Chapter 2

Working with a Project Team

"It takes a whole village to raise a child."
—*African proverb*

A free and open source software (FOSS) project is distinguished by a development methodology that combines agile techniques with high levels of interaction among developers and users.

This chapter introduces the key ideas behind FOSS development, including agile techniques, the use of frameworks, code reading, documentation, and teamwork.

Two types of FOSS projects, which we call "client-oriented" and "community-oriented" projects, are characterized in this chapter. Participating in a client-oriented project requires a personal level of communication with team members and users. Participating in a community-oriented project requires the use of a different set of communication channels and conventions.

By completing this chapter, readers should be prepared to begin working on a client-oriented FOSS project team or contributing to an ongoing community-oriented FOSS project. The remainder of this book provides guidance for continuing that effort and making a real contribution to an ongoing FOSS project.

2.1 Key FOSS Activities

A number of key activities play important roles in any effective FOSS development effort, whether it be client-oriented or community-oriented. These strategies are summarized in this section and they are illustrated often in later chapters.

2.1.1 Agile Development

The software development world has recently embraced a powerful paradigm called *agile development*. Agile development emerged in response to the many failures of the traditional software process that were noted in Chapter 1.

In a traditional development process, each stage – requirements gathering, design, coding, testing, and delivery – is viewed as a single discrete event. One stage typically does not begin until the previous stage is completed. Typically, the client is involved in the beginning and ending stages of the process, but not in the crucial middle stages. This is illustrated in Figure 2.1.

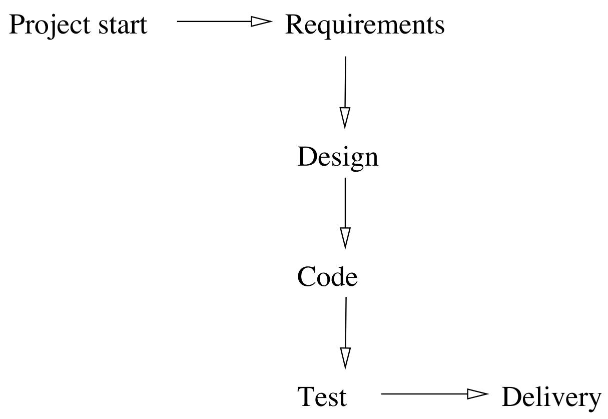


FIGURE 2.1: The traditional software development process.

The agile process is more fluid, in the sense that each stage has a smaller time scale and all stages repeat in a continuing cycle until the project is complete. The agile development cycle requires that the user be engaged continuously, thus allowing new features to be added from user-provided scenarios and test cases throughout the development period. Thus, the software product evolves from the bottom up, rather than from the top down. This process is pictured in Figure 2.2.

Since users are involved throughout the agile development process, they play a critical role at each repetition of the design, coding, testing, and review stages. Because these stages are repeated again and again, each iteration provides a new opportunity for *debugging* and *refactoring* the code base in preparation for adding new functionality in the next iteration.

Debugging means finding and correcting errors in the program. Bugs, or instances of incorrect behavior, result from programming errors. Such errors can often be notoriously difficult to find and correct, even when working with a small code base.

Users play a key part in debugging, since they are the ones who most often identify bugs, both during and after the development process has been completed. Continuous communication between users and developers is essential for debugging to be effective. FOSS development, which is an agile process,

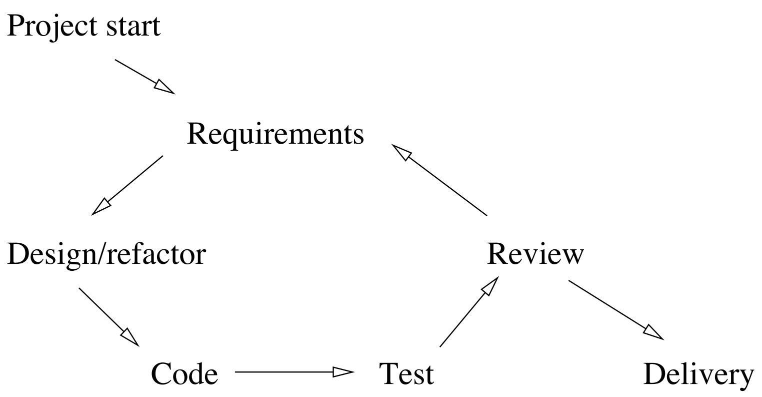


FIGURE 2.2: The agile software development process.

is especially effective in this regard, since it relies on continuous interaction between users and developers.

Refactoring a program means to read the code, find instances of poor programming practice (from either a readability or an efficiency standpoint), and reorganize (usually simplify) the code so that it performs the same functions in a more readable and/or efficient way. No new functionality is added during refactoring, only an improvement in the quality of the code.

The ability to read code with a critical eye, especially code not written by oneself, is a fundamental skill in software development. Software is seldom written by a single person from scratch, contrary to what might have been inferred in introductory programming courses. Instead, software is usually developed incrementally by many developers, each one adding new code to an existing "code base," thus adding a new feature to its overall functionality.

2.1.2 Using Patterns

The overall architecture of a software system refers to the organization of its code base in a way that best reflects the system's functionality, supports its systematic development and deployment, and allows effective distribution of programming tasks among the team's developers.

A *design pattern* is an abstraction from a concrete programming form that keeps reappearing in different coding contexts. Once such an abstraction is identified, it is described and given a name by which programmers can refer to and reuse it.

Examples of design patterns include: Strategy, Visitor, Observer, Mediator, Factory and Prototype (for object creation), and Iterator (for traversals). For more discussions about these and other software design patterns, see **en.wikipedia.org/wiki/Design_pattern**.

An *architectural pattern* moves the idea of a design pattern to the architectural level of a software artifact. At this level, we have patterns for implementing computer-user interaction, such as Client-Server and Model-View-Controller (MVC), as well as others like Multi-Tier, Pipe-and-Filter, and Peer-to-Peer. See http://en.wikipedia.org/wiki/Architectural_pattern) for more discussion of archtectural patterns. The Client-Server, Multi-Tier, and MVC architectural patterns are more fully discussed and illustrated in Chapters 4 and 8.[1]

Once a system's architecture is determined, the code base can be organized appropriately. For example, software organized using a client-server or multi-tier pattern often has the following natural directory structure for its code base.

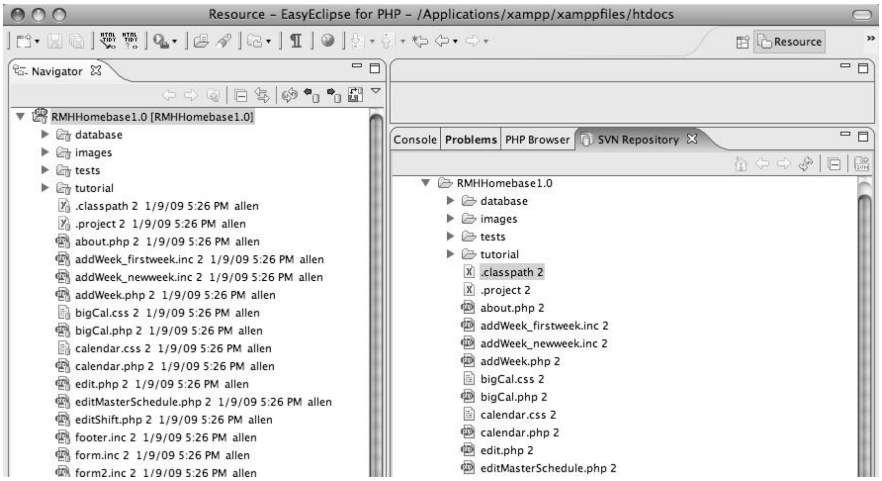
Domain modules/classes These define the key elements of the application. The domain modules define the name space upon which all other modules are derived. They reflect terminology familiar to users as they exercise the software. Names must be chosen carefully, so as to promote clarity of communication among developers and users. Development of the domain modules in a software system is the subject of Chapter 6.

Database modules/classes These define the tables and variables that comprise the persistent data in the system. The database resides on the server in a client-server software system. Each table is related to one or more core classes/modules. Development of the database modules is the subject of Chapter 7.

User interface modules These implement the system-user interactions that take place on the client side of the software system. The user interface implements the functionality given by the use cases in the requirements document. Development of the user interface is the subject of Chapter 8.

Unit test modules Each of these modules is developed in conjunction with a core, database, or user interface module. Together, the unit test modules comprise a "test suite" for the software system. Whenever any module in the system is changed (refactored, added, or expanded), the test suite must be run to be sure that the system's functionality is not compromised by that change. Unit testing strategies are revisited in each of Chapters 6, 7, and 8.

[1]By contrast, a *software framework* is an organizational generalization for a specific type of software application. It is language- and system-specific, and its code inevitably contains instances of various design and architectural patterns. For example, Web application frameworks are implemented in PHP, Java, Ruby, and C++. Well-known PHP frameworks are called CakePHP and the Zend Framework. Well-known Java frameworks are called Apache Struts and Spring. For more discussion of software frameworks, see http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks.


Resource – EasyEclipse for PHP – /Applications/xampp/xamppfiles/htdocs
Resource
Navigator
Console
Problems
PHP Browser
SVN Repository

```
RMHHomebase1.0 [RMHHomebase1.0]
  database
  images
  tests
  tutorial
  .classpath 2 1/9/09 5:26 PM allen
  .project 2 1/9/09 5:26 PM allen
  about.php 2 1/9/09 5:26 PM allen
  addWeek_firstweek.inc 2 1/9/09 5:26 PM allen
  addWeek_newweek.inc 2 1/9/09 5:26 PM allen
  addWeek.php 2 1/9/09 5:26 PM allen
  bigCal.css 2 1/9/09 5:26 PM allen
  bigCal.php 2 1/9/09 5:26 PM allen
  calendar.css 2 1/9/09 5:26 PM allen
  calendar.php 2 1/9/09 5:26 PM allen
  edit.php 2 1/9/09 5:26 PM allen
  editMasterSchedule.php 2 1/9/09 5:26 PM allen
  editShift.php 2 1/9/09 5:26 PM allen
  footer.inc 2 1/9/09 5:26 PM allen
  form.inc 2 1/9/09 5:26 PM allen
  form2.inc 2 1/9/09 5:26 PM allen
```

```
RMHHomebase1.0
  database
  images
  tests
  tutorial
  .classpath 2
  .project 2
  about.php 2
  addWeek_firstweek.inc 2
  addWeek_newweek.inc 2
  addWeek.php 2
  bigCal.css 2
  bigCal.php 2
  calendar.css 2
  calendar.php 2
  edit.php 2
  editMasterSchedule.php 2
```

FIGURE 2.3: Using the Eclipse environment with Subversion.

On-line help tutorials These modules are typically developed last, and in conjunction with user training that accompanies delivery of the software. Their structure tends to mirror that of the user interface modules, so that there will be one help screen for each distinct use case that users will be exercising with the system. Writing on-line help modules is covered carefully in Chapter 9.

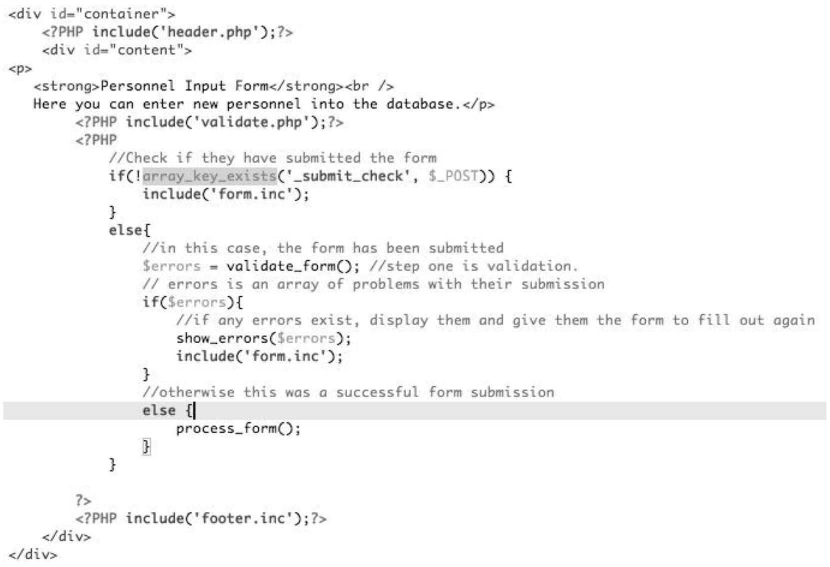
Organization of individual modules within the code base is initially done by the lead developers. Refinements to this code base are made by individual developers on the project team. To facilitate collaboration, and to ensure the integrity of the code base when several developers are making changes to it simultaneously, a "version control system" is used to synchronize their work.

As shown in Figure 2.3, the code base for *RMH Homebase* follows this general organizational scheme. Figure 2.3 also shows how several developers can synchronize their work using a version control system (SVN in this case). More discussion of version control systems appears in Chapter 3.

2.1.3 Reading and Writing Code

Program reading and writing is, of course, the central activity within software development, since the program is the software. The ability to read and write program code in the language(s) of the application is a key requirement for all the developers in a software project. A corollary requirement is that developers must be able to learn new language features and skills as needed to support the timely implementation of new software elements.

Many fine programming languages are in use for developing contemporary software. This book's examples are shown in PHP and MySQL, although



```
<div id="container">
    <?PHP include('header.php');?>
    <div id="content">
<p>
   <strong>Personnel Input Form</strong><br />
   Here you can enter new personnel into the database.</p>
        <?PHP include('validate.php');?>
        <?PHP
            //Check if they have submitted the form
            if(!array_key_exists('_submit_check', $_POST)) {
                include('form.inc');
            }
            else{
                //in this case, the form has been submitted
                $errors = validate_form(); //step one is validation.
                // errors is an array of problems with their submission
                if($errors){
                    //if any errors exist, display them and give them the form to fill out again
                    show_errors($errors);
                    include('form.inc');
                }
                //otherwise this was a successful form submission
                else {
                    process_form();
                }
            }

        ?>
        <?PHP include('footer.inc');?>
    </div>
</div>
```

FIGURE 2.4: Example code from *RMH Homebase*.

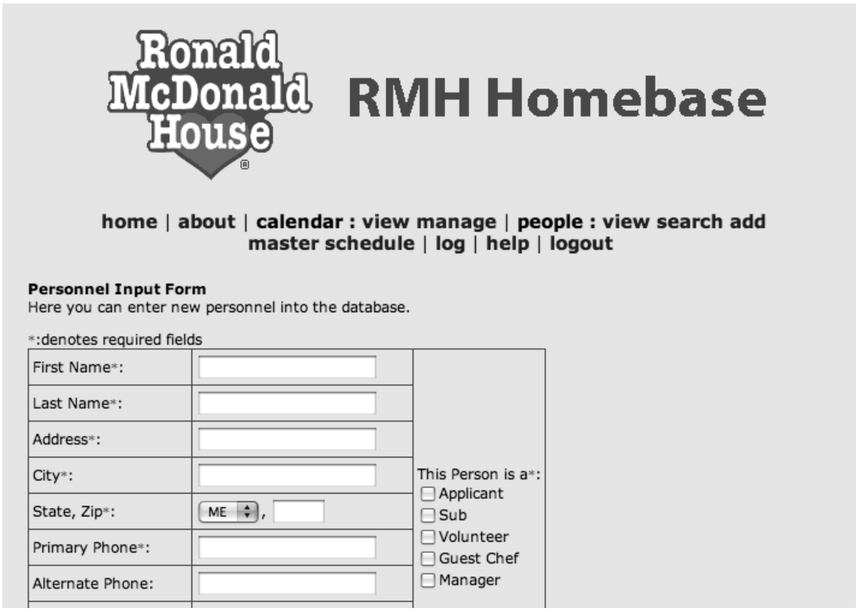
readers who are familiar with Java or C++ should have no trouble understanding their meaning. Other projects and tutorials appearing at the book's Web site myopensoftware.org/textbook use different languages, such as Java, so that readers working with different languages in their projects should look there for additional support.

Reading code is an especially important skill. The code you read in published programming textbooks is, unfortunately, not typical of the code you find in most software applications.

In general, code published in textbooks tends to be more uniform and readable than code found in real applications. By contrast, code that appears in software applications often reflects more than one programming style and may contain elements that are inefficient, unnecessarily verbose, or just plain difficult to read.

For example, Figure 2.4 shows some example PHP code from the *RMH Homebase* project, and Figure 2.5 shows what it produces in a Web browser. PHP code can be embedded inside the HTML of a Web page by using the HTML tags <?PHP and ?>. Whenever such a page is rendered, the PHP code is executed. In this example, the embedded PHP code calls functions that display the header (which includes the menu bar), the applicant's information form, and a footer (not shown in Figure 2.5).

Writing good code requires using conventionally accepted coding and documentation practices. This is especially important when the software is being developed by a team. Unfortunately, much of the code that underlies actual software products does not reflect the use of good practices. Thus, some of



RMH Homebase

home | about | calendar : view manage | people : view search add
master schedule | log | help | logout

Personnel Input Form
Here you can enter new personnel into the database.

*:denotes required fields

First Name*:		This Person is a*: ☐ Applicant ☐ Sub ☐ Volunteer ☐ Guest Chef ☐ Manager
Last Name*:		
Address*:		
City*:		
State, Zip*:	ME ,	
Primary Phone*:		
Alternate Phone:		

FIGURE 2.5: Output of the example code in Figure 2.4.

a programmer's work includes rewriting poorly written code to make it more readable and receptive to the addition of new features.

Below is a list of widely used coding standards for common program elements. These standards are illustrated here in PHP, though similar standards exist for Java, C++, or any other contemporary programming language.

- Naming and spelling—Class, function, variable and constant names should be descriptive English words. Class names should be capitalized; if they consist of more than one word, each word should be capitalized—e.g., Person, SubCallList.
  - Multiple-word function and variable names should separate their adjacent words by an underscore—e.g., `$primary_phone`. Alternatively, these names can be written with each non-first word capitalized—e.g., `$primaryPhone`.
  - Global variable names should be written in all-caps and begin with an underscore—e.g., `_MY_GLOBAL`.
  - Constant names should be written in all-caps—e.g., `MY_CONSTANT`.
- Line length—A line generally contains no more than one statement or expression. A statement that cannot fit on a single line is broken into two or more lines that are indented from the original.

```
/**
 * fill a vacancy in this shift with a new person
 * @return false if this shift has no vacancy
 */
function fill_vacancy($who) {
    if ($this->vacancies > 0) {
        $this->persons[] = $who;
        $this->vacancies=$this->vacancies-1;
        return true;
    }
    return false;
}
```

FIGURE 2.6: Documentation with indented blocks and control structures.

- Indentation—Use consistent indentation for control structures and function bodies. Use the same number of indentation spaces (usually four) consistently throughout the program—e.g., see Figure 2.6.

The use of coding standards such as these is sometimes met with resistance by programmers. They argue that pressures to complete projects on time and on budget prevent them from the luxury of writing clear and well-documented code all the time.

This argument breaks down when the software being developed is subject to later revisions and extensions, especially by other programmers who need to read the code as they revise and extend it. This is especially true in the open source development world, where the source code typically passes through several sets of programmers' eyes as it evolves.

2.1.4 Documentation

As a general rule, effective software development requires that the code not only be well written but also be well documented.

Software is well documented if a programmer unfamiliar with the code can read it alongside its requirements statement and gain a reasonable understanding of how it works. Minimally, this means that the code should contain a documentary comment at the beginning of each class and non-trivial method, as well as a comment describing the purpose of each instance variable in a class. Additionally, each complex function may contain additional documentation to help clarify its tricky parts.

When reading a code base for the first time, a new developer may find a shortage (sometimes a complete absence) of commentary documentation. If the code is well written, the reader may still be able to deduce much of its functionality from the code itself. In fact, it is a good exercise for a developer new to a code base to add commentary documentation in places where it is

```
/**
 * class Shift characterizes a time interval for scheduling
 * @version May 1, 2008
 * @author Alex and Malcom
 */
 class Shift {
    private $mm_dd_yy; // String: "mm-dd-yy".
    private $name;     // String: '9-12', '12-3', '3-6',
                       // '6-9', '10-1', '1-4', '12-2', '2-5'
    private $vacancies; // no. of vacancies in this shift
    private $persons;  // array of person ids filling slots,
                       // followed by their name,
                       // e.g. "Malcom1234567890+Malcom+Palmer"
    private $sub_call_list; // "yes" or "no" if SCL exists
    private $day;      // string name of month "Monday"...
    private $id;       // "mm-dd-yy-ss-ss" is a unique key
    private $notes;    // notes written by the manager
    ...
```

FIGURE 2.7: Inserting comments in the code.

lacking. That is, it improves one's own understanding and it contributes to the project by improving future readers' understanding of the code.

A good way to begin refactoring an under-documented code base is to add documentation in places where it is lacking. This helps not only to improve the quality of the code for future readers, but also to reveal bad smells and improve the code's receptiveness to the addition of new features.

Documentation standards exist for most current programming languages, including PHP and Java. These latter are supported by the JavaDoc and PHPDoc tools, respectively. They implement a standard layout for the code and its comments, and they automatically generate a separate set of documentation for the code once it is fully commented.

For example, consider the Shift class in the *RMH Homebase* application discussed above. Its documentation contains a stylized comment of the form:

```
/**
 *
 */
```

at the head of each class and each non-trivial method, as well as an in-line comment alongside each of its instance variables. This is shown in Figure 2.7.

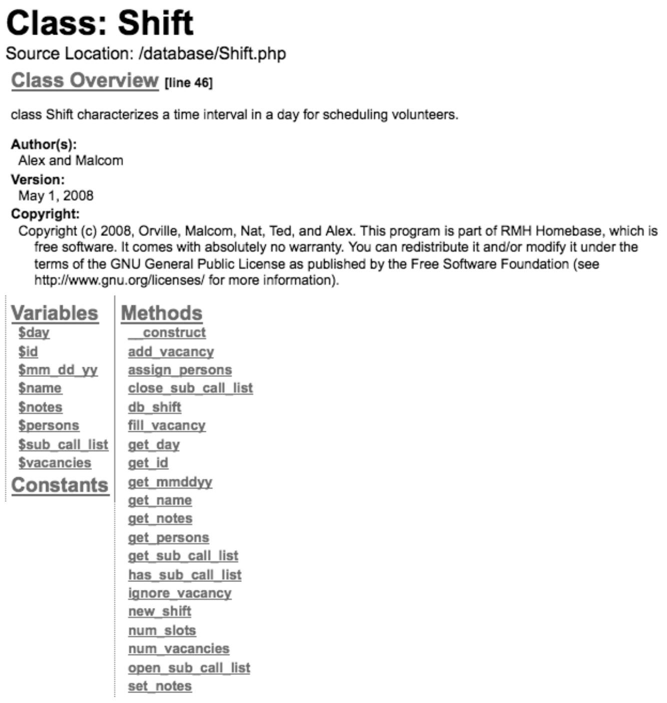
Notice that this documentation contains so-called *tags*, such as `@author` and `@version`. When used, each of these tags specifies a particular aspect of the code, such as its author, the date it was created or last updated, and the value returned by a method.

TABLE 2.1: Some Important PHPDoc Tags and Their Meanings

Tag	Meaning
@author	name of the author of the class or module
@version	date the class or module was created or last updated
@package	name of the package to which the class or module belongs
@return	type and value returned by the function or method
@param	name and purpose of a parameter to a function or method

A short list of the important PHPDoc tags with a brief description of their meanings is shown in Table 2.1.[2] Other languages, such as Java, have similar tagging conventions. It is important for developers using those languages to learn those tagging conventions before writing any documentation.

Once a code base is completely documented, a run of PHPDoc (or JavaDoc, as the case may be) will generate a complete set of documentation for all its classes and modules. For example, Figure 2.8 shows the documentation that is generated for the `Shift` class shown in Figure 2.7.



Class: Shift

Source Location: /database/Shift.php

Class Overview [line 46]

class Shift characterizes a time interval in a day for scheduling volunteers.

Author(s):
Alex and Malcom

Version:
May 1, 2008

Copyright:
Copyright (c) 2008, Orville, Malcom, Nat, Ted, and Alex. This program is part of RMH Homebase, which is free software. It comes with absolutely no warranty. You can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation (see http://www.gnu.org/licenses/ for more information).

Variables	Methods
$day	__construct
$id	add_vacancy
$mm_dd_yy	assign_persons
$name	close_sub_call_list
$notes	db_shift
$persons	fill_vacancy
$sub_call_list	get_day
$vacancies	get_id
Constants	get_mmddyy
	get_name
	get_notes
	get_persons
	get_sub_call_list
	has_sub_call_list
	ignore_vacancy
	new_shift
	num_slots
	num_vacancies
	open_sub_call_list
	set_notes

FIGURE 2.8: PHP documentation generated for the Shift class.

[2]A more complete list can be found at http://phpdoc.org.

2.1.5 On-Line Help

Documentation serves the needs of developers who are interested in understanding the functionality of a software system. However, it does not serve the needs of users who want to learn the system. For this purpose, separate elements are needed, which often take the form of on-line help tutorials, or equivalently a printable manual that teaches the user how to perform each use case supported by the software.

So, as a separate step from documentation, open source software development requires that on-line help tutorials be developed and integrated within the software itself. These tutorials are written in a language and style familiar to the user, and they should use terminology that is common to the user's domain of activity.

Each help tutorial should correspond to a single user activity, called a "use case,"[3] which typically appears as a group of interactive forms. Thus, the word "Help" in the navigation bar should take the user to that particular tutorial corresponding to the form with which the user is currently working. The tutorial itself should describe a short sequence of steps, with illustrative examples, that shows how to accomplish an instance of the use case that the form implements.

For example, the *RMH Homebase* software has nine distinct use cases (see Appendix A). One of these is called "Change a Calendar" which allows the user to find and fill a vacancy for a shift on a particular day of the week. The corresponding form for that use case is shown in Figure 2.9.

If the user needs assistance, the help tutorial for filling a vacancy can be selected on the menu bar, and the user receives the step-by-step instructions shown in Figure 2.10.

The help tutorial opens in a separate window from the user's current form, so that he/she can work with the form while simultaneously reading the screen.

2.2 Client-Oriented vs Community-Oriented Projects

Prior to the emergence of FOSS development, large-scale proprietary software projects were characterized by a tightly controlled, top-down, hierarchical development process. The newer FOSS development process is characterized by a loosely controlled, bottom-up, distributed community that is highly cooperative and democratic.

This atmosphere of openness and collaboration cannot be emulated in the proprietary development world, since that world demands complete secrecy

[3]See Chapter 5 and Appendix A for more discussion and examples of use cases.