CHAPTER 1

Managing Projects with FogBugz

Communication is the lifeblood of a software project. Whether you're building an application for commercial sale or developing it for internal corporate use, there's no way that a software project can be successful without customer feedback. Communication within the team that's building the application is equally important. Developers, testers, and managers all need to coordinate their activities with each other to make sure the product gets out the door on time and with the right features.

Software teams manage this communication with a variety of tools and technologies: whiteboards, e-mails, wikis, phone calls, sticky notes, instant messages, hallway meetings, formal review sessions, and more. But there's a danger to using too many tools to manage a software project: the more places you have to store information and the more ways you have to pass it around, the easier it is for important messages to fall through the cracks. In this book, I'll show you how to use one tool—Fog Creek Software's FogBugz 6.0—to collect and manage all of the communication between users, managers, developers, and testers. With FogBugz in place, you'll spend less time hunting for information and trying to remember who was doing what, and have more time to finish the product on time and within budget.

FogBugz from the Mountain Top

As you work through this book, you'll learn all the nitty-gritty details of working with FogBugz. But before setting out on this journey, it's good to know where you're going. FogBugz came from some simple notions of what a bug-tracking tool should do. From those roots, though, it's grown into a robust project management tool. You may also need some other tools (for example, something to graphically display the project schedule and dependencies), but FogBugz can form the core of a successful project management strategy for most software projects.

Note The early versions of FogBugz did indeed concentrate on tracking bugs—hence the name of the product. But the more robust capabilities of FogBugz 6.0 move it beyond the bug-tracking category to make it more of a project-tracking tool. It's too late to rename the product, though.

Understanding the FogBugz Philosophy

FogBugz is based on two core principles:

- Track as much product-related communication as possible in a single tool.
- Keep everything as simple as possible (but no simpler!).

By sticking to these principles, Fog Creek has been able to deliver a tool that can be installed in minutes and that the average developer can start using immediately. Unlike some products in the field, FogBugz does not let you customize everything. Excessive flexibility can lead to an organizational paralysis while people debate which bug statuses to use, how to organize workflow, and so on. In contrast, FogBugz lets you customize the things that really vary between organizations (like the name of the project that you're working on), while delivering a robust set of core features that just work.

■ **Note** For more information on installing FogBugz, see the Appendix.

Surveying FogBugz

FogBugz is a client-server system with a Web client. The information that you store in FogBugz is tracked in a database and presented through a series of scripted pages on a Web server (depending on your choice of platform, the scripting language is either ASP or PHP). To the users, this means a FogBugz installation looks like any other Web site. You can interact with FogBugz through any modern Web browser, including Internet Explorer, Firefox, Safari, or Opera. You can even use a mobile device such as a PocketPC or a SmartPhone to work with FogBugz (though you may find using the small screen challenging).

The features of FogBugz break up into five categories:

- Case tracking
- Project scheduling
- Documentation management
- E-mail management
- Discussion group management

I'll briefly discuss each of these areas in turn.

Case Tracking

Cases are the key unit in the database that FogBugz maintains. Each case is assigned to one of three categories:

- *Features*: New functionality to be added to the product
- *Bugs*: Existing functionality that doesn't work right
- *Inquiries*: Questions from customers or other stakeholders

Figure 1-1 shows a typical case (it happens to be a bug) in FogBugz. As you can see, each case is characterized by a variety of properties, including its priority, the release that needs to include the fix, and the history of work on the case.

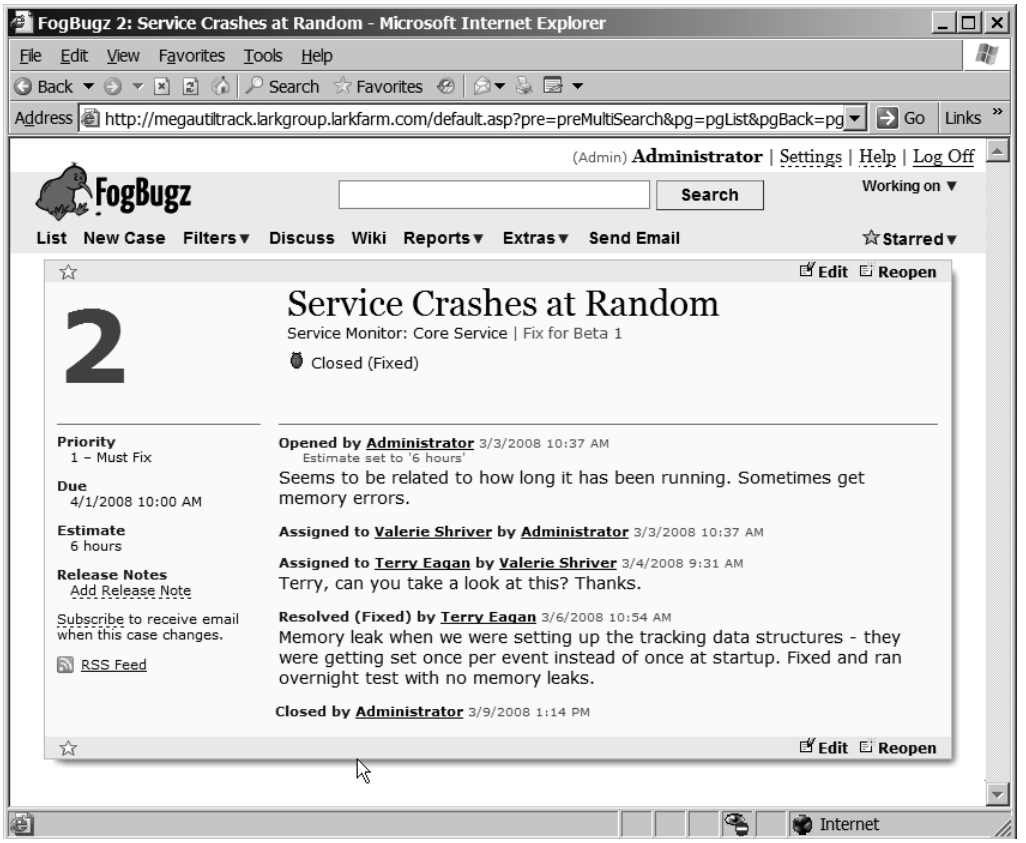

Figure 1-1. *A typical case in FogBugz*

Cases can be entered manually by any licensed user of FogBugz. They can also be created in several other ways. For example, cases can be automatically created from e-mail received at a specific address, created by a site administrator based on a discussion group thread, or even automatically added to the database by special code in an application that you've already shipped.

FogBugz lets users filter cases to find only the set they're interested in at the moment (for example, all open cases assigned to you that are due for the next release). You can use the FogBugz Web interface to adjust the properties of a case or assign an estimate to it. When you're done fixing a bug, implementing a feature, or handling an inquiry, you can mark it resolved. This automatically assigns the case back to its originator, who can close it (thus removing it from the list of active cases).

■ **Tip** Only the originator of a case can close it. If you can't convince the person who spotted a bug that it's fixed, then it's not fixed.

FogBugz also offers other features related to case management. If you're interested in a particular case, you can subscribe to it so as to receive e-mail notification whenever the case is changed. You can also subscribe to RSS feeds that provide an overall view of case activity. FogBugz integrates with a number of popular source code control systems so that you can track which code fixes are related to which bugs. You can even create a set of release notes automatically from the cases that were fixed for a particular release.

■ **Note** You'll learn more about managing FogBugz cases in Chapter 2.

Project Scheduling

FogBugz 6.0 implements a new *evidence-based scheduling* system. This system of project scheduling is based on two simple principles:

- Because most software projects contain some uncertainty, and estimates cannot be exact, it is impossible to calculate an exact delivery date before the software is actually delivered.
- The best way to judge how accurate a particular developer's estimates are is to look at how accurate their estimates have been in the past.

The more information it has, the better FogBugz can predict schedules for you. As you assign features and bugs to developers, they should enter initial estimates, predicting how long it will take to do the work involved. As they do the work, developers can use the FogBugz timesheet to enter actual elapsed time. These two pieces of data provide enough information for FogBugz to judge the accuracy of estimates from individual developers.

The other piece of information you need to give FogBugz (in order to get accurate estimates) is a list of the tasks involved in a project. This includes not only the features involved in the product, but also tasks such as integration and debugging, which can be entered into FogBugz as *schedule items* and estimated separately.

With information on the work to be done and the accuracy of estimates, FogBugz will produce and track status reports for your project. A key feature of these status reports is the Ship Date graph, shown in Figure 1-2. As you can see, FogBugz presents information on ship dates as a range of possibilities, rather than as a single hard-and-fast date.

■ **Note** Chapter 4 covers evidence-based scheduling in detail.

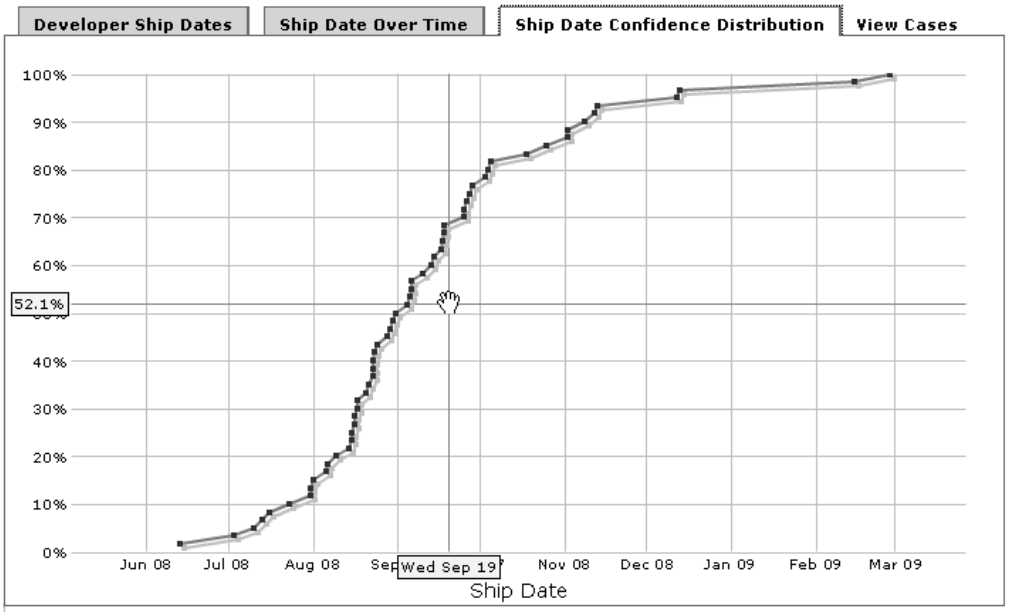

Figure 1-2. *The Ship Date graph*

Documentation Management

Development projects tend to generate a number of different types of documentation. To name just a few of the more obvious documentation artifacts, a typical project might generate

- Specifications
- Procedural documentation
- Status reports
- Release notes
- Support articles

To handle your documentation needs, FogBugz includes a complete wiki system. A wiki lets you build and edit documentation directly in your Web browser, with complete control over content and formatting. FogBugz lets you create both internal wikis (for use by those with accounts on your FogBugz server) and externally accessible wikis (that your clients and customers can contribute to). You can use a FogBugz wiki to create and polish documentation while a project is underway, publish it at the conclusion of the project, and maintain it after a product has been released.

■ **Note** You'll learn more about the wiki and managing documentation in Chapter 5.

E-Mail Management

FogBugz also helps you manage incoming product-related e-mail from your customers. This isn't a substitute for your existing e-mail server, but a way to handle e-mail sent to specific addresses. For example, you might use CustServ@megautil.com as a general customer service e-mail address, and ServMonBugs@megautil.com as an address to accept bug reports on your Service Monitor application.

You can set up FogBugz to monitor any number of POP3 mailboxes for incoming mail. When mail arrives, FogBugz applies a series of rules to sort it appropriately. First, spam is automatically discarded. For other messages, you have a choice of manual sorting or autosorting. If you choose to manually sort messages, FogBugz will create a new case in the project of your choice for each incoming message. Autosort is much more sophisticated. You can create a set of categories, and autosort will learn by example which messages belong in which category. With autosort, you start by moving messages manually, but FogBugz soon takes over the job, creating new cases in categories just as you would have done yourself.

Customers who send e-mail to a properly configured FogBugz POP3 address will get an automatic reply return, with a URL where they can check the progress of their case in the system. Any member of the development team can respond to e-mail messages and see the whole history of communications with the user when doing so. The system can automatically assign a due date to make sure that customers get replies in a timely fashion. To make it easier to generate those replies, you can also create predefined text snippets that can be inserted into a return e-mail with just a few keystrokes.

Discussion Group Management

E-mail is good for one-on-one communication, but there are times when a conversation benefits from wider input. For example, you might have a group of developers and testers who want to discuss how a particular feature should work, or a group of customers with feedback and suggestions for future versions of the application. To handle this sort of many-to-many communication, FogBugz includes support for online discussion groups.

FogBugz discussion groups are simple. You can set up any number of groups on your server and give them each a distinct name. Each group contains threads, and each thread contains messages. Messages are presented as a chronologically ordered list, without any branching; this makes it easy to catch up with any conversation by starting wherever you left off.

The discussion group implementation includes anti-spam technology to prevent junk from cluttering up the real discussion, and moderation to help weed out disruptive messages or unruly users. You can customize the appearance of a discussion group so that it fits in with your corporate Web site. A single button click will turn a discussion group message into a case, so that problems and suggestions reported via discussion group don't get lost.

Getting Down to Business

To get a better understanding for how FogBugz can help with your development cycle, let's follow a couple of typical cases from start to finish.

For these examples, I'll introduce MegaUtilities Corporation, a fictional company that writes and (with any luck) sells software utilities for the Microsoft Windows market. Their products include Network Enumerator (a general-purpose network browser), ScriptGen (an automated generator for command scripts), and Service Monitor (a Windows service that

monitors the event log and sends e-mail when it recognizes a particular message). MegaUtilities has a comparatively small staff: two administrators, a single project manager, a customer service representative, four developers, and three testers.

Moving a Bug Through the System

During the beta period for Service Monitor, Robert Evers (who is the company's customer service rep) logs on to FogBugz and, as part of his daily duties, reviews the new postings to the Service Monitor discussion group. He finds the new thread shown in Figure 1-3 (fortunately, there are lots of other threads from happy customers, so the beta isn't a complete disaster).

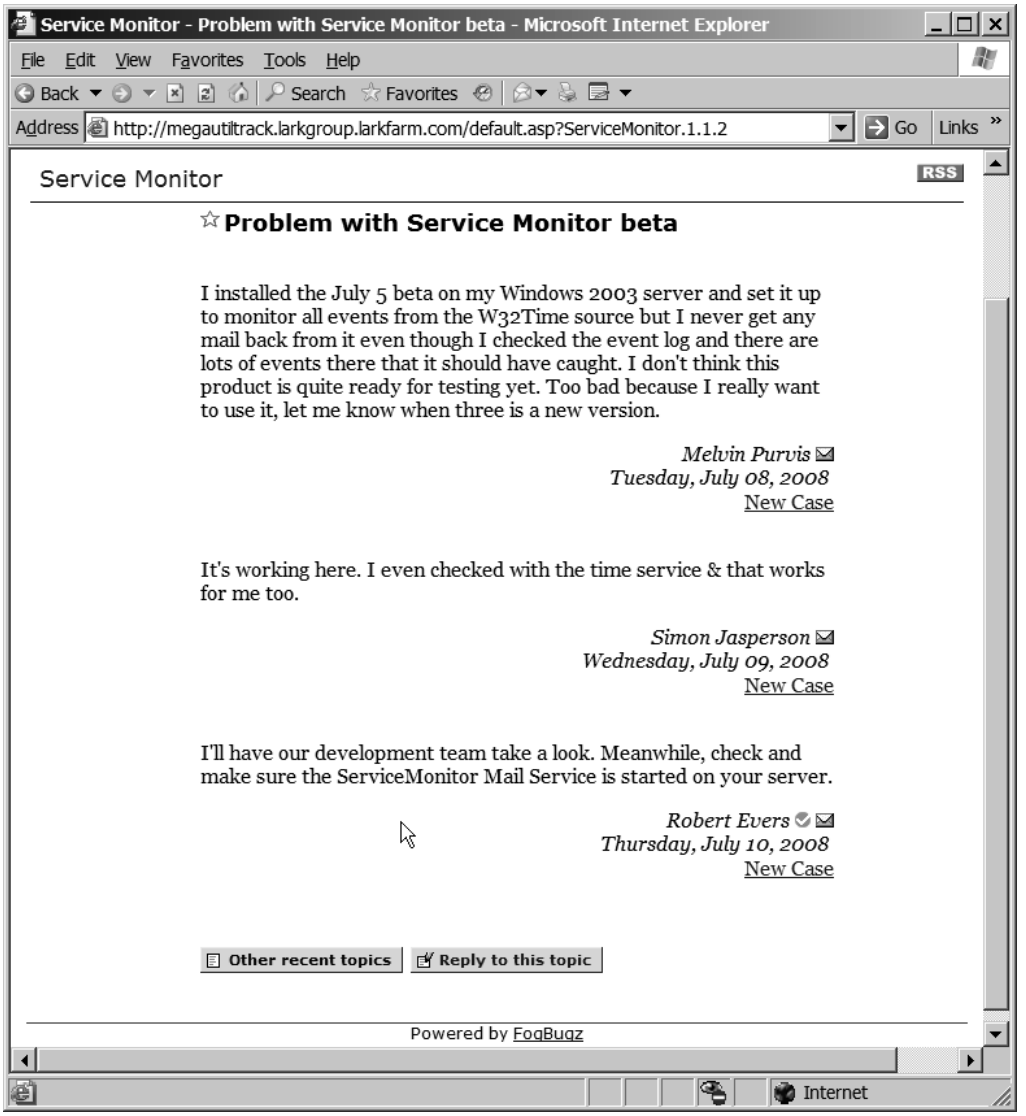

Figure 1-3. *A FogBugz discussion thread*

MegaUtilities has the sensible policy of never ignoring customer complaints, no matter how outlandish or ungrammatical they are. Even though another customer has already replied to the first poster, Robert uses the New Case hyperlink (which only appears because he's logged in as a user on the server) to create a new FogBugz case to track this particular bug. FogBugz automatically grabs the title and description from the discussion group posting, so all Robert has to do is fill in other information and click OK to create the case.

■ **Tip** This example shows in a small way one of the benefits of discussion groups: other customers will do some of your support for you. The more effort you expend in building a broad base of users on your discussion groups, the more chance there is that frequent posters will emerge and answer questions before your own paid staff can even read them.

Robert chooses the Service Monitor project, and FogBugz automatically assigns the bug to the project lead. Because this particular bug is a direct failure of the core functionality of the product, he marks it as a priority 1 bug for the 1.0 release version of the product. Robert then returns to the discussion group thread, clicks the Reply To This Topic button, and types a reply to let the original poster know that his problem is being looked at.

Meanwhile, FogBugz itself has not been idle. As soon as the new case gets created and assigned to the project lead, Valerie Shriver, FogBugz sends her e-mail to tell her that there's something new on her plate. Because Valerie is a Type A personality who always keeps her e-mail running, she gets this notification in short order. Clicking the link in the e-mail takes her directly to the case. Although she doubts that even a beta could get out the door if it wasn't working at all (and she knows that it's happily monitoring events in the company lab in any case), Valerie also knows that she can't just ignore a prospective customer. Fortunately, she has development staff to take care of these things for her. So she clicks the Assign button, assigns the bug to Paige Nagel, and adds a comment guessing at the cause of the bug.

■ **Tip** When you want to assign a case to another user, choose the Assign button rather than the Edit button. While either one will move the case over and allow you to add comments, only the Assign button will automatically send an e-mail notification as well.

Of course, FogBugz e-mails Paige to tell her the bug is on her plate now. Paige doesn't see how this bug can be happening either, but she dutifully sets up Service Monitor on one of the lab machines and tells it to monitor for W32Time events. She deliberately assigns a bogus time server to the machine so that events will end up in the event log. Sure enough, she gets the notification e-mails just as she should. Paige has other things to do, and this one really looks like pilot error to her, so she clicks the Resolve button. She chooses "Resolved (Not Reproducible)" as the status and saves her changes.

But remember, resolving a bug doesn't get rid of it. Instead, it goes back to Robert, who was the one who entered the case into the system in the first place. Robert feels strongly about protecting his customers, and he's not going to take "not reproducible" as a resolution without a fight. He thinks about some of the issues they saw when alpha testing, looks at the

customer's e-mail address, and comes to his own conclusion about the possible cause of the bug. So he reactivates the bug, adds a comment, and shoots it back over to Paige.

This time Paige grumbles a bit about the added effort to set up a good repro case (after all, she has new features to implement, not just bugs to fix!), but she gets to work. A few minutes later she has a test Hotmail account of her own, and a few minutes after that, she verifies that she can reproduce the problem. This puts her 90% of the way to fixing it. She changes the format of the message so that it looks less "spammy" and marks the bug as "Resolved (Fixed)." This bounces it back to Robert again.

Robert uses the private e-mail feature of the discussion group to send a message to the original poster, asking him to check his Hotmail spam folder for the missing messages. Sure enough, there they are, and Robert logs back into FogBugz to close the case. Figure 1-4 shows the final state of the closed case. As you can see, it preserves the entire history and lets you see just what happened, beginning with the original report.

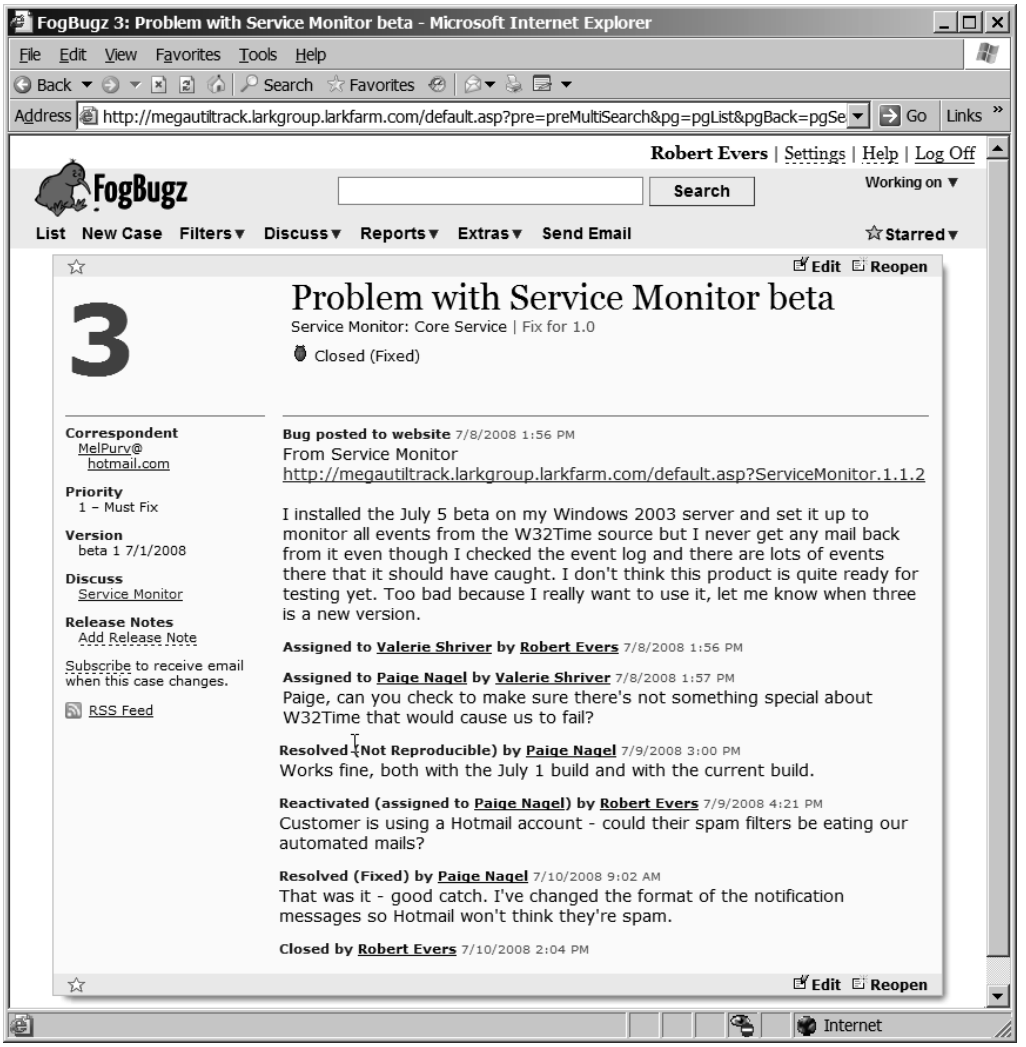

Figure 1-4. *A closed case in FogBugz*

Responding to a Customer Inquiry

My second example highlights the e-mail management features of FogBugz. This time I'll follow what happens when a customer inquiry arrives by e-mail. Simon Jasperson, who happens to be a long-time user of MegaUtilities products, had planned to be an enthusiastic tester of Service Monitor. But when he tried to install the product, the installation failed. From reading the release notes with the beta, he knows that he can send e-mail to CustServ@megautil.com describing his problem, so he does so.

Back at MegaUtilities, Randy Wilson happens to be logged on to the FogBugz server when he notices a new case in the Inbox project. This is a general-purpose project that exists to handle incoming mail that isn't automatically assigned. Any FogBugz user can review and deal with incoming mail, so Randy clicks through to the message. It looks like a legitimate bug to him, so he moves it over to the Service Monitor project and lets FogBugz assign it to the primary contact, Valerie Shriver.

Meanwhile, FogBugz doesn't forget about the customer. Simon gets back an automatically generated e-mail that not only tells him his message has been received, but gives him a URL for tracking what's going on with it. Clicking through to the URL gives him a read-only view of the case, as shown in Figure 1-5.

Figure 1-5. *Customer view of an open case*

The bug proceeds through the usual process at MegaUtilities. Valerie adds a due date to the bug and assigns it over to Terry Eagan, another of her developers. Terry is pretty swamped right now, but she opens the bug and puts in an initial estimate of 6 hours to solve the problem. All of these changes continue to be reflected on the status page that Simon can check out, assuring him that the company is working on his problem.

In a few days, Terry has time to track down the actual problem, and fixes the code so that it works—at least on her test machine. She marks the bug as fixed, and FogBugz assigns it to the FogBugz administrator (because it can't be assigned to a customer). The administrator later signs on to look at the bug on behalf of the customer. Seeing that the bug is fixed, she clicks the Reply button in the bug's header, which automatically composes an e-mail back to the customer, as shown in Figure 1-6. The administrator uses this e-mail to close the loop back to the original customer, letting him know that the bug is fixed in the latest build, and then closes the bug.

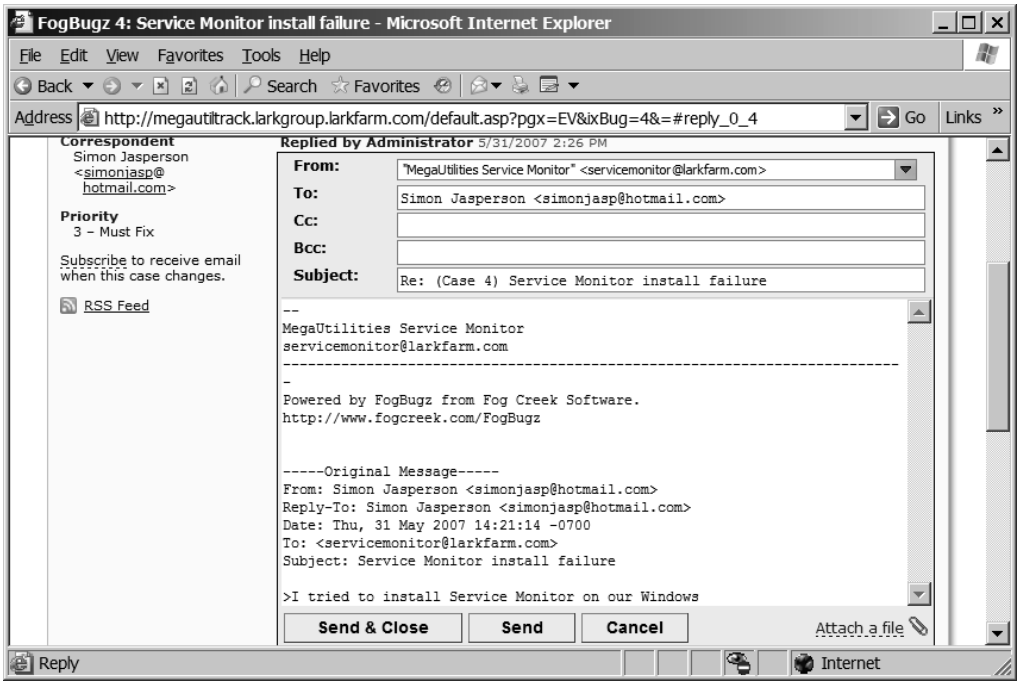

Figure 1-6. *Sending e-mail back to a customer*

Making Effective Use of FogBugz

FogBugz is a great program, but it isn't miraculous. No amount of management technology will make your software projects successful all by itself. FogBugz can help by making it easy to track and resolve issues, and by making sure that communication with the all-important customer doesn't get ignored. But you also need to do some work on the social level, to make sure that FogBugz gets used, and used well.

Bringing FogBugz into Your Company

The first hurdle to using FogBugz is to get it used at all. Here, the first few cases may be the hardest. After people see how easy FogBugz is to use, and how much it helps them work, they'll probably start using it on their own. So how do you jump-start the process?

If you sign the paychecks, you can just tell everyone to use the new bug-tracking system. But this may not be the most effective option. Telling programmers what to do has frequently been referred to as "herding cats," and it is just about as effective. Instead, start using FogBugz yourself. Put in some bugs or feature requests (if you don't know enough about your own company's products to make sensible feature requests, why are you in charge?). Use the wiki to hold the specs for the next project coming down the pike. Then you can suggest to people that they'd better keep an eye on the system if they don't want to lose your valuable input. This will usually get through.

If you're a manager, and nobody seems to be using FogBugz, start assigning new features to your team using FogBugz. Eventually they'll realize that instead of coming into your office every few days saying "What should I do next?" they can just see what's assigned to them in FogBugz. If you're using an agile process where new features get assigned during a consensual meeting, have someone enter the features into FogBugz right there at the meeting.

■ **Tip** Use the new Add Case link in list view to quickly add features during a meeting.

If you're a developer, and you're having trouble getting testers to use FogBugz, just don't accept bug reports by any other method. If your testers are used to sending you e-mail with bug reports, just bounce the e-mails back to them with a brief message: "Please put this in the bug database. I can't keep track of e-mail." If you're a developer, and only some of your colleagues use FogBugz, just start assigning them bugs. Eventually they'll get the hint.

If you're a tester, and you're having trouble getting programmers to use FogBugz, just don't tell them about bugs—put them in the database and let the database e-mail them. This can be especially effective if you can also convince the manager for the project to subscribe to the RSS feed for the bugs. Most developers have at least enough political savvy to want to stay as informed as their boss.

Writing Good Bug Reports

It's not enough to get everyone in the company using FogBugz. You also need to get them using it *well*. The key here is to teach people to write good bug reports. For starters, every bug report should contain these three essential pieces of information:

- How to make the bug happen
- What happened
- What should have happened

Put that way, it looks easy, right? But if you've ever spent time trying to actually respond to bug reports, you know that writing good bug reports is harder than it looks. People post all

sorts of crazy things to bug databases (and that's not even counting spam, which FogBugz fortunately does a good job of weeding out before it gets to you).

When you think you've found a bug, the first step is to record the information about how to reproduce the bug. The key problem here is to include just the necessary information. Determining what's necessary is an art. What you had for breakfast this morning probably doesn't have a bearing on the bug (unless you're testing dietary analysis software). The operating system on your computer and the amount of RAM could have a bearing. The last action you took in the application almost certainly does make a difference.

Whatever information you can collect, please organize it sensibly. Programmers tend to be focused on careful instructions, so step-by-step details are very useful. Figure 1-7 shows a bug with a good set of reproduction steps.

Figure 1-7. *A bug report to make a developer happy*

Sometimes it's just not possible to come up with good repro steps. Perhaps you used the application for quite a while and it suddenly crashed, and you don't remember what you were doing. In that case, write down everything you remember. Perhaps you thought you had the steps, but they don't always make the bug happen. In that case, record the steps, but please also tell the developer that the bug is intermittent. Otherwise, it'll probably just get closed as not reproducible.

The second piece of information you need to supply is what happened that made you think this was a bug. Did the program destroy all the files on your hard drive? Did Windows expire in the fabled Blue Screen of Death? Did the developers just spell a word wrong on screen? This information often makes the best title for the bug as well; something like

"Shift-Enter fills hard drive with temporary files" is easy to recognize when you're just scanning down a list of titles.

Finally, tell the developer what should have happened instead. "Sausauge should be spelled sausage" will help guide the poor developer who wasn't at the top of his English class. At times you can omit this piece of information because it will be implied by the description of what happened. If you report it as a bug that the program failed to install on a particular operating system, it's a safe bet that you expected it to install. But when in doubt, be explicit.

Writing Good Feature Requests

Feature requests, too, require careful composition. When you enter a feature request into the FogBugz database, you're saying one of two things:

- "This feature is part of the spec, and now it's your job."
- "I know no one thought of this earlier, but I think it's really, really important."

Any project that requires enough work to justify using FogBugz at all should have a formal, written specification. Specs are enormously important, because they represent the shared vision of how the product should work when it's finished. Developers refer to the spec to figure out how all the pieces fit together, testers use it to figure out what to test (and to see whether a particular piece of behavior is a bug or a feature), managers use it to help schedule the project, and so on. The product's spec should be a collective, comprehensive, up-to-date vision of the way that the product will work.

■ **Note** For more detail on writing good functional specifications, see Joel Spolsky's book *Joel on Software* (Apress, 2004).

But although the spec is important, it doesn't quite get you all the way to working code. Someone has to take all of those features in the spec and assign them to individual developers to implement. Now, you (assuming you're the program manager) could do that with e-mail or notes on a whiteboard or orders shouted down the hall, but with FogBugz installed, you've got a better solution: enter them as feature requests in FogBugz. You can cut and paste the appropriate part of the spec into the feature request, and include a hyperlink to the full spec on your network or in the FogBugz wiki (you do have the spec stored on a server where everyone can read it, right?).

An added benefit of using FogBugz to assign features is that it will help you schedule the entire project. If your developers use the estimating features of FogBugz, you can look at the total amount of work left to be done, and adjust your schedule (or your feature set) as necessary.

■ **Note** For more information on estimating in FogBugz, see Chapter 4.

The other use for feature requests in FogBugz is to handle features that you didn't think of when you were designing the product. In this case, the feature request needs to contain three essential pieces of information:

- What the feature should do
- Why this is important
- Who wants the feature

For anyone to implement the feature, you need to describe what it should do. This description should be as detailed as if you were writing a spec. For example, if your feature requires a new message box to pop up, you need to supply the text that you want to see in the message box. Without this level of detail, the developer who ultimately gets assigned the feature request won't know what to build (and the testers won't know what to test, and so on).

■ **Tip** If you're the manager for a product, you should keep your specs up to date by incorporating any feature requests that are entered directly into FogBugz.

Justifying new features is particularly important. For many organizations, software development is a zero-sum game: with an announced release date, adding a new feature means throwing some other feature out (or reducing the product quality, which is usually a bad idea). You should be prepared to argue why your particular feature is essential. Does it take care of some unforeseen scenario where the application crashes or destroys data? Does it bring the product to feature parity with an important competitor? Is it something that will leapfrog all the competition and make the program sell like hotcakes?

Finally, make it clear who wants this feature. All other things being equal, a feature request from Joe Tester is less likely to meet the approval of management for this version than a feature request relayed from Mr. Megabux, your largest customer.

Keeping It Simple

Remember that I said that one of the underpinnings of FogBugz is to keep everything as simple as possible? To use the product successfully, you need to keep that rule in mind. FogBugz itself is customizable (after all, it's a set of ASP or PHP pages; there's nothing to prevent you from mucking about in the source code), but you shouldn't waste time fixing things that aren't broken. In addition to keeping the program itself simple, you also need to think about keeping your bug-tracking process simple. I'll close this chapter with a few concrete suggestions on how to do that.

On the application side, avoid the temptation to add new fields to FogBugz. Every month or so, somebody will come up with a great idea for a new field to put in the database. You get all kinds of clever ideas, for example, keeping track of the file where the bug was found; keeping track of how often the bug is reproducible; keeping track of how many times the bug occurred; keeping track of which exact versions of which DLLs were installed on the machine where the bug happened. It's very important not to give in to these ideas. If you do, your new

bug entry screen will end up with a thousand fields that you need to supply, and nobody will want to enter bug reports anymore. For the bug database to work, everybody needs to use it, and if entering bugs "formally" is too much work, people will go around the bug database. At that point, your ability to track what's going on goes out the window. When bugs are swapped by e-mail, hallway conversation, and cocktail napkin, you can't trust the estimates coming out of FogBugz, you can't search effectively for duplicate bugs, and you're generally in trouble.

On the process side, you should consider training testers to write good bug reports. Meetings between the test team and the development team can also be helpful (that is, if your organization somehow enforces distance between these two teams, which I think is a bad idea anyhow). A good tester will always try to reduce the repro steps to the minimal steps to reproduce; this is extremely helpful for the programmer who has to find the bug. Developers can also give the testers an idea of the sort of information that they find extraneous, and the sort that they find necessary, in bugs that have already been entered.

Keep track of versions on a fine-grained basis. Every build that goes off the build machine should have a unique version number. Testers need to report the version where they found a bug, and developers need to report the version where they fixed the bug. This avoids pain all around.

Everyone should be responsible for keeping the customers happy. Everyone on the team needs to at least dip into the discussion groups to see what customers are talking about, and should feel free to open cases based on discussion group comments. You may also want to have everyone review the e-mail inquiries that have ended up in the inbox project, and sort them to the correct project. This helps ensure a timely response to customers. For larger projects and teams, though, it's a better idea to have one person (or a small team of people) whose job it is to explicitly sort the incoming inquiries.

Summary

The story of every bug is a variation on this theme:

- Someone finds it and reports it.
- The bug gets bounced around from person to person until it finds the person who is really going to resolve it.
- When the bug is resolved, it goes back to the person who originally reported it for confirmation.
- If, and only if, they are satisfied with the resolution, they close the bug, and you never see it again.

In this chapter, you've seen how FogBugz enables you to work through this process with a minimum of overhead. You've also learned a little about how to use the system most effectively by writing good bug reports and feature requests, and by not cluttering the process up with unnecessary overhead.

If you've installed FogBugz, you're probably ready to dive in and start entering cases now. Great! But there's still plenty more to learn about FogBugz that might not be apparent at first. So after you get those first few cases cooking, read on. In the next chapter, I'll dig into case management in more detail, and show you some of the other tools that FogBugz has to offer.