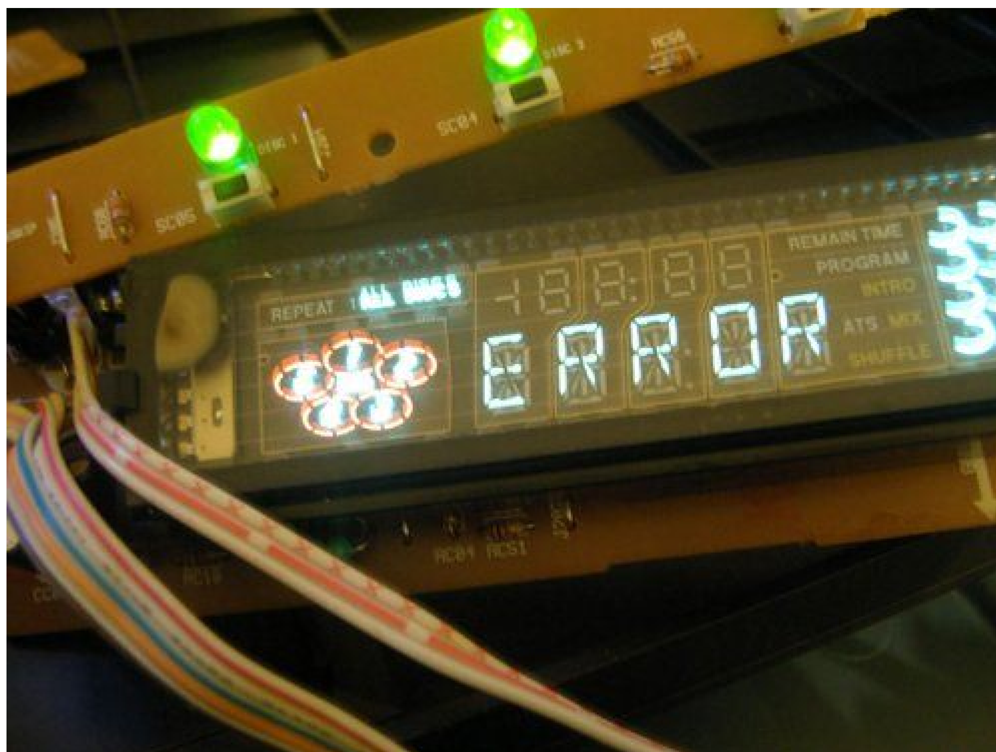
Top 21 PHP Programming Mistakes

Part 3: Seven Deadly Mistakes



By Sterling Hughes

January 14, 2001

This eBook was designed and published by Free PDF Ebooks.

For more free eBooks visit our Web site at http://www.acrobatplanet.com.

Free PDF Ebook at http://www.acrobatplanet.com

Table of Contents

Top 21 PHP Programming Mistakes – 1
Part 3: Seven Deadly Mistakes..........1
Intended Audience..........4
Introduction..........4
1. Getting Lost in Time..........4
SterlingWonderfulToyland.com: An Example 5
The Plain Jane Section..........6
The E-commerce Portion..........6
Selling deadlines..........6
When you don't allocate enough time..........7
2. Not Sticking to a Project Plan..........8
Project Phases..........9
Requirements Analysis Phase..........9
Determining User Requirements..........9
Determining Technology Requirements..........10
Program Design Phase..........10
Model it..........10
Illustrate It..........10
Draft Pseudo Code..........11
Testing Phase..........12
Regression Testing..........12
Stress Testing..........13
3. Excluding the User from the Design Process..........13
Continuous User Feedback..........14
Prototyping..........14
When should milestones be set?..........15
Beta Testing..........15
4. Hacking at Design Flaws..........15
Design Flaw Indicators..........16
Correcting Design Flaws..........16
5. Not Doing a Code Review..........17
6. Not Having Project Code Guidelines..........18
A Sample Project Guideline Document..........19
Style Guidelines for DesignMultimedia.com..........19
Introduction..........19
File Structure..........20
Page Headers and Footers..........20
Code Documentation..........21
Commenting Style..........21

Free PDF Ebook at http://www.acrobatplanet.com

Implementation Guidelines..21
Variable Naming..21
7. Cut and Paste Coding: The Wrong Way..21
Doing It the Right Way: Learn first, Then Copy..22
Libraries are Fine...22
Summary..22
About The Author...23

Free PDF Ebook at http://www.acrobatplanet.com

Intended Audience

This article is intended for the PHP programmer interested in avoiding some of the most common mistakes when applying PHP. The reader is expected to at least be familiar with PHP syntax, but should have a working knowledge of PHP functionality as well.

Introduction

One of PHP's greatest strengths happens to be one of its greatest weaknesses as well: PHP is easy to learn. Many people are attracted to the language because of this, not realizing that it's a lot tougher to learn how to do it *right*.
There just hasn't been enough emphasis on good programming practice. Inexperienced coders are being asked to create and distribute complex web applications. Mistakes that an experienced programmer would avoid are all too common, such as the improper use of the printf() function or the misapplication of PHP's semantics.
In this three part article series, I present a list of 21 mistakes that I believe are frequently made and ranging in severity from non-critical down to those that can break the farm. I offer solutions, suggestions and/or comments on how to solve and prevent these errors, in addition to other tricks of the trade that I have gained over the years.
The series is comprised of the following three articles:

- **Part 1:** Covers the first 7 "textbook" mistakes (#21-15, in reverse order of severity) on our rating list. Committing one of these mistakes, while not critical, will lead to slower and less maintainable code.

- **Part 2:** Covers the next 7 "serious" mistakes, representing # 14-8 on our rating list. Committing one of these mistakes will lead to drastically slower run times and less secure scripts, in addition to less maintainable code.

- **Part 3:** Covers the last 7 "deadly" mistakes. These mistakes are conceptual in nature and can represent the root cause for committing any one of the mistakes listed in Series Parts I and II. They include blunders such as not allotting enough time for a project and not having a thorough code review.

Free PDF Ebook at http://www.acrobatplanet.com

1. Getting Lost in Time

The number one mistake is a timeless problem that programmers have had to deal with – getting lost in time. Face it, we are optimists. It is natural for us to assume that a software project will take just as long as it *should* take. We don't account for *everything* that could go wrong. For example, the fact that we might have trouble with the implementation of something might not even cross our minds.
So, we must suppress our gut feelings and crunch those numbers. As a rule of thumb whenever I contract a software project, I always take into account every factor that I can think of, and come up with an estimated time. Then I *double* this estimate. It is hardly ever inaccurate, and companies are rarely disappointed when you finish your software ahead of time (whereas if you're late, they're not quite as happy).
This "armchair" method of estimating helps me set an ample time to do a quality job on the program without falling behind schedule. But, in truth, it depends on the accuracy of my initial estimate.
Every programmer underestimates (or in rare cases overestimates) the time necessary to complete a project by a certain amount, for me the mean is about 2x, for someone else it might be 1.5x, and a third person might underestimate actual time required by 3x. The trick is to figure out what your average error is when estimating the time necessary for projects and consistently tacking on that error margin to all of your projects.
Not allocating enough time to your project will have a few effects (which I know of all too well):

- You will forfeit much of your social life until the project is finished. All your days and nights will be spent programming, leaving little time for relaxation (not that programming isn't fun, but programming *non-stop* is horrible).
- You will be rushed, meaning that you will spend less time on making your code secure, readable and fast. Instead, your focus will only be on banging out your code by the deadline.
- You may have to skip crucial steps, such as a code review or debugging, which, in turn, will lead to irate clients and even *more* work.
- You may have to add more manpower, which costs you more money, and may not help (according to Brook's Law, especially on large projects).

Free PDF Ebook at http://www.acrobatplanet.com

Always make sure to allocate enough time in your projects for a lengthy debugging process (just as long or longer than the development process, itself), and 1/3 of the total project time for planning. These are the two most important parts of the development process.

SterlingWonderfulToyland.com: An Example

Let's take a sample site, SterlingWonderfulToyland.com. This is a simple e-commerce site that has been designated to sell its limited selection of Playstation 2's and Video Games over the Internet at discount prices.
Being an old fashioned kind of guy, the owner John Giuseppe, also wants to encourage customers to come into the toy store itself and pick out the toys at the store. Therefore, the site has two main sections:

- An e-commerce end of the site.
- The plain jane portion of the site - giving you the directions to the Toy store and a store layout.

When planning, I would come up with separate estimates for each section of the Web site. I would then combine the two resulting deadlines, add around a week to integrate the two sections and lastly pitch the deadline to the client for the whole site (stating the deadline as if there was just one unified section).

The Plain Jane Section

For the simple, non-dynamic portion of the web site, I would take a stab at how long it would take to make each page with an editor such as Macromedia's Dreamweaverä. Since there is really nothing dynamic about it, I would multiply that estimated time by the total estimated number of pages (of course this assumes that the design of the Web site has already been done). I would then double that estimate (as my personal error margin is usually 2x). For this section no real planning is needed (I mean heck, once you have a diagram of how it should look, given to you by your designer, all you have to do is lay it out in Dreamweaver).
Note: Factor in up to an additional 1/3 of the time you expect will be spent implementing the application. This amount is for planning the application. For example, if you expect that it will take 9 days to implement a credit card validation program, then tag on an extra 2-3 days.

The E-commerce Portion

Free PDF Ebook at http://www.acrobatplanet.com

For the e-commerce portion, planning out the required time is pretty tricky. What I find that helps is to simply break the task up into even simpler components (such as credit card transactions, one-click ordering, product browser, product management, etc), and estimate the different components *without* applying the margin of error. I would then combine them, and multiply the sum by my margin of error. Then at this point, I would allocate an additional 1/3 of the time I expect it to take to complete the project, solely to testing and debugging, to make sure that I deliver the client a bug free product.
Note: Factor in up to an additional 1/3 of the time for planning, as for the Plane Jane component.

Selling deadlines

Even if you can appropriately set a project date, your boss or client will not be as happy as you are with your deadline. Sometimes a deadline that is too long can be a deal-breaker. Other contractors will give a deadline that they can't possibly keep to get the deal and then simply be late with the project. So how do you set a deadline that your boss/client will be happy with and you can honestly keep? Here's where you've got to sell yourself and your deadlines... What can you offer the client with deadlines? What makes your deadlines feasible and why are your deadlines set as they are? It often helps to role-play, pretending you are the client, and asking yourself what would be fair to expect from a potential programmer? Some points that I have found to be important include:

- *Do it right*. Other programmers may do it a little faster, but you'll do it correctly which saves the client money and time in the long run (this is especially useful if you charge by the hour).

- *Stick to your word.* The deadlines that you are setting are deadlines that you intend to keep. Other programmers may give deadlines that look better, but they have no intention of keeping to them.

- *Provide Client recommendations.* Your previous jobs speak for themselves, have your satisfied clients give you recommendations that you can then show to your boss/potential clients.

- *Be thorough!* Run your proposals through spell-check, have someone proof them for grammar. Add diagrams, and clearly state everything that you plan to do and how long each of the tasks you define will take.

- *Give ranges*. Don't just tell clients how long you think it will take - provide them with scenarios. What is the best-case scenario? What is the worst-case scenario?

Free PDF Ebook at http://www.acrobatplanet.com

- *Set short deliverable dates.* The further in the future a target date is set, the more likely it will *not* be met.

When you don't allocate enough time

Sometimes you don't allocate enough time for a project, leaving you very far behind... When this happens you still have many options (that aren't always clear at the time).

- *Communicate!* Communication between you and your client is the most important thing possible. Constantly inform them of what your currently doing. If you see that you may need to extend a deadline a little it won't be as hard as if you haven't communicated all along.

- *Show your boss/client a partial version of the Web site*: It often helps when asking for time extensions to show the work that you've already done. Demonstrate that you have been working on the project.

- *Blame Canada*: Take full responsibility for being late with the project, but don't forget to mention any external factors that caused you to be late.

- *Cut corners*: This is nasty, but if you're really late with a project cut some corners to get a working version of your application for the client, with the hacks in your code clearly marked, then, during your testing phase and later on, go back and fix those hacks.

- *Go GUI*: If I'm creating an application I really prefer to handcraft my HTML, I don't trust the output generated by any WYSIWYG editors. However, if you are in a time crunch consider constructing your HTML with the WYSIWYG editors.

Free PDF Ebook at http://www.acrobatplanet.com

2. Not Sticking to a Project Plan

Many software projects these days eventually lead to "making it up as you go along". On one of my first projects, for example, I created an application based on a 40-minute phone call. Now, while the application turned out ok, the risk of failure was far greater than had I put in the proper time to plan and design the application *before* I went ahead and implemented it.
For example, when I finished the project, nothing was abstracted. This means that if the user had wanted to use MySQL instead of MSSQL, the application would have needed a re-write. Moreover, much of the security was a hack implementation. I stored all of the application data in the cookie. There were a couple of other flaws - some too embarrassing to mention...
When Fred Brooks outlined the software development cycle in *The Mythical Man Month*, he gave the following as an optimal schedule:

- **1/3** Planning: Here is where you design the way that your application works, what are the different components of the application (and the sub-components) and how do they interact? What should the application do? Answering these fundamental questions is the basis of planning an application.
- **1/6** Coding: What we all love to do. Transform the design into reality.
- **1/4** Component test and early system test: When developing large applications there comes a point when the application itself might not be complete but the application is finished enough that testing of the basic functionality can begin.
- **1/4** System test all components in hand: This is the final stage where we have the finished application, now the application must be tested and re-tested to make sure it is as bug free as possible.

Today we are lucky if even 1/6 of the dedicated project time is spent on planning. Programmers begin right away, furiously pumping out code without having a clear idea of the requirements, an appreciation for the problem, or a viable approach to solving that problem. This scenario is analogous to writing a term paper or an article without having an outline.
Coding should just be the process of placing down what you have already planned. A lot of programmers (me being one of them) go as far as writing their entire application (or tricky parts of their application) in pseudo-code prior to coding it with a real language such as PHP.

Free PDF Ebook at http://www.acrobatplanet.com

Note: The process of planning the look, feel and functionality requirements of an application is what is known as information architecture.

Project Phases

There is a lot that can be said about the stages of a project plan. For example, mention can be made of debugging, flowcharting, modeling, project management and setting target dates. However, I will only cover a basic outline.
A Standard Project Plan will include a:

- Requirements Analysis Phase
- Program Design Phase
- Testing Phase

Requirements Analysis Phase

The first part of planning a project is to create a requirement analysis; this is where you define exactly what is needed. You specify exactly what the program must do and how the program should work. This is one of the most important phases of designing a web application.

Determining User Requirements

So how do you find out what exactly the user requires for an application? Take the role of a consultant to determine their needs, namely get to know your client well. For example:

- What do they do?
- What makes their product superior (or unique)?
- How do they want to present themselves to their audience?
- What features in the site will help them reach their market?

This last one I've found is a great way to make your projects larger. Can you find a way that adding functionality to their web site will help them? If so, you've just got a more satisfied customer, as well as a larger (and better paying) project on your hands.

Free PDF Ebook at http://www.acrobatplanet.com

Methods of research can vary from distributing surveys or questionnaires to interviewing the key decision makers in the company. In whatever manner you go about collecting your information, it is critically important to keep the above points in mind.

Determining Technology Requirements

Here is where you decide what technology your application will need to use and how you will need to use it. The overriding question is do we have the capability and know-how to meet the user requirements? This can include for example, the programming language (in addition to PHP), the OS, the server speed, connection speed, etc.

Program Design Phase

You have the specifications. Here is where you decide how to write the application, what happens when; perhaps even some pseudo-code if you encounter a tricky point of implementation. In short, you will need to:

- Model it
- Illustrate it
- Draft Pseudo-code

Model it

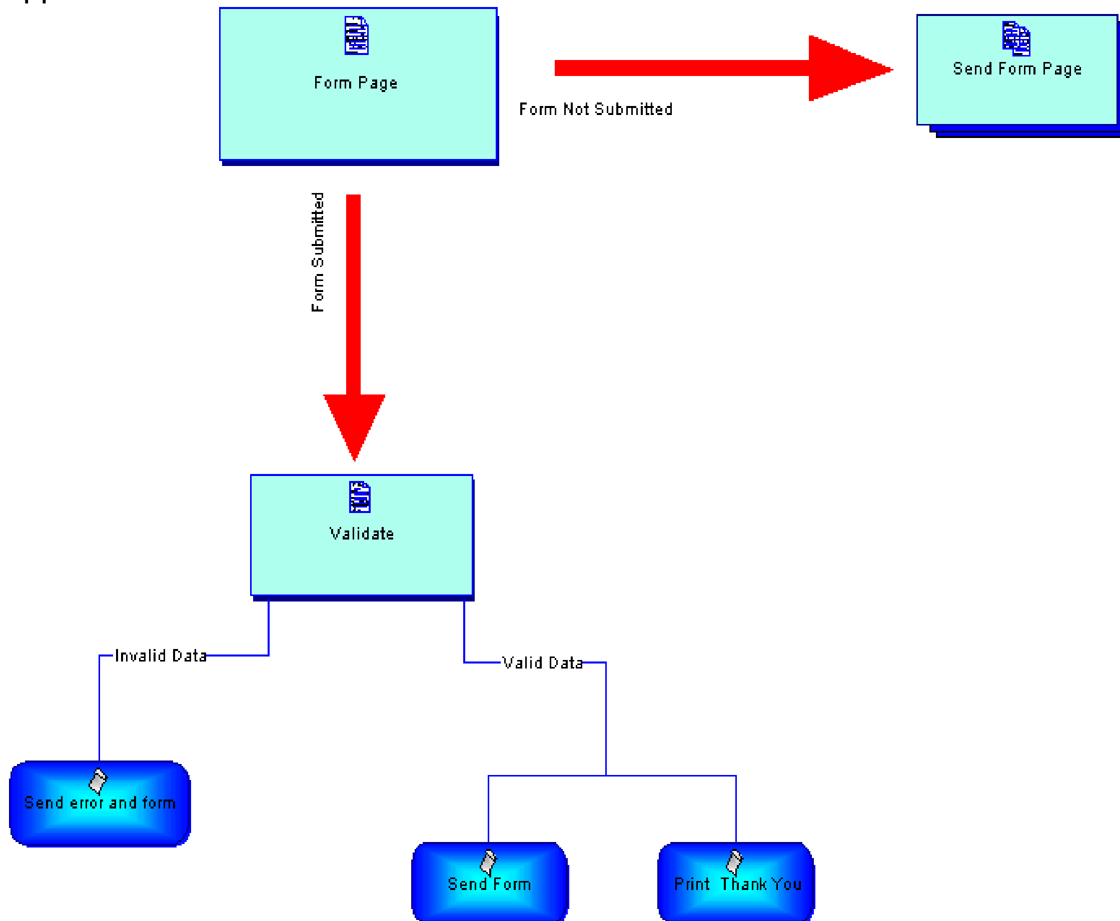
Before you code an application, conceptualize how the different parts of your application will interface with each other.
For example, let's design a simple form application:
We consider the possible actions a user can do when accessing a simple mail sending script. In the most abstracted view of the application, there are two possible options. They have submitted form data or they have *not*.
If they have not submitted any form data, then we should send them the initial form page. Otherwise, we should (again) do one of the following two things.

- If the data submitted is valid (meaning it conforms to the criteria for valid form data) then send the mail to the user and output a thank you message.
- Otherwise, send error message informing them of what went wrong, and giving them the option to correct this error.

Free PDF Ebook at http://www.acrobatplanet.com

Illustrate It

A very simple diagram showing the flow of an application is displayed below. It was done in Microsoft Visioä and is perfectly suitable for a simple form mail application.



However, when it comes to more advanced applications, it is often useful to have a specialized tool that will help you diagram your applications. For information on three popular tools, check out: Dia, Visio, and UML.

Draft Pseudo Code

Writing *pseudo* code, meaning code that describes the application but does not really work, is a common practice that many developers often use. Generally it is employed when they're stuck on a tricky bit of implementation or they can't fully see how a particular aspect of the application would fit in. I've also found that pseudo-code is useful when defining application interfaces, namely, where other

programmers will need to use the interfaces that I design. It helps to see what will be necessary and the easiest ways to go about it.
For example, the pseudo-code below can be drafted when creating our simple Web site form-mail application.

```
if (formSubmitted)
{
   valid_name(name) or error() and output_form();
   valid_email(email) or error() and output_form();
   valid_message(message) or error() and output_form();

   message = name & email;

   send_mail(email, message);

   print "Thank you for your feedback";
}
else
{
   print "Send us feedback";
   formelement(name);
   formelement(email);
   formelement(message);
}
```

The pseudo-code above defines the basic structure of the script and shows all of the specific required elements. The code does not have to be valid running PHP code (although it can be). What pseudo-code should do is define the different tasks of the application and perhaps the theory behind those tasks.
The level at which you abstract your psuedo-code is of course up to you. Personally, I'm more inclined to a less abstracted form of psuedo-code than many people. It all depends on how comfortable you are with programming.
Finally, once you have planned the whole application you can then start to code your application knowing all the steps you're going to need to take and what exactly you have to create.

Testing Phase

One of the most important stages of application development that is often left out is the (final) testing phase. Often due to time and/or management pressure, final testing is shortened or bypassed with the application being deemed production ready.
Let's be honest, programmers hate testing. It's probably one of the most tedious and annoying stages of developing an application. It often consists of wild goose chases, hours of debugging, and testing all of the different bounds to make sure that your code works correctly in most cases. To top this off, you'll never have bug

Free PDF Ebook at http://www.acrobatplanet.com

free code! You can always expect to miss something. You know, the one thing you would *never* think would happen, takes place anyway.

Regression Testing

Applications are perpetual "works in progress". It is important to ensure that when you add new functionality, you don't compromise the old functionality that your users have come to depend on. Therefore, you need what is known as a *Regression Testing Suite*. This is a set of tests that makes sure that functionality that currently works does not break down due to the new changes that you make. PHP itself has a regression-testing suite to make sure that all functions and processes work correctly so when you make a change to PHP you're sure that you haven't broken another part of PHP. This helps PHP not only keep backwards compatibility (i.e., when new features are added, your old scripts don't break). But it also is a way of making sure that none of the changes made actually break functions (not just change their behavior).

Stress Testing

OK, so your application works great with 1 user - everything seems to be going perfectly and with tremendous speed. But what about when your application is used by 20 users? 30 users? Or even a 100 users simultaneously? How fast is your application then? Does your application suddenly emit strange errors? Whenever you test an application you should always make sure to stress test it to make sure that it does not implode under high loads and varying conditions.
This does **not** mean passing over the testing phase to the user. You know, just get it out and wait for the bug reports and feedback. Beta test it (as mentioned in the previous mistake). Moreover, there are some automated tools that emulate the testing over a large user pool. One that comes to mind is Apache's *ab* tool. AB, or Apache Benchmark, will perform a certain amount of requests on your Web page and return the success rate, failure rate, the average time, etc.
I suggest that you use *ab* on all of your Web pages (that is, if you've made the infinitely wise decision to go with Apache). Then you can identify and optimize those pages that are memory hogs or take a long time to load.
(Also, look into purchasing Zend's powerful script caching utility – **The Zend Cache**).

Free PDF Ebook at http://www.acrobatplanet.com

3. Excluding the User from the Design Process

Have you ever had the following happen to you?
You've been assigned the task of designing and producing a corporate wide in-house application tailored specifically to your organization. You've spent hours fact finding and documenting pages of requirements. You cost out the project, assign tasks, and do it.
Three months later you present your working model only to gain the following user feedback:

- "That's not what we wanted".
- "The requirements have changed".
- "Good, but...".
- "Err...what application?" (the original user contact left the company!!!!).

The ultimate judge of your application's quality will be the user. By definition, they are the ones who will be using your application (and trying to abuse it in many cases). Many programmers create applications that are masterpieces in and of themselves, but nonetheless fail to meet expectations. This is often due to one or more "misunderstandings".
Misunderstandings come about when you divorce the user from the design process. When creating an application, always keep your users in mind. Always bear in mind what they want you to do and how the application is meant to achieve that desired goal. Most importantly, however, stay in contact with them by making time for:

- Continuous User Feedback
- Prototyping
- Beta Testing

Continuous User Feedback

As Benjamin Franklin wrote in Poor Richard's almanac "A stitch in time saves nine." The same holds true with applications. If you want to save time in the long run it is important to have constant user feedback, having the user tell you whether the

Free PDF Ebook at http://www.acrobatplanet.com

application is useful. What do they like? What don't they like? What would make the application better?

Prototyping

Prototyping is a structured process of testing and soliciting user feedback over the course of an application's development. Testing a web application under prototyping, then, is an ongoing phase. You begin this process by defining a user test group.
Test the application against user requirements, but be sure to solicit direct feedback as well. Many programmers make the mistake of only testing the application after it has been completed. This can be a recipe for failure, as there often will be discrepancies between what the user really wants and what you have put together. Moreover, users get a better appreciation for what they really want when they see some tangible example. In short, user requirements cannot be written in stone (even though that's what every programmer wants).
I suggest that you define milestones over the course of your application's development. At the end of every milestone ponder over the following:

- Does the work that you've done provide benefits to the user?
- What would they like to see in the application that isn't already there?
- Would they use this functionality that you've added?
- What is the net gain for the application?
- Do your improvements make it better or worse?

When should milestones be set?

Usually it is a good practice to define your milestones whenever (another) significant portion of the UI (user interface) has been completed.
For example, I often place my first milestone when the interface to the application is first completed. This is the point when the designers have a basic template of the site laid out. The next UI test could be when you've gotten a very basic demo of the applications functionality to work.
Thereafter, I would test the UI after the completion of every application "module" or "component". A module or component might be something like the user management system of an application, or perhaps a site search engine. At these milestones, I would take my initial test group (as well as any new testers that

Free PDF Ebook at http://www.acrobatplanet.com

might be especially relevant to what I'm currently testing), and have them go over the original questions I proposed. Doing so enables you to see the "entire" effect of the modifications you've just made. You can define additional sets of questions at every milestone or stick with a standard set.

Beta Testing

This is a common form of testing that has the flavor of prototyping, but is often left to the very end of the process. Sample customers are given the opportunity to test out the application and submit their comments and bug reports. It is not as interactive as prototyping and really should be undertaken much earlier in the process. Nonetheless, it is a necessary task that developers should undertake before going live with their applications.

Free PDF Ebook at http://www.acrobatplanet.com

4. Hacking at Design Flaws

You create an application and then realize that certain things were not done as well as they should have been. Hacking at design flaws is when you put in temporary patches instead of addressing the flaw's underlying, more serious problem.
When you commit this type of mistake, it can lead to working code with serious flaws that will compromise both the speed and security of your application.

Design Flaw Indicators

Naturally, when you initially plan your project, you think that you're going about things in the right manner. You might not realize that you have taken a wrong path until you have already gotten into the specifics of the application's design (or within its parts thereof).
Here are two indicators that your project plan has gone astray:

- **Excessive Kludging:** You are "kludging" the code. A "kludge" is a solution that makes the code work, but doesn't fit into the design of your program. Alternatively, it might not be the most optimal solution, but it is the best solution you can get within your current design.

- **Over-Complicated Solutions.** You find yourself doing complex operations to implement simple systems. Take the following example that uses a **for** loop to print out a string:

```
<?php

$GeorgeBush = "If  you look at the trends, I think you'll see that most of our imports come
from other countries";

for ($idx = 0; $idx < strlen($GeorgeBush); $idx++)
{
   print $GeorgeBush[$idx];
}
?>
```

The above code is well written; it loops through a string and prints out a quote from George Bush. It has proper indentation and correct syntax. Nevertheless, the same purpose could have been achieved by simple using a print statement on the entire string.

Free PDF Ebook at http://www.acrobatplanet.com

Correcting Design Flaws

When you realize that your program has been written with faults or with an non-optimal design, the necessary corrective steps can range anywhere from leaving your program as it is, to modifying only *parts* of the program design, all the way to re-designing the entire application.
In most cases, it is a good idea to have someone independent of your program's development, review it and evaluate what needs to be done.
Let's look at three categories of mistakes:

- **Small, localized design flaws:** Sometimes the flaws in your program design may not be that critical. Small flaws won't break the farm and aren't worth the time and money required to re-implement the flawed design.

  Corrective Action: In this case, you should take note of the flaw or the hack for further inspection. In the event that you decide to redesign and implement your application in the future, you can implement any changes at that time. An example of this might be using the incorrect type of data-structure for a particular part of your program (a numerically indexed array where an associative array would be appropriate, or a stack where a tree would be appropriate.)

- **Significant, localized design flaws:** In other cases, it turns out that only part of your application really needs to be re-designed. Perhaps you're creating an Operating System, and your window manager has a bunch of hacks in it, but the underlying OS code is fine.

  Corrective Action: All you'll really need to re-design is the Window Manager (not the entire OS). This is probably the most common case, where parts of the design are flawed, but the overall structure of the design is fine.

- **Significant, global design flaws:** Under the most extreme case, the application's infrastructure design itself is flawed.

  Corrective Action: When the infrastructure is flawed, this usually requires a complete re-organization of the code and how the different parts of your program interact. This is by far the most complex and time-consuming case, and is rarely done with projects that have progressed far enough that hacks become necessary. An example of this might be to use flat files to store all the information for a major search engine like Yahoo.

Free PDF Ebook at http://www.acrobatplanet.com

5. Not Doing a Code Review

The idea for this article series stemmed from a code review that I conducted for a friend. When reading his code I was able to cut the number of variables by one third, giving a 400% speed increase in database access time. Moreover, I cut the number of lines of code in half as well as other improvements which lead to a whopping 1000% overall speed increase (10 times faster).
What's the moral of this story? Having another experienced programmer go over your code with a fine-toothed comb will greatly increase the quality, speed and security of your code. They will find bugs that you did not even realized existed and will identify easier ways of doing things. Furthermore, they will identify areas of your code that are slowing PHP down or that could be causing major security flaws.
One of the reasons that PHP, as an open source language, makes terrific business sense, is that other developers can review code that has been committed to the PHP source. Thousands of free code reviews take place examining the PHP source code for flaws, potential crashes and leaks, compatibility breaks and speed. Therefore, by the time that a new PHP version has been released, at least 2 or 3 expert programmers have looked at committed source code.
Ideally, scripts for a medium-sized/large project should be reviewed by at least two different programmers who were not involved with creating the source code. As with writing, fresh perspectives to a snippet of code can only be helpful. In most cases, however, you can get away with having one expert programmer reviewing the code.
A qualified reviewer is one who can quickly assess the workings of your code and provide constructive suggestions for both code content *and* implementation.
Often, it is helpful to draft a short questionnaire for the reviewer. Some examples that I have found particular helpful include:

- What is the purpose of XXX code?
- How does XXX file relate to the rest of the project?
- What is the program's standard error checking mechanism?
- Can we trace the actions of a standard user working with this program?
- Where might the user encounter errors?

Free PDF Ebook at http://www.acrobatplanet.com

6. Not Having Project Code Guidelines

Once, when I was just starting programming, I worked on a basic project (in Perl) with three other programmers. As I was young (and I wasn't the lead developer on the project), we didn't have any coding guidelines for the project. We were each assigned a portion of the project and went off separately to do our part. When we finally got together to implement the final product (tie all the different components together), each of the different parts of the project looked completely different.
For example, one of the programmers I worked with preferred the **studlyCaps** style of naming functions and variables. I, myself, preferred using underscores to delineate my functions and variables. The project leader had an even more interesting programming style. Depending on his mood he'd either use **studlyCaps** or underscores to delineate words in variable and function names (which actually caused some namespace conflicts which were a headache in and of themselves).
In short, the project was a mess. It required around 20 hours more than it should've taken, if we'd taken the time to plan it out correctly and set program wide coding guidelines.
Code guidelines are a way of defining the structure and appearance of your code, they describe the method and the style you should use when implementing your project.
Every project should follow a set of guidelines. This would include anywhere from general issues such as how to divide the source code (i.e., file structure) to more specific issues such as the exact variable naming sequences (i.e., prefixes and suffixes, upper-casing global variables).
Guidelines define a standard set of conventions that should be followed regardless of the programmers individual style. They can serve to define all of the unknowns in your code such as:

- Which variables are global and how they should be marked as global.
- The document structure, meaning when a certain file should be placed in the **lib** directory as opposed to the **src** directory.
- Commenting style and conventions.
- Documentation procedures.
- Line width.

Free PDF Ebook at http://www.acrobatplanet.com

Guidelines should be written up in a formal document and adhered to by every programmer in the project. When the project is finished, this document would be stored for future reference.

A Sample Project Guideline Document

Let's create a very short sample guidelines document. We won't go into the specifics, but rather create a skeletal outline. It will have all the required elements but will not go into detail. An actual guidelines document could easily run anywhere from 10-40 pages. Note that you would not have to recreate these documents for every project. The template could always be modified a little bit when necessary.

Style Guidelines for DesignMultimedia.com

Here's an example of a Style Guidelines outline. Once you have this document completed, you can focus solely on implementing your project. You should not have to worry about inconsistent source code, naming conventions or site organization when you implement your site.

Introduction

This document contains the following information:

- File Structure
- Page Headers and Footers
- Code Documentation
- Commenting Style, Comment Meanings and Definitions
- Implementation Guidelines
- Variable Naming

File Structure

The DesignMultimedia.com application has the following structure:
<A description of how the files for your application should be stored (i.e., what goes where), as well as the naming conventions for each file.>
Explanation of the structure:

Free PDF Ebook at http://www.acrobatplanet.com

<Explain each of the individual conventions so there are no misunderstandings as to what, exactly, you mean.>

Page Headers and Footers

Every page in the program should have the following header:
<A page header would go here. It could be anything from some code snippet to a simple copyright.>
For example, all the .c and .h files in PHP4 have the following standard header:

```
/*
+----------------------------------------------------------------------+
| PHP version 4.0                                                      |
+----------------------------------------------------------------------+
| Copyright (c) 1997, 1998, 1999, 2000 The PHP Group                   |
+----------------------------------------------------------------------+
| This source file is subject to version 2.02 of the PHP license,      |
| that is bundled with this package in the file LICENSE, and is        |
| available at through the world-wide-web at                           |
| http://www.php.net/license/2_02.txt.                                 |
| If you did not receive a copy of the PHP license and are unable      |
| obtain it through the world-wide-web, please send a note to          |
| license@php.net so we can mail you a copy immediately.               |
+----------------------------------------------------------------------+
| Authors: Sterling Hughes                                             |
+----------------------------------------------------------------------+
*/
```

And the following footer:
<Here you would put your sites standard footer, for example the following is found at the bottom of all the .c and .h files in PHP):>

```
/*
* Local variables:
* tab-width: 4
* c-basic-offset: 4
* End:
*/
```

Explanation if necessary:
<Explain the conventions for the headers and footers, and perhaps why they're required (depending on the content of your header and footer).>

Code Documentation

Free PDF Ebook at http://www.acrobatplanet.com

<Here is where you lay out the specifics for how code will be presented in the application, whether it is by using the javadoc documentation style or XML documentation using the Docbook stylesheets.>

Commenting Style

<This describes the different types of comments used in addition to defining what any abbreviations or "phrases" mean.>

Implementation Guidelines

<This is the site's do's and don'ts: For example, don't keep different classes in the same file or do keep every class in its own file.>

Variable Naming

<For example, you can write in the following:>
This site follows the following naming conventions:
[1] All classes are of mixed case.
[2] All functions are lowercase with words separated by a underscore (_).
[3] More rules on the specific naming conventions.

Free PDF Ebook at http://www.acrobatplanet.com

7. Cut and Paste Coding: The Wrong Way

I have seen novice programmers copy code that, for example, validates an e-mail address, sends an e-mail, and takes the values of a form to build an e-mail message. They would paste it all into their program with the result being a working mess of code statements that insecurely sends a form.
While the code would work under optimal conditions, it would fail under any real test for good code. The patchwork job would ***not*** be:

- **Extendible**: The code will look as if it were bits of unrelated snippets slapped together. Ask an experienced programmer to modify the script and they'll prefer to re-write the entire thing. Un-readable code is un-extendible code.

- **Secure**: You would be placing other people's code into your scripts without fully understanding what the code does. Think about this. What if the code you copied had a bogus system call, which removed all the files from your hard drive? Moreover, the same code would not always be secure on different systems and configurations. Lastly, your program would be vulnerable to bugs inherent in other people's code.

- **Fast**: When code is pasted together it probably isn't very fast, since it doesn't have a logical progression - the most important thing when it comes to creating fast scripts.

Doing It the Right Way: Learn first, Then Copy

Study another programmer's code thoroughly before copying. Analyze what was done. Only when the code is readable, consistent with your program's logic, and free of errors, should it be considered as a candidate for copying. Integrating the code at this point will enable you to fit it with the rest of your script more smoothly.

Libraries are Fine

Only use PHP libraries from a trusted source such as PEAR, or the PHP Classes Repository. For pre-packaged API's, it is fine to use the extra functionality these API's can provide your program. In fact, if you can find a pre-written library from a trusted source, it is most often better to use those libraries in your code.

Summary

Free PDF Ebook at http://www.acrobatplanet.com

- **Cut and Paste Coding - The Wrong Way:** It's generally a bad idea to "cut and paste" other people's code into your application. It's fine if you take the basic ideas and algorithms from their code and integrate them into your application. Alternatively, it's o.k. to use their code as "libraries". But do not **blindly** copy people's code.

- **Not Having Project Code Guidelines:** Having coding guidelines is essential to any project. They can ensure that all of the code conforms to a certain style, organization and documentation. This helps to avert future misunderstandings.

- **Not Doing a Code Review:** Always have someone else look over your code. Another set of eyes can often spot out things like non-optimized SQL, incorrect coding practices, unnecessarily complex PHP code, security holes and other flaws.

- **Hacking at Design Flaws:** Don't hack at any significant design flaw. If you see that you've designed something incorrectly (or not at all), you will generally be better off re-designing and re-implementing it. You'll be grateful in the long run.

- **Excluding the User from the Design Process:** Never exclude the user from the design process. After all is said and done, they will be the ultimate judges of your application's usefulness. Involving them over the course of the application's development will avert future misunderstandings.

- **Not Sticking to a Project Plan:** Do not "make it up as you go along". Allocate time for application planning. An example of a basic project plan would include a Requirements Analysis Phase, a Program Design Phase and a Testing Phase.

- **Getting Lost in Time:** Allocate enough time to your projects! Having to rush to a deadline will only increase the likelihood of you committing any of the previous 20 mistakes. Don't find yourself lost in time.

About The Author

Sterling Hughes is a software developer for Pentap Technologies AG. He's been developing dynamic Web applications for the past six years for companies such as Sarah Lee, UDV and Marriott. He is the author of The PHP Developer's Cookbook, as well as one of the contributors of the PHP Documentation. He has written PHP's

Free PDF Ebook at http://www.acrobatplanet.com

SWF, CURL, Sablotron and Bzip2 extensions and is one of the authors of the Sockets extension. You can reach him at sterling@php.net.

Free PDF Ebook at http://www.acrobatplanet.com