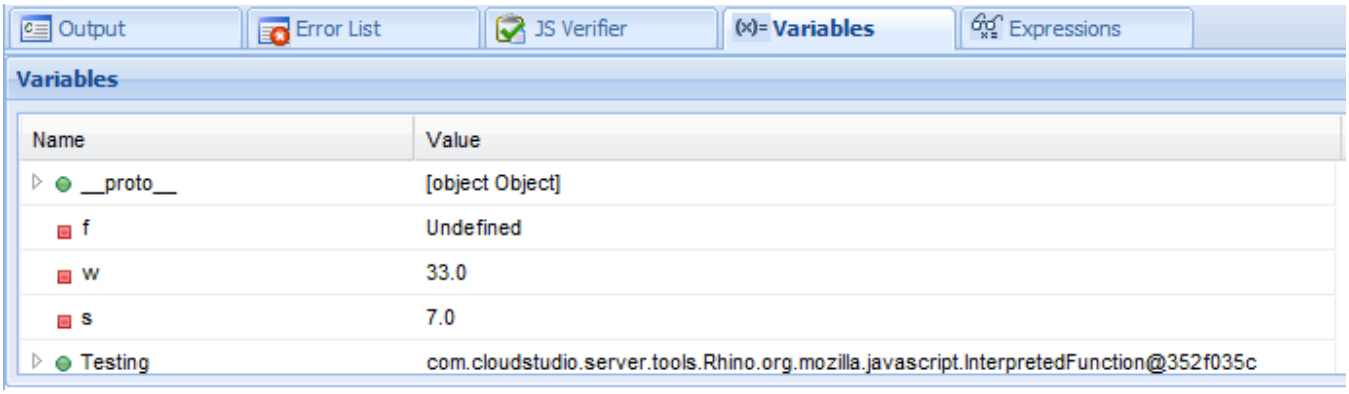


2.If the debugging mode is single control the new user will have the debugger toolbar with the button of the debugger disabled :



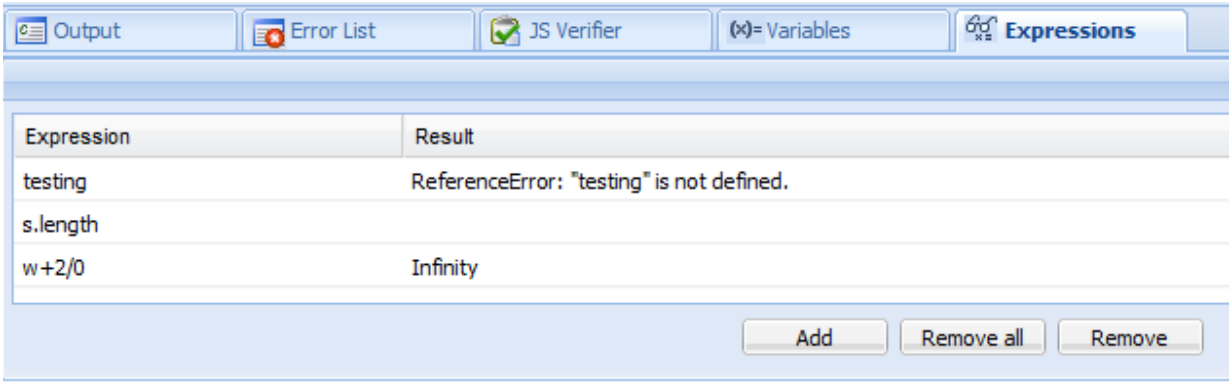
Java Script debugger Watch tabs :

1. Variables Tab



This tab is initially disabled, but whenever the debugger is started this table become able and focused for all the users who are sharing the debugging session. all the variables, which are defined after the debugger current point, hold as a value “Undefined”. the variables initially are ordered alphabetically by their name. but it is able to order them also by the values.

2. Expressions evaluation:



this tab is initially disabled, but whenever the debugger is started this tab become able for the users who are sharing the debugging session.

Add Expression: will create new row in the expressions list for add the new one for evaluating.

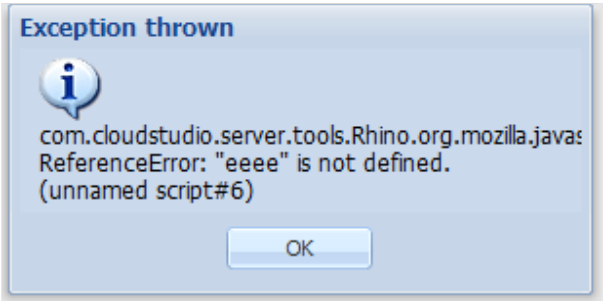
Remove: removing a specific expression

Remove all : remove all the expression in the list.

any user who is sharing the debugger session can add expressions, and also see the expressions added by the other users. The expressions are synchronized with the debugger flow.

Errors/Exceptions handling :

Whenever the code contains error or exception. the debugger will show error notification which contain the type of the exception, the line of exception and a brief description. after an exception is thrown the debugger will terminate for all the users which are sharing the debugger session.



.6 Case study questionnaire

In this appendix the questionnaire that the participants filled in after the case study is attached.

JavaScript Debugger-Cloudstudio

Thank you for participating in our Case Study.
We would like you to answer a few questions about your experience of using

a) a Shared Screen for debugging
b) the CloudStudio (CS) debugger

* Required
Your complete name *

General Information

Please provide your group's Letter: *
(Your group letter was part of the instruction-email you received.)

- A
- B
- Don't know

Provide your Email address if you'd like to be informed about the results of this study:
Optional.

How many people were in your team for this study (including you)? *

- 2
- 3
- 4
- Don't know

Experience

What is your experience level as a programmer? *

	1	2	3	4	5	
No experience	○	○	○	○	○	Years of experience

What is your experience level with debugging software? *
(Any type of debugging, including different tools and languages.)

	1	2	3	4	5	
No experience	○	○	○	○	○	Years of experience

What is your experience level with the JavaScript language? *

	1	2	3	4	5	
No experience	○	○	○	○	○	Years of experience

How often have you participated in a collaborative debugging session before? *
That means debugging software together with >1 other person, either remotely or on the same computer.

- ○ 0 times
- ○ 1 - 3 times
- ○ 3 - 10 times
- ○ 10 - 20 times
- ○ > 20 times

What kind of collaborative debugging sessions did you have?
You can skip this question if you answered "0 times" for the previous question.

- ☐ Together in front of the same computer
- ☐ Software that allows to watch another person's screen
- ☐ Remote access to a machine (all participants cloud control mouse, keyboard and debugger)
- ☐ Other: _____

Debugging using a Shared Screen

This section is about collaboratively debugging software using a Shared Screen (e.g. through TeamViewer, Skype etc.).

What was your role when debugging using Shared Screen? *
For example, suggesting breakpoints or expressions.

- ◯ I controlled the debugger
- ◯ I was watching the debugging

Considering your role, how actively were you involved in the debugging session (using Shared Screen)? *
Role is either "watching the debugging" or "controlling the debugger".

	1	2	3	4	5	
Not at all	◯	◯	◯	◯	◯	Very much

How often did you tell the person who controlled the debugger what to do? *
For example, telling the person to add a breakpoint, or investigate an expression.

- ◯ Nerver, as I was the person in control of the debugger
- ◯ 0 times
- ◯ 1 - 3 times
- ◯ 4 - 7 times
- ◯ 8 - 12 times
- ◯ > 12 times

How much did you miss that everybody can add breakpoints (using Shared Screen)? *

	1	2	3	4	5	
Not at all	◯	◯	◯	◯	◯	Very much

How much did you miss that everybody can add expressions (using Shared Screen)? *

	1	2	3	4	5	
Not at all	◯	◯	◯	◯	◯	Very much

How much did you miss that everybody can see variables (using Shared Screen)? *

	1	2	3	4	5	
Not at all	◯	◯	◯	◯	◯	Very much

How much did you miss that everybody can browse the code individually (using Shared Screen)? *
That is, you can only see whatever person controlling the debugger is seeing.

	1	2	3	4	5	
Not at all	◯	◯	◯	◯	◯	Very much

How much did you miss that everybody can control the debugger (using Shared Screen)? *
That is, everybody has control over "step over", "step into" etc.

	1	2	3	4	5	
Not at all	○	○	○	○	○	Very much

How efficient is collaboartive debugging using a Shared Screen? *

	1	2	3	4	5	
Not at all	○	○	○	○	○	Very much

Did you miss any other type of feature during Shared Screen debugging?
Optional.

Debugging using CloudStudio Collaborative Debugger

This section is about collaboratively debugging software using the CloudStudio debugger with its full functionality.

How actively were you involved in the debugging session (using the CS debugger)? *
For example, setting breakpoints, adding expressions etc.

	1	2	3	4	5	
Not at all	○	○	○	○	○	Very much

How often did you control the debugger, e.g. add breakpoints, investigate expressions etc.? *

- ○ 0 times
- ○ 1 - 3 times
- ○ 4 - 7 times
- ○ 8 - 12 times
- ○ > 12 times

How useful was it that everybody could add/share breakpoints (using the CS debugger)? *

1	2	3	4	5

Not at all ○ ○ ○ ○ ○ Very much

How useful was it that everybody could add/share expressions (using the CS debugger)? *

1 2 3 4 5

Not at all ○ ○ ○ ○ ○ Very much

How useful was it that everybody could navigate variables individually (using the CS debugger)? *

1 2 3 4 5

Not at all ○ ○ ○ ○ ○ Very much

How useful was it that everybody can browse the code individually (using the CS debugger)? *

Compare this to Shared Screen where everybody is looking at the same code.

1 2 3 4 5

Not at all ○ ○ ○ ○ ○ Very much

How useful do you consider the "with users single control" mode of the CloudStudio debugger? *

"with users single control" mode: one person (the master) controls the debugger ("step", "step into"); others can add breakpoints, expression etc.

1 2 3 4 5

Not at all ○ ○ ○ ○ ○ Very much

How useful do you consider the "with user multi control" mode of the CloudStudio debugger? *

"with user multi control" mode: every person in the debugging session has full control over the debugger.

1 2 3 4 5

Not at all ○ ○ ○ ○ ○ Very much

How efficient is collaboartive debugging using the CloudStudio debugger? *

1 2 3 4 5

Not at all ○ ○ ○ ○ ○ Very much

Did you miss any other type of feature during CloudStudio debugging session?

Optional.

Other Information

Which collaborative debugging do you prefer? *

- ( ) Shared Screen debugging
- ( ) CloudStudio collaborative debugging

Any other comment or feedback about this case study or the CloudStudio debugger?
Optional.

Submit

.7 Communication templates

In this appendix mails that are sent to the participants before the case study are attached.

.7.1 Email Sent to group's members one day before the assignments date

Hi all:

Thank you very much for accepting to participate in the evaluation our tool. We have organized the groups; the exercise will be done:

ADD DATE and TIME HERE

You will participate in this exercise with

ADD the NAMES Here

Please, make sure you share the Skype IDs, or any other chat program.

REQUIRED TOOLS FOR THE EXERCISE:
Here is a list of the required tools. Please make sure to install them before the exercise.
1. Chrome. You can install it from:
http://support.google.com/chrome/bin/answer.py?hl=en&answer=95346

2. TeamViewer (or any desktop remote sharing). To install it:
http://www.teamviewer.com/en/download/windows.aspx

3. Skype account or any chat program. Please add each other before the exercise.

NOTES:
Here are some important notes about the exercise:

1. We appreciate the time you take for helping us. Please, be careful about all the provided documents and notes. Follow the tasks and steps **EXACTLY** as we describe it.
2. We did not developed the whole cloudstudio framework. Please do not use other tools besides the debugger. using other tools might lead to errors or unexpected behavior that we cannot fix.
3. **DO NOT** change the JavaScript code we will provide.
4. During the exercise, some operation like opening the Javascript files or starting the debugger and terminating it, it takes time (it is performance issue for the cloudstudio in general) so please be patient in these cases. Do not execute the commands several times..
5.Do not forget to use Google Chrome.

DEBUGGER INFORMATION:
The debugger is easy to use. In case you need more information, here is a short guide:
https://docs.google.com/document/d/1Zlr8ef_UFURYOf8MnrYCwzwTXjSKvRySoj-4BBOAiIk/edit?pli=1
NEXT STEPS:
5 minutes before the exercise start, you will get an e-mail from us with two documents explaining the assignments. These documents describe what you should do during the exercise.

Please confirm us your participation; we had already organized the groups based on your preferences.
Thank you very very much for your time,
cheers
Rand&Mert

.7.2 Email sent to group's members before assignments starting time

Hi guys,

Before you start the assignment, please check you have installed:
1. Chrome: http://support.google.com/chrome/bin/answer.py?hl=en&answer=95346
2. TeamViewer (or any desktop remote sharing). http://www.teamviewer.com/en/download/windows.aspx
3. Skype account or any chat program. Please add each other before the exercise.

HOW TO PROCEED: :

You have to solve 2 different assignments. Below, you find a link to the first assignment; the second assignment will be sent once you finish the first one.

The solving of the assignments will take about 90 minutes (depending of each team, it might take more or less time. If you have not finished the FIRST assignment after 1:15hs, finish the task you are solving and do not continue with the other tasks. In a similar way, if you have not finished the SECOND assignment after 1:00hs, finish the task you are solving and do not continue with the other tasks.

When you finish the Assignmet 1, you can go to your shared documents in google drive and you will find the document of assignment2. After finishing the 2 assignments, we will send you an e-mail with a link to a questionnaire. Please, fill it in. It will request your group, you are:

Group [ADD GROUP LETTER HERE]

ASSIGNMENT 1:

Here is the link to assignment 1:

ADD HERE THE LINK TO THE GOOGLE DOC

Please when you finish everything remember to send us you skype chat logs

We will be online during the case study, In case of any help, you can contact us via mail, facebook, skype, etc.

Email
- Rand:
- Mert:

Skype:
- Rand:
- Mert:

Mobile numbers
- Rand:
- Mert:

PLEASE DON'T EDIT THE JAVASCRIPT CODE PROVIDED IN THE PROJECTS.

We appreciate a lot your collaboration for our thesis,
Thank you very much

Rand&Mert

Bibliography

[1] AceGWT - An integration of the Ajax.org Code Editor (ACE) into GWT. http://github.com/daveho/AceGWT.

[2] AceGWT - coderun. http://www.coderun.com/.

[3] Acm. http://www.acm.org/.

[4] Acm conferences for cscw. http://dl.acm.org/event.cfm?id=RE169&CFID=208279252&CFTOKEN=57895217/.

[5] Ajax.org Cloud9 Editor. http://ace.ajax.org/.

[6] Amazon simpledb. http://aws.amazon.com/simpledb/.

[7] Amy editor. http://www.amyeditor.com/.

[8] Bitbucket. https://bitbucket.org/.

[9] Cary l bates. http://patent.ipexl.com/inventor/Cary_L_Bates_1.html.

[10] Cloud9. https://c9.io/.

[11] Cloudsibiboduction. http://se.inf.ethz.ch/research/cloudstudio/.

[12] codebeamer. https://codebeamer.com/cb/user.

[13] Coderun. http://www.coderun.com/.

[14] Codiad. http://codiad.com/.

[15] Collabode. http://groups.csail.mit.edu/uid/collabode/.

[16] Collabode phd thesis. http://groups.csail.mit.edu/uid/other-pubs/maxg-thesis.pdf.

[17] Collide. http://code.google.com/p/collide/.

[18] Compilr. https://compilr.com/.

[19] Debuglive. http://www.breakpoints.com/.

[20] Debuglive launches first web-based debugging environment. http://pilot.us.reuters.com/article/2010/02/16/idUS79252+16-Feb-2010+BW20100216.

[21] ecco. http://ecco.sourceforge.net/.

[22] Exo ide. http://cloud-ide.com/.

[23] Gforge advanced serve. http://gforge.org/gf/.

[24] Git. http://git-scm.com/.

[25] Github. https://github.com/.

[26] Google Web Toolkit (GWT). http://developers.google.com/web-toolkit/.

[27] heroku. http://www.heroku.com/.

[28] Ibm jazz. https://jazz.net/.

[29] ideone. http://ideone.com/.

[30] joyent. http://joyent.com/.

[31] Js bin: Built for sharing, education and real-time rendering. http://coding.smashingmagazine.com/2012/07/23/js-bin-built-for-sharing-education-and-real-time/.

[32] Jsbin. http://jsbin.com/.

[33] Kodingen. https://kodingen.com/.

[34] Meebo. https://www.meebo.com/.

[35] Mercurial. http://mercurial.selenic.com/.

[36] Rational team concert. https://jazz.net/products/rational-team-concert/.

[37] Shiftedit. http://shiftedit.net/.

[38] Visual studio. http://www.microsoft.com/visualstudio/ita.

[39] Web2project. http://web2project.net/.

[40] Windows azure. http://www.windowsazure.com/it-it/.

[41] Cary L. BATES. Collaborative software debugging in a distributed system with collaborative step over operation, 04 2012.

[42] MN US) Bates, Cary L. (Rochester. Collaborative software debugging in a distributed system with symbol locking, October 2012.

[43] Lotus Development Corporation. *Groupware: Communication, Collaboration and Coordination*. Lotus Development Corporation, 1995.

[44] Charles A. Findley. Computer-supported cooperative work: history and focus. 1994.

[45] Jonathan Grudin. Why cscw applications fail: problems in the design and evaluationof organizational interfaces. In *Proceedings of the 1988 ACM conference on Computer-supported cooperative work*, CSCW '88, pages 85–93, New York, NY, USA, 1988. ACM.

[46] Jonathan Grudin. Computer-supported cooperative work: history and focus. 1994.

[47] Jonathan Grudin. Human-computer interaction. chapter Groupware and social dynamics: eight challenges for developers, pages 762–774. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[48] Bertrand Meyer, Arno Fiva, Ilinca Ciupa, Andreas Leitner, Yi Wei, and Emmanuel Stapf. Programs that test themselves. *IEEE Computer*, 42(9):46–55, 2009.

[49] Martin Nordio, Cristiano Calcagno, Bertrand Meyer, Peter Müller, and Julian Tschannen. Reasoning about Function Objects. In Jan Vitek, editor, *TOOLS-EUROPE*, LNCS. Springer-Verlag, 2010.

[50] Martin Nordio, H.-Christian Estler, Carlo Alberto Furia, and Bertrand Meyer. Collaborative Software Development on the Web. 2011.

[51] Martin Nordio, H.-Christian Estler, Bertrand Meyer, Julian Tschannen, Carlo Ghezzi, and Elisabetta Di Nitto. How do Distribution and Time Zones affect Software Development? A Case Study on Communication. In *6th International Conference on Global Software Engineering*. IEE, 2011.

[52] Martin Nordio, Carlo Ghezzi, Bertrand Meyer, Elisabetta Di Nitto, Giordano Tamburrelli, Julian Tschannen, Nazareno Aguirre, and Vidya Kulkarni. Teaching Software Engineering using Globally Distributed Projects: the DOSE course. In *Collaborative Teaching of Globally Distributed Software Development - Community Building Workshop (CTGDSD)*, New York, NY, USA, 2011. ACM.

[53] Martin Nordio, Roman Mitin, and Bertrand Meyer. Advanced Hands-on Training for Distributed and Outsourced Software Engineering. In *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, New York, NY, USA, 2010. ACM.

[54] Martin Nordio, Roman Mitin, Bertrand Meyer, Carlo Ghezzi, Elisabetta Di Nitto, and Giordano Tamburelli. The Role of Contracts in Distributed Development. In *Software Engineering Approaches for Offshore and Outsourced Development*, volume 35 of *LNBIP*, Berlin, Heidelberg, 2009. Springer-Verlag.

[55] Yu Pei, Yi Wei, Carlo A. Furia, Martin Nordio, and Bertrand Meyer. Code-based automated program fixing. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 392–395. IEEE, 2011.

[56] J.P. RodriÌĄ andguez, C. Ebert, and A. Vizcaino. Technologies and tools for distributed teams. *Software, IEEE*, 27(5):10 –14, sept.-oct. 2010.

[57] Lucy A. Suchman. Office procedure as practical action: models of work and system design. *ACM Trans. Inf. Syst.*, 1(4):320–328, October 1983.

[58] John C. Tang, Chen Zhao, Xiang Cao, and Kori Inkpen. Your time zone or mine?: a study of globally time zone-shifted collaboration. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, CSCW '11, pages 235–244, New York, NY, USA, 2011. ACM.

[59] Julian Tschannen, Carlo A. Furia, Martin Nordio, and Bertrand Meyer. Usable verification of object-oriented programs by combining static and dynamic techniques. In *Proceedings of the 9th International Conference on Software Engineering and Formal Methods*, SEFM '11. Springer, 2011.

[60] Julian Tschannen, Carlo A. Furia, Martin Nordio, and Bertrand Meyer. Verifying Eiffel Programs with Boogie. In *First International Workshop on Intermediate Verification Languages (BOOGIE 2011)*, 2011.

[61] Yi Wei, Hannes Roth, Carlo A. Furia, Yu Pei, Alexander Horton, Michael Steindorfer, Martin Nordio, and Bertrand Meyer. Stateful Testing: Finding more errors in code and contracts. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 440–443. IEEE, 2011.

[62] P. Wilson. *Computer Supported Cooperative Work:: An Introduction*. Springer, 1991.