IEEE SOFTWARE

Best Practices

***IEEE Software*, Vol. 13, No. 4, July 1996**

Daily Build and Smoke Test

If you want to create a simple computer program consisting of only one file, you merely need to compile and link that one file. On a typical team project involving dozens, hundreds, or even thousands of files, however, the process of creating an executable program becomes more complicated and time consuming. You must "build" the program from its various components.

A common practice at Microsoft and some other shrink-wrap software companies is the "daily build and smoke test" process. Every file is compiled, linked, and combined into an executable program every day, and the program is then put through a "smoke test," a relatively simple check to see whether the product "smokes" when it runs.

BENEFITS. This simple process produces several significant benefits.

It minimizes integration risk. One of the greatest risks that a team project faces is that, when the different team members combine or "integrate" the code they have been working on separately, the resulting composite code does not work well. Depending on how late in the project the incompatibility is discovered, debugging might take longer than it would have if integration had occurred earlier, program interfaces might have to be changed, or major parts of the system might have to be redesigned and reimplemented. In extreme cases, integration errors have caused projects to be cancelled. The daily build and smoke test process keeps integration errors small and manageable, and it prevents runaway integration problems.

It reduces the risk of low quality. Related to the risk of unsuccessful or problematic integration is the risk of low quality. By minimally smoke-testing all the code daily, quality problems are prevented from taking control of the project. You bring the system to a known, good state, and then you keep it there. You simply don't allow it to deteriorate to the point where time-consuming quality problems can occur.

It supports easier defect diagnosis. When the product is built and tested every day, it's easy to pinpoint why the product is broken on any given day. If the product worked on Day 17 and is broken on Day 18, something that happened between the two builds broke the product.

It improves morale. Seeing a product work provides an incredible boost to morale. It almost doesn't matter what the product does. Developers can be excited just to see it display a rectangle! With daily builds, a bit more of the product works every day, and that keeps morale high.

USING THE DAILY BUILD AND SMOKE TEST. The idea behind this process is simply to build the product and test it every day. Here are some of the ins and outs of this simple idea.

Build daily. The most fundamental part of the daily build is the "daily" part. As Jim McCarthy says (*Dynamics of Software Development*, Microsoft Press, 1995), treat the daily build as the heartbeat of the project. If there's no heartbeat, the project is dead. A little less metaphorically, Michael Cusumano and Richard W. Selby describe the daily build as the *sync pulse* of a project (*Microsoft Secrets*, The Free Press, 1995). Different developers' code is allowed to get a little out of sync between these pulses, but every time there's a sync pulse, the code has to come back into alignment. When you insist on keeping the pulses close together, you prevent developers from getting out of sync entirely.

Some organizations build every week, rather than every day. The problem with this is that if the build is broken one week, you might go for several weeks before the next good build. When that happens, you lose virtually all of the benefit of frequent builds.

Check for broken builds. For the daily-build process to work, the software that's built has to work. If the software isn't usable, the build is considered to be broken and fixing it becomes top priority.

Each project sets its own standard for what constitutes "breaking the build." The standard needs to set a quality level that's strict enough to keep showstopper defects out but lenient enough to dis-regard trivial defects, an undue attention to which could paralyze progress.

At a minimum, a "good" build should

- compile all files, libraries, and other components successfully;
- link all files, libraries, and other components successfully;
- not contain any showstopper bugs that prevent the program from being launched or that make it hazardous to operate; and
- pass the smoke test.

Smoke test daily. The smoke test should exercise the entire system from end to end. It does not have to be exhaustive, but it should be capable of exposing major problems. The smoke test should be thorough enough that if the build passes, you can assume that it is stable enough to be tested more thoroughly.

The daily build has little value without the smoke test. The smoke test is the sentry that guards against deteriorating product quality and creeping integration problems. Without it, the daily build becomes just a time-wasting exercise in ensuring that you have a clean compile every day.

The smoke test must evolve as the system evolves. At first, the smoke test will probably test something simple, such as whether the system can say, "Hello, World." As the system develops, the smoke test will become more thorough. The first test might take a matter of seconds to run; as the system grows, the smoke test can grow to 30 minutes, an hour, or more.

Establish a build group. On most projects, tending the daily build and keeping the smoke test up to date becomes a big enough task to be an explicit part of someone's job. On large projects, it can become a full-time job for more than one person. On Windows NT 3.0, for example, there were four full-time people in the build group (Pascal Zachary, *Showstopper!*, The Free Press, 1994).

Add revisions to the build only when it makes sense to do so. Individual developers usually don't write code quickly enough to add meaningful increments to the system on a daily basis. They should work on a chunk of code and then integrate it when they have a collection of code in a consistent state-usually once every few days.

Create a penalty for breaking the build. Most groups that use daily builds create a penalty for breaking the build. Make it clear from the beginning that keeping the build healthy is the project's top priority. A broken build should be the exception, not the rule. Insist that developers who have broken the build stop all other work until they've fixed it. If the build is broken too often, it's hard to take seriously the job of not breaking the build.

A light-hearted penalty can help to emphasize this priority. Some groups give out lollipops to each "sucker" who breaks the build. This developer then has to tape the sucker to his office door until he fixes the problem. Other groups have guilty developers wear goat horns or contribute $5 to a morale fund.

Some projects establish a penalty with more bite. Microsoft developers on high-profile projects such as Windows NT, Windows 95, and Excel have taken to wearing beepers in the late stages of their projects. If they break the build, they get called in to fix it even if their defect is discovered at 3 a.m.

Build and smoke even under pressure. When schedule pressure becomes intense, the work required to maintain the daily build can seem like extravagant overhead. The opposite is true. Under stress, developers lose some of their discipline. They feel pressure to take design and implementation shortcuts that they would not take under less stressful circumstances. They review and unit-test their own code less carefully than usual. The code tends toward a state of entropy more quickly than it does during less stressful times.

Against this backdrop, daily builds enforce discipline and keep pressure-cooker projects on track. The code still tends toward a state of entropy, but the build process brings that tendency to heel every day.

Who can benefit from this process? Some developers protest that it is impractical to build every day because their projects are too large. But what was perhaps the most complex software project in recent history used daily builds successfully. By the time it was released, Microsoft Windows NT 3.0 consisted of 5.6 million lines of code spread across 40,000 source files. A complete build took as many as 19 hours on several machines, but the NT development team still managed to build every day (Zachary, 1994). Far from being a nuisance, the NT team attributed much of its success on that huge project to their daily builds. Those of us who work on projects of less staggering proportions will have a hard time explaining why we aren't also reaping the benefits of this practice.

Editor: Steve McConnell, Construx Software, 11820 Northup Way #E200, Bellevue, WA 98005.
E-mail: steve.mcconnell@construx.com - WWW: http://www.construx.com/stevemcc/

Email me at stevemcc@construx.com.