

12-2011

# A Web-Integrated Environment for Component-Based Software Reasoning

Charles Cook

Clemson University, [ctcook@g.clemson.edu](mailto:ctcook@g.clemson.edu)

Follow this and additional works at: [https://tigerprints.clemson.edu/all\\_theses](https://tigerprints.clemson.edu/all_theses)

 Part of the [Computer Sciences Commons](#)

---

## Recommended Citation

Cook, Charles, "A Web-Integrated Environment for Component-Based Software Reasoning" (2011). *All Theses*. 1230.  
[https://tigerprints.clemson.edu/all\\_theses/1230](https://tigerprints.clemson.edu/all_theses/1230)

This Thesis is brought to you for free and open access by the Theses at TigerPrints. It has been accepted for inclusion in All Theses by an authorized administrator of TigerPrints. For more information, please contact [kokeefe@clemson.edu](mailto:kokeefe@clemson.edu).

A WEB-INTEGRATED ENVIRONMENT FOR  
COMPONENT-BASED SOFTWARE REASONING

---

A Thesis  
Presented to  
the Graduate School of  
Clemson University

---

In Partial Fulfillment  
of the Requirements for the Degree  
Master of Science  
Computer Science

---

by  
Charles Cook  
December 2011

---

Accepted by:  
Dr. Murali Sitaraman, Committee Chair  
Dr. Jason Hallstrom  
Dr. Brian Malloy

## ABSTRACT

This thesis presents the Web IDE, a web-integrated environment for component-based software reasoning. The Web IDE is specifically tailored to emphasize the relationships among various components in component-based software engineering (CBSE) and to facilitate reasoning. It allows students to use RESOLVE, a component-based, integrated specification and programming language, to build components and systems, providing real-time feedback that can be used to reason about the correctness of their component implementations. Real-time interaction and relationship focused component presentation reinforces CBSE and reasoning principles in a way not possible with traditional programming exercises and file management systems.

The Web IDE has gone through several stages of development, getting feedback from users and adding new functionality at each step. It has kept pace with web browser development by incorporating browser features, such as the file API and local storage, to provide enhanced functionality to users. Several undergraduate software engineering courses at Clemson and elsewhere have successfully used the Web IDE for both reasoning and team-based component development exercises, demonstrating the robust and useful nature of the Web IDE.

## ACKNOWLEDGMENTS

I would like to thank Murali Sitaraman, whose guidance made this project possible, and committee members Jason Hallstrom and Brian Malloy for their comments. I would also like to acknowledge and thank the other members of the Clemson RESOLVE Software Research Group for beta testing of the system. Their feedback and support has been a valuable asset throughout the duration of the project. And finally, I would like to thank my wife Michelle, kids Ethan and Olivia, and the rest of my immediate family for their never ending support throughout this endeavor.

This research was funded in part by NSF grants DUE-1022941 and DUE-0633506.

## TABLE OF CONTENTS

	Page
TITLE PAGE .....	i
ABSTRACT .....	ii
ACKNOWLEDGMENTS .....	iii
LIST OF TABLES .....	vi
LIST OF FIGURES .....	vii
CHAPTER	
I. INTRODUCTION .....	1
Component-Based Software Engineering .....	1
Reasoning and Education .....	2
Software Development Alternatives .....	3
Thesis Goals and Format .....	5
II. WEB IDE DESIGN OVERVIEW .....	6
Concept Development .....	6
Application Client Design .....	6
Application Server Design .....	7
Client/Server Interaction Overview .....	8
III. APPLICATION DESIGN AND DEVELOPMENT .....	11
Overview .....	11
RESOLVE Components and Relationships .....	11
Stage 1 .....	12
Stage 2 .....	15
Stage 3 .....	18
Summary of Improvements .....	22

Table of Contents (Continued)

	Page
IV. COMPONENT-BASED SOFTWARE DEVELOPMENT AND REASONING .....	24
Developing New Components .....	24
Component Reasoning .....	28
Building a System .....	30
V. EXPERIMENTATION AND EVALUATION .....	33
Team Software Development Assignment .....	33
Reasoning and Proving Assignment .....	37
Usability Survey.....	39
VI. RELATED WORK .....	42
Web-Based Software Development.....	42
Software Reasoning Environments.....	46
VII. CONCLUSIONS AND FUTURE WORK.....	49
Conclusions.....	49
Future Work .....	49
APPENDIX.....	52
REFERENCES .....	54

## LIST OF TABLES

Table		Page
1	Usability Study Likert Item Statements and Response Average Analysis Breakdown.....	52
2	Usability Study Likert Item Statements and In-Depth Response Details.....	53

## LIST OF FIGURES

Figure		Page
1	Client/Server Interaction Scheme .....	8
2	Application Server/RVC Interaction.....	10
3	RESOLVE Component Relationships in UML representation .....	12
4	Web-based IDE Stage1 .....	13
5	Verification Conditions for Selected Component.....	14
6	Web-based IDE Stage 2 .....	15
7	Stage 2 UI with component selected.....	16
8	Web-based IDE Stage 3 .....	19
9	Enhancement dropdown box.....	20
10	Stage 3 VCs tab.....	21
11	Team Assignment Component .....	37
12	Reasoning-based Assignment Questions .....	38



## CHAPTER ONE

### INTRODUCTION

Component-based software development and reasoning are becoming two important topics of software engineering research and they are beginning to find their way into undergraduate computer science curricula. However, there are no educational tools or environments available specifically designed to facilitate and emphasize component-based software engineering. This thesis presents a web-integrated environment that is designed from the ground up to provide an environment for CBSE and reasoning suitable for use in undergraduate computer science courses.

#### Component-based Software Engineering

Most modern software is built from components in a team development environment. The focus of component-based software engineering (CBSE) is on developing reliable components that can be reused within larger applications without the need for modifications specific to a particular application. The techniques used to develop components, however, must be repeatable and the applications created must have predictable properties [1].

An important aspect of CBSE is the relationship between the components. To be able to successfully create a finished product, the relationships between each component must be fully understood. These relationships affect the logical reasoning that must be used to predict the expected behavior of the program [2], and are often not understood

clearly by beginning software developers. For more advanced developers, a framework that organizes these relationships is beneficial. Many languages include built-in support for component development, such as the use of object inheritance with specific keywords like implements and extends, but an environment that manages and exploits the relationship among components can increase the effectiveness and efficiency of component developers and component users.

### Reasoning and Education

While several languages and systems exist to build component-based software, few facilitate reasoning about such software. Combining reasoning and software development is critical in this age of technological abundance. Almost without exception, people interact with electronic devices and, knowingly or unknowingly, put their lives in the hands of devices that rely on properly written software instructions to function without incident. There have been cases where billions of dollars, and in some instances lives, have been lost due to software programming issues. To mitigate the potential for software engineering disasters in the future, developers must also be able to use mathematical reasoning to ensure and understand the correctness of their programs [3].

The need to incorporate these skills into the next generation of software developers is not lost on computer science educators. The notion of integrating logical reasoning into software engineering is not new. Researchers have long recognized that using mathematical models and applying reasoning techniques [4] will allow students to more fully understand their code. This understanding is further reinforced as students

gain the ability to recognize potential models of representation on their own and realize the importance of mathematical reasoning [5]. Pioneers of these ideas have worked relentlessly to bring formal reasoning into the undergraduate curriculum [6]. As a result, many educational institutions now use discrete math and other courses [7] early in the curriculum for introducing computer science students to reasoning tools and techniques that can later be applied in software engineering courses. Some have begun offering discrete math courses taught by CS faculty and with a focus on software engineering.

Recognizing the importance of and building upon the foundation of material covered in these discrete math courses, researchers from several different institutions have identified and created an inventory of the basic mathematical reasoning principles needed to create high-quality applications [8]. Elements of the inventory have already been incorporated into existing undergraduate software engineering courses to provide a framework for teaching specification and reasoning principles that can easily be adapted for use at other institutions [9].

### Software Development Alternatives

In developing a suitable setting for teaching reasoning, one other choice to be made is between two ways of creating software—using a command line interface (CLI) or an integrated development environment (IDE). There are advantages and shortcomings to both. Sometimes the choice of which to use is a matter of personal preference, but at other times, the choice is based on the circumstances and situation surrounding the programming task at hand.

The CLI has long been a staple of both experienced and novice software developers. It provides straightforward access to tools for writing, compiling, and running computer programs and is generally the first, and often the only, tool used by undergraduate computer science students during their studies. These command line tools are usually sufficient for relatively simple student projects, but for more complicated software projects involving several components, managing everything from the CLI can become cumbersome.

The other option is to use an IDE. Modern IDE's are typically all encompassing, providing developers with an all-in-one, point-and-click solution to write, compile, and run programs. IDE's often contain extra features like file and package managers, collaborative tools, and add-on programming language modules that make them ideal for use in team-based development projects. Some software companies may even require project team members to use specific development environments and supply them with workstations deployed with duplicated system images. Research groups, small companies, and individuals, however, often do not have the same IT resources available and must manage each workstation individually. Installing, configuring, and maintaining the core IDE, any required development plugins, and any supporting software packages can be a time-consuming and challenging process. Many software developers still opt to utilize an IDE, because having the ability to easily access and manipulate project components is often enough to offset the extra effort required to get an IDE set up.

## Thesis Goals and Format

Given the above background, this thesis makes the following contribution: It will show that it is possible to create a user-friendly web-integrated environment for teaching component-based software development and reasoning for effective use in computer science courses. The thesis goal is supported through significant experimentation in classrooms. The environment has also been used in software engineering research, though no formal data on usage is available at this time.

Ubiquitous high-speed internet connections and modern web browsers allow us to remove the hassles associated with using the CLI or an IDE to develop component-based software. The Web IDE will be an environment that presents components based on their relationship, not simply on their file system location. It will use RESOLVE, a component-based, integrated specification and programming language [10], to teach and reinforce specifications, component relationships, and reasoning principles to the modern computer science student.

This thesis will present an overview of the technologies used to implement the Web IDE in chapter 2. Chapter 3 discusses the details involved during the development and implementation of the Web IDE. Chapter 4 presents demonstrations of two typical usage scenarios—creating components and component verification. Chapter 5 discusses utilizing the Web IDE in several courses and then provides the results from a usability survey. Chapter 6 presents related work and chapter 7 presents conclusions and discusses future directions for the Web IDE.

## CHAPTER TWO

### WEB IDE DESIGN OVERVIEW

This chapter presents an overview of the design of the Web IDE, a client-server application. It begins with a general overview of the two parts of the application, the user interface and the application server. In the process, it motivates the choice of programming languages used for implementation. It then goes on to describe how the parts interact to create the overall application.

#### Concept Development

From the beginning the Web IDE was envisioned as a fully web-based client/server application. In a typical scenario, a user would download and install a client program to their computer, which would then communicate with a centralized server over the internet. As much logic and processing as possible is offloaded to the client-side program, allowing the application to take full advantage of the client's hardware while minimizing the use of server resources and thus maximizing the potential of the server. This project, as a web application, uses the client's web browser as an OS-agnostic client program and an application server running on a dedicated server at Clemson.

#### Application Client Design

Historically, the web browser has not been the ideal choice to use as a cross-platform client program. Many web applications make use of Adobe's Flash or Microsoft's Silverlight plugins to run fully within the browser, but these plugins are not

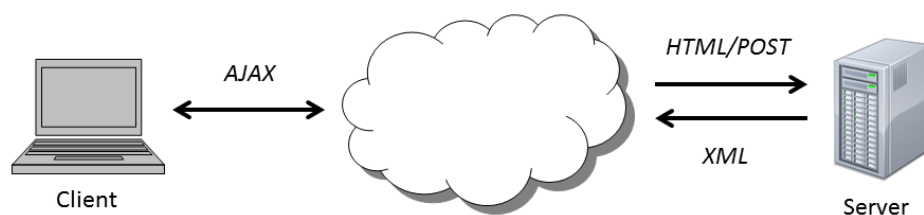
native to the browser and must be downloaded and installed by the user and are often operating system and browser dependent. Additionally, the JavaScript engines built into older browsers were not fast or advanced enough to be used for complicated real-time tasks. However, all modern web browsers now have native JavaScript engines that are fast and more than capable of running complex applications fully within the web browser. To exploit these capabilities, JavaScript was chosen as the primary language for client-side user interface development (UI) for this project.

### Application Server Design

The primary job of the application server would be to provide two-way communications between the web browser-based UI and the server machine. There are several robust programming languages capable of handling this task. The choice for this project is dictated by the need to host the RESOLVE Verifying Compiler (RVC). The RVC is capable of generating and proving verification conditions for RESOLVE code, and translating RESOLVE programs to Java [10], and will provide compilation services for the application. Thus, the application server would also need to act as an intermediary between the UI and the RVC, easily and reliably able to communicate with both. Because the RVC is written in Java, Java Servlets were the natural choice to use to implement the application server software.

## Client/Server Interaction Overview

The two components of the application, the JavaScript client frontend and the Java Servlet backend would work together using the communication protocols in Figure 1 to provide a dynamic cloud-computing experience. The UI would consist of a single, static HTML web page providing the basic formatting for the UI. JavaScript would then be used to load the application and dynamically change page content. Users would interact with the application through controls embedded directly within the page, without needing to reload the page or use the browser's navigational controls. Asynchronous JavaScript and XML (AJAX) would be used handle communication with the server. Clicking a button requiring content to be retrieved from the server would trigger a JavaScript AJAX function, sending a POST request to the server and defining a handler function to be executed after the server responds. The request would be sent asynchronously, allowing the function to complete without locking up the user's browser while the server processes the request. The server would receive the request, prepare the content, and transmit it back as XML (Extensible Markup Language). When the client receives the XML response, the handler function is automatically called, extracting and processing the content and updating the user interface as needed.

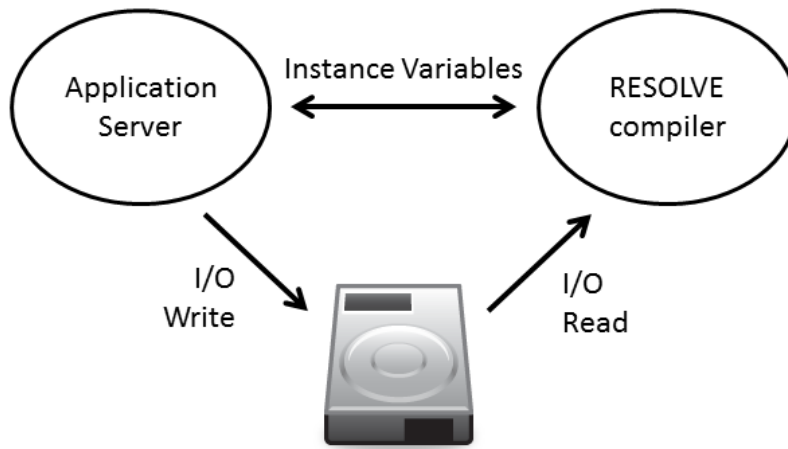


**Figure 1 Client/Server Interaction Scheme**



Developing the application server would be less straightforward than the UI because it would need to interact with the RVC, a tool designed to be executed from the command line interface. Because both are implemented in Java, the server would be able to instantiate the RVC, but there would be no means to pass information between the two. Minor modifications would need to be made to the RVC, enabling the application server to directly send commands currently meant to be received as command line arguments, properly invoking the RVC.

Another issue to overcome was that, like most compilers, the RVC is designed to compile source files saved on the local file system. With no way to feed user submitted files directly to the RVC, the server would need to write them all to the file system before they could be compiled. Thus, the server would first extract user-submitted files from the POST data of the HTTP request, then save them with randomly generated filenames. While this would make multi-user, temporary file storage feasible, it would add an extra layer of complexity due to the strict source file naming policy enforced by the RVC. The actual name of a RESOLVE source file must exactly match the module name in the source code. To overcome this issue, the server application would examine the content of the submitted file, looking for the module name and replacing it with the randomly generated name before writing it to disk. After making the filename replacement and writing the file to disk, it would then invoke the RVC and execute the compile job. Once complete, the server application would delete the saved user file, encode the RVC output as XML, and send the XML results back to the client.



**Figure 2 Application Server/RVC Interaction Diagram**

## CHAPTER THREE

### APPLICATION DESIGN AND DEVELOPMENT

This chapter begins with an overview of the overall goals of the project and the development process. It goes on to present the goals, important implementation and design details, and outcomes for the three stages of development.

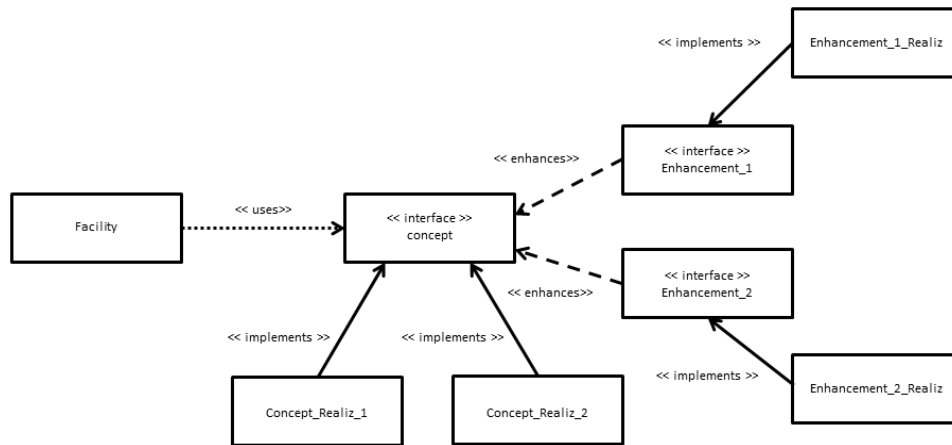
#### Overview

Web IDE development began two years ago in an effort to provide a better platform for CBSE research and education using RESOLVE at Clemson. The ultimate goal of the project was provide users with an easier way to make use of the RVC and without having to install and maintain any additional software. It was meant to provide easy and uniform access to the RESOLVE library of components. The library would not be presented to users as a normal file system hierarchy, but rather in a way that emphasized and reinforced the relationship between the components.

#### RESOLVE Components and Relationships

RESOLVE uses four types of components—concepts, enhancements, realizations, and facilities. A RESOLVE concept is similar to a Java Interface. It contains signatures for operations and specifications used for proving their correctness. A concept can be implemented by multiple realizations, files containing implementations for the operations in the concept. A concept can be extended by adding an enhancement, a component providing additional operations and specifications. Like a concept, an enhancement can

also be implemented by multiple realizations. The final component type, the facility, instantiates and uses the other component types.



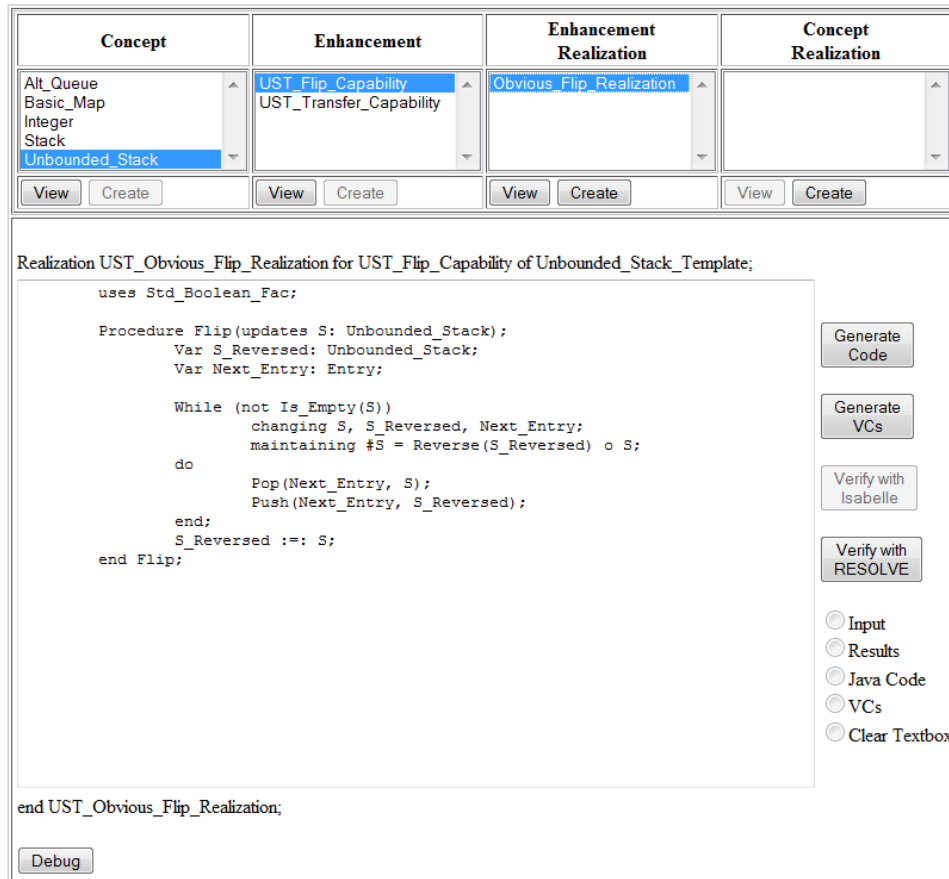
**Figure 3 RESOLVE Component Relationships in UML representation**

### Stage 1

The first stage of development, completed and ready for use in spring 2010, resulted in a basic, table-based user interface that allowed online users to perform the same basic RVC tasks available to stand-alone users. Users were able to easily browse through the RESOLVE library of components; after browsing to the Web IDE, users were greeted with a list of concept modules. Clicking a concept name would generate a request to the application server to query the server's file system and get the contents of the specific concept file, as well as the filenames of any enhancements or realizations related to it. The UI would then display the names of any available enhancements or realizations as options in dedicated select boxes, which the user could then also click to

select and view. The contents of files were presented to users by way of a textbox, which allowed them to be able to modify the code, compile, and see the effects in real-time.

Figure 4 shows the UI layout with a component selected.



**Figure 4 Web-based IDE Stage1**

At this stage of user interface design, users were also able to create and compile new components; however these new components, and any changes made to existing components, only persisted while the component was selected. Selecting another component would bring its contents into view, overwriting whatever was previously

there. With a component selected and in view, the user was able to generate verification conditions (VCs), verify, or translate to Java as appropriate for the selected component type. When the RVC output was received, the RESOVLE code was temporarily stored in a JavaScript variable for later use, and the output was displayed in the textbox. The user could then use the radio buttons located beneath the command buttons to easily compare the RVC input and output. See Figure 5 for an example of RVC generated VCs.

Concept	Enhancement	Enhancement Realization	Concept Realization
Alt_Queue Basic_Map Integer Stack Unbounded_Stack	UST_Flip_Capability UST_Transfer_Capability	Obvious_Flip_Realization	
View Create	View Create	View Create	View Create

Realization UST\_Obvious\_Flip\_Realization for UST\_Flip\_Capability of Unbounded\_Stack\_Template;

```

//
// Generated by the RESOLVE Verifier, March 2009 version
// from file: a53132d1d.rb
// on:      Fri Sep 25 13:32:59 EDT 2009
//
Free Variables: S:*Modified_String_Theory.Str(*Entry), ?Next_Entry:*Entry,
?S_Reversed:*Modified_String_Theory.Str(*Entry),
?S:*Modified_String_Theory.Str(*Entry),
??S:*Modified_String_Theory.Str(*Entry), Next_Entry:*Entry,
S_Reversed:*Modified_String_Theory.Str(*Entry)

VC: 0_1

true
=====>
S = (Reverse(empty_string) o S)

VC: 0_2

(S = (Reverse(?S_Reversed) o ??S) and
??S /= empty_string and
.....)

end UST_Obvious_Flip_Realization;

```

Generate Code

Generate VCs

Verify with Isabelle

Verify with RESOLVE

Input

Results

Java Code

VCs

Clear Textbox

Debug

**Figure 5 Verification Conditions for Selected Component**

## Stage 2

With the successful completion of stage one development and a functional proof-of-concept in hand, the project was able to continue. The goal of the second stage of development was to transform the Web IDE from a simple and utilitarian tool into a more visually appealing and user-friendly web application.

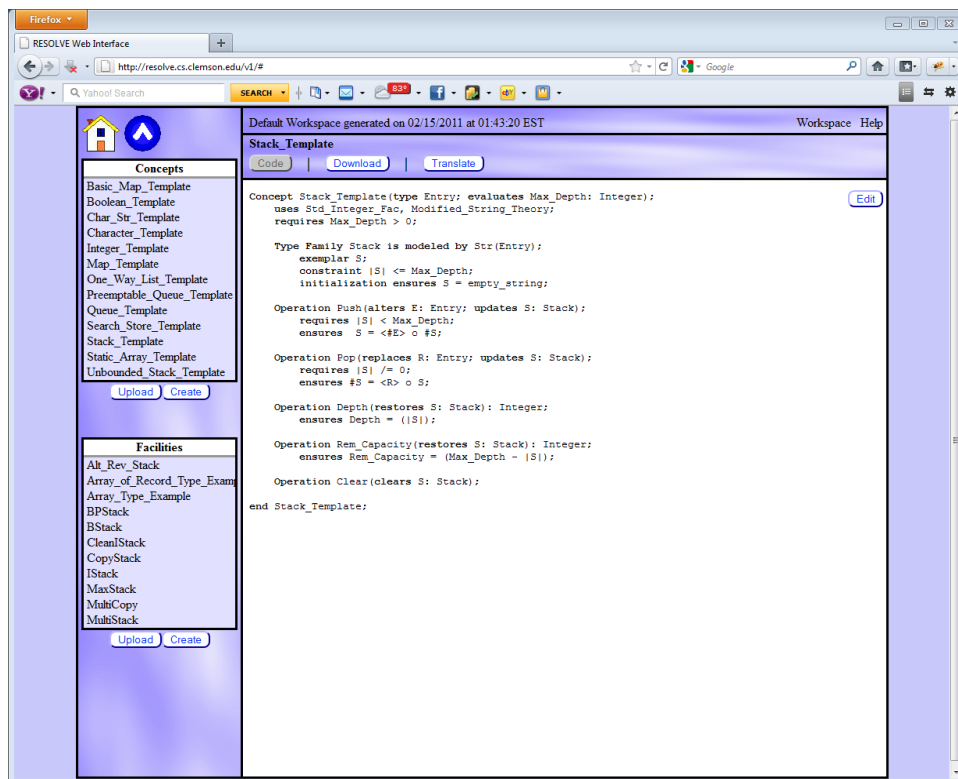


Figure 6 Web-based IDE Stage 2

Development of the second version of the application began with an overhaul of the page framework and was completed at the end of summer 2010. The tables used to format the UI in stage 1 were removed in favor of HTML div elements that could be dynamically changed using JavaScript and CSS (Cascading Style Sheets). This allowed

greater visual control of the content seen by the user and easier expansion and modification of the design of the base HTML page. The UI was redesigned to improve space management (Figure 6); the dedicated realization, enhancement, and enhancement realization select boxes that were often empty and under-used during normal usage in the previous design were removed from the default view. The concept list was re-positioned to the left side of the page. Clicking a concept now would dynamically hide the list of concepts and any available enhancements and realizations would appear instead. Figure 7 presents the UI with a component selected.

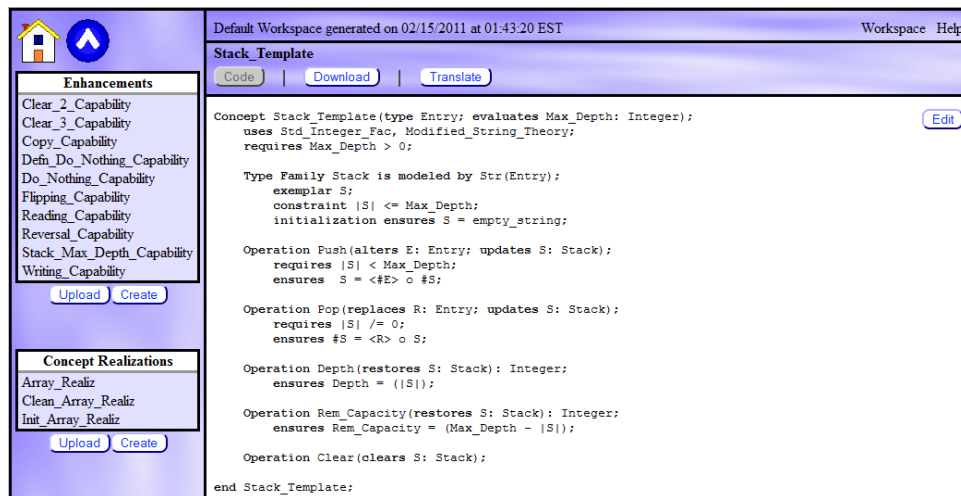


Figure 7 Stage 2 UI with component selected

To improve readability, syntax highlighting was added. RESOLVE source code was analyzed by the client-side JavaScript and keywords were highlighted in bold and displayed as HTML. An edit button was also added which when clicked would dynamically remove the HTML-formatted RESOLVE code and replace it with an



editable textbox. This version also introduced the ability to view and compile RESOLVE facilities.

In addition to the visual UI redesign, changes were made to behind-the-scenes handling of the RESOLVE component library. Previously, the user interface would query the server each time a filename was clicked. This method resulted in a high volume of server requests, each of which required a disk read on the server. For this stage a new technique was devised. An XML workspace file containing a hierarchical representation of the entire component library would be pre-generated and stored on the server. When a user entered the URL of the Web IDE, the server would only need to open one file. The workspace file would be read in and its contents would be sent directly to the client. The server application was also modified to use Java Sessions. Using session information, the server could now distinguish between new and returning browser users. When a new user session was detected, it would first open and read the workspace file into session memory, then send it to the user. Otherwise, it would send the configuration file already stored in memory. This increased server capacity by minimizing time-consuming disk I/O.

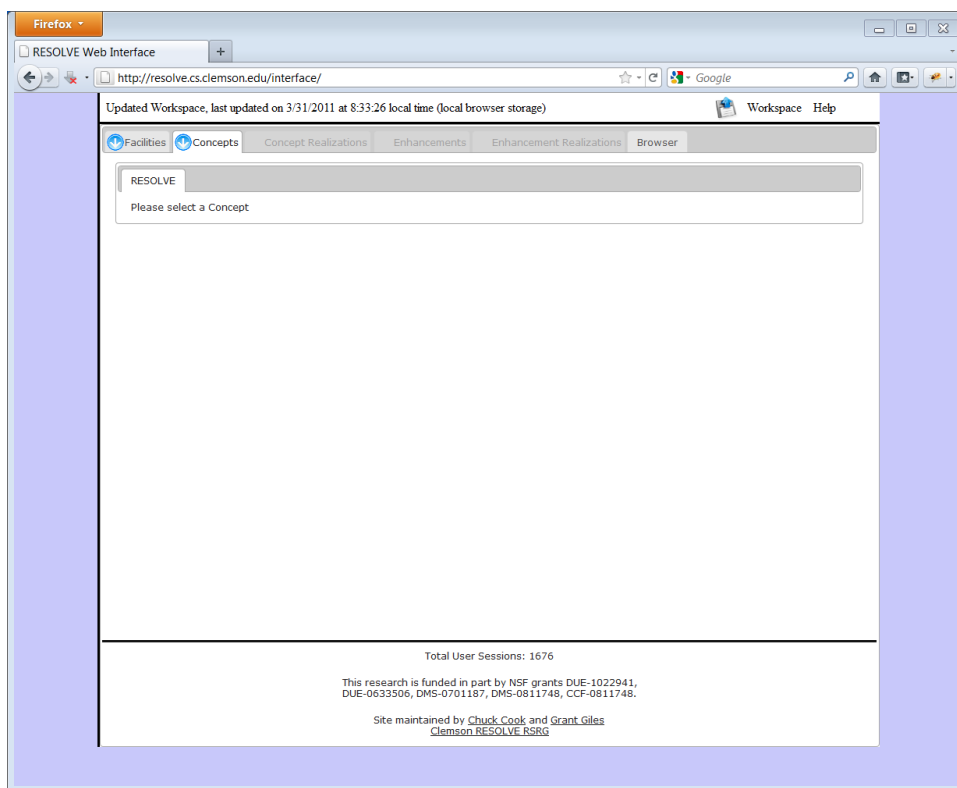
Changes to the application server meant the user's browser would receive the entire component library at once, allowing it to be stored and manipulated as a hierarchy within JavaScript. Clicking on a component name now did not initiate any communication with the application server; the content and related components were retrieved locally, increasing the responsiveness of the UI. This improvement also allowed user-created components and changes to existing components to be stored. Functionality

was added to regenerate the XML workspace file contents from the locally stored component library hierarchy and send it back to application server session memory. This meant that local changes to the component library would now persist between page reloads. A dropdown menu of workspace control options was added to the UI. This gave users the ability to download and save a local copy of the workspace file. A saved workspace file could also now be uploaded to the server, which would replace the default workspace in session memory, and be loaded into the UI. This update allowed users to be able to save the state of their work between browser and computer restarts. In addition to the workspace file, individual RESOLVE component files could also be downloaded or uploaded and added to the workspace.

### Stage 3

While the second stage of development resulted in a UI that was a dramatic improvement over the previous version, it did not make use of the advanced capabilities included in the most recent browser releases. It consisted of a simple page design that lacked the look and feel of a modern web application. Therefore, like the previous stage, this one also started out with a reworking of the UI and was ready for use in spring 2011. In addition, a third party JavaScript library was used for the first time. By using the JQuery library to handle common and simple tasks, like button click handling and dynamic page manipulation, time and effort could be dedicated to the design and implementation of the functionality required for the proper operation of the UI.

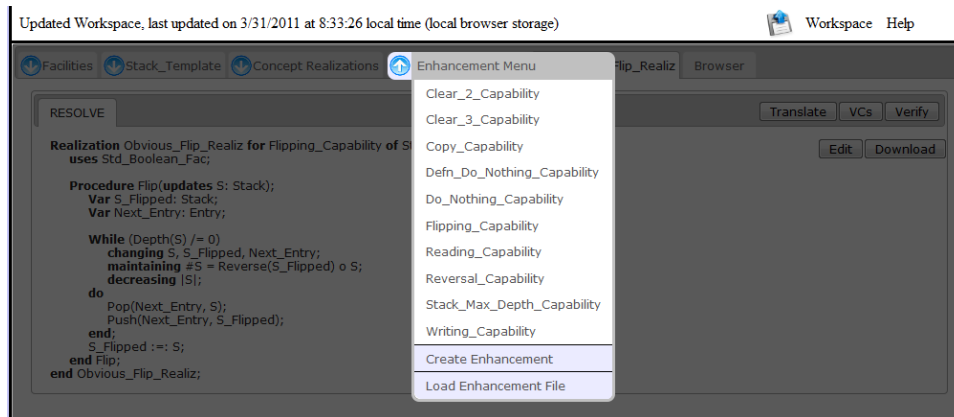
The UI itself was redesigned to mimic the basic feel of the modern web browser by using tabs to manage page content. Using Tabs allowed users to quickly and intuitively manipulate the interface, changing the content currently visible. To implement the tabbed page design, the JQuery UI plugin was used. The entire lower portion of the user interface was replaced with a horizontal row of primary tabs (see Figure 8).



**Figure 8 Web-based IDE Stage 3**

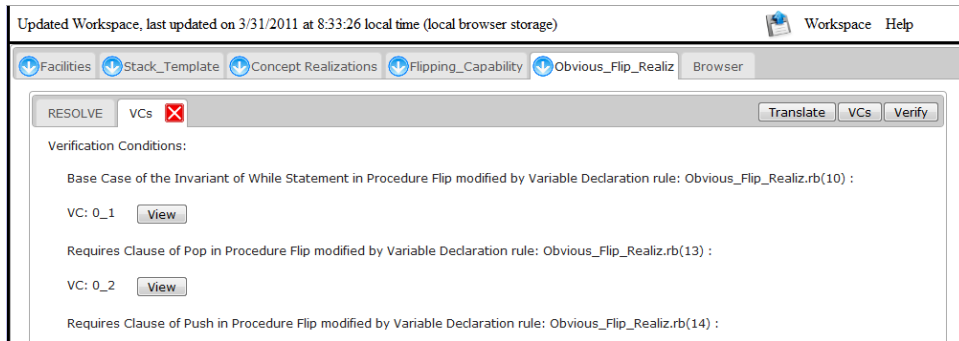
The first five tabs were dedicated to a specific type of module—facility, concept, concept realization, enhancement, and enhancement realization. Each tab contained an icon that when clicked would display a dropdown list of components available to select. Using dropdown lists instead of the select boxes used previously allowed the content

viewable at any time to be maximized. When a component was selected, its content would appear in the tab body and the dropdown icon for any available associated components would be activated. Figure 9 shows an enhancement dropdown list for a selected concept.



**Figure 9 Enhancement dropdown box**

In addition to the main module tabs at the top, the component content, editor textbox, and any RVC results were now displayed in sub-tabs (see Figure 10). This design gave users the ability to edit one component while simultaneously referring to previous RVC results or other module types without having to save, navigate away, navigate back, and reopen the component.



**Figure 10 Stage 3 VCs tab**

The last primary tab introduced a new feature, the browser. The browser tab was designed to present the component library in a more traditional, file browser-type view. It allowed users to select and view any number of files in sub-tabs of the browser, regardless of component type. It also allowed users to view RESOLVE theory files for the first time.

Along with the visual UI update, changes were made to take advantage of new browser functionality; further minimizing interaction between the user's browser and the application server. Previously, when a user loaded a saved workspace file or individual component file into the UI, the browser automatically uploaded the file to the server, which simply turned the file around and sent it back to the browser to be added to the UI. This was inefficient, but JavaScript engines in older web browsers were unable to manipulate local files. However, this has recently changed; all the major browsers now include an implementation of the File API. This API now gives JavaScript direct access the contents of a local file. For this stage, all file upload functionality has been modified to make use of the File API, removing the need to communicate with the application server when a user loaded a component from a file.

Another new browser feature is also exploited for this stage of development. With the emergence of a significant number of popular web applications, browser developers have recognized the need to store more user data within the browser. This has resulted in three ways to store local data within the browser—session storage, local storage, and database storage, being included as part of the upcoming HTML5 specification [11]. Browser developers have taken different positions on which type of storage is preferred, resulting in varying degrees of implementation of each type across the range of browsers. However, one of the types is supported by all of the major browsers—local storage. Because of this fact, the UI was modified to store the workspace file in browser local storage instead of in server session memory. This approach has two important advantages. Updated workspace files are no longer sent to the application server, which in turn has resulted in a further reduction of client/server communications and has improved server capacity by reducing the memory footprint of each user session. The second advantage is that the state of the user’s current workspace persists between computer and browser restarts without the need to download and save the workspace file locally.

### Summary of Improvements

This chapter discussed the changes and improvements made through the stages of development. More features and complexity were added as development progressed. The Web IDE began as a simple, HTML table-based proof of concept user interface for the RVC and morphed in to a modern and intuitive web application taking advantage of

state-of-the-art browser technologies like HTML 5, CSS, and local storage to provide a hassle-free, persistent CBSE and reasoning environment.

## CHAPTER FOUR

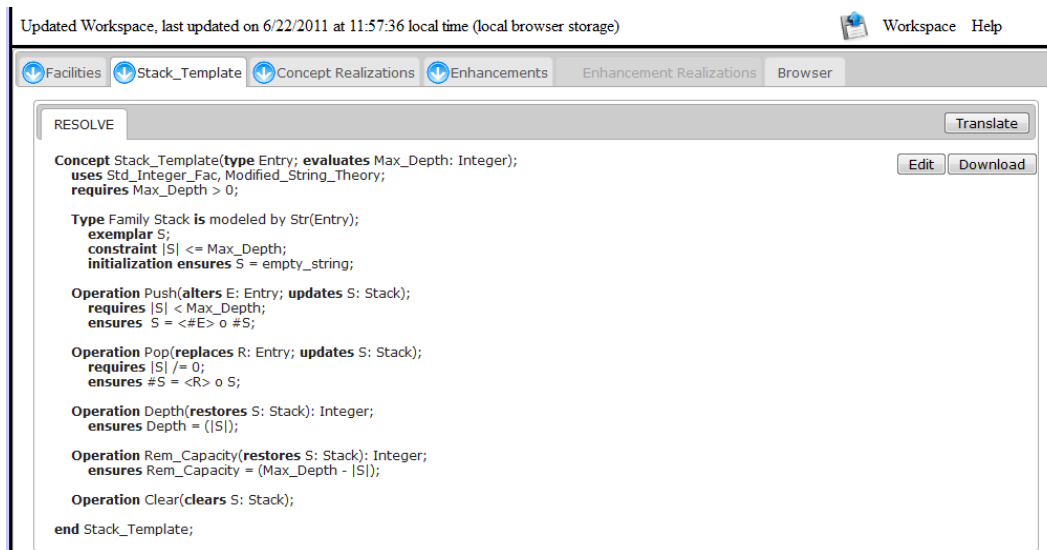
### COMPONENT-BASED SOFTWARE DEVELOPMENT AND REASONING

This chapter introduces the general functionality and usability of the Web IDE by describing three typical usage scenarios. The first scenario involves adding new workspace components. The second illustrates generating verification conditions (VCs) and proving the correctness of an example realization. The final scenario demonstrates building a system from the newly added components.

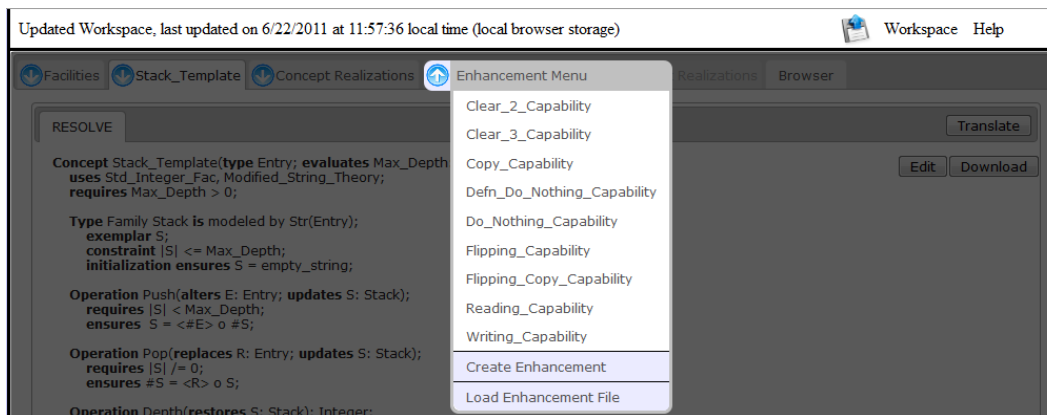
#### Developing New Components

The first step is to open a web browser and navigate to the Web IDE URL, <http://resolve.cs.clemson.edu>. The initial screen prompts users to select a concept. Clicking the blue and white dropdown icon in the tab labeled “Concept” brings the concept dropdown list into view. If, for example, the concept “Stack\_Template” is located and clicked, the specification of Stack\_Template is brought into view in the RESOLVE tab and the dropdown icons for the “Concept Realization” and “Enhancement” tabs are enabled. These tabs can be used to browse or create implementations or enhancements for Stack\_Template.

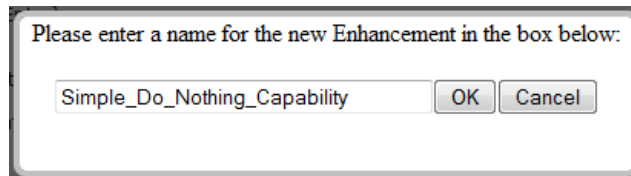




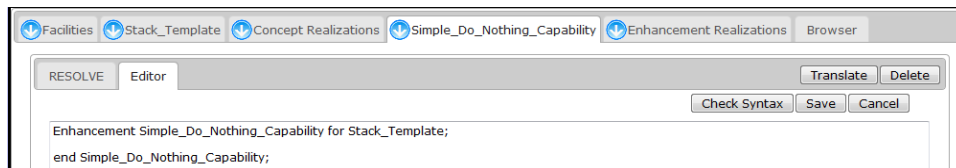
To create a Stack\_Template enhancement, the “Enhancement” dropdown icon is clicked and the enhancements available for Stack\_Template are displayed.



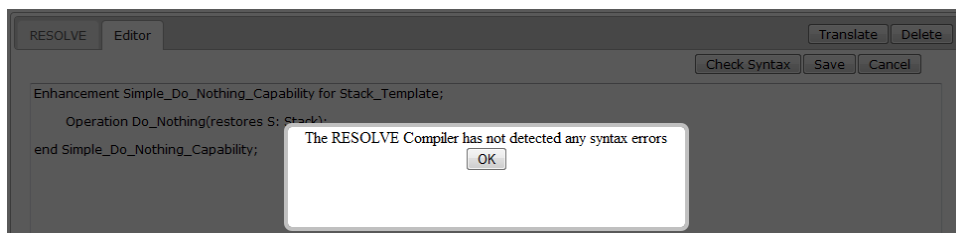
To create a new enhancement, the “Create Enhancement” option may be clicked. This opens up a dialog box requesting the name of the new enhancement; “Simple\_Do\_Nothing\_Capability” is typed in and “OK” is clicked.



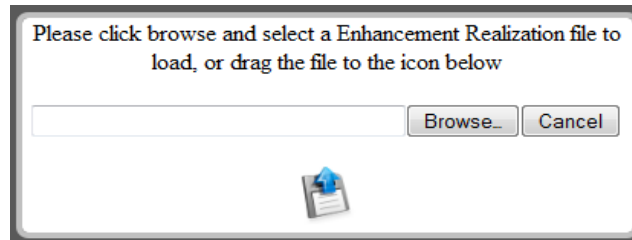
The interface then creates a stub for Simple\_Do\_Nothing\_Capability and automatically opens it in the component editor tab.



After typing in the specification “Operation Do\_Nothing(restores S: Stack);” for the enhancement operation Do\_Nothing, the “Check Syntax” button is clicked to make sure there are no syntax errors.



Because the Web IDE reports no syntax errors, “Save” is clicked, saving the changes made to the enhancement stub. Next, an implementation for the newly created enhancement must be added. This time, instead of creating the component manually, suppose that a realization file has been prepared in advance and needs to be loaded into the workspace. To accomplish this task, the dropdown icon on the “Enhancement Realization” tab is clicked and the “Load Enhancement Realization File” option is selected. This opens a dialog box asking for the file location.



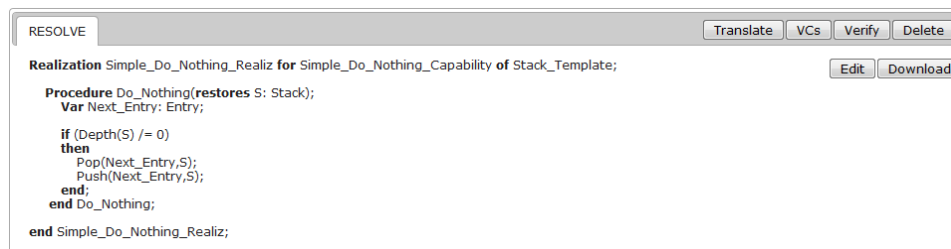
The browse button can be used to find and select the file with the operating system’s file browser, or the file can be dragged and dropped onto the disk icon. After dragging the file Simple\_Do\_Nothing\_Realiz.rb and dropping it on the disk icon, its code is displayed on the RESOLVE tab with keywords highlighted in bold.

```
RESOLVE Translate VCs Verify Delete  
  
Realization Simple_Do_Nothing_Realiz for Simple_Do_Nothing_Capability of Stack_Template; Edit Download  
Procedure Do_Nothing(restores S: Stack);  
Var Next_Entry: Entry;  
  
  if (Depth(S) /= 0)  
  then  
    Pop(Next_Entry,S);  
    Push(Next_Entry,S);  
  end;  
end Do_Nothing;  
end Simple_Do_Nothing_Realiz;
```

## Component Reasoning

To incorporate formal reasoning into CBSE, the Web IDE is able to generate verification conditions (VCs) for RESOLVE realizations. The Web IDE uses the specifications, or pre- and post-conditions, found in concepts and enhancements to generate the VCs for the implementation code found in realizations. A VC for a line of code is made up of two parts—the givens, a list of facts known to be true prior to that line of code, and the goal needed to be proven for the line of code to be correct. For the entire block of code to be considered correct, the goal for each and every VC must be proven using only the givens for the particular VC.

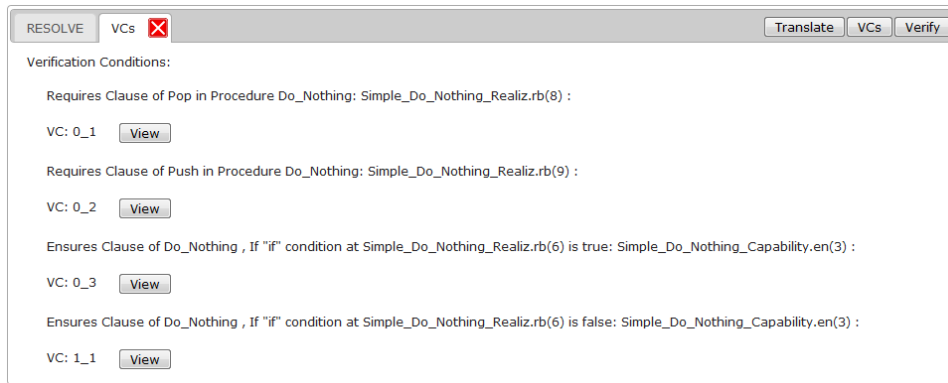
To use the reasoning features of the Web IDE, a realization must be selected. Building upon the previous section, suppose `Simple_Do_Nothing_Realiz` was still selected.



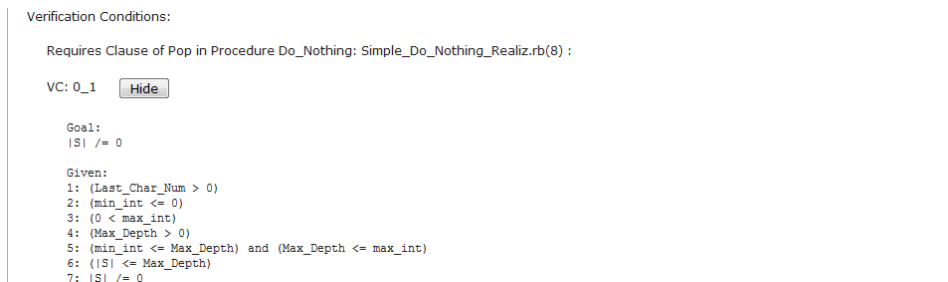
The screenshot shows a web browser window with a tab titled "RESOLVE". The interface includes a toolbar with buttons for "Translate", "VCs", "Verify", and "Delete". Below the toolbar, the text "Realization Simple\_Do\_Nothing\_Realiz for Simple\_Do\_Nothing\_Capability of Stack\_Template;" is displayed, followed by "Edit" and "Download" buttons. The main content area contains the following code:

```
Procedure Do_Nothing(restores S: Stack);
  Var Next_Entry: Entry;
  if (Depth(S) /= 0)
  then
    Pop(Next_Entry,S);
    Push(Next_Entry,S);
  end;
end Do_Nothing;
end Simple_Do_Nothing_Realiz;
```

The “VCs” button may be clicked to begin VC generation for the component. When complete, an overview of the VCs is displayed in a new tab.



This tab contains an ID number, justification with line number, and “View” button for each VC generated. The “View” button for an ID may be clicked to expand the page and show the givens and goal for that VC. The “View” button is dynamically replaced with a “Hide” button that may be used to minimize the VC.



The “Verify” button may be clicked to attempt to automatically prove the VCs for Simple\_Do\_Nothing\_Realiz. The Verify tab appears, displaying a summary of the VCs and the current prover status for each VC.



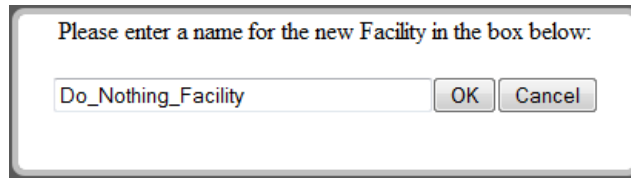
The statuses are updated in real-time as the RVC complete proofs for each VC. When complete, the VC label is updated with either a green checkmark if proven or a red X icon otherwise and the time taken for the proof.



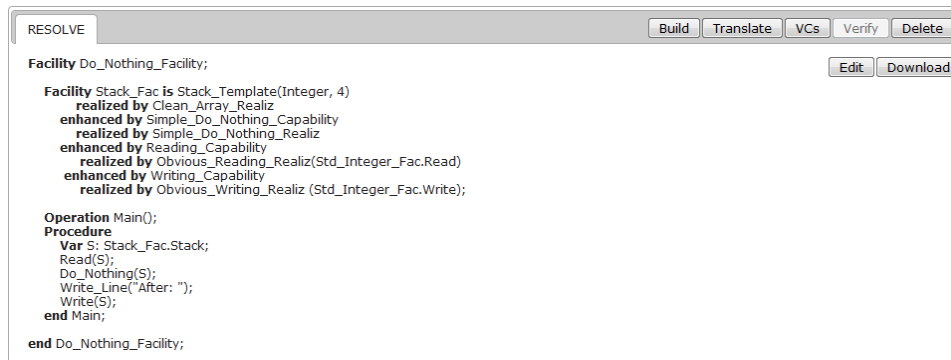
Because each of the four VCs was successfully proven, the component Simple\_Do\_Nothing\_Realiz is reasoned to be correct.

### Building a System

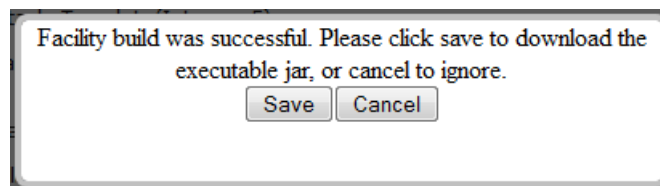
To instantiate and use concepts and their enhancements in a larger component-based system, a facility must be created. To create a facility using Stack\_Template and the enhancement and realizations previously added, the “Facility” dropdown icon is clicked and the “Create Facility” option is selected, opening the facility name dialog box. “Do\_Nothing\_Facility” is typed into the textbox, and “OK” is clicked.



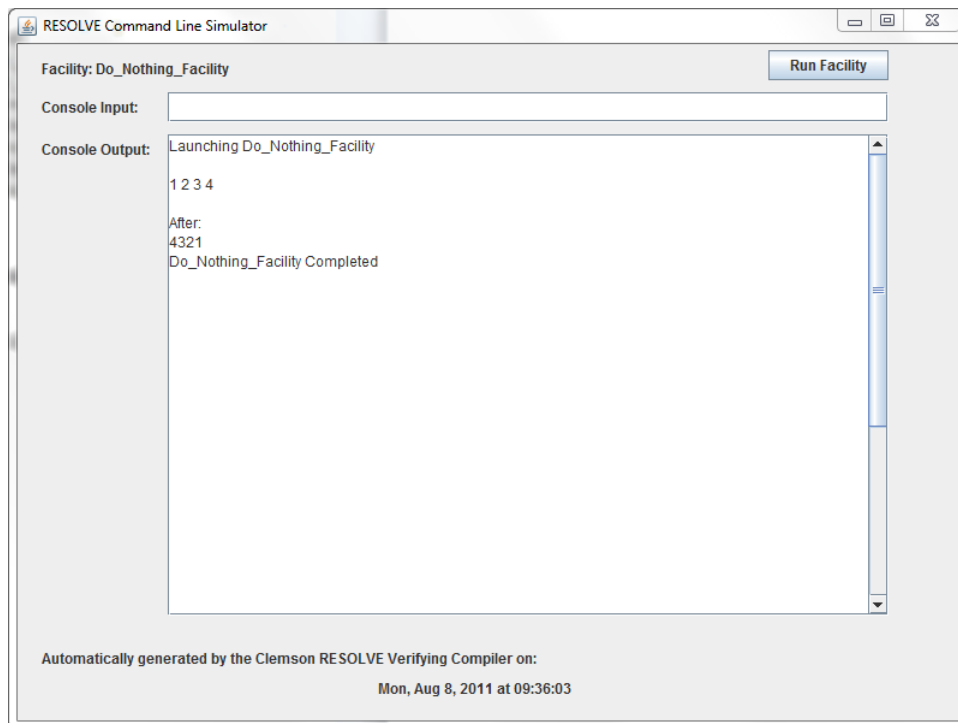
Like the enhancement created earlier, the Web IDE automatically generates and opens the stub for the new facility in the text editor tab. The facility code is added, checking syntax as needed until the code contains no syntax errors. When finished, clicking “Save” adds the facility to the workspace.



With the facility complete, “Build” can be clicked, translating the RESOLVE files to Java and generating an executable jar. When the operation is completed, a box appears claiming success and prompting to download the jar.



Clicking on “Save” begins the file download. After the file has been downloaded, the jar, “Do\_Nothing\_Facility.jar,” is double-clicked to execute the file and launch the packaged RESOLVE Command Line Simulator. Clicking “Run Facility” starts the program.



The Facility builds a stack of four integers that are read in from the command line. Four integers, “1 2 3 4,” are typed into the Console Input box, and enter is pressed. The simulator repeats the input sequence and executes the rest of the program, creating and populating the stack, calling the Do\_Nothing enhancement operation, and printing out the contents of the stack to the Console Output box.



## CHAPTER FIVE

### EXPERIMENTATION AND EVALUATION

This chapter begins with a presentation of two different course assignments that were designed to use the Web IDE to exercise and reinforce software engineering principles. It describes each assignment and summarizes the outcomes from using each in an undergraduate software engineering course. It ends with a discussion of a usability survey given to students in two courses who used the Web IDE for coursework.

#### Team Software Development Assignment

##### Assignment Overview

This assignment was designed to introduce students to team-based component design and design-by-contract and was given to students taking CPSC 372: Introduction to Software Engineering at Clemson in the fall 2010. The class was broken down into teams of three and the students were required to use the Web IDE to complete the assignment. Each team was responsible for turning in a single workspace file for each of the three parts of the assignment. Each part included tasks to be completed by individual team members, requiring them to independently develop components while relying solely on given contracts. When complete, the components could be combined into a system with minimal integration costs, demonstrating a core tenet of CBSE, the assignment, and the Web IDE.

## Part 1

The first part involved three tasks, each to be completed individually by a team member:

- i. Develop realization `Circular_PQ_Realiz_1` for `Preemptable_Queue_Template`
- ii. Develop realization `Circular_PQ_Realiz_2` for `Preemptable_Queue_Template`
- iii. Develop facility `PQ_Test_Facility`

Tasks (i) and (ii) were to develop realizations for a given concept. The students were given the internal contracts for both realizations, which were identical except for a slight difference in the correspondence clauses. This difference would require each to be implemented differently. Task (iii) was to develop a test facility that would fully exercise the realization implementations from (i) and (ii) by declaring a `Preemptable_Queue` of `Integers`, calling every `Preemptable_Queue` operation, and creating a local operation that would write out and clear the `Preemptable_Queue`.

## Part 2

This part required the teams to complete four tasks, the first three as individuals and the last as a group:

- i. Develop realization `Stack_Based_Realiz` for `Preemptable_Queue_Template`
- ii. Develop realization `PQ_Copying_Realiz` for `PQ_Copying_Capability` of `Preemptable_Queue_Template`

- iii. Develop realization Linear\_PQ\_Search\_Realiz, for PQ\_Rotating\_Search\_Capability of Preemptable\_Queue\_Template
- iv. Revise facility PQ\_Test\_Facility

Task (i) was to create a new realization for Preemptable\_Queue\_Template that used a Stack to implement a Preemptable\_Queue. There was a stipulation that the team member that worked on task (iii) from part one was required to complete this task and was given only the internal contract. Tasks (ii) and (iii) were to implement realizations for the listed enhancements, which already existed in the RESOLVE component library. Task (iv) was to modify the test facility to test the operations implemented in (ii) and (iii). This task was to be completed collectively as a team.

### Part 3

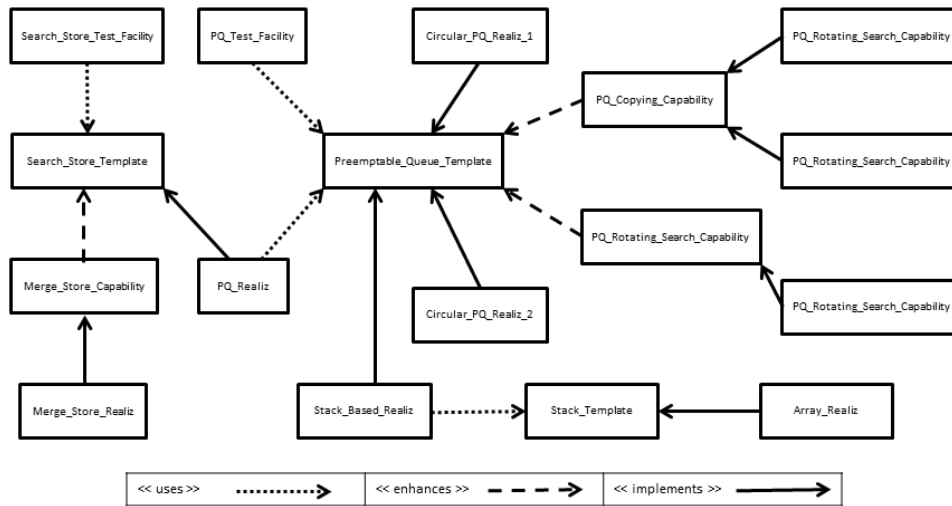
The final part of the assignment also consisted four tasks, three to be done individually and one as a group:

- i. Develop realization PQ\_Recursive\_Copying\_Realiz for PQ\_Copying\_Capability of Preemptable\_Queue\_Template
- ii. Develop realization Merge\_Store\_Realiz for Merge\_Store\_Capability of Search\_Store\_Template;
- iii. Develop facility Search\_Store\_Test\_Facility
- iv. Implement concept Search\_Store\_Template using Preemptable\_Queue\_Template and its rotating search enhancement

Task (i) required a recursive implementation for the given enhancement to be completed by the team member who did task (i) of part two. Task (ii) required an implementation for an enhancement of `Search_Store_Template` and task (iii) required the development of a facility that would test all `Search_Store` operations implemented in (ii). The final task was to collectively implement a `Search_Store` using a `Preemptable_Queue`.

### Assignment Results

For each of the parts in this assignment, every team was required to download the workspace file containing all the implemented RESOLVE files and submit it to the instructor via email. The instructor could then easily view their completed RESOLVE components and test their facilities by generating and running an executable jar file. Each of the teams was able to complete the assignment and turn it in on time, resulting in a class average of 92.7%. The assignment for the spring 2011 CPSC 372 course followed the same basic format; however students were required to implement different components of equivalent difficulty. The average score for these students was 85.6%. For both the fall 2010 and spring 2011 semesters, the average results for the assignment exceeded the overall averages for the courses of 85% and 78%, respectively. In addition to developing new components, the teams were required to create and deliver a UML diagram demonstrating their understanding of the relationships between the components they implemented, as seen in Figure 11.



**Figure 11 Team Assignment Component UML Diagram**

## Reasoning and Proving Assignment

### Assignment Overview

This was an individual assignment made up of six questions designed to exercise the student’s reasoning skills and make use of the Web IDE’s verification condition (VC) generation and proving capabilities. It was used in CS 315: Software Engineering at the University of Alabama, in spring 2011. The questions are listed in Figure 12 below.

### Assignment Results

For this assignment, students were required to use the Web IDE in support of the reasoning and proving questions and turn in the completed assignment to the instructor. The majority of the questions involved using the Web IDE to generate VCs for a specified component and then attempting to manually prove the goal, using only the provided given statements. One of the questions involved making changes to the

component and noting the changes in the generated VCs. All of the eighteen students who turned in the assignment were able to generate the VCs and attempt the manual proof; most students were able to successfully prove the assertions. This assignment was also given in software engineering courses at Western Carolina, DePauw, and Cleveland State in spring 2011, as well as in CPSC 372 at Clemson in summer 2011. A similar assignment was given to the students in CPSC 372 after completing the Team Software Development Assignment. These students used the Web IDE to generate and attempt to manually prove VCs for the realizations they developed. They were required to correct the code for any unprovable VCs and turn in an explanation of why they were incorrect. For the reasoning assignment, the fall 2010 and spring 2011 CPSC 372 students averaged 89% and 92%, respectively, scores that were once again higher than the overall course averages.

1. Develop a reasoning table for Stack\_Template Clear\_2 operation with specification in Clear\_2\_Capability and implementation in Clear\_2\_Realiz. You should assume and confirm assertions for each state.
2. Generate VCs for the above example and compare the generated VCs with the assertions in your table. Write a proof of each VC, i.e., prove the goal in VC using the givens.
3. Generate VCs for Queue\_Template Remove\_Last operation with specification in Remove\_Last\_Capability and implementation in Remove\_Last\_Realiz. Write a proof for each generated VC.
4. Consider the previous question. Make each of the following changes to the code, generate VCs, and state which VC(s) become unprovable.
  - a. Comment out the first Dequeue operation.
  - b. Change the maintaining clause (loop invariant) to  $\#Q = \langle E \rangle \circ Q$ ;
  - c. Change the decreasing clause to  $|T|$ ;
5. Generate VCs and attempt proving Integer\_Template Find\_Max operation with specification in Int\_Max\_Capability and implementation in Int\_Max\_Realiz. In this implementation  $I := J$  denotes a swap statement that exchanges the value of I and J. Explain how you'd fix this error.
6. Generate VCs for the Integer\_Template Add\_to operation with specification in Adding\_Capability and implementation in Recursive\_Add\_to\_Realiz. Prove all VCs.

**Figure 12 Reasoning-based Assignment Questions**

## Usability Survey

To gauge the effectiveness of the Web IDE as a teaching tool in computer science software engineering courses a usability survey was created. The survey was administered at the end of the spring 2011 semester using SurveyMonkey, a web-based service for data collection, and was made up of sixteen questions. The first four questions were designed to collect background information on the respondents, such as school affiliation, course ID, and programming language familiarity. The next set of questions were five-level Likert items meant to assess the students' level of agreement with ten statements pertaining to the design, use, and functionality of the Web IDE. The final two questions asked about what specific features the students made the most use of and for feedback about how to make the tool more user-friendly. Twenty-two responses were collected.

The respondents came from two different groups of students—seventeen undergraduate students from the previously discussed CPSC 372 at Clemson, and five graduates students who used the tool during ECE 693 at Cleveland State University. This section will present a subset of the collected data. A detailed, statement by statement breakdown of the results is shown in Table 1 in the Appendix, and the full response set can be seen in Table 2. The combined score for all of the questions over all of the respondents was 3.81 out of 5. The average scores for the graduate and undergraduate groups were 4.61 and 3.63, respectively. The standard deviation for both groups for each question is about 1, suggesting that the mean is a reasonable representation of student opinions.

The statement that received the highest level of agreement from the respondents was: “The web interface has helped reinforce the principles discussed in class,” with a score of 4.18 out of 5. The high level of agreement with this statement comes as no surprise; access to a tool that provides an easy way to experiment with and put to use the topics covered in lectures will improve students understanding of the materials. The component editor in the Web IDE allowed students to easily make changes to components and see the effects in real-time without needing any additional software installed.

Another statement, “The web interface has helped me understand the relationship between specifications and implementations,” received a score of 4.09. The high level of agreement for this statement seems to validate some of the key goals for the Web IDE. Students were able to make use of the specifications provided in concepts and enhancements to understand how groups of developers could independently implement components that could then be seamlessly combined and work together properly. Providing an environment specifically designed to support component development based on the relationships between them helped make this possible.

The responses to the statement “The web interface has helped me understand the relationship between mathematical reasoning and software development,” receiving a score of 3.86, also provides support for a successful field test of the Web IDE. Having a tool available that can be used in conjunction with reasoning techniques to verify the



correctness of a block of code helps students to see the connection between software engineering and discrete math principles.

While the sample size for this usability survey was too small to provide any statistically conclusive evidence, the generally positive nature of the responses indicates that the Web IDE was seen as beneficial by students who used the tool in their courses.

## CHAPTER SIX

### RELATED WORK

This chapter discusses and presents two different types of related work. First, similar web-based software development tools are discussed. Then tools that have been designed for reasoning and verification of software correctness are presented.

#### Web-based Software Development

Along with the continuing development of traditional software development tools, there are other ongoing, similar efforts to use the web as a software development platform. These efforts can be broadly divided into two main groups—service-oriented and research-oriented IDE's.

#### Service-oriented IDE Development

There are many individuals and groups actively developing online software development platforms touted as web-based IDE's. Like the Web IDE, these tools have been designed to only require a web browser, using JavaScript to create and manipulate the UI. While all of the IDE's of this group are free to use, they can be divided into two types. Some of them require users to register, while others are open to use by everyone.

Cloud9 IDE, ShiftEdit, and Compilr are three IDE's that require users to register and sign-in to use. Of these, Cloud9 IDE is the most recent offering. Users are able to create open source projects that are viewable and usable by all other users. This IDE supports development using JavaScript, Java, C++, HTML, PHP, and Python using a file

editor with syntax highlighting. Real-time code analysis and debugging tools are available for JavaScript-based web application development. Files and projects are either hosted by the service or sent to Github or BitBucket repos and can be managed within the UI. Cloud9 supports team development through simultaneous file editing and a built-in instant messaging system [12].

The second registration-required service is ShiftEdit. ShiftEdit includes support for HTML, PHP, CSS, JavaScript, Ruby, Python, Perl, and Java. However, syntax highlighting is only available for HTML, JavaScript, CSS, and PHP. It includes automatic bracket and tag closing, similar to standalone IDEs, but real-time syntax error checking is only available for PHP and JavaScript. ShiftEdit does not host any user developed files on its servers, but provides support for uploading to FTP/SFTP servers and to Dropbox. It also allows optional collaboration via sharing and revision history. It is free to register for ShiftEdit, but it is ad-supported so usable screen space is compromised. Also, users are limited to three projects [13].

Compilr requires registration for the service as well. Compilr is geared more toward compiled languages, such as C#, Visual Basic, Java, and C++, though there is some support for interpreted languages PHP and Ruby. The free account offered by Compilr is not unlimited; users are only allowed to create three projects, all of which are publicly available through the Compilr website. These public projects can be viewed by anyone using a read-only, guest version of the IDE. All of the files in a project can be downloaded as a single zip archive through the user interface. Registered users are able to download executable binaries of their compiled projects [14].

While the Web IDE is similar to these, there are some important differences. The most obvious is that no user registration is necessary for the Web IDE. There are, however, others that are also registration-free—Coderun Studio and Idone.com. Coderun is a free service, currently in open beta, which supports C#, JavaScript, and PHP development using a code editor and built-in file browser. The editor includes live syntax highlighting, code completion, and debugging tools for some of the supported languages, but not all. To save projects and files to the server, users must register and log in. Saved projects are only accessible by the owner by default, but can be shared with other users. It also supports downloading projects and source files for offline use [15].

Idone is a service that has a very large catalog of over 40 languages that can be developed through their web-based IDE and executed on the server side through a submission-based user interface. Registered users are able to retrieve and edit past submissions. They are allocated more server resources for program execution and are not submitted to the ads visible in the non-logged in user interface. Additionally, registered users have access to the Idone API, allowing them access to Idone's submission-based backend within their own applications [16].

Like the Web IDE, these services do not require users to register. In contrast, though, registration is required to save any user-created content. To allow registration-free use of the Web IDE and still accommodate user-content, all user information is stored within the user's web browser, with a downloadable version available for backup and collaboration. Because it is component-based, the Web IDE presents all user-created and library modules in a way that focuses on the relationship between the modules; not

simply by file or project location. Unlike most of the similar web-based IDE's, the Web IDE allows users to compile and download an executable file for a RESOLVE program.

### Research-oriented IDE Development

There have also been university-level research projects investigating the use of web-based IDE's. Last fall, three graduate students in the Master's program at San Jose State University, Department of Computer Science completed theses outlining their research into web-based IDEs. Two of the students worked together to create a web-based IDE for PHP web application development. The IDE was written in PHP using the CakePHP framework and was designed as an environment where users with limited web programming experience could easily create web applications. Using the framework allowed the web application to take advantage of CakePHP's Model-View-Controller (MVC) architecture [17]. The IDE itself required user registration and login and used a database for server-side data storage. The CakePHP framework also facilitated the creation of databases for user developed server applications without the need for any knowledge of databases or SQL queries [18]. The third student developed a similar web-based IDE designed to help users create dynamic web pages with no knowledge of server-side languages. This IDE used Java Servlets, HTML, and JavaScript to create the user interface as well as Struts, an open-source framework based on MVC architecture [19].

Though similar to the Web IDE, these tools were designed with a different focus in mind—to create web applications with little or no programming experience in web

development. In contrast to the Web IDE, these tools all require credentials to login and store all user files on the server. These are tools that have been created solely for research into web-based IDE's and have not been deployed or tested on a larger scale.

### Software Reasoning Environments

Researchers at the Universite Catholique de Louvain have devised a tool for use in the computer science classroom. Development of their tool was based on a simple premise—that existing tools were too complicated to be used in a teaching environment. Their tool was designed to be used with a simple programming containing pre- and post-conditions. The tool was used by student in labs to help write code given specifications and vice-versa. In contrast the Web IDE, this tool was a standalone program that needed to be installed on the computers the students were using [20].

Another institution that has deployed a software tool to reason about software correctness is Pace University. Researchers there have created a tool called JAIDE (Java Integrated Development Environment). They incorporated formal notation, UML diagrams, and Java code to create a tool to type check, prove theorems, and analyze the source code. JAIDE was used for a multi-semester limited field-test in software engineering courses. Though similar to the Web IDE in functionality, this tool was not easily and openly available. It was provided to a limited number of students through an issued laptop pre-configured with the software [21].

Unlike the previous group of researchers, some have built systems to combine or enhance existing reasoning tools. One such system is ProverEditor, created by

researchers in Europe. ProverEditor is a plugin used to integrate the theorem proving capabilities of Coq with the file management and editing capabilities of Eclipse. It adds Coq-specific highlighting to Eclipse's file editor and works with other plugins to interact with the Coq theorem prover [22].

Other European researchers use Coq as a theorem proving backend for a web-based proof assistant. ProofWeb uses AJAX to communicate with a server-based version of Coq modified to produce output encoded as XML. The interface allows users to manually input or load Coq files and view the output in real time, as well as present users with a visual representation of proof trees. ProofWeb has been tested in several graduate and undergraduate level logic courses at European Institutions [23].

Another theorem prover that is often used as a backend for reasoning environments is Z3 [24]. One such environment is VeriFast, a verification system for C and Java. VeriFast is a tool that allows users to add specifications embedded within comments to Java and C source code. VeriFast will recognize and use these specifications and the included Z3 prover to debug and diagnose errors in the source code [25].

JMLEclipse is a verification environment that is built upon Eclipse, a popular IDE. Like VeriFast, JMLEclipse embeds specifications within Java comments; but it uses Java Modeling Language (JML) specifications. Using JML to specify and verify Java components allows it to be used for contract-based component development [26].

Jahob is a specification and verification system built for a subset of Java. Its specification language is JML-like and generates verification conditions in formats that

are acceptable to a diverse set of automated provers. Jahob allows its users to add in-line hints suitable for various prover backends [27].

KeY [28] is an environment geared toward generating and verifying object-oriented programs written in Java Card, a subset of Java designed for on embedded devices. Like VeriFast, KeY provides support for specifications embedded with Java comments, but uses JML and UML notations. It generates proof obligations that can be automatically verified using its built-in theorem prover, or exported for use in a third party prover, such as Z3.

The Ohio State University verification system also used RESOLVE for component verification and also provides a web-based user interface with access to a library of components. Verification conditions can be generated, but only for the implementations found in the library. This system also supports automated proving of VCs using Isabelle [29] or SplitDecision, an OSU developed proving tool that uses simplification techniques to prove or disprove VCs [30].

While all of the environments discussed in this subsection have been specifically designed to reason about the correctness of code, almost all of them are standalone programs that users must download and install that feature typical file management systems. The Web IDE differs from these not only in its web integration, but on its emphasis on component relationships, modular reasoning capabilities, and features for studying the interactions between specifications, implementations, and verification.



## CHAPTER SEVEN

### CONCLUSIONS AND FUTURE WORK

#### Conclusions

The primary goal of this research project was to create an installation-free, web-based integrated development environment ideally suited for reasoning-based component development in both research and classroom environments that emphasized the relationship between components. Over the past two years, the Web IDE has undergone changes that have transformed it from a simple brainstorming idea into a robust, full-featured tool ready to be used in the classroom. It has been used in computer science courses at Clemson and elsewhere to help reinforce the principles of design-by-contract, team-based collaborative module development, component relationships, software specification, and verification condition generation and proving. It has also been used by graduate students at multiple institutions for research and experimentation and for demonstrating reasoning at several mathematical reasoning workshops. The results from software development class assignments, Web IDE usability surveys, and classroom and workshop experiences have given confidence that the time and effort put into development of the project was worthwhile and that the goals of the project have been fulfilled and even surpassed.

#### Future Work

While the Web IDE has been successfully field testing and is currently in use in software engineering courses and for research, there are improvements that can be made

to increase the usefulness of the tool. Taking advantage of additional CSS and JavaScript libraries could allow the built-in RESOLVE code editor to take a more prominent role in the functionality of the Web IDE. Currently, the editor is purely text-based; syntax highlighting is not applied to RESOVLE code until the user saves their changes and switches to the code tab. The functionality of the both the code and editor tabs could be combined. Advanced CSS formatting and user-editable HTML elements can simulate the functionality of the traditional textbox, while providing on-the-fly syntax highlighting to code in view. This real-time keyword highlighting would be a valuable asset for users developing new components. Similarly, non-editable Java syntax highlighting could be applied to RVC generated Java code when needed.

Another feature that could be updated is the syntax checker. Right now, the code being edited is sent to the server application and the RVC is invoked to check for syntax errors. Using the most current version of ANTLR, the parser generator used to generate the RVC's parser module; it is now possible to use the RVC's ANTLR grammar source files to generate a JavaScript parser for RESOLVE. This JavaScript parser could be incorporated into the Web IDE and used to analyze the RESOLVE code in view and detect syntax errors. This would completely remove the need to communicate with the application server in this case and improve the performance of the user interface.

An additional way to make the Web IDE more useful in an educational setting is to add more support for online user collaboration. Currently, collaboration is supported through the ability to load and save both the entire workspace and individual component files. It is the user's responsibility to share the files as needed. The Web IDE could be

changed to support user logins and group membership. Collaborating users could be part of the same group and be able to have access to a common workspace stored on the server. Functionality similar to repository commits and updates could be added to the user interface to keep users in sync with workspaces stored on the server.

Another way to make the Web IDE a more effective tool in undergraduate software engineering courses would be to provide a way to render RESOLVE facilities line by line as Java source code. Presenting an equivalent to RESOLVE code in a language that undergraduate computer science students are more familiar with will allow them to connect with and relate to the reasoning principles introduced in class more effectively.

In addition to improving the application itself, outside exposure to the Web IDE will be increased. Partnerships will be formed with educators at a large number of other institutions. Data collecting functionality will also be added to application server. These new partners will provide an arena for extensive field use and evaluation that will be used along with usage data collected by the system to fine-tune the application and optimize the educational experience of the user.

## Appendix

### Usability Survey Results

**Table 1 Usability Study Likert Item Statements and Response Average Analysis Breakdown**

	All Responses	All SD	Undergrad Responses	Undergrad SD	Grad Responses	Grad SD
1. The web interface has helped me understand the relationships between specifications and implementations.	4.10	1.00	3.88	1.02	4.80	0.40
2. The web interface has helped me understand the relationship between mathematical reasoning and software development.	3.86	1.14	3.76	1.21	4.20	0.75
3. The web interface has helped reinforce the principles discussed in class.	4.18	0.94	4.08	1.00	4.60	0.49
4. Real-time feedback through the web interface has aided in module development.						
a. Syntax Checker	4.15	0.91	4.06	0.97	4.50	0.50
b. Translator/Builder	3.89	0.81	3.81	0.81	4.50	0.50
c. Verification Condition (VC) Generator	3.95	1.12	3.69	1.10	5.00	0.00
5. It has been easy to learn to use the web interface.	3.86	1.25	3.63	1.17	4.60	0.49
6. The screencasts available under web interface help have been useful in learning to use the interface.	3.54	0.93	3.36	0.88	4.50	0.50
7. The web interface has made collaborative, team development easy.	3.11	1.20	2.93	1.14	4.50	0.50
8. Overall, I am satisfied with the RESOLVE web interface.	3.62	1.00	3.31	0.91	4.60	0.49
9. I have found the user interface design to be visually pleasing.	3.65	0.96	3.38	0.86	4.75	0.43
10. I have found the tabbed page design to be easy to understand and navigate.	3.85	0.91	3.63	0.86	4.75	0.43

**Table 2 Usability Study Likert Item Statements and In-Depth Response Details**

	Strongly Disagree	Disagree	Neither Agree or Disagree	Agree	Strongly Agree	N/A
1. The web interface has helped me understand the relationships between specifications and implementations.	0.0%	13.6%	4.5%	40.9%	40.9%	0.0%
2. The web interface has helped me understand the relationship between mathematical reasoning and software development.	4.5%	9.1%	18.2%	31.8%	36.4%	0.0%
3. The web interface has helped reinforce the principles discussed in class.	0.0%	9.1%	9.1%	36.4%	45.5%	0.0%
4. Real-time feedback through the web interface has aided in module development.						
a. Syntax Checker	0.0%	9.5%	4.8%	42.9%	38.1%	4.8%
b. Translator/Builder	0.0%	0.0%	33.3%	28.6%	23.8%	14.3%
c. Verification Condition (VC) Generator	4.8%	9.5%	4.8%	42.9%	33.3%	4.8%
5. It has been easy to learn to use the web interface.	4.8%	9.5%	14.3%	38.1%	33.3%	0.0%
6. The screencasts available under web interface help have been useful in learning to use the interface.	0.0%	9.5%	19.0%	23.8%	9.5%	0.0%
7. The web interface has made collaborative, team development easy.	9.5%	14.3%	33.3%	14.3%	14.3%	0.0%
8. Overall, I am satisfied with the RESOLVE web interface.	0.0%	19.0%	19.0%	42.9%	19.0%	0.0%
9. I have found the user interface design to be visually pleasing.	0.0%	15.0%	25.0%	40.0%	20.0%	0.0%
10. I have found the tabbed page design to be easy to understand and navigate.	0.0%	10.0%	20.0%	45.0%	25.0%	0.0%

## WORKS CITED

- [1] Bass L., et al. “Volume I: Market Assessment of Component-Based Software Engineering”. Carnegie Mellon University. 2001.
- [2] Edwards S. H., et al. “Software Component Relationships”. In *Proc 8<sup>th</sup> Annual Workshop on Software Reuse*. Columbus, OH. 1997.
- [3] Leonard D. P., et al. “Injecting Rapid Feedback and Collaborative Reasoning in Teaching Specifications”. In *Proc. 40<sup>th</sup> ACM Symposium on Computer Science Education*. ACM. 2009. 524-528.
- [4] Henderson P. “Mathematic Reasoning in Software Engineering Education”. *CACM*. 2003. 46(9). 45-50.
- [5] Bucci P., et al. “Do We Really Teach Abstraction”. *The 32<sup>nd</sup> SIGCSE Technical Symposium on Computer Science Education*. ACM. 2001. 26-30.
- [6] Henderson P., et al. “Striving for Mathematical Thinking”. *ACM SIGCSE Bulletin (ITiCSE 2001 Working Group Reports)*. ACM. 2001. 33(4). 114-124.
- [7] Marion B. and Baldwin D. “On the Implementation of a Discrete Mathematics Course”. SIGCSE Committee Report. 2007.
- [8] Krone J., et al. “A Reasoning Concept Inventory for Computer Science”. Clemson University. 2010.
- [9] Sitaraman M., et al. “Engaging Students in Specification and Reasoning: “Hands-On” Experimentation and Evaluation”. In *Proc. 14<sup>th</sup> Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*. ITiCSE. 2009.
- [10] Sitaraman M., et al. “Building a Pushbutton RESOLVE Verifier: Progress and Challenges”. In *Formal Aspects of Computing*, Springer. 2011. 607-626.
- [11] WC3. 2011. Web Storage. [online] Available at: <http://dev.w3.org/html5/webstorage>
- [12] Cloud9 IDE, Inc. 2011. Cloud9 – Your code anywhere, anytime. [online] Available at: <http://cloud9ide.com>
- [13] ShiftEdit. ShiftEdit – Online IDE. [online] Available at: <http://shiftedit.net>

- [14] Compilr. 2011. Online C#, PHP, C, C++, Ruby, VB, Java IDE & Compiler. [online] Available at: <http://compilr.com>
- [15] Coderun Studio. 2009. Web Development and Deployment Tools: Coderun. [online] Available at: <http://www.coderun.com>
- [16] Sphere Research Labs. 2011. Idone.com | Online IDE & Debugging Tool. [online] Available at: <http://ideone.com>
- [17] Vagesna S., “View Component of Web-based IDE to develop Web Applications in CakePHP” (2010). *Master’s Projects*. Paper 5. [online] Available at: [http://scholarworks.sjsu.edu/etd\\_projects/5](http://scholarworks.sjsu.edu/etd_projects/5)
- [18] Widjaja S., “Web-Based IDE to Create Model and Controller Components for MVC-based Web Applications on CakePHP” (2010). *Master’s Projects*. Paper 1. [online] Available at: [http://scholarworks.sjsu.edu/etd\\_projects/1](http://scholarworks.sjsu.edu/etd_projects/1)
- [19] Palvai T., “Web-Based IDE for Interfacing View Controller” (2010). *Master’s Projects*. Paper 11. [online] Available at: [http://scholarworks.sjsu.edu/etd\\_projects/11](http://scholarworks.sjsu.edu/etd_projects/11)
- [20] Dony I. and Le Charlier B. “A Tool For Helping Teach A Programming Method”. In *Proc. 11<sup>th</sup> Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*. ITiCSE. 2006.
- [21] Skevoulis S. and Makarov V. “Integrating Formal Methods Tools Into Undergraduate Computer Science Curriculum”. In *Frontiers in Education Conference, 36<sup>th</sup> Annual*. 2006. 1-6.
- [22] Charles J. and Kiniry J. R., “A Lightweight Theorem Prover Interface for Eclipse”. INRIA Sophia Antipolis. 2008.
- [23] Hendricks M., et al. “Teaching Logic Using a State-of-the-Art Proof Assistant”. *Acta Didacta Napocensia*. 3(2). 2010. 35-48.
- [24] De Moura L. and Bjorner N. “Z3: An Efficient SMT Solver”. In *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 4963. 2008. 337-340.
- [25] Jacobs B., et al. “A Quick Tour of the Verifast Program Verifier”. In *Proc. APLAS 2010, tool paper track*. LNCS 6461. 2010. 304-311.
- [26] Chalin P., et al. “Towards an Industrial Grade IVE for Java and Next Generation Research Platform for JML”. In *Formal Aspects of Computing*. Springer. 2011.

- [27] Kunac V. and Rinard M. “An overview of the Jahob Analysis System – Project Goals and Current Status”. In *NSF Next Generation Software Workshop*. 2006.
- [28] Ahrendt W., et al. “The Key tool: Integrating object oriented design and formal verification”. LNCS 4334. Springer-Verlag. 2007.
- [29] Nipkow T., et al. “Isabelle/HOL—A proff Assistant for Higher\_Order Logic”. LNCS 2283. Springer. 2002.
- [30] Kirschenbaum J., et al. “Automated Full Functional Verification of Clients of User-Defined Abstract Data Types”. The Ohio State University. 2010.