1

Keeping It Simple

‘‘Keep It Simple’’ is really a synonym for ‘‘Get a Grip.’’

If you are reading this book, you probably have some degree of responsibility in getting a website application up and running. If you are reading this chapter and you are searching for a series of steps you can actually assimilate and follow toward fulfilling that goal, then you are in the right place.

First of all, in a software project (and because of its complexity, that is what a website application is), you are either controlled by circumstances or you succeed — but only if you can maintain your grip on things. That’s only if, after receiving all the advice, you are able to fashion your own means of zooming into detail, return to the overview, keep it going, and know at all times where your bookmarks are . . .and if you can pilot the process of each layer making up the project, on every front: the purpose, the design, the usability, the navigation, the function, the data, the push and pull and flow of actions and results, the emission and reception of messages, the completion of tasks, the updating, the classification, and relation of content.

Which is to say, if you keep it simple and keep it all in view, or at least know where to look for it, then you can marshal your own approach to truly leveraging a powerful, open-ranging, and dynamically productive framework such as Drupal, the ‘‘Community Plumbing’’ Content Management System framework middleware powerhouse, featuring:

- ❑ Significant off-the-shelf functionality
- ❑ Tremendous extensibility through nearly 3,500 contributed modules
- ❑ Based on one of the most active Open Source communities in existence

Drupal is all of these things.

Add to the mix that Drupal itself is evolving at a fairly brisk pace, as you’ll see in later chapters, and you definitely need to come to Drupal with your own approach.

Because you are using Drupal for world domination (a favorite geek metaphor among Drupaleros), then you had better have a program. And you had better make sure that everyone involved gets on that program and stays there.

Getting with the "Program"

The ''program'' means that you must start out with a clear idea of how your client defends her interests with the website application in the works. In the program, keeping it simple does not mean splitting it apart and losing the richness of vision, nor does it mean oversimplifying.

This chapter lays out a method that you follow throughout the rest of the book. Then, you can either adopt it lock-stock-and-barrel or roll your own. But we definitely recommend following some kind of Agile approach and have developed a lean, mean methodology checklist. We find that this means, at a bare minimum, maintaining a policy for:

- ❑ **Vision and Scope** — The business vision and scope
- ❑ **Visitors and Users** — Who's going to use the website?
- ❑ **User Stories** — Narratives telling us what the users are going to use the website for
- ❑ **Analysis and Design** — What needs to be done so they can do that?
- ❑ **Planning and Risk Management** — When should you do that?
- ❑ **Design and Usability** — What should it look like?
- ❑ **Tracking and Testing** — Making sure you're getting what you really want
- ❑ **Technology Transfer and Deployment** — Turning over the helm to those who will be managing the website application each and every day

Figure 1-1 shows a basic main process workflow for this book's example project. The workflow is strongly influenced by Mike Cohn's book *User Stories Applied* (`http://amazon.com/User-Stories-Applied-Development-Addison-Wesley/dp/0321205685`).

The Perl programming language, in common with Drupal, has been one of the major Open Source success stories of all time, answering a burning need in an intelligent and synthetic way, backed by an extremely active community led by very smart people. And, like Drupal, given a problem, it provides an enormous number of alternatives offering themselves as solutions. ''There's more than one way to do it'' has always been their slogan, and the same holds true with Drupal: there is always more than one way to do it. So, of course, you can substitute your own process workflow and find your own solutions along the way. The important thing is to recognize that the development of a website application is a complex process. To get it done right and to leverage a powerful, dynamic, and productivity-enhancing framework like Drupal, you need to develop your own independent approach and method as you gain experience yourself. The method you'll use throughout this book is a ''starter set'' you will adapt and tailor to your own needs, as you develop the Literary Workshop community website.

In a nutshell, the main process workflow makes the first task the identification of the customer and, by extension, the business vision and scope of the project as well as the complete list of stakeholders involved. Then comes the identification of the roles — the different kinds of users who will use the site. For each role, you write a series of user stories, identifying all the possible interactions the role will have with the website application. Doing it this way (asking who will use the site, and, for each of the roles, what they are going to do when they interact with it) guarantees that you can cover all the functionality required and come up with a complete list of user stories.

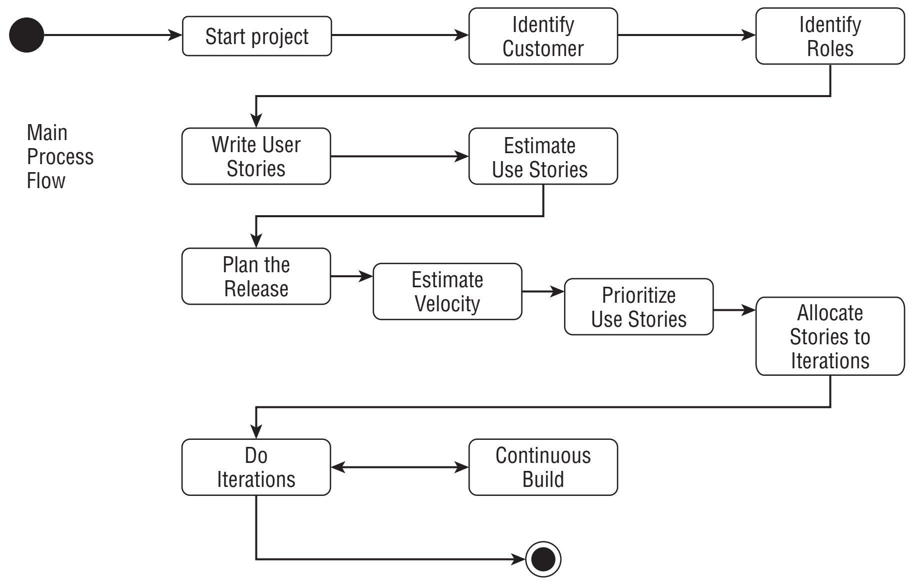

Figure 1-1

At this point, you have all your user stories, perhaps written on 3×5 cards and spread out on a table in front of you, or on a magnetic board, or taped up to the wall, or whatever. So you can do the planning. This involves making an initial estimate for each user story, taking advantage of the fact that each user story is a semi-autonomous chunk of functionality that can be dealt with independently. Then, you create a way of putting the estimates in context on the basis of the velocity of the team. (Is this our first time? Any extra-special technical areas of difficulty, like dealing with a text messaging gateway, or with specialized web services?)

Next, you are ready to prioritize the user stories. If they are indeed 3×5 cards, this means unshuffling the deck and putting them in order. The two most significant criteria for this should be: which ones does the client think are the most essential, and which ones need to be tackled first because they involve some kind of risk that needs to be mitigated at as early a stage as possible.

This process dovetails into the next important planning task, which is allocating the stories to iterations. You want to have several iterations, at least four to six for a medium site, even more for a large site, following the Agile principle of ‘‘frequent releases.’’ One reason for this is so that the client, who should be considered part of the development team, can give really effective feedback in time for the architecture of the site not to be adversely affected by any ‘‘surprises’’ that may crop up: If implementation is straying far from the client expectations of what the website is supposed to actually do, you want to find out about that sooner rather than later. Another is so that work can be expressed as much as possible using the semantics of the solution domain, rather than the problem domain — which means that people can think much more clearly when something concrete is up and running, rather than being forced to work in the abstract.

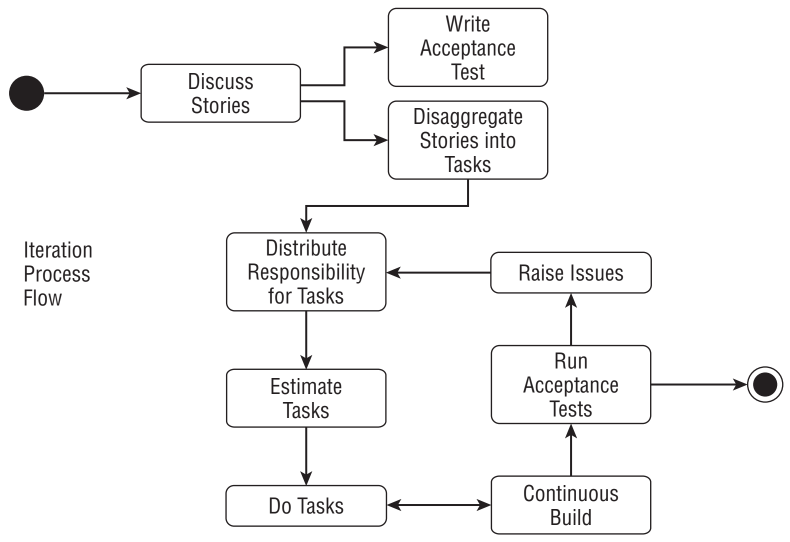

Figure 1-2

So now, you have planned your iterations, and you have on the table (and hopefully on the wall), or else entered into your favorite issue tracking system, essentially four to six piles of no more than five user stories (more iterations and more user stories per iteration if it is a bigger website, also depending on estimated team velocity).

Basically, you want to grab the first pile (the first iteration) and implement it. Now, for each planned iteration, or phase (sometimes people group iterations in phases), you use the workflow shown in Figure 1-2.

To do this, you take each story and discuss it, the client takes a major responsibility for writing the acceptance test for it, and you list all the tasks that need to be carried out in order to actually implement the functionality involved in the user story. The acceptance test is basically a semi-formal to formal statement of precise criteria according to which the work has actually been done right.

According to the Extreme Programming website (`http://extremeprogramming.org` — a great starting point to finding out more about a lot of the methodology we are talking about and using in this book, as is Kent Beck's ground-breaking work on the subject, *Extreme Programming Explained: Embrace Change*; `http://amazon.com/Extreme-Programming-Explained-Embrace-Change/dp/0201616416`):

> Acceptance tests are black box system tests. Each acceptance test represents some expected result from the system. Customers are responsible for verifying the correctness of the acceptance tests and reviewing test scores to decide which failed tests are of highest priority. Acceptance tests are also used as regression tests prior to a production release.

Essentially ''getting your site done right'' means that all the acceptance tests pass for all user stories. You'll be learning about acceptance tests and other forms of testing, like unit testing, below in the book.

In any case, it should be clear that the acceptance tests must be written before any work is actually done. Once written, and all the tasks necessary to implement a given user story are listed, each task can be taken up by a pair (hopefully) of developers. (It's much more productive for two people to work together on a task, but we won't get involved in any religious wars here; also, you might be working by yourself, so just be prepared to put on a lot of hats!) The developers will take the task, make an estimate, carry it out, and immediately integrate their work into the development test site. Later, the client will run the acceptance tests for the user story, and if they pass, the work just got done! Otherwise, an issue is raised in whatever issue tracking system you are using (more on this later), and the work is recycled, until the acceptance tests all pass.

Let's get our lean-and-mean methodology checklist straight now, starting with the task of mapping the business vision and scope.

Starting with a Map for Business Vision and Scope

Experience at succeeding has shown that to achieve the benefits of the ''KISS'' approach, you actually have to dig down deep to the roots of what is an organic, dynamic process. It is not a question of over-simplification for the sake of simplification.

To succeed, you need to stand ''commonsense'' on its pointy head, at least for a while, and acquire a deep vision: It is not only that those who lack a business plan will not enjoy financial success. While true, what you are concerned with here is that without first identifying the business plan, there is no way you can build a website application that meets your clients' needs and fits right into their regular activity. Real needs cannot be translated into an analysis and design, analysis and design into implementation, implementation into a working model for testing, a working tested model into a deployed website application — the website application the client needs will never be born.

In traditional Information Technology terms, this is called a *Business Model* (see Wikipedia). The importance of business rules is present in the context of Agile Modeling, as well. Business Modeling is a difficult subject to master in its own right, but thankfully, you can cut to the chase here, and draw yourselves a Web 2.0 picture of the relationship between the business rules, the feature list, and the offerings of a website application: a meme map.

> *For more on Business Models, see Wikipedia at* `http://en.wikipedia.org/wiki/Business_model`*. For more on Agile Modeling, see* `http://agilemodeling.com/artifacts/businessRule.htm`*. For more on meme maps, see ''Remaking the Peer-to-Peer Meme,'' by Tom O'Reilly,* `http://oreillynet.com/pub/a/495`.

A meme map shows the deep relationship between the internal activities of a business or organization, their strategic positioning, on the one hand, and that business' outward, public face, on the other, including the website applications and their functionality, which is what it is you actually have to develop. Everything is clear at a glance. This is just what you need to get started. Look at Figure 1-3 (which shows a meme map for a Drupal-based Literary Workshop website application) and the comments following it.

- ❑ At the top, there are three bubbles containing the main public functionality of the website application.
- ❑ In the middle, the core is shown as a rectangle housing the positioning strategy and guiding principles (which may well differ with someone attempting a similar kind of site, but which will have a big impact on what you will be doing anyway).

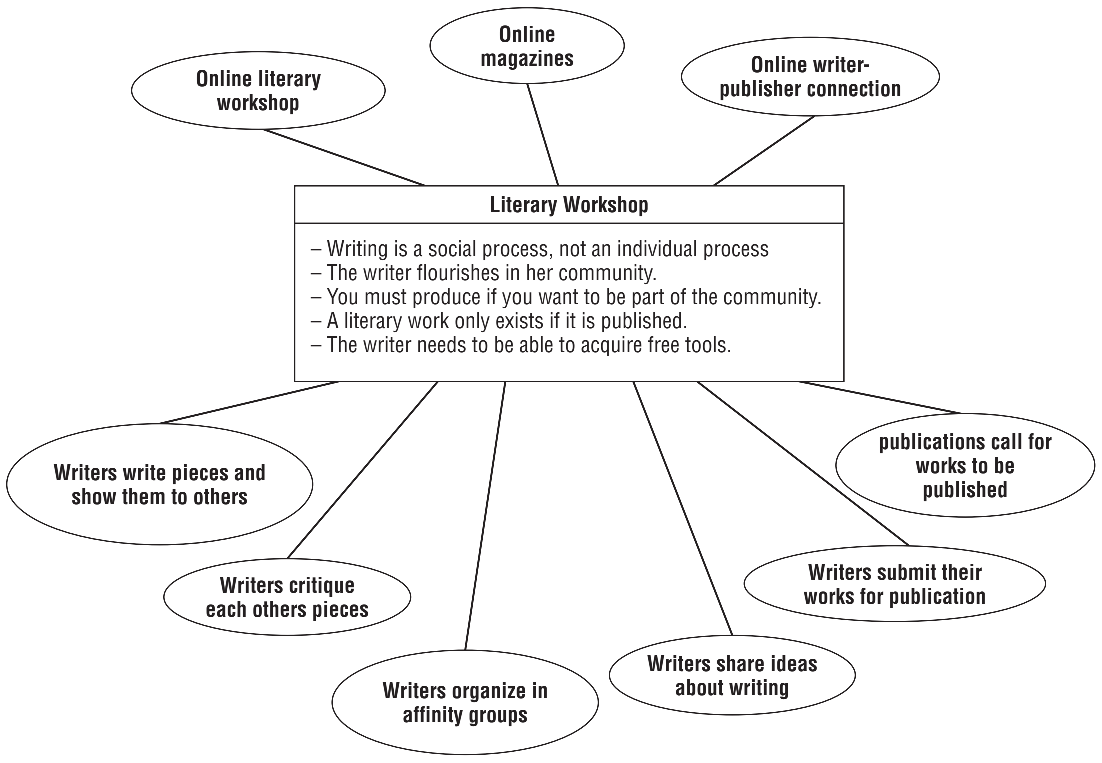
Online literary workshop
Online magazines
Online writer-publisher connection
Literary Workshop
– Writing is a social process, not an individual process
– The writer flourishes in her community.
– You must produce if you want to be part of the community.
– A literary work only exists if it is published.
– The writer needs to be able to acquire free tools.
Writers write pieces and show them to others
publications call for works to be published
Writers critique each others pieces
Writers submit their works for publication
Writers organize in affinity groups
Writers share ideas about writing

Figure 1-3

- ❑ Below are the regular business activities housing and forming the material basis and personal interactions supporting the rest.

From this point on, ''Keeping It Simple'' is going to mean banishing anything that isn't directly connected to your business vision, automatically and constantly getting rid of the fluff.

Going to the heart of the matter, and keeping that present, will enable you to get a grip right from the start.

There is a website where you can make your own meme maps (Do It Yourself Meme Map Generator, `http://web.forret.com/tools/mememap.asp`), so you can try it out yourself. Or you can use any diagram drawing tool. Or use pencil and paper (that will work!). In any case, I strongly recommend that you follow along in this book by actually developing your website application as I develop mine. Practice makes perfect.

Who's Going to Use the Site?

This question really goes to the identification of the actual users of the website itself, and also the users of the website in a business sense.

Perhaps a sector of the back office, for example, will never actually use the site as such, but will be interested in receiving periodic statistics, say, as a weekly email. They must be included, of course, in the list of roles. Here's a list of roles for the Literary Workshop website application:

Role	Description
Workshop Leader	The person who actually runs the workshop, decides who to accept, monitors whether members are complying with requirements, and also participates along with the other members
Workshop Member	Someone who has joined the workshop and actively participates in it
Publisher	Someone publishing a magazine, on and off the site
Webmaster	Technical administrator of the website

The main thing is that every possible user of your website application needs to be taken into consideration in order to truly capture the complete set of requirements that need to be met in the implementation of the project. At the same time, a complete list of all interactions with the site (their user stories) for each of these users completes the picture.

What Are They Going to Use It For?

Let's make a list of user stories, then, for each of the Roles we have previously identified.

Role	User Story
Workshop Leader	Can approve applications to join the workshop (from members and magazine and book publishers)
	Can suspend members and publishers
	Can manage affinity groups
	Can broadcast messages to members
	Can do everything workshop members and publishers can do
Workshop Member	Can post literary pieces
	Can make any post public, private, or visible to an affinity group
	Can critique public posts
	Can browse public pieces and critiques
	Can send and receive messages to and from all members, publishers, and the workshop leader
	Can start an affinity group with its own forums
	Can post to forums
	Can maintain their own literary blogs

Role	User Story
Publishers	Can browse public content
	Can broadcast a call for pieces to be submitted for a publication
	Can select content for inclusion in a publication
	Can manage an on-line publication
	Can manage an on-line blog
Webmaster	Can administer the website configuration
	Can install new updates and functionality

What Needs to Be Done So They Can Do That?

You will be taking each user story and doing some analysis and design aimed at discerning what can be reused from the giant Drupal storehouse of core and contributed functionality, and what needs to be added on — perhaps contributing back to the community in the process (you'll learn why this is a great idea later on in the book).

But that isn't enough. The answer to this question is actually to be found during the course of the iteration planning workflow as well as in the user story implementation workflow. During the planning stage, when we are prioritizing user stories and assigning them to iterations, we would do well to bear in mind the organization of iterations established by both the Rational Unified Process (see www-306.`ibm.com/software/awdtools/rup` as well as `http://ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf`) and the Open Unified Process (see `http://epf.eclipse.org/wikis/openup/index.htm`) into four phases, or groups of iterations:

Phase	Iterations	Description
Inception	Usually a single iteration, with a resulting prototype	Vision, scope, and feasibility study enables the initiation of the project based on cost and schedule estimates. Initial requirements, risks, and mitigation strategies are identified, and a technical approach is agreed on.
Elaboration	Usually two iterations, prototype confirming architectural decisions	During the elaboration phase, the requirements baseline and the architectural baseline are set. Iterations are planned. Planning, costs, and schedule are agreed on. A lot of work gets done, and the test site is up and running with successive development releases.
Construction	Enough iterations to get the job done	Change management is in force from the onset of this phase. Alpha and Beta release will be completed, and all Alpha testing completed by phase end.

Phase	Iterations	Description
Transition	Usually a single iteration	Release 1.0 of the website application has been deployed on the live site and accepted by the client and is in production. All documentation and training have been completed, and an initial maintenance period has elapsed.

Well, here we are getting to some pretty rigorous language. But, these phases actually occur in any project, and it is best to be conscious of the whole process, so as to manage it instead of being managed.

The main thing to understand here is that as the basic workflow is followed, two baselines emerge, relatively early in the project — a requirements baseline (the sum of all user stories) and an architectural baseline. Now, the decision to use Drupal in the first place settles a slew of architectural decisions. But you need to decide exactly how required functionality will be supported. Here are a few examples:

❑ Which modules will support the use of images? Will images be Drupal *nodes* in their own right, or fields in a node?

❑ What editing facilities will different kinds of users have at their disposal? Will a wiki-style markup approach be chosen, or will a rich text editor be necessary? Which one? And once it is chosen, how will it be configured?

❑ Will part of the site's content find its origin in external sources? Will those sources be news feeds? Will specialized multimedia modules be necessary? Will that content simply be listed, or will it be incorporated into the database also?

❑ To what extent will the website need to scale?

❑ In terms of support for foreign languages, will there be a need for localization (the process of adapting the user interface, including menus, currency and date formats, to a particular locale, or language, locality, and their customs, commonly abbreviated as l10n)? Will there be a need to make the content of the site multilingual through the use of internationalization modules (the process of making one or more translations into various different languages available for each content item, commonly abbreviated as i18n)?

❑ What is the basic content model? What classes of content and what content types need to exist, and what is the relationship between them?

And then there is also a whole other area of things that need to be attended to that are ongoing throughout the project, namely, setting up development, testing, and production sites; setting up a build and deployment procedure, including a version control system; and setting up an environment for everyone working on the project, with all the necessary (compatible) tools and workstations. You guessed it — this will be dealt with in detail in upcoming chapters.

To delve even further into the whole question of software development process engineering (really getting it done right!), check out the CMMI website (''What Is CMMI?'' at `http://sei.cmu.edu/cmmi/general`*). There are also books on the subject, specifically* CMMI Guidelines for Process Integration and Product Improvement*, by Mary Beth Chrissis, Mike Konrad, and Sandy Scrum (*`http://amazon.com/CMMI-Guidelines-Integration-Improvement-Engineering/dp/0321154967`*), as well as* CMMI Distilled *(*`http://amazon.com/CMMI-Distilled-Introduction-Improvement-Engineering/dp/0321461088`*). These sources give* *a good overview and grounding for this model. This model has been proven totally compatible with Agile approaches (*`www.agilecmmi.com/`* is just one example), and while it may definitely be overkill for most readers of this book, it may make all the difference in the world for some.*

When Should You Do That?

The answer to this question is: during the whole project! There will be constant imbalance and balance struck again between two apparent opposites: the need to decide what to do and then do it, on the one hand, and the need for change, on the other. So this calls for an incremental and iterative approach, providing frequent opportunity for client feedback and for taking stock, and providing entry points for change and its impact to be repeatedly evaluated.

The mistake has been made time and time again, of using the so-called *waterfall model* approach to website development as a way of keeping things simple. ''We will decide exactly what we want to do, and then we will do it.'' Experience has shown that this is a recipe for disaster and must be replaced with an incremental and iterative approach.

For more information on the Standard Waterfall Model for Systems Development, see `http://web.archive.org/web/20050310133243` *and* `http://asd-www.larc.nasa.gov/barkstrom/public/The_Standard_Waterfall_Model_For_Systems_Development.htm`.

Now, keeping it simple is actually the opposite of banishing change. Change is actually the mechanism by means of which clients understand what they really want and make sure the final product embodies their needs. That is why there need to be frequent iterations.

So, progress in the project means that various models are actually being built at the same time.
The architectural big picture emerges and takes shape on a par with the requirements baseline, sometime in the third or fourth iteration, which is to say, together with actual deliveries of prototypes. By that time, all the user stories are written, estimated, and prioritized, and the iterations to implement them are planned.

But as the team starts plucking the next user story off the wall and starts seeing how to implement it, and as the client begins to see his or her dream more and more in the flesh, subtle and not so subtle changes will occur. If the planning has been good, then the user stories with the biggest likelihood of affecting the architecture will be among the first to be tackled. Then their impact will be positive and will help to shape the architectural baseline.

The catastrophe scenario to be avoided (and it is not always so easy to avoid) is a user story that gets implemented very late in the game and turns out to have a huge impact on architecture. You find out, for some reason, that the editing user interface simply has to be done in Flash, so we need to solve the problem of how to integrate Adobe's Remoting protocol with Drupal via, say, the Drupal Services module. Well, it's all good, but you really need to know about things that have that kind of impact earlier on.

The more the work is planned around iterations that are constructed in terms of a basic architectural vision and constantly checked by all concerned, the less likelihood there is of that kind of high-cost impact catastrophe occurring.

What Should It Look Like?

Isn't it nice that this is just one more little section of this chapter? Too often a project is reduced to its bells and whistles (see the next section, ''Making Sure You're Getting What You Really Want'').

Well, because having a ''Web 2.0 design'' really is a concern of many clients, a good way of understanding what that means and what elements go together to constitute it, is the article ''Current Web Design'' (`http://webdesignfromscratch.com/current-style.cfm`).

But, most of all, you should be concerned about usability. You should concern yourself about form following content and being dictated by content. The best way to do that is to get the functionality of the site going first, and then and only then imposing the graphic design. That is the method you will be using in this book.

> *The obligatory read here is Steve Krug's book,* Don't Make Me Think *(*`http://sensible.com`*). However, I have recently seen this book cited in the Drupal forums as a reason why people in general, including developers, shouldn't have to think. No, for the end-user of a website not to have to think (that is what it's about), a lot of thinking has to go on: Drupaleros have to do a lot of thinking to get their websites done right.*

The main lessons are:

- ❑ The importance of usability testing
- ❑ The need to start out with a clear home page instead of ending up with a home page that is out of control
- ❑ The importance of usability testing
- ❑ The need to base your site navigation and design on how people really use the Internet
- ❑ The importance of usability testing

Drupal is great for this kind of approach — first, because, as you shall see, it is its own greatest prototyper, and second, because of its great theming system. With Drupal, the functionality is really skinnable, on a high level of detail and in a very flexible manner. But, of course, you have to know what you are doing. However, once you learn the secrets, you can leverage an extremely powerful theming system that has also proven itself to be very SEO friendly.

Making Sure You're Getting What You Really Want

You should be concerned about testing, with the discipline of avoiding being driven by the bells and whistles instead of by what you really need. You should also be concerned about ''Feature Creep,'' with quality control, and with building, which is understood as the integration of dependable blocks and units.

There are two basic principles involved here, and getting what you really want depends on both of them being observed:

- ❑ Unit testing forms an important part of the responsibility of implementing a piece of functionality. Unit tests must be written in such a way that they cover the maximum possible number of key functional points in the code.

- ❑ The whole process of development should be test-driven, by which we mean acceptance test-driven. Acceptance tests are black box tests; they test how the website application should behave. In website applications, usability testing forms an important part of acceptance testing.

While there are other forms of testing that should be included, such as stress and load testing, these two — unit tests and acceptance tests — are two you absolutely cannot do without. Indeed, the PHP SimpleTest framework is becoming part of Drupal. The module page can be found at `http://drupal.org/project/simpletest`, while great documentation is centralized at `http://drupal.org/simpletest`.

We have already defined acceptance tests, and here simply need to stress that they should be written and executed by the client, even though he or she will need to count on your assistance throughout the project in order to do so.

Turning Over the Helm

At some point, the artist must be dragged kicking and screaming from her masterpiece and told: ''It's done. It's not yours any longer; it belongs to the final user.'' This, too, must be planned for and implemented throughout the project. Its easy accomplishment is another beneficial result from getting the client involved, early and actively, in the project. Seeing the client as someone to keep at a distance like a bull in a china store will result in a difficult delivery of the website to those who will be using it thereafter.

Again, an iterative approach will help, and is actually indispensable, in order to avoid the all-too-often witnessed scenario of finishing a site and having no one to deliver it to, or else, delivery constituting itself as an almost insurmountable obstacle.

From the start, the responsibilities must be clear. One example scheme could be the following:

- ❑ Client is responsible for testing and acceptance.
- ❑ Client is responsible for administering website users and setting their permissions.
- ❑ Developer is responsible for updating Drupal core and modules.
- ❑ Client is responsible for contracting hosting.
- ❑ Developer is responsible for initial installation on production site.

This is why the phase is called *Transition* in the Unified Process approach. The website application itself must be deployed after all testing is completed. A maintenance plan must be in place. But there must also be documentation (manuals and/or on-line Help) and training in order to empower the client and the final users to actually use what they have acquired, for them to really take ownership.

In the case of website applications, the tight schedules they usually have and the fact that the resources actually required are generally overwhelming compared to what the client may actually have been thinking at the outset, so the more gradually this is all done, the better.

In this book, therefore, there will be a fictionalized client who will also be very much present throughout the project and who will actually motorize everything.

Information Architecture and an Agile Approach for the Rest of Us

Best practices fans and refugees will discern throughout this chapter a dependency — a ''standing on the shoulders of giants'' — in relation both to the Agile approach to software development and to the discipline of information architecture (see `http://webmonkey.com/tutorial/Information_Architecture_Tutorial`; see also a more advanced article: `http://articles.techrepublic.com.com/5100-22_11-5074224.html`). Here I am drawing from a huge body of materials, and given the practical character of this book, I run the risk of treating these subjects superficially.

I hope that in the course of working through this book, it will be clear that I am not ''name-dropping'' buzzwords, but, rather, extracting from vast areas of expertise just what you need, and no more, in order to succeed at a task that is sold to you as simple and straightforward and that is simple and straightforward compared to doing everything from scratch, but that is neither simple nor straightforward.

So everything mentioned in this chapter will be used thoroughly, and you will gain a practical familiarity with all these tools.

So whether you are a project manager with specialized departments working under you, or someone who practically has to do the whole project alone, your responsibility in getting this site done right will very much make you a Renaissance person. You will learn much more about CSS and tools like Firebug than you may care to, more about navigation and menu systems than you may care to, even more about ''templates'' and PHP than you may care to. Indeed, in using Drupal, you may learn much more about ''Views,'' ''content types,'' ''Taxonomy,'' and ''clean URLs'' than you ever dreamed of. You may find yourself checking out from CVS and SVN repositories and apt-get installing whole operating systems, or organizing and/or supervising others who do that as part of their everyday work.

You will find yourself involved in sending in ''patches'' to module maintainers. You may even be involved in theme or module development. You will find yourself concerned about unit test ''coverage'' and usability tests.

It is hoped that you will end up with a site done right together with a stack of passed acceptance tests that truly document the system requirements.

The Example Used throughout This Book

As mentioned, you will be developing the Literary Workshop website application as an example project to illustrate the material presented in this book. Using a version control system and a standardized deployment procedure, you will be able to move forwards and backwards in time to all stages of development as part of your practical exploration.

As well, in the course of working your way through this book, you'll discover a whole series of what are termed reusable ''Website Developer Patterns.'' These are collectible challenges, solutions, and secrets that crop up time and time again and need to be dealt with in project after project, and a systematic approach must be found for them in order to get your site done right.

So, let's get to it.

Summary

In this chapter, you have been introduced to the methodology to be followed in this book in order to get the most out of the Drupal CMS framework, and to the nontrivial example you will be working with throughout. The methodology is based on an Agile approach to any kind of software development, and has been tailored to the development cycle required to develop a Drupal website application. In the next chapter, you will take your first practical steps and get the functional prototype up and running.