Creating an Environment for Reusable Software Research: A Case Study in Reusability

Matt Curtin
The Ohio State University
Department of Computer and Information Science
<cmcurtin@interhack.net>

August 4, 1999
OSU-CISRC-8/99-TR21

Abstract

When an architectural change forced an evaluation of the environment used to develop RESOLVE/C++, it was determined to rewrite the environment from scratch. In starting afresh it was possible to apply reusable software principles, which led to a significant reduction in code, an increase in maintainability, and an elimination of bugs.

Keywords: Reusability, RESOLVE, C++, Lisp, XEmacs, GNU Emacs.

1 Introduction

RESOLVE is a software engineering framework, language, and discipline that focuses on the design, specification, construction and use of components.[6] The discipline emphasizes the importance of parameterization, composition, comprehensibility, and performance. We applied these principles to the RESOLVE/C++ [2] [7] [3] [1] [4] development environment and reaped a large reward.

2 The Environment

The work of the Reusable Software Research Group (RSRG) at The Ohio State University's Department of Computer and Information Science (OSU-CIS) is done in a Unix-based environment. Tools used include *make*, various language compilers, and GNU Emacs. For our purposes, the environment of the students taking CS1 and CS2 and using RESOLVE/C++ is essentially the same.

In 1998, OSU-CIS began a large migration from one Unix variant to another. In so doing, we left behind many years of baggage, including some ancient applications, window managers, and the like. At the same time, it was decided

that a migration from GNU Emacs to XEmacs would be undertaken. Where GNU Emacs was available in the old, XEmacs is available in the new.

RESOLVE/C++ and RESOLVE/Ada development and maintenance was supported in the old environment with a locally-customized version of the Ada, C++, and font-lock modes that are a part of GNU Emacs. A simple test demonstrated that the old Emacs Lisp for RESOLVE/C++ would not work under XEmacs, forcing us to consider this code and the functionality it provided.

3 Out With the Old

Because RESOLVE/Ada is not being pursued actively, we were able to limit our scope to deal with RESOLVE/C++. The old code to support this environment was composed of 2,468 Emacs Lisp s-expressions spread across five files. These files include "forked" versions of *c++-mode* and *font-lock-mode* from between 1996 and 1998. Their elements were incorporated into various files in the RESOLVE/C++ package and were maintained locally. This presented a number of problems.

Stagnation. Because the code was a forked version of another package, it was not practical to incorporate the bug fixes, performance improvements, functionality enhancements, etc., that occurred over the years. The code that was written in 1996 remained almost exactly as it was.

Maintainability. The maintainability of the code suffered a great deal. It was very difficult, even for a proficient Lisp programmer, to read the source code and readily understand it. Numerous interdependencies made it necessary to have several files open at once to trace the functionality of the mode from beginning to end. The only updates made to the environment since 1996 were in the form of language keywords being added or removed.

Fragility. Changes made in the code had a strange tendency to affect the rest of the system in unpredictable ways. Sometimes, parts of the code would simply stop working, for unknown reasons. In practice, the fix would typically be to comment-out the nonworking blocks of code.

Exclusivity. Because the RESOLVE/C++ environment was a modified version of the "normal" C++ environment, it was impossible to use both RESOLVE/C++ and "vanilla" C++. Having the RESOLVE/C++ code in one's *load-path* means being unable to use the "normal" C++ environment.

4 In With the New

Requirements for the RESOLVE/C++ environment were clear upon examination of the state of the system.

- Must be easily comprehensible, especially as Lisp expertise is scarce.[1] Nonexperts must be able to understand the code.
- Must perform efficiently. With the heavy use of thin clients and support of remote users, many instances of the application could be running on a single processor. Users should not believe that their environment is "pokey".
- Must be customizable. Preferences for such things as default colors must not be hardwired into the mode itself.
- Must not require local maintenance. Because of funding constraints, it is not possible to retain a Lisp guru to maintain the system. The nature of open-source software is dynamic, where bugs are fixed and new features are added.[5] As XEmacs and its C++-supporting Lisp packages are upgraded, those changes need to be incorporated into the RESOLVE/C++ environment. The only acceptable local changes are where the RESOLVE/C++ language itself slowly changes and the environment needs its keywords list to be updated. Those keywords must be in a single, easy-to-maintain list.

Given these requirements, a number of decisions were reached.

- Rather than hardwiring color and font preferences into the environment, we parameterized these preferences, putting them in the *.xemacs-options* file that researchers and students new to RSRG or its classes can inherit. This enables the user to make his own customizations without having XEmacs go through the process of setting these preferences twice (once, when starting the mode, and again when evaluating the preferences the user set).
- Forked versions of existing modes are unacceptable.
- The Major Mode to support RESOLVE/C++ would be a "derived mode"[2] of *cc-mode*, a package which is used to support C, C++, Objective C, Java, CORBA IDL, and Pike code.

Implementation of a new major mode to support RESOLVE/C++ followed. The result is a single source file of 125 s-expressions. Our requirements have been met, and we have seen the new XEmacs portion of the RESOLVE/C++ environment live through a major upgrade (XEmacs 20.4 to 21.1p2). A new platform (XEmacs 21.1p2 under Win32) was tested, and the mode worked as expected.

It is now possible to choose whether to use RESOLVE/C++ (*rcpp-mode*) or standard C++ (*c++-mode*).

[1] In the author's eyes, scarcity of Lisp expertise is a crime, but it is more productive to recognize this sad fact and to deal with it than to ignore it.

[2] GNU Emacs and derivatives support a form of inheritance for major editing modes. In essence, a derived mode is one that inherits from another, in the same way that a class in an object oriented programming language would inherit from another.

5 Conclusions

Success of *rcpp-mode* demonstrates that reusable software does work outside of the laboratory. By constructing a component that took advantage of its environment through clearly defined interfaces and mechanisms provided, we were able to provide the same functionality with about 5% of the code needed to support the same functionality using the "fork-and-customize" method. Other gains have been in speed, maintainability, and portability.

6 Acknowledgments

Thanks to Bruce Weide and Tim Long for support of my efforts, testing, and helpful comments on an earlier draft. Additional thanks to Shaun Rowland and Jim Hill for building and testing Win32 XEmacs[3] and *rcpp-mode*.

A Source Code: rcpp-mode

```
;;                                      -*- Emacs-Lisp -*-
;; Author          : C Matthew Curtin <cmcurtin@interhack.net>
;; Created On      : <1999/04/26 17:22:40 cmcurtin>
;; Last Modified By: Author: cmcurtin
;; Last Modified On: Date: 1999/07/13 18:00:08
;; Version         : Revision: 0.4
;; Maintainer      : cmcurtin+rcpp-mode@interhack.net
;; Status          : Functional

;; Copyright (C) 1999 The Ohio State University
;; Copyright (C) 1999 Matt Curtin <http://www.interhack.net/people/cmcurtin/>

;; rcpp-mode is free software; you can redistribute it and/or modify
;; it under the terms of the GNU General Public License as published
;; by the Free Software Foundation; either version 2, or (at your
;; option) any later version.

;; rcpp-mode is distributed in the hope that it will be useful, but
;; WITHOUT ANY WARRANTY; without even the implied warranty of
;; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
;; General Public License for more details.

;; You should have received a copy of the GNU General Public License
;; along with XEmacs; see the file COPYING.  If not, write to the Free
;; Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
```

[3]Doing so under Linux!

```
;; Commentary:

;; RESOLVE/C++ (aka RCPP) is a discipline for component-based software
;; engineering in C++.  The discipline is enforced in a variety of
;; tools that collectively comprise the RCPP development environment.
;; RESOLVE functionality is given to C++ through the use of templates
;; which extend the language in various ways to make it more suited
;; to the task of building software in components.  This is a major
;; mode that will provide a useful RESOLVE/C++ editing environment,
;; adding RESOLVE/C++ keywords to the known keywords while maintaining all
;; of the other cool stuff that comes from the excellent "CC Mode", of
;; which this is a derived mode.
;;
;; If you're unfamiliar with the concept of a "derived mode", you can
;; get hip to the idea one of two ways:
;;  o Learn how it really works by checking the Emacs help on major
;;    modes, particularly the bit about derived modes.
;;  o Believe me when I say that it's an Emacs-Lisp sort of inheritance.
;;
;; This mode was developed for use at The Ohio State University's
;; Computer and Information Science Department, by both students and
;; researchers.  More information about RESOLVE can be found at the
;; OSU Reusable Software Research Group's web page at
;; http://www.cis.ohio-state.edu/rsrg/.
;;
;; The latest and greatest release of rcpp-mode is available on the
;; web at http://www.interhack.net/projects/rcpp-mode/.

;; Code:

(require 'font-lock)
(require 'cc-mode)
(c-initialize-cc-mode)

(defvar rcpp-mode-syntax-table nil
  "Syntax table used in rcpp-mode buffers.")

(if rcpp-mode-syntax-table
    ()
  (setq rcpp-mode-syntax-table (make-syntax-table))
  (c-populate-syntax-table rcpp-mode-syntax-table))

(defvar rcpp-version
  (substring (substring "$Revision: 1.2 $" 10) -2 1))

(setq c-version
```

```
  (concat c-version " with rcpp-mode " rcpp-version " derived mode"))

(setq auto-mode-alist
      (append
       (list
        '("\\.cc$"                 . rcpp-mode)
        '("\\.C$"                  . rcpp-mode)
        '("\\.cpp$"                . rcpp-mode)
        '("\\.h[r]?[0-9]*[a-z]?$"  . rcpp-mode))
       auto-mode-alist))

(defun rcpp-mode ()
  "Major mode for developing RESOLVE/C++ code."
  nil)

(define-derived-mode rcpp-mode c++-mode "RCPP"
  "Major mode for editing RESOLVE/C++ code.
\\{rcpp-mode-map}"
  ;; set up some default look and feel
  (setq c-basic-offset 4
        c-default-style "user")
  (c-set-style "bsd")
  (make-local-variable 'compile-command)
  (setq compile-command (concat "/usr/class/sce/rcpp"
                                (getenv "RCPPVERSION")
                                "/tools/rcpp-make"))
  (delete-menu-item '("C++")) ; kill the C++ menu from C++ Mode.
  ;; Replace the killed C++ menu with the RESOLVE/C++ menu, defined thusly:
  (add-submenu nil
               '("RESOLVE/C++"
                 ["Comment Out Region"     comment-region (mark)]
                 ["Uncomment Region"
                  (comment-region (region-beginning) (region-end) '(4)) (mark)]
                 ["Refresh Highlighting"   font-lock-fontify-buffer t]
                 ["Fill Comment Paragraph" c-fill-paragraph t]
                 "---"
                 ["Backslashify"           c-backslash-region (mark)]
                 ["Indent Line"            c-indent-command t]
                 ["Indent Region More"     rcpp-indent-region t]
                 ["Indent Region Less"     rcpp-outdent-region t]
                 "---"
;                ["Up Conditional"         c-up-conditional t]
;                ["Backward Conditional"   c-backward-conditional t]
;                ["Forward Conditional"    c-forward-conditional t]
                 ["Backward Statement"     c-beginning-of-statement t]
                 ["Forward Statement"      c-end-of-statement t]
```

```
                "---"
                ["Build Project"          compile t]
                ["First Error"            first-error t]
                ["Next Error"             next-error t]
                ["Previous Error"         previous-error t]
                ["Run Program"            rcpp-run-program t]
                ))

  (set-syntax-table rcpp-mode-syntax-table)
  (setq major-mode 'rcpp-mode
        mode-name "RCPP"
        local-abbrev-table rcpp-mode-abbrev-table)
  (use-local-map rcpp-mode-map))

(defvar resolve-things (concat (regexp-opt '("abstract_template"
                                             "concrete_template"
                                             "abstract_instance"
                                             "concrete_instance"
                                             "toolkit_class"
                                             "implements"
                                             "extends"
                                             "checks"
                                             "instantiates"
                                             "encapsulates"
                                             "specializes"
                                             "employs"
                                             "procedure"
                                             "function"
                                             "is_abstract"
                                             "toolkit_procedure"
                                             "toolkit_function"
                                             "toolkit_object"
                                             "global_procedure"
                                             "global_function"
                                             "local_procedure"
                                             "local_function"
                                             "procedure_body"
                                             "function_body"
                                             "alters"
                                             "consumes"
                                             "preserves"
                                             "produces"
                                             "object"
                                             "catalyst"
                                             "and"
                                             "or"
```

```
"not"
"mod"
"enumeration"
"Boolean_constant"
"Character_constant"
"Integer_constant"
"Real_constant"
"Text_constant"
"self"
"NULL"
"default_value"
"no_parameters"
"standard_abstract_operations"
"standard_concrete_operations"
"field_name"
"rep_field_name"
"number_of_fields"
"number_of_displayable_fields"
"standard_assignment_operator"
"standard_equality_operators"
"custom_equality_operators"
"standard_comparison_operators"
"custom_comparison_operators"
"select"
"Accessor_Position"
"current"
"assert"
"checking_assert"
"are_distinct_objects"
"debug"
"breakpoint"
"cond_debug"
"cond_breakpoint"
"trace"
"serr"
"Can_Convert_To_Boolean"
"Can_Convert_To_Character"
"Can_Convert_To_Integer"
"Can_Convert_To_Real"
"To_char"
"To_int"
"To_double"
"To_Boolean"
"To_Character"
"To_Integer"
"To_Real"
```

```
                                                "To_Text"
                                                "Boolean"
                                                "Character"
                                                "Integer"
                                                "Real"
                                                "Text"
                                                "Character_IStream"
                                                "Character_OStream"
                                                "Character_Error_OStream"
                                                "m_pi"
                                                "m_e"
                                                "Minimum_Character"
                                                "Maximum_Character"
                                                "Minimum_Integer"
                                                "Maximum_Integer"
                                                "Minimum_Real"
                                                "Maximum_Real"
                                                "Sin"
                                                "Cos"
                                                "Tan"
                                                "Arc_Sin"
                                                "Arc_Cos"
                                                "Arc_Tan"
                                                "Sinh"
                                                "Cosh"
                                                "Tanh"
                                                "Abs"
                                                "Sign"
                                                "Exp"
                                                "Power"
                                                "Ln"
                                                "Log"
                                                "Sqr"
                                                "Sqrt"
                                                "Ceiling"
                                                "Floor"
                                                "end_user_command_line"
                                                "command_line_arguments")))
  "RESOLVE/C++ keywords to highlight with font locking.")

(defvar rcpp-font-lock-keywords-1 c++-font-lock-keywords-1
  "Subdued level highlighting for rcpp-mode.")

(defvar rcpp-font-lock-keywords-2
  (append c++-font-lock-keywords-2 (list (cons
                                          (concat
```

```
                                          "\\<\\(" resolve-things "\\)\\>")
                                         font-lock-keyword-face)))
  "Medium level highlighting for rcpp-mode.")

(defvar rcpp-font-lock-keywords-3 c++-font-lock-keywords-3
  "Gaudy level highlighting for rcpp-mode.")

(defvar rcpp-font-lock-keywords
  (append c++-font-lock-keywords
          rcpp-font-lock-keywords-2
          (list (cons
                 (concat "\\<\\(" resolve-things "\\)\\>")
                 font-lock-keyword-face))
          rcpp-font-lock-keywords-3)
  "Keywords for rcpp-mode.")

(define-key rcpp-mode-map [return] 'newline-and-indent)
(define-key rcpp-mode-map [(control c) b] 'compile)
(define-key rcpp-mode-map [(control c) r] 'rcpp-run-program)
(define-key rcpp-mode-map [(control c) <] 'rcpp-outdent-region)
(define-key rcpp-mode-map [(control c) >] 'rcpp-indent-region)

(defun rcpp-run-program (&optional prog-name)
    "Interactively run an executable program in a separate shell
buffer.  This command will be run in the same directory as the file
contained in the current buffer.  If an executable with the same base
name as the current file exists, it will be run; otherwise, the user
will be prompted for the name of the executable to run."
    (interactive)
    (let ((explicit-shell-file-name
           (substring (buffer-file-name) 0
                      (string-match "\\." (buffer-file-name)))))
      (shell)))

(defun rcpp-outdent-region (&optional argp)
  "Outdents region by amount equal to value of universal-arg or c-basic-offset."
  (interactive "P")
  (let ((start (point))
        (end   (mark)))
    (if (> start end)
        (progn
          (setq start end)
          (setq end (point))))
    (indent-rigidly start end (- 0 (if argp (prefix-numeric-value argp)
                                     c-basic-offset)))))
```

```
(defun rcpp-indent-region (&optional argp)
  "Indents region by amount equal to value of universal-arg or c-basic-offset."
  (interactive "P")
  (let ((start (point))
        (end   (mark)))
    (if (> start end)
        (progn
          (setq start end)
          (setq end (point))))
    (indent-rigidly start end (if argp (prefix-numeric-value argp)
                                c-basic-offset))))

;; HISTORY
;; Log: rcpp-mode.el,v
;; Revision 0.4  1999/07/13 18:00:08  cmcurtin
;; Changes per Bruce:
;;  o Keyword changes
;;  o Added "indent more" and "indent less", including function
;;    definitions for them (‘rcpp-indent-region’ and
;;    ‘rcpp-outdent-region’).
;;  o Changed the order of a few options (find first error, find next
;;    error)
;;  o Made compile command now take note of environment’s $RCPPVERSION so
;;    that if $RCPPVERSION is -sun then the compile command will be
;;    /usr/class/sce/rcpp-sun/tools/rcpp-make, if it’s -hp, it’ll be
;;    /usr/class/sce/rcpp-hp/tools/rcpp-make, if it’s undefined, it’ll be
;;    /usr/class/sce/rcpp/tools/rcpp-make, which should be a symlink to
;;    something that makes sense in the general case.
;;  o Keyboard bindings for some RCPP commands (build project, run
;;    program, rcpp-indent-region, rcpp-outdent-region).
;;  o Bruce notes that indentation on formal parameters isn’t what they want.
;;    What we’ve got:
;;      procedure Foo (
;;          alters Integer& i
;;          );
;;    What they want:
;;      procedure Foo (
;;              alters Integer& i
;;          );
;;    There isn’t a really easy way to do this without getting into the
;;    issue of treating formal comments specially, so we’ll put this off
;;    until we do the formal comment handling.
;;
;; Revision 0.3  1999/05/31 13:24:07  cmcurtin
;; Minor cleanup: addition of provide clause, "ends-here" comment, and
;; pointer to the official distribution point.
```

```
;;
;; Revision 0.2  1999/05/26 02:13:26  cmcurtin
;; First complete implementation.
;;
;; Revision 0.1  1999/04/27 14:25:39  cmcurtin
;; *** empty log message ***
;;

(provide 'rcpp-mode)
;;; rcpp-mode.el ends here
```

References

[1] P. Bucci, T.J. Long, and B.W. Weide. Teaching software architecture principles in CS1/CS2. In *Proceedings 3rd International Software Architecture Workshop*, pages 9–12. ACM, November 1998.

[2] T.J. Long and B.W. Weide. Weaving software engineering into the fabric of CS1 and CS2. In *Proceedings 4th International Workshop on Software Engineering Education*, pages 66–69, May 1997.

[3] T.J. Long, B.W. Weide, P. Bucci, D.S. Gibson, J.E. Hollingsworth, M. Sitaraman, and S.H. Edwards. Providing intellectual focus to CS1/CS2. In *Proceedings 29th SIGCSE Technical Symposium on Computer Science Education*, pages 252–256. ACM, February 1998.

[4] T.J. Long, B.W. Weide, P. Bucci, and M. Sitaraman. Client view first: An exodus from implementation-biased teaching. In *Proceedings 30th SIGCSE Technical Symposium on Computer Science Education*, pages 136–140. ACM, March 1999.

[5] Eric S. Raymond. The Cathedral and the Bazaar, July 1999. [online] http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar.html.

[6] M. Sitaraman and B.W. Weide. Component-based software using RESOLVE. In *Software Engineering Notes 19, 4*, pages 21–63. IEEE, October 1994.

[7] M. Sitaraman, B.W. Weide, T.J. Long, and W.D. Heym. Teaching the essential role of mathematical modeling in understanding and reasoning about objects. Technical Report OSU-CISRC-9/97-TR43, Department of Computer and Information Science, The Ohio State University, September 1997.