Extracted from:

Ship It!

A Practical Guide to Successful Software Projects

Develop in a Sandbox

How do you share code with your teammates? A surprising number of teams never answer this question explicitly and instead just get a big, old shared disk drive, with all of their source code and other files lying around. Any act by any developer—from simply editing a file to compiling code—will instantly affect every other developer on the team. Their life is now filled with constant, unpleasant surprises.

It's just like a crowded kitchen on Thanksgiving, with everyone throwing something else into the mix, and it makes for a pretty frustrating work environment. While many teams continue to operate this way, you can take a safer and more professional stand. This will have a deep effect on your tools and infrastructure, so you need to get this straight right from the beginning.

There's only one fundamental rule to keep in mind: isolate others from the effect of your work until you are ready. That's why we call this *sandbox development*: every developer has their own sandbox to play in without disturbing other developers.

That may sound easy enough, especially in terms of isolating source code (see Practice 2, *Manage Assets*, on page 19), but the real trick is to remember that it applies to *all* resources: source code, database instances, web services on which you depend, and so on.

Your own development machine should be designed to contribute to your own productivity.[1] It should not contribute anything to the global build process—no one else should have to rely on your machine directly for anything.

But how do other developers get your code? Code is shared via the *repository*. Think of the repository as a big shared disk, but one that's managed by a librarian. The librarian ensures that everyone has the right version of any file (or other resource) that they need and that everyone can work without clobbering each other. Every developer uses a software tool to check in and check out files (just like a real book library) so they can work on them locally.

repository

On your own developer machine, you edit local copies of source code files, compile, build, and test in splendid isolation from your team-

[1] This means it's perfectly okay for different developers to use different code editors or even Integrated Development Environments (IDEs).

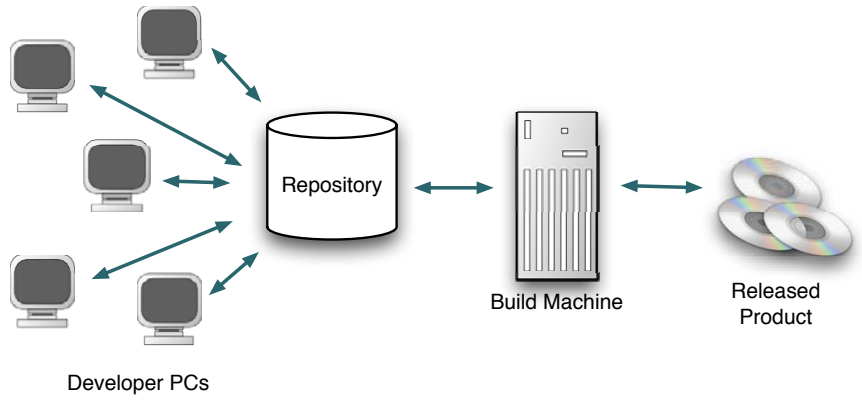


Figure 2.2: Sandbox Development Setup

mates. If you need to use a database, a web server, or any other resource while developing, make sure that you're the *only* one using it. When you're satisfied that you're finished with a piece of code, you check it back into the repository.

But then how do customers get the finished product? In addition to the developer machines and the repository, you have a *build machine*. The build machine is an unattended server that simply gets all of the latest source code from the repository, builds, and tests it, over and over again. The result of this build is the product *release*.

build machine

release

Most of the time, this release will just be thrown away after each build, but every so often this is the pile of bits that you'll ship to your customers and end users. It's built the same whether it's the usual 10:00 a.m. build or it's the final release after months of toil and sweat.

It's always consistent, because the build machine is an independent entity: it never looks at individual developer machines for any reason. The input to the build is the repository, and the output from the entire process is from a designated build machine. This system works great as long as developers don't cheat.

TIP 2

Stay in the sandbox

Joe Asks...

Where Do Releases Come From?

Your build machine may or may not be the box where you build releases—the code that you ship to your customers. However, the build box and the shipping product build box both use the same scripts, use the same repository as their source, and so on.

Some of the differences might be that a shipping build creates a new branch or tag within the repository to mark a known, released set of code, or perhaps that the shipping build wraps the code in installers for various platforms.

Sometimes it's hard to "stay in the sandbox," especially if database licenses or web server ports are in short supply. You may be able to use a single database but create separate instances for each developer. Or, if forced to use one database with one instance, you may be able to partition the data space (for example, Joe is assigned test account data for accounts 1000–1999, and Sue is assigned accounts 2000–2999, and so on). This still leaves you open to risk of interference, but it's better than nothing.

For other resources such as web services, every developer should have a clear shot at their own instance (whether they are providing the service or testing against it).

With this basic idea of isolation in mind, let's take a look at some of the tools and other bits of infrastructure you'll need to achieve the sandbox effect.

When Not to Experiment

Never have a vital part of your product cycle (such as the build system) written in a niche or noncore technology, especially if only one developer knows it. Use a technology that anyone in the shop can configure and maintain. Technology playgrounds are fine, and necessary for professional development, but this isn't the place for them. Experiments must exist outside of the critical path.[15]

In one startup, the build script had been written in a new language; it was a learning project for the developer who wrote it. Worse, it was a general-purpose scripting language, not a build system. It contained more than 25 pages of spaghetti code that used every possible obscure language feature. Needless to say, the code was incomprehensible. It featured hard-coded dependencies on one developer's machine, on specifically setup network drives, and on specific versions of software components. The program was just a mess.

Never let a critical technology (like your build system) be created as a technology experiment. Use a tool designed for builds to create your builds, not the cool new technology that a team member wants to learn. There are plenty of noncritical areas for technology learning to take place. Never create automated tools that run on only one machine. Never hard-code network drive dependencies. Put everything you need in your SCM system, and the network drives become unimportant.

For example, if you are working in a Java shop, consider using Ant as your build script. Ant's whole purpose in life is to build Java programs, so it's a lot easier to write Java build scripts in Ant than in a language such as Python. Python is a great language, but it doesn't know anything about Java. Leverage your existing expertise when selecting your build system.

You gain a lot by adhering to this rule; it makes maintenance of the tool much easier. Anyone in the shop will be able to work with the technology and make adjustments. Also, by requiring experience, you avoid getting caught by a technology that looks great for a given situation but actually isn't.

[15] Your project's *critical path* contains anything that can slow down your project. Your SCM system and build scripts are good examples of items in your critical path. When they break, everything else stops as well.

But This Stuff Is Sooo Cool!

If you think a certain technology should be the exception to the rule (never make core technologies an experiment), then spend the time and money to get everyone trained. Don't just bring in the new technology and assume it's so great that everyone will learn it. It never happens that way. Use feedback to make sure people learn what you expect them to learn. Remember that such feedback reflects the talent of the teacher, not the stupidity of the student.

Only experience can tell you about a given technology's shortcomings.

One other danger worth mentioning involves letting any code wizard (or build script) do anything for you that you don't understand yourself. It's fine to let a tool handle details for you, but only if you already understand those details. If you don't, you'll be completely lost when something breaks. "Don't use wizard code you don't understand" [HT00].

Tip 13

Keep critical path technologies familiar

Work from The List

Many times we use a to-do list to track our work. The List formalizes the to-do concept so we can use it in a team setting.

In the past, when working on smaller projects, we used legal pads and notebooks to track our work. The List began as our personal to-do list of things we didn't want to forget. As we transitioned into more leadership roles, The List we were using began to contain teamwide items and paper became inefficient. After you show other team members The List enough times, you start looking for alternative ways to record and share The List. A white board works fine for a small group, especially in a large room with cubes (can you say *startup*?). These days we tend to use web pages or wikis. Some people use a spreadsheet[1] for The List. Everyone can access a web page, and it's easy to edit as well.

The List is how you set your daily and weekly agendas. You order your work with The List, as does the entire team (it's fractal!). When you get swamped, overwhelmed, or scattered, you come back to The List and use it to regain your focus. If you get stuck on a tough problem and you need to step away for a while, The List gives you a readily available set of items to use as filler. This ensures that you're working on the most important item, rather than the proverbial "squeaky wheel."

Why You Need The List

How often have you worked on a project where everyone was staying busy but the product was never complete? Important features were forgotten, team members were held up waiting for features that weren't done, and developers were stuck, not knowing what to work on next.

Someone needs to write down all the features, sort them by priority, and let team members grab their next job from the top of The List. You create a central point of organization for your team without the overhead of most heavyweight processes.

Team members that have The List never run out of work. When they finish their current task, they check out The List, pick a feature within the top-priority items, and go to work. Developers can pick their next

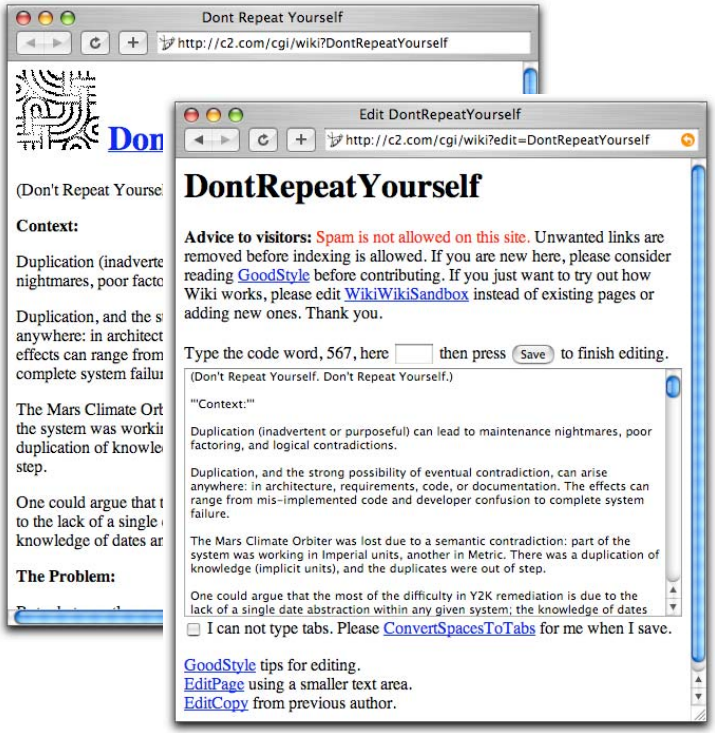
[1] Joel Spolsky uses an Excel spreadsheet to track his work list, and he has a lot of good reasons that you should also use something just as simple. See http://www.joelonsoftware.com/articles/fog0000000245.html

Joe Asks...

What's a Wiki?

A *wiki* is an easy way for people to create web pages. Wiki pages are written in a simple markup, not HTML, so anyone can add content. Wikis are designed to encourage group collaboration.

A Wiki page looks like any other web page initially, except that there's an "edit this page" link at the bottom. Clicking on that link puts the content of the current page in a simple HTML text edit box, so you can make changes to page very easily.



Visit `http://www.wiki.org` for more information.

job so they can still pick the work that is most interesting to them, but the tech lead knows that the features that are the highest priority are in progress or next on The List.

Since all developers are at different skill levels, the tech lead makes exceptions as needed, but generally, no second-priority item can be touched until the first-priority items are complete.

The List (as a team tool) gives management and customers something concrete to look at and evaluate the product before the time is invested adding the features. It's always cheaper to remove a feature before you've spent a week adding it! How many times have you finished a product and then had the customer say, "The product is okay, but I would have really loved it if you had put in feature A instead of X, Y and Z"? You create a piece of lightweight documentation that you can show people early in the development cycle when you use The List.

The List also provides a great deal of agility to your team. The List assures that you've done some basic design work up front by ensuring that you've decomposed your product into features and features into list items, as needed. Also, because your product is separated into features, you can drop out items or add others as needed.

Most companies have an individual or two who will rush in and demand to know why feature X was not finished or is not being worked on. (Their viewpoint: isn't the feature that I care about the most important one?) With a defined and prioritized set of tasks in hand, you can show them what features you're developing and why they are more important to the core product. This explanation is usually enough to satisfy the person and show them that you're doing useful work.

Your comprehensive list of features with priorities that make sense is a confidence builder for the team, for management, and for other teams that might depend on your efforts. Having The List several items deep demonstrates you are thinking ahead and planning your next steps.

Tip 14

Work to The List

How Should I Use The List?

You can use The List for your own work or for your entire team. Either way is easy and very effective. We use it both ways.

The List As an Organizational Tool

Through the years we've worked in a variety of roles, but we've rarely worn a single hat. As a general rule we, like most people, end up with lots of different jobs to do and very little time to get them done. When we try to juggle all the work in our heads, we tend to spread our time so thin on a lot of tasks and end up getting no substantial work done.

Our solution is to use The List as a personal organizer as well. While we always use the group list for the entire team, we've found that having our own copy of what we have to do is very helpful. Make a list each morning of what is on your plate and then prioritize it. At the end of the day, review what you did (or didn't) finish. Decide if you didn't get everything done because you were overly optimistic about what you thought you could get done or because you were distracted during the day. For a more complete discussion of using your own list to order your personal work list, get a copy of *The Seven Habits of Highly Effective People* by Stephen Covey.

Getting started with your own copy of The List is easy. First, create a list of every task you are working on (or have pending). Then, with your tech lead, assign a priority to each item. Finally, put a time estimate with each item. Don't worry about getting the time estimates perfect the first time, you'll improve over time.

Getting your team started on The List isn't hard either if your product is already well-defined. It's actually a great group activity that can help the entire team understand the project's overall direction.

1. Put every feature that you are adding to your project on a white board. This can take a while and often takes more than one white board.
2. Assign priorities to each feature. Be sure to include the proper stakeholders (management, customers, etc.) in this process. It's ideal to have your entire team in on the process, but if you have strongly opinionated team members, it may be smoother to just include the tech lead and the stakeholders.
3. Rewrite all of the features, sorted by priority.
4. Attach time estimates to each item.

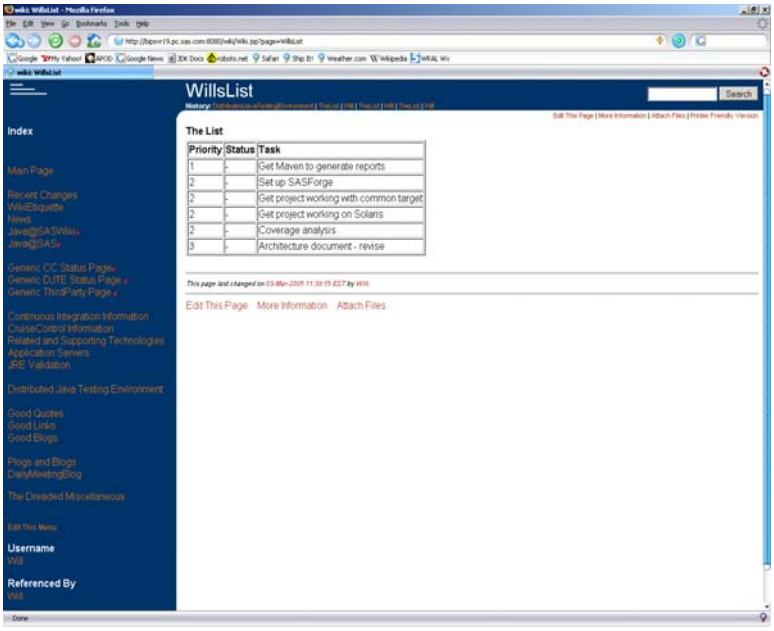


Figure 3.2: The List on an intranet web page

Until the current top priority items are completed, no one can work on the lower-priority items. This ensures that all the priority-one items are in progress before any of the priority-two tasks are touched.

As you can see, getting started using The List is fairly easy. However, to be effective, The List must adhere to a number of rules. It must be all of the following:

- Publicly available
- Prioritized
- On a time line
- Living
- Measurable
- Targeted

Next we'll look at what each of these rules means, and what it means to us and our team.

Joe Asks...

What's RSS?

Depending on who you ask, RSS stands for either "Rich Site Summary" or "Really Simple Syndication" or "RDF Site Summary." It's a way of sharing changes to content. It's very popular with web sites with dynamic content (like news sites or build status). When a web site shares their changes using RSS, it's called an *RSS feed*. An RSS feed is an XML file that lists changes or new content.

An *RSS reader* is a program that checks on all the RSS feeds you've subscribed to and shows you the new stuff. RSS feeds are made available on web servers, so an RSS reader is really just looking at a file on a web site and showing you the changes.

RSS readers are a really convenient way to get your news in a digest format. They collect news until you are ready to read it.

Publicly Available

Your team's List must be publicly available. A secret list doesn't help collaboration. Put The List on your white board or web site, make an RSS feed for it, or otherwise make it very easy and obvious for people to read. Keeping The List in front of you helps you maintain your focus. It gives you an easy review of pending work that you can scan quickly—especially when you're scattered or distracted during a hectic day. Keeping it publicly visible helps your manager keep track as well.

Prioritized

The List must be prioritized. It's very important to recognize the different types of features involved in a product: necessary features, desired features, and fluff features. You *must* make these distinctions when prioritizing The List, or you will be wasting your time. There will always be a core set of tasks that must be accomplished before the product can ship; these are the top-priority features. For instance, these might include the login screen, the installer, or a working database. You simply cannot ship your product without them. Having a new and

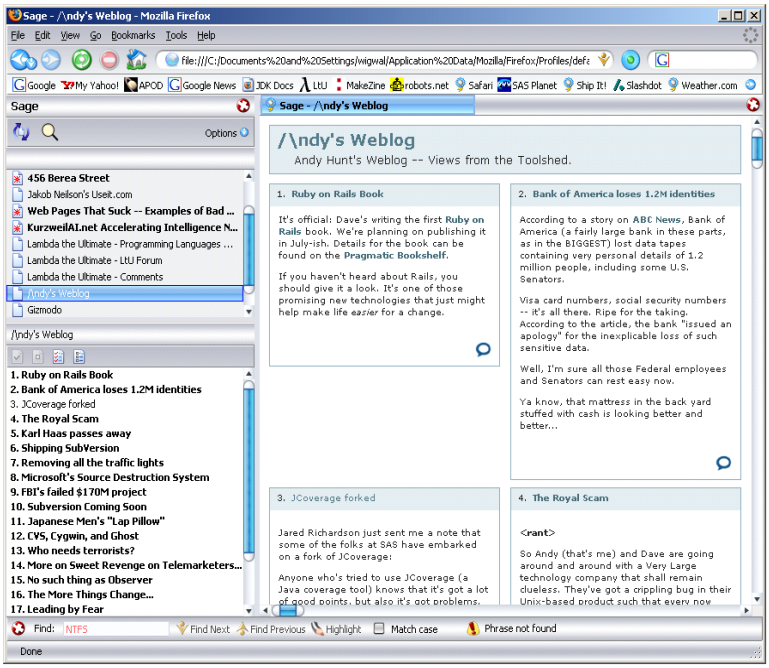


Figure 3.3: An RSS newsreader

improved background color for the About dialog box would probably be considered be a fluff feature.

Never, ever, bypass the priorities you've set. Finish all the higher-priority items before working on lower-priority items, unless there's *good*—and widely publicized—reason to temporarily put one on hold.

Time Estimates

The List always has a time line associated with it. The time line should not be set in stone, but it should include estimates for how long each item on the list should take to complete. Then, as you complete an item, record how long it actually took and pay attention to the difference.

Over time, you—and everyone on the team—will get very good at estimating how long a given task will take. After a few iterations, the tech lead should be able to create rough project time lines based on individual team member's lists, and the project manager should be able to do

the same thing. *There are no wrong answers when estimating.* Some estimates will be closer than others. Don't worry about how much you miss an estimate at first. Like a muscle, this skill grows as you use it.[2]

Living

To be effective, it must be a living list. Your team must be able to adapt to change. The tech lead will adjust feature priorities as the project progresses; new features will appear while others fade away. Priorities change. This is a Good Thing! It can be frustrating until you get used to it, but remember that your company is trying to be competitive in a changing marketplace. They depend on you to be flexible as well. Instead of fighting it, work with it.

In fact, changes to The List usually mean that your customers and stakeholders are looking at the project and are actually giving mind share—and valuable feedback—to it. Most customers wait until the project is finished to look at your work, but by then it's too late. It's always better to get feedback earlier, even if it might be frustrating to see The List frequently change. If The List hasn't evolved in a while, it probably doesn't reflect the current priorities of the project.

Measurable

In order to be effective, every item on The List must be measurable. After all, you must be able to determine whether the item is done if you want to mark it off your list.

This criteria eliminates vague items like "performance improvements" but encourages "Make login complete in less than five seconds" or "Generate report X in less than ten seconds." By creating a goal with a binary state, you make it possible to tell when it's done. An open-ended goal like "performance improvements" can last the life of the product and end up being a black hole.

If you currently have items on The List that are not measurable, take the time to look at what the real requirement is. Did the item come from a request for faster reporting or faster startup times? Break down the item into defined, binary items, and then get the person who asked

[2] If your team is having trouble with estimates, try to limit the choices; for instance, every estimate must be one day, one week, two weeks, or four weeks. Allow no other choices at first.

Feature-Boxed Iterations Instead of Time-Boxed Iterations

The problem with time-boxing is that we ship product features, not calendar days, to our customers.

When your team is using The List, and all the items are ordered by their priority, your releases become feature boxed, not time boxed. Management can look at the The List and draw a line beneath the features that they need in the next release. You then add up the time estimates for those features and calculate the release date. Your individual product and industry will dictate how long to schedule for your internal testing and beta programs, but you can concretely schedule your development *code freeze*.

When your sales team decides that a given feature must be included, that feature's priority on The List can change, and it can migrate into the shipping features. However, the time associated with the feature *must* be added to the ship date.

This way of working gives your sales force and management team a clear and defined way to understand the trade-off between specific features and time. They are no longer trying to make decisions about ship dates and features in a vacuum. Instead of trying to abstractly weigh features that take more time, they are weighing two specific features, with specific time frames.

We feel that this approach gives you the product when it's ready, as soon as it can be ready, instead of letting your company dictate an arbitrary release date that you miss. Our industry is famous for missing deadlines, and no wonder given the way we write software. Instead of trying to push an arbitrary feature set into an arbitrary deadline, companies should let their development team tell them what they can do! If the developers can't hit the mark that sales wants, is it better to find out now and adjust your plans or find out later, when you miss the deadline? And if the development team can hit the mark *early*, wouldn't it be nice to know so that the company can add features to the release or get it out the door to customers sooner?

The first time you release a product this way, management will be nervous. The second time, they'll be relaxed. The third time, management will have learned to trust their software teams to deliver what they promised!

Joe Asks...

What's a Code Freeze?

A *code freeze* is when your code base stops changing. During your development cycle, code is fluid, like water, and changes constantly. However, after the code freezes, the changes stop. Only major bug fixes can be made after a code freeze. Feature additions and minor bug fixes are not allowed.

Oftentimes a "code freeze" degrades into more of a "code slush" as ill-considered changes seep into the release.

for the original item to look at the items. This review will make sure that you are actually addressing the customer's need.

If an item can't be translated into measurable goals, then bump it to the lowest priority and get working on the higher-priority items. Removing the item entirely can be a mistake if the original idea for the item was a good one; it simply needs to be boiled down to the measurable pieces.

Targeted

You've probably noticed by now that we have talked about both team lists and individual lists. Each type of list is very important and must be targeted at the proper audience. The List for your team will be a lot larger and have all the outstanding work for the entire project on it. Your individual version of The List will contain fewer team items (sometimes only a single item for the project), but as soon as you are done, you copy an item from the team's list and put it on yours.

Although it is very simple, The List is a powerful tool on many levels. It keeps you organized and on-track and keeps your management chain informed and involved in your work direction. Creating and prioritizing The List makes you think through your work and map out your next steps. Any great pool player will tell you that they have their next eight shots planned out; so will any great developer!

Your List Looks Like This

Here is an example of what your personal list might look like, sorted by priority:

1. Add a new report that displays widgets produced per day.
2. Add a new report that displays widgets produced per employee.
3. Look at bug #12345 (widgets per month shows zero when viewing five-month report).
4. Install development tools on my new workstation.
5. Check out cool new WhatChaMuhCallIt-Reports. . . might make a good addition to the next edition's report system.

Notice that the most important items are at the top. In this case the new features are more important than the bug fix, but that's not always the case. Also, the computer upgrade and research project are at the bottom of the list. These items are for filler when you get some down-time (perhaps you are waiting on someone else?) or you need a break. It's important to have the filler items on the list so that lower-priority items don't get forgotten.

How to Get Started

1. For an entire day, write down every task as you work on it (this will be your "finished" list).
2. Organize whatever daily task list you do have into a formal copy of The List.
3. Ask your tech lead to help you prioritize your work and add rough time estimates.
4. Start working on the highest-priority item on The List—no cheating! If some crisis forces a lower-priority item higher, record it.
5. Add all new work to The List.
6. Move items to your finished list as you complete tasks (this makes surviving status reports and "witch-hunts" much easier).

The act of creating The List forces you to organize and prioritize your work. Just as keeping a diary helps you think through and understand what you've been doing, The List helps you sort out your current workload but in a fairly high-level, lightweight way.

Review The List every morning. Update it whenever new work pops up. . . especially the last-minute crisis tasks; you're likely to forget about those when someone asks you what you on earth you did all last week.

You're Doing It Right If...

- Is every one of your current tasks on The List?
- Does The List accurately portray your current task list?
- Did the tech lead or customer help you to prioritize The List?
- Is The List publicly available (electronically or otherwise)?
- Do you use The List to decide what to work on next?
- Can you update (and publish) The List quickly?

Warning Signs

- You fail to add tasks to The List because you're "too busy."
- More time is spent updating The List than completing the tasks.
- It takes weeks for team members to complete individual items on their personal lists (hint: the items are too big).
- The List is updated less than once a week.
- Priorities on The List don't match "real" priorities.
- The List is a closely held secret, not visible to anyone outside your team.
- In addition to the team's list, there are other publicly available versions that differ.