

CHAPMAN & HALL/CRC INNOVATIONS IN  
SOFTWARE ENGINEERING AND SOFTWARE DEVELOPMENT

# SOFTWARE DEVELOPMENT

AN OPEN SOURCE APPROACH

ALLEN TUCKER  
RALPH MORELLI  
CHAMINDRA DE SILVA



CRC Press

Taylor & Francis Group

Boca Raton London New York

---

CRC Press is an imprint of the  
Taylor & Francis Group, an **informa** business

The photo on the cover symbolizes the central themes of free and open source software development: the management of chaos (Raymond's "bazaar" metaphor), communities, collaboration, openness, teamwork, and agility.

CRC Press  
Taylor & Francis Group  
6000 Broken Sound Parkway NW, Suite 300  
Boca Raton, FL 33487-2742

© 2011 by Taylor and Francis Group, LLC  
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works

Printed in the United States of America on acid-free paper  
10 9 8 7 6 5 4 3 2 1

International Standard Book Number: 978-1-4398-1290-7 (Hardback)

This book contains information obtained from authentic and highly regarded sources. Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access [www.copyright.com](http://www.copyright.com) (<http://www.copyright.com/>) or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

**Trademark Notice:** Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

**Visit the Taylor & Francis Web site at**  
**<http://www.taylorandfrancis.com>**

**and the CRC Press Web site at**  
**<http://www.crcpress.com>**

---

# *Contents*

List of Figures . . . . .	xi
List of Tables . . . . .	xv
Preface . . . . .	xvii
Acknowledgments . . . . .	xxv
Authors . . . . .	xxvii
<b>1 Overview and Motivation</b>	<b>1</b>
1.1 Software . . . . .	1
1.1.1 Types of Software . . . . .	2
1.1.2 The Changing Landscape . . . . .	5
1.1.3 Who Are the Developers? . . . . .	7
1.1.4 Strategic Choices . . . . .	8
1.2 Free and Open Source Software (FOSS) . . . . .	11
1.2.1 Origins and Growth . . . . .	12
1.2.2 Licensing . . . . .	16
1.2.3 Worldwide Impact . . . . .	18
1.2.4 Humanitarian FOSS . . . . .	19
1.3 Two Case Studies . . . . .	20
1.3.1 RMH Homebase . . . . .	20
1.3.2 Sahana . . . . .	20
1.4 Summary . . . . .	22
Exercises . . . . .	23
<b>2 Working with a Project Team</b>	<b>27</b>
2.1 Key FOSS Activities . . . . .	27
2.1.1 Agile Development . . . . .	27
2.1.2 Using Patterns . . . . .	29
2.1.3 Reading and Writing Code . . . . .	31
2.1.4 Documentation . . . . .	34
2.1.5 On-Line Help . . . . .	37
2.2 Client-Oriented vs Community-Oriented Projects . . . . .	37
2.2.1 Project Evolution . . . . .	40
2.2.2 Similarities and Differences . . . . .	42
2.3 Working on a Client-Oriented Project . . . . .	44
2.3.1 Members, Roles, and Tasks . . . . .	44
2.3.2 Team Dynamics . . . . .	47
2.3.3 Scheduling, Milestones, and To-Do Lists . . . . .	48
2.4 Joining a Community-Oriented Project . . . . .	50

2.4.1	Project Selection . . . . .	52
2.4.2	First Contact with the Project . . . . .	53
2.4.3	Norms for Good Citizenship . . . . .	56
2.4.4	Becoming a User First . . . . .	58
2.5	Summary . . . . .	60
	Exercises . . . . .	60
<b>3</b>	<b>Using Project Tools</b>	<b>63</b>
3.1	Collaboration Tools . . . . .	63
3.1.1	Asynchronous Communication . . . . .	64
3.1.2	Synchronous Communication . . . . .	65
3.1.3	Shared Documents . . . . .	66
3.2	Code Management Tools . . . . .	67
3.2.1	The IDE . . . . .	68
3.2.2	The Software Stack . . . . .	70
3.2.3	The Version Control System . . . . .	72
3.2.4	The Bug Tracker . . . . .	77
3.3	Run-Time System Constraints . . . . .	82
3.3.1	Performance . . . . .	82
3.3.2	Web Hosting . . . . .	83
3.3.3	Licensing . . . . .	83
3.3.4	Platform . . . . .	84
3.4	Summary . . . . .	84
	Exercises . . . . .	85
<b>4</b>	<b>Software Architecture</b>	<b>87</b>
4.1	Architectural Patterns . . . . .	87
4.2	Layers, Cohesion, and Coupling . . . . .	89
4.2.1	Using Metrics to Evaluate Cohesion and Coupling . . . . .	93
4.3	Security . . . . .	95
4.3.1	Architectural Vulnerabilities . . . . .	96
4.3.2	User-Level Security . . . . .	97
4.4	Concurrency, Race Conditions, and Deadlocks . . . . .	100
4.5	Summary . . . . .	105
	Exercises . . . . .	105
<b>5</b>	<b>Working with Code</b>	<b>107</b>
5.1	Bad Smells and Metrics . . . . .	108
5.1.1	Identifying Bad Smells . . . . .	108
5.1.2	Software Metrics . . . . .	110
5.2	Refactoring . . . . .	111
5.2.1	Example 1: Removing Useless Functions . . . . .	114
5.2.2	Example 2: Removing a Layering Violation . . . . .	114
5.3	Testing . . . . .	117
5.3.1	Unit Testing Tools . . . . .	119

5.3.2	Test Case Design . . . . .	120
5.3.3	A Strategy for Sequencing Unit Tests . . . . .	128
5.4	Debugging . . . . .	129
5.4.1	Tool Use vs Developer Skill . . . . .	130
5.4.2	Example 1: A User Interface Bug . . . . .	131
5.4.3	Example 2: A Multi-Level Bug . . . . .	133
5.5	Extending the Software for a New Project . . . . .	134
5.5.1	A New Use Case . . . . .	135
5.5.2	Impact on the Code Base . . . . .	136
5.5.3	Team Discussions . . . . .	139
5.6	Summary . . . . .	140
	Exercises . . . . .	140
<b>6</b>	<b>Developing the Domain Classes</b>	<b>143</b>
6.1	Understanding the Current System . . . . .	143
6.1.1	Reading a Design Document . . . . .	144
6.1.2	Reading Code . . . . .	147
6.1.3	Examining the Domain Classes . . . . .	150
6.2	Adding New Features . . . . .	151
6.2.1	Top-Down Analysis/Bottom-Up Development . . . . .	154
6.2.2	Modifying the Domain Classes . . . . .	155
6.2.3	Documentation and Bulletproofing . . . . .	158
6.3	Class Design Principles and Practice . . . . .	162
6.3.1	Using What’s Already There . . . . .	163
6.3.2	Adding a New Domain Class . . . . .	164
6.4	Managing the Ripple Effect . . . . .	166
6.4.1	Unit Testing the New Code . . . . .	166
6.4.2	Refactoring the New Code Base . . . . .	169
6.5	Summary . . . . .	171
	Exercises . . . . .	171
<b>7</b>	<b>Developing the Database Modules</b>	<b>173</b>
7.1	Design Principles and Practice . . . . .	174
7.1.1	Database Creation . . . . .	175
7.1.2	Connecting the Program to the Database . . . . .	176
7.1.3	Tables . . . . .	178
7.1.4	Normalization and Keys . . . . .	180
7.1.5	Backup and Recovery . . . . .	181
7.2	Working with a Database . . . . .	182
7.2.1	Table Creation . . . . .	184
7.2.2	Table Searching . . . . .	185
7.2.3	Table Insertion, Deletion, and Updating . . . . .	187
7.3	Database Security and Integrity . . . . .	188
7.3.1	Database-Level Permissions . . . . .	189
7.3.2	User-Level Permissions . . . . .	189

7.3.3	Controlling Concurrency . . . . .	192
7.4	Adding New Software Features: Database Impact . . . . .	193
7.4.1	Items 1 and 9d: Volunteer Status and Application . . . . .	195
7.4.2	Item 3: Calendar View . . . . .	198
7.5	Summary . . . . .	201
	Exercises . . . . .	202
<b>8</b>	<b>Developing the User Interface</b>	<b>205</b>
8.1	Design Principles and Practice . . . . .	205
8.1.1	The Model-View-Controller Pattern . . . . .	207
8.1.2	Sessions, Query Strings, and Global Variables . . . . .	210
8.1.3	Ensuring Security at the User Interface . . . . .	213
8.2	Working with Code . . . . .	218
8.2.1	Reading Deeply . . . . .	219
8.2.2	Debugging as a Community Activity . . . . .	224
8.3	Adding New Features: User Interface Impact . . . . .	230
8.3.1	Item 1: Volunteer Status . . . . .	230
8.3.2	Item 2: Make Active/Inactive . . . . .	235
8.3.3	Item 3: Calendar View . . . . .	237
8.4	Summary . . . . .	239
	Exercises . . . . .	240
<b>9</b>	<b>User Support</b>	<b>243</b>
9.1	Technical Writing . . . . .	243
9.1.1	Knowing Your Audience . . . . .	244
9.1.2	Principles of Good Writing . . . . .	246
9.2	Types of User Support . . . . .	249
9.2.1	On-Line Help . . . . .	249
9.2.2	Reference Manuals . . . . .	251
9.2.3	Open Discussion Forums . . . . .	253
9.2.4	User Training and Feedback . . . . .	257
9.3	Example: RMH Homebase On-Line Help . . . . .	258
9.3.1	Help and the Code Base . . . . .	258
9.4	Summary . . . . .	263
	Exercises . . . . .	263
<b>10</b>	<b>Project Governance</b>	<b>265</b>
10.1	Origins and Evolution . . . . .	265
10.1.1	Starting a Client-Oriented Project . . . . .	267
10.1.2	Quality Assessment . . . . .	271
10.2	Evolving into a Democratic Meritocracy . . . . .	273
10.2.1	Incubation . . . . .	274
10.2.2	Organization . . . . .	277
10.2.3	Decision Making and Conflict Resolution . . . . .	281
10.2.4	Domain Constraints . . . . .	282

10.3	Releasing Code . . . . .	284
10.3.1	Licensing . . . . .	284
10.3.2	Finding a Project Host . . . . .	285
10.3.3	Release Strategies . . . . .	287
10.4	Summary . . . . .	289
	Exercises . . . . .	290
<b>11</b>	<b>New Project Conception</b>	<b>291</b>
11.1	Requirements Gathering . . . . .	292
11.1.1	Domain Analysis . . . . .	292
11.1.2	User Stories . . . . .	295
11.1.3	Use Cases . . . . .	297
11.2	Initial Design . . . . .	303
11.2.1	Domain Classes . . . . .	303
11.2.2	User Interface . . . . .	304
11.2.3	Performance and Platform . . . . .	305
11.2.4	System Architecture . . . . .	307
11.2.5	Design Alternatives . . . . .	308
11.2.6	Design Document . . . . .	308
11.3	Summary . . . . .	309
	Exercises . . . . .	309
	<b>Appendices</b>	<b>311</b>
<b>A</b>	<b>Details of the Case Study</b>	<b>311</b>
A.1	Requirements . . . . .	311
A.1.1	Domain Analysis . . . . .	312
A.1.2	Use Cases . . . . .	317
A.1.3	System Requirements . . . . .	327
A.2	Design . . . . .	328
A.2.1	Goals . . . . .	329
A.2.2	Software Architecture . . . . .	329
A.2.3	Domain Classes . . . . .	330
A.2.4	Database Design . . . . .	330
A.2.5	GUI Design . . . . .	333
A.2.6	Implementation Schedule . . . . .	336
A.2.7	User-System Interaction . . . . .	336
<b>B</b>	<b>New Features for an Existing Code Base</b>	<b>341</b>
B.1	Starting with a Request from the Client . . . . .	341
B.2	Impact on the Design and the Code Base . . . . .	343
B.3	Defining a Project that Implements these Features . . . . .	350
	<b>References</b>	<b>351</b>
	<b>Index</b>	<b>355</b>



---

## List of Figures

1.1	Market shares during the “Browser Wars” . . . . .	3
1.2	Relationships among common FOSS licenses . . . . .	17
2.1	The traditional software development process. . . . .	28
2.2	The agile software development process. . . . .	29
2.3	Using the Eclipse environment with Subversion. . . . .	31
2.4	Example code from <i>RMH Homepage</i> . . . . .	32
2.5	Output of the example code in Figure 2.4. . . . .	33
2.6	Documentation with indented blocks and control structures. . . . .	34
2.7	Inserting comments in the code. . . . .	35
2.8	PHP documentation generated for the Shift class. . . . .	36
2.9	Form for filling a vacancy on a shift. . . . .	38
2.10	Help screen for filling a vacancy. . . . .	39
3.1	Developing within the Eclipse environment. . . . .	69
3.2	Software stack for Web-based applications. . . . .	70
3.3	A LAMP stack for Web-based applications. . . . .	71
3.4	The code synchronization problem. . . . .	75
3.5	Resolving the problem: Copy-modify-merge. . . . .	76
3.6	Life cycle of a bug, as defined in Bugzilla . . . . .	78
3.7	Showing the open source license notice in the user interface. . . . .	83
3.8	Displaying the open source license notice in the source code. . . . .	84
4.1	The client-server pattern. . . . .	89
4.2	Layers in a multi-tier architecture. . . . .	90
4.3	Multi-tier architecture of <i>RMH Homepage</i> . . . . .	91
4.4	User interface code that violates the layering principle. . . . .	92
4.5	A secure login form. . . . .	98
4.6	Excerpt from the underlying login code. . . . .	99
4.7	<i>RMH Homepage</i> Applicant, Volunteer, and Manager menus. . . . .	100
4.8	Code for generating menus. . . . .	101
4.9	Deadlock at a traffic intersection. . . . .	103
5.1	Refactoring, testing, and debugging are interrelated. . . . .	108
5.2	Example bad smell—duplicate code. . . . .	109
5.3	Example bad smell removal. . . . .	112
5.4	A new search function for the dbPersons module. . . . .	116

5.5	A second new search function for the dbPersons module. . . .	116
5.6	A typical PHP unit for testing. . . . .	118
5.7	Elements of a unit test for a PHP class or module. . . . .	118
5.8	A series of unit tests. . . . .	119
5.9	Results of running a series of unit tests. . . . .	119
5.10	A partial unit test for the Shift class. . . . .	122
5.11	A unit test for the dbShifts module. . . . .	124
5.12	The Shift form in the user interface. . . . .	124
5.13	A unit test for the editShift module. . . . .	126
5.14	A calendar form for the ChangeACalendar use case. . . . .	127
5.15	Locating a bug in the calendar.php module. . . . .	131
5.16	Locating a bug in the dbDates module. . . . .	134
5.17	Use case <b>Housecleaning</b> . . . . .	137
5.18	A new calendar housecleaning form. . . . .	138
6.1	New instance variables for the Person class. . . . .	156
6.2	New functions for the Person class. . . . .	156
6.3	Revising the instance variables for the Shift class. . . . .	158
6.4	Revising the constructor for the Shift class. . . . .	159
6.5	Adding a new function to the Shift class. . . . .	160
6.6	Original function to return a Shift name. . . . .	161
6.7	Instance variables for the Week class. . . . .	164
6.8	Instance variables for a new Month class. . . . .	165
6.9	Constructor for the new Month class. . . . .	166
6.10	Augmenting the unit test for the Shift class. . . . .	168
6.11	New unit test for the Month class. . . . .	169
6.12	Generating the dates for a new Month. . . . .	169
7.1	MySQL commands to create a new user and database. . . . .	175
7.2	Creating a new database using LAMP. . . . .	176
7.3	Connecting to the <i>RMH Homebase</i> database. . . . .	177
7.4	A view of the dbDates table. . . . .	178
7.5	Shifts in the database for September 2, 2009. . . . .	180
7.6	Backing up a database using LAMP. . . . .	182
7.7	Template for MySQL table creation. . . . .	184
7.8	Creating the dbDates table in the rmhDB database. . . . .	185
7.9	Deleting a date from the dbDates table. . . . .	191
7.10	New attributes defined in dbPersons table creation. . . . .	196
7.11	Inserting a new person into the dbPersons table. . . . .	197
7.12	Testing modifications in the dbPersons.php module. . . . .	198
7.13	New <code>setup_dbDates</code> function in dbDates.php refactoring. . .	200
7.14	New <code>get_shift_name_from_id</code> function for dbShifts.php. . .	201
7.15	Upgraded unit test for the dbShifts module . . . . .	202
8.1	The Model-View-Controller pattern. . . . .	207

8.2	The Shift view in <i>RMH Homebase</i> . . . . .	208
8.3	Overview of the editShift.php module. . . . .	211
8.4	Controlling navigation via \$_POST variables. . . . .	213
8.5	Ensuring security via \$_POST and \$_SESSION variables. . . . .	215
8.6	Password checking during <i>RMH Homebase</i> login. . . . .	215
8.7	Part of the code base for handling a SubCallList. . . . .	219
8.8	Using the SubCallList form. . . . .	220
8.9	Using the Shift form to generate an SCL. . . . .	220
8.10	Code snippet for removing a person from a Shift. . . . .	222
8.11	Reproducing the bug. . . . .	225
8.12	Locating the defect. . . . .	226
8.13	Designing the fix. . . . .	227
8.14	Testing the fix: viewing a person and editing a person. . . . .	227
8.15	Showing a person's status. . . . .	231
8.16	Changing personForm.php to show a person's status. . . . .	231
8.17	Updating a person's database entry. . . . .	232
8.18	Showing a person's status in the view. . . . .	232
8.19	Changing viewPerson.php to show a person's status. . . . .	233
8.20	Searching for "inactive" volunteers and results. . . . .	233
8.21	searchPeople.php code that finds "inactive" volunteers. . . . .	234
8.22	Changing editMasterSchedule.php to list only "active" volunteers. . . . .	234
8.23	Listing only "active" volunteers when filling a vacancy. . . . .	234
8.24	Listing volunteers who have not worked recently. . . . .	235
8.25	Code for listing volunteers who have not worked recently. . . . .	236
8.26	New function to search for active or inactive persons. . . . .	237
8.27	A view of the new Move Shift feature. . . . .	238
8.28	Selecting new start and end times for a Shift. . . . .	239
9.1	First page of the <i>Firefox manual</i> , including Help and FAQs. . . . .	252
9.2	Accessing the <i>Sahana</i> FAQ list. . . . .	253
9.3	Point of access to the <i>Sahana</i> forums. . . . .	254
9.4	Accessing the <i>Sahana</i> bug tracking list. . . . .	255
9.5	Point of access to the Drupal forums. . . . .	256
9.6	Accessing the Firefox user forum. . . . .	256
9.7	The main help menu in <i>RMH Homebase</i> . . . . .	259
9.8	Stepping through the task of filling a vacancy. . . . .	260
9.9	Showing a screen shot by selecting its thumbnail. . . . .	261
9.10	Integrating help pages within the code base. . . . .	262
9.11	HTML code for Step 1 of the help page for filling a vacancy. . . . .	262
10.1	The staging server: Client-Developer interaction. . . . .	268
10.2	Organizational levels in the <i>Sahana</i> project. . . . .	281
11.1	RMH guest referral form. . . . .	294

11.2	RMH guest registration card. . . . .	295
11.3	RMH guest room schedule. . . . .	296
11.4	<i>RMH Homeroom</i> use cases. . . . .	301
11.5	Some of the initial domain classes for <i>RMH Homeroom</i> . . . . .	304
11.6	Room view screen draft for <i>RMH Homeroom</i> . . . . .	305
A.1	RMH Volunteer application form: 12/2007. . . . .	314
A.2	RMH master schedule—Group One. . . . .	315
A.3	Typical handwritten monthly Volunteer calendar. . . . .	316
A.4	Typical scheduled day on the handwritten calendar. . . . .	316
A.5	RMH use case diagram. . . . .	317
A.6	Use case <b>UpdateApplicantList</b> . . . . .	318
A.7	Use case <b>UpdateSubList</b> . . . . .	319
A.8	Use case <b>UpdateVolunteerList</b> . . . . .	320
A.9	Use case <b>GenerateACalendar</b> . . . . .	321
A.10	Use case <b>ChangeASchedule</b> . . . . .	322
A.11	Use case <b>ChangeACalendar</b> . . . . .	323
A.12	Use case <b>FindASub</b> . . . . .	324
A.13	Use case <b>GenerateReminders</b> . . . . .	325
A.14	Use case <b>ViewAList</b> . . . . .	326
A.15	Key instance variables for the Person class. . . . .	330
A.16	Key instance variables for the Shift class. . . . .	331
A.17	Key instance variables for the SCL class. . . . .	331
A.18	Key instance variables for the RMHdate class. . . . .	332
A.19	Key instance variables for the Week class. . . . .	332
A.20	Group One master schedule. . . . .	333
A.21	First weekend's master schedule. . . . .	334
A.22	Example calendar view. . . . .	334
A.23	Shift view for 12–3 pm on May 1. . . . .	335
A.24	SubCallList view for 12–3 pm on May 1. . . . .	335
A.25	Filling the 12–3 pm vacancy on May 1. . . . .	336
A.26	Display of a Volunteer in the database. . . . .	337
A.27	Manager home page. . . . .	338
A.28	Volunteer home page. . . . .	339
A.29	On-line Volunteer application form. . . . .	339
B.1	Use case <b>Housecleaning</b> . . . . .	349

---

## *List of Tables*

2.1	Some Important PHPDoc Tags and Their Meanings . . . . .	36
2.2	Client-Oriented vs. Community-Oriented Projects . . . . .	40
4.1	Possible Race Condition Outcomes for Updating a Shift . . . . .	102
6.1	Two Entries in the dbPersons Table . . . . .	151
6.2	Formatting Codes Used by the PHP <code>date</code> Function . . . . .	164
7.1	MySQL Language Connections . . . . .	176
7.2	Common Relations Used in a MySQL Query . . . . .	184
7.3	Common Attribute Types in MySQL Tables . . . . .	185
8.1	Functions in the dbShifts.php Module . . . . .	223
9.1	<i>RMH Homebase</i> User Questionnaire and Results . . . . .	258
A.1	Overall Structure of the <i>RMH Homebase</i> Code Base . . . . .	329



---

## ***Preface***

To understand the principles and practice of software development, there's no better motivator than participating in a software project that has real world value and a life beyond a single academic semester. The *free and open source software* (FOSS) movement provides a fertile ground for identifying such projects and an exciting new environment for teaching the principles of modern software development.

This book uses a model for teaching software development that combines FOSS principles, agile techniques, modern collaboration tools, community involvement, and teamwork as central themes. Together with its accompanying Web site [myopensoftware.org/textbook](http://myopensoftware.org/textbook), this book is designed for use as the main text in a one-semester course on software development.

By completing this course, students should gain a rich appreciation for the principles and practice of FOSS development. As a by-product, they should also become better writers, programmers, and software community members. Our primary goal is that students appreciate the value of collaboration as a fundamental paradigm for software development. They should learn that an effective development team can create a level of software quality that an individual working in isolation cannot typically match.

---

### **Why FOSS?**

The software development world has evolved rapidly in recent years. The traditional life cycle model is now viewed to have serious weaknesses as a basis for developing, maintaining, and upgrading today's software products. So software developers are using new paradigms to meet these new challenges and they are enjoying significant success. A key component of this evolution is the development of FOSS.

There are many advantages to using FOSS as the medium for a software development course. The main advantage is that FOSS allows source code to be freely read, evaluated, modified, and shared without the licensing constraints that accompany the development of proprietary software. Second, FOSS allows the life of a software project to extend beyond the boundaries of one semester or one institution. Third, FOSS development techniques require a level of openness and collaboration among software developers that propri-

etary software techniques do not typically match. We discuss these ideas more fully in Chapter 1.

Many FOSS projects are designed to respond to real public and humanitarian needs. Thus, a FOSS focus can add a valuable service learning component to a software development course.<sup>1</sup>

All the FOSS projects discussed in this book respond to real public and humanitarian needs. In this setting, we distinguish between two types of projects, which we call *client-oriented* and *community-oriented*.

- A client-oriented project is one that is in an incubation stage and is usually tailored to fit the needs of a single client. As such, it usually has a small development group and a modest set of design goals, a relatively small initial code base, and a relatively brief but complete requirements statement.
- A community-oriented project is one that has matured to a point that it fulfills the needs of a large group of clients. Such a project usually has a large (often international) development and user group, a large existing code base, and a continuing need for updating and adding new features.

From a software development point of view, both types of projects share a common body of principles and practices. For example, both require developers to be skilled communicators (among themselves and with users) and adept at using modern collaboration tools (such as working with a shared code base). Developers should also be comfortable participating in Web-based discussion threads and finding solutions to technical problems via those threads. Moreover, they should be familiar with modern programming tools and languages, interactive development environments, and database principles.

Both client-oriented and community-oriented projects actively engage the ultimate users of the software as participants in all phases of the development process—not just during requirements analysis and acceptance testing, as in the traditional life cycle model. This process is a derivative of *agile programming*, since it treats the user as a full-fledged member of the development team and it iterates through the development cycle frequently and continuously until the project is complete.

---

<sup>1</sup>Some have argued further that adding service learning activities to the computer science curriculum will improve the attractiveness of computer science as a major field of study, thus increasing enrollment by women and other underrepresented groups. This, for example, is a major theme of the HFOSS project (see [hfoss.org](http://hfoss.org) and [MTD<sup>+</sup>09] for more information).

---

## Course Organization

The aim of this book is to immerse students directly into an agile FOSS software development process in the setting of a one-semester course. The projects discussed in this book (and the Web site [myopensoftware.org/textbook](http://myopensoftware.org/textbook)) are tailored so that they can be completed in a 13-week semester.<sup>2</sup>

Some of the projects are accompanied by a description of new features to be added to an existing code base (see Appendix B for an example). Other projects are accompanied by a design statement, which includes use cases and an initial code base (see Chapter 11 and Appendix A for examples). Either type of project provides an effective starting point for students to implement, adapt, or extend a software artifact during the course of a semester. Other community-oriented project activities can be crafted by connecting students to an ongoing community-oriented project, such as *Sahana*.

Here are two alternative approaches for using this book in a software development course.

One approach is for the instructor to engage students in a client-oriented project. For example, the client can be a local organization on or off campus and the team should include someone familiar with the application that the software will eventually serve (ideally a member of the organization itself). Alternatively, the project can be the *RMH Home-base* project itself, which is the main example used throughout this text. Other examples of this type of project can be found at the book's Web site [myopensoftware.org/textbook](http://myopensoftware.org/textbook).

A second approach is for students to contribute to an ongoing community-oriented project, such as *Sahana* (see [sahanafoundation.org](http://sahanafoundation.org)). In this case, the domain experts are the community of developers and clients who are contributing to the project from many different locations around the world. Thus, student developers who wish to join this community can begin by participating in Web-based discussion threads at the project's Web site. Other examples of community-oriented projects can be found at [hfoss.org](http://hfoss.org).

For either approach, this book can be used in a mid-level undergraduate software development course, an advanced projects course, or a service learning course for non-majors who have prior programming experience. The important prerequisite is that students have some programming maturity (for instance, by completing the introductory programming and data structures

---

<sup>2</sup>Since these projects utilize open source (non-proprietary) code, their code base can be freely downloaded and reused.

courses) in advance. A course using this book can either precede or follow a software engineering course.

In this course, we recommend that students begin by covering the material in Chapters 1–5. There, they learn the principles of open source software development that can be applied in either a client-oriented setting or a community-oriented setting. For a client-oriented approach Sections 2.3 and 3.4.1 can be skipped, while for a community-oriented approach Sections 2.2 and 3.4.3 can be skipped.

For a client-oriented approach, we recommend that students continue by covering Chapters 6–9, which provide detailed treatments of coding challenges for three different levels of architecture. If the project chosen for the course uses a language other than PHP (e.g., Java, C++, or Ruby), students will need to adapt those PHP-dependent discussions to the language they are using. Eventually, we expect to expand the book’s Web site so that other language options are more directly covered.

For a community-oriented approach, we recommend that students continue by covering Chapters 9–11, which provide additional discussions about forum participation, bug tracking, community building, and project governance. Chapters 10 and 11 provide a kind of “sneak preview” to a software engineering course, since they introduce some principles of project management and requirements analysis that are normally covered in that course.

For either approach, this text provides a team-based, hands-on, agile FOSS development experience, which is its overriding theme.

---

## Sample Syllabi

Below are two sample syllabi that show how a software development course can be structured using the materials in this book. Each one assumes a 13-week semester in which the class meets for 3 hours per week. Both these models are based on syllabi used by the authors while teaching recent software development courses. These syllabi are illustrated in greater detail at the book’s Web site [myopensoftware.org/textbook](http://myopensoftware.org/textbook).

The first sample syllabus shows the organization of a course where students focus on a *client-oriented project* and work in teams throughout the semester.

Week	Milestone	Topics	Readings (chapters)
1		intro to project, design,	1-2
2		collaboration tools platform/design review, team formation	3
3	Domain classes	refactor, design,	4
4		team meeting code,	5
5		team meeting unit test, client review	6
6	Database modules	refactor, design,	7
7		team meeting code,	
8		team meeting unit test, client review	
9	GUI modules	refactor, design,	8
10		team meeting code,	
11		team meeting unit test, client review	
12	Client documentation	system test,	9
13		team meeting system refactoring	
Exam week	Project presentation		

As shown in this syllabus, a team meeting occurs almost every week. This meeting can be a videoconference (e.g., using Skype), an AIM chat, or a face-to-face meeting in which all team members are present. Each meeting should have representation from users as well as developers.

The second sample syllabus (next page) shows the organization of a course where students engage in a *community-oriented project* and work in teams during much of the semester.

During the first few weeks of this course, students complete a few assignments to familiarize themselves with software development tools, database principles, and the details of an ongoing community-oriented project. Thereafter, the team project dominates weekly assignments and culminates in a project presentation at the end of the course.

With either syllabus, the most important outcome is that students gain new skills as team players and developers for a FOSS project. Thus, students should be graded at least in part on the quality of their contributions to the team project.

Week	Milestone (assignment)	Topics	Readings (chapters)
1	Browse projects, Blog post	Free software vs Open Source	1-2
2	A simple interactive program	3-tier architecture, Software Stack and IDE	3
3	A login script	FOSS methodology, database essentials	4
4	Multideveloper application	Project collaboration, version control systems	5
5	HFOSS project reviews	Project management, documentation	9
6	Project launch	Software architecture	
7	Code review, testing, bug reports	Project code reading	10
8	Project testing, bug reports	FOSS licenses	
9	Project testing, bug reports	Open Source 2.0	11
10	Project patching, bug fixing	Mozilla	
11	Project testing, bug fixing	OS 2.0 beyond open source	
12		Term paper assignment	
13	Project release		
Exam week	Project presentation		

---

## Other Support and Guidance

Students using this text should have completed an introductory programming course and a data structures course, or their equivalent. From that experience, they should be comfortable working with classes and objects, basic data structures (including hash tables or associative arrays), and files.

Students should therefore be familiar with a modern programming language such as Java or C++, and they should be prepared to learn other languages (e.g., PHP and MySQL) depending on the code base for the software project that they select. Familiarity with a modern integrated development environment (IDE) such as Eclipse is also helpful. Students should also be prepared

to learn how to use on-line tutorial materials and troubleshooting/debugging support for the languages and tools with which they are less familiar.

A variety of supporting materials are available at the book's Web site [my-opensource.org/textbook](http://my-opensource.org/textbook):

- Downloadable FOSS development projects, including their design documents, use cases, and code bases
- A discussion forum for instructors and students to share their experiences and exchange ideas about particular issues raised by these projects
- Supporting materials for common FOSS development tasks, such as setting up a version control system (VCS), an IDE, a project code base, a unit test suite, and so on
- Additional exercises beyond the ones appearing in this book, reflecting a wide variety of software projects and other activities conducted by people using this book

The supporting materials will be tied to individual chapters where they are required by specific exercises. Solutions to end-of-chapter exercises will also be available to instructors via secure password to the book's Web site [my-opensource.org/textbook](http://my-opensource.org/textbook).

The software tools used by students in this course should also be open source to the greatest extent possible. This preference is not only a cost-effective way to equip a software lab, it also encourages students to become familiar with downloading and installing useful software tools on their own computers. The resources that are referenced directly in this book are:

Activity	Language/tool	Web source
Programming	PHP	<a href="http://php.org">php.org</a>
	MySQL	<a href="http://mysql.org">mysql.org</a>
Documentation	PHPDocumentor	<a href="http://phpdoc.org">phpdoc.org</a>
Testing	PHPunit	<a href="http://phpunit.de">phpunit.de</a>
IDE	Eclipse	<a href="http://eclipse.org">eclipse.org</a> , <a href="http://eclipseclipse.org">eclipseclipse.org</a>
Code synchronization	Subversion (SVN)	<a href="http://apache.org">apache.org</a>
Discussion threads	Google groups	<a href="http://groups.google.com">groups.google.com</a>
Team meetings	Skype	<a href="http://skype.org">skype.org</a>

While this list is PHP-oriented, we expect that students with strong Java or C++ programming experience will be able to assimilate the PHP syntax and easily follow the examples presented in this book.

The choice of development platform that students make will depend on the programming language required by the project that they choose. For example, many open source projects require Java, so that the tools listed above will need

to be replaced by a Java-specific development platform. More extensive and detailed descriptions of these alternatives are provided at the book's Web site [myopensource.org/textbook](http://myopensource.org/textbook).

This book's content is consistent with the Software Engineering recommendations of *Computing Curricula 2008* [ACM08]. It also covers all the topics in the Software Development course described in the *Liberal Arts Model Curriculum* [Con07].

With regard to *Computing Curricula 2008* [ACM08], this book covers all the topics in the Software Engineering section of the core body of knowledge—software design, using APIs and other tools, software processes, requirements, specifications, validation, evolution, and project management. It also covers topics beyond that core body of knowledge—team programming, open source, agile methods, component-based computing, software reliability, and software security.

Finally, students using this text and contributing to an active FOSS project might consider applying for FOSS certification. More information about the FOSS Certificate can be found at the Web site <http://hfoss.org>.

---

## *Acknowledgments*

The authors would like to thank the following reviewers, who gave many thoughtful and insightful suggestions on the selection, ordering, and presentation of material in this text: Bob Cupper (Allegheny College), Ed Gehringer (North Carolina State University), Greg Hislop (Drexel University), Charles Kelemen (Swarthmore College), Richard LeBlanc (Seattle University), Bob Noonan (College of William and Mary), Oliver Radwan, Linda Seiter (John Carroll University), and Laurie Williams (North Carolina State University).

This text benefits from work conducted under the *Humanitarian FOSS Project* at Trinity College (<http://hfoss.org>), with support from the National Science Foundation under Grant No. 0939034. We also thank Bowdoin College for its support, including especially the four Bowdoin students—Oliver Radwan, Maxwell Palmer, Nolan McNair, and Taylor Talmadge—who developed the original release of the *RMH Homepage* software. The staff of the Ronald McDonald House in Portland, Maine, especially Gabrielle Little, provided valuable insight and feedback throughout this project.

Finally, the authors would like to acknowledge and thank our editor Alan Apt for his patient guidance and encouragement throughout the development of this book.



---

## *Authors*

**Allen Tucker** is the Anne T. and Robert M. Bass Professor Emeritus at Bowdoin College. He has a BA in mathematics from Wesleyan University and an MS and PhD in computer science from Northwestern University. He is an ACM Fellow and Distinguished Lecturer. Professor Tucker has authored publications in the areas of programming languages, software development, natural language processing, and curriculum development. He has been a Fulbright lecturer in Ukraine and a visiting lecturer in New Zealand, France, and Germany. He is a member of the Humanitarian FOSS Project's executive committee.

**Ralph Morelli** is professor of computer science at Trinity College in Hartford, Connecticut, where he has been teaching since 1985. He has a BA in mathematics from the University of Connecticut and an MA and PhD in philosophy and an MS in computer science from the University of Hawaii. In addition to a textbook on Java, Professor Morelli has authored publications in the areas of artificial intelligence, FOSS, and computer science education. He is currently one of the principal investigators of the Humanitarian FOSS Project.

**Chamindra de Silva** has been a project lead and architect on the *Sahana* Free and Open Source Disaster Management Project following the 2004 Asian tsunami. He is the CTO and Director of the Sahana Foundation and has participated in project deployment for governments and NGOs in Pakistan, Philippines, Peru, the U.S., China and Haiti. He co-founded the Humanitarian FOSS community and is on the advisory board of the Humanitarian FOSS Project. He is also an Apache Committer, an OSI Member and a contributor to OLPC. He has represented Sri Lanka in Asian Open Source forums such as UNDP-IOSN and AsiaOSSS. He has an MEng in engineering and computer science from Oxford University.