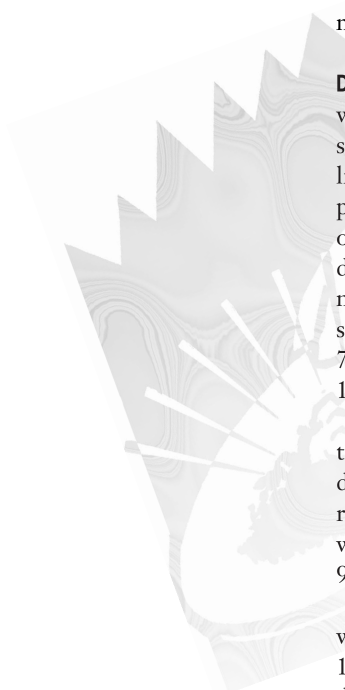
Gauging Software Readiness with Defect Tracking

Prospecting for programmer's gold.



Editor:
Steve McConnell
Construx Software Builders
PO Box 6922
Bellevue, WA 98008
stevemcc@construx.com

IN THE COMPETITIVE COMMERCIAL software market, companies feel compelled to release software the moment it is ready. Their task is treacherous, treading the line between releasing poor-quality software early and high-quality software late. Finding a sound answer to the question, "Is the software good enough to release now?" can be critical to a company's survival. That answer is sometimes based on gut instinct, but several techniques can put this judgment on firmer footing.

DEFECT DENSITY. One of the easiest ways to judge whether a program is ready for release is to measure its defect density—the number of defects per line of code. Suppose the first version of your product, GigaTron 1.0, consisted of 100,000 lines of code. Further suppose that you detected 650 defects prior to the software's release, and that 50 more defects were reported after release. The software therefore had a lifetime defect count of 700 defects, and a defect density of 7 defects per 1,000 lines of code (KLOC).

Suppose that Version 2.0 had 50,000 additional lines of code and that you detected 400 defects prior to release and another 75 after release. The total defect density of that release would be 475 defects divided by 50 KLOC, or 9.5 defects per KLOC.

Now suppose that you're trying to decide whether GigaTron 3.0 is reliable enough to ship. In 100,000 new lines of code you've detected 600 defects so far, or 6 defects per KLOC. Unless you have good reason to think that your development process has improved with this project, your experience should lead you to expect 7 to 10 defects per KLOC. The number of defects you should attempt to find will vary depending on the level of quality you're aiming for. If you want to remove 95 percent of all defects before shipping, you will need to detect 650 to 950 defects. This technique suggests that the product is not quite ready to ship.

The more historical project data you have, the more confident you can be in your prerelease defect density targets. If you have data from only two projects and the range is as broad as 7 to 10 defects per KLOC, that leaves a lot of wiggle

The more historical project data you have, the more confident you can be in your prerelease defect density targets.

room for judging whether the third project will be more like the first or the second. But if you've tracked defect data for 10 projects and found that their average lifetime defect rate is 7.4 defects per KLOC, with a standard deviation of 0.4 defects, you have a great deal of guidance indeed.

DEFECT POOLING. Another simple defect prediction technique is to separate defect reports into two groups; let's call them Pool A and Pool B. You then track the defects in these two pools separately. The distinction between the pools is arbitrary: you could put all the defects discovered on Mondays, Wednesdays, and weekends into Pool A, and the rest into Pool B. Or you could split your test team down the middle and put each subgroup's reported defects into its own pool. It doesn't matter how you divide the group as long as both subgroups operate independently and both test the full scope of the product.

You then track the number of defects reported in each pool and—here's the important part—the number of defects reported in *both* pools. The number of unique defects reported at any given time is

Continued on page 135

Continued from page 136

$$\text{Defects}_{\text{unique}} = \text{Defects}_{\text{A}} + \text{Defects}_{\text{B}} - \text{Defects}_{(\text{A+B})}$$

The number of total defects can then be approximated by the simple formula

$$\text{Defects}_{\text{total}} = \frac{\text{Defects}_{\text{A}} \times \text{Defects}_{\text{B}}}{\text{Defects}_{(\text{A+B})}}$$

If the GigaTron 3.0 project has 400 defects in Pool A, 350 defects in Pool B, and 150 of the defects in both pools, the number of unique defects detected would be 400 + 350 – 150 = 600. The approximate number of total defects would be 400 × 350 / 150 = 933. This suggests that about 333 defects are yet to be detected (about a third of the estimated total); quality assurance on this project still has a long way to go.

DEFECT SEEDING. In defect seeding, one group intentionally inserts defects into a program for detection by another group. The ratio of the number of seeded defects detected to the total number of seeded defects provides a rough idea of the total number of unseeded defects.

Suppose you intentionally seed GigaTron 3.0 with 50 errors. For best effect, the seeded errors should cover the full breadth of the product's functionality and the full range of severities—ranging from cosmetic to crashing errors. When you think testing is almost complete, you look at the seeded-defect report: 31 seeded and 600 indigenous defects have been found. You can estimate the total number of defects with the formula

$$\text{IndigenousDefects}_{\text{total}} = \frac{\text{SeededDefects}_{\text{planted}}}{\text{SeededDefects}_{\text{found}}} \times \text{IndigenousDefects}_{\text{found}}$$

This technique suggests that GigaTron 3.0 has a total of about 50/31 × 600, or 967, defects.

To use this technique, you must seed the defects before you begin the tests whose defect detection rate you want to ascertain. If your testing uses manual methods and has no systematic way of covering the same testing ground twice, you should seed defects before testing begins. If your testing uses fully automated regression tests, you can seed defects virtually any time to estimate the number of defects left undetected by the automated tests.

A common problem with defect seeding programs is forgetting to remove the seeded defects. Another common problem is that removing the seeded defects introduces new errors. To prevent these problems, be sure to remove all seeded defects prior to final system testing and product release. A useful implementation standard is to require that errors are planted only by *adding* one or two lines of code; this ensures that you can remove the seeded errors safely by simply removing the erroneous lines of code.

No news is more often the result of insufficient testing than of superlative development practices.

DEFECT MODELING. A colleague of mine recently added several hundred lines of code to an existing program in one sitting. The first time he compiled the code, he got a clean compile with no errors—his initial coding appeared to be flawless. When he tried to test the new functionality, however, he found that it didn't exist. When he reexamined his new code, he found that his work had been embedded in a preprocessor macro that deactivated the new code. When he moved the new code outside the scope of the macro, it produced the usual number of compiler errors.

With software defects, no news is usually bad news. If the project has reached a late stage with few defects reported, there is a natural tendency to think, "We finally got it right and created a program with almost no defects!" In reality, no news is more often the result of insufficient testing than of superlative development practices.

Some of the more sophisticated software project estimation and control tools can predict the number of defects you should expect to find at each point in a project. By comparing defects actually detected to the number predicted, you can assess whether your project is keeping up with defect detection, or lagging behind.

COMBINATIONS. Evaluating combinations of defect density, defect pools, and defect seeding will give you more confidence than you could have using any one technique. Examining defect density alone on GigaTron 3.0 suggested that you should expect 700 to 1,000 total lifetime defects, and that you should remove 650 to 950 before product release to achieve 95 percent prerelease defect removal. If you had detected 600 defects, the defect density information alone might lead you to declare the product "almost ready to ship." But defect pooling analysis estimates that GigaTron 3.0 will produce about 933 defects. Comparing the results of these two techniques suggests that you should expect a total defect count toward the high end of the defect density range. Because the defect seeding technique also estimates a total number of defects in the 900s, GigaTron 3.0 appears to be a relatively buggy program.

RESOURCES. The popular software development literature doesn't have much to say about defect prediction. A notable exception is Glenford Myers' *Software Reliability* (John Wiley and Sons, 1976). Lawrence H. Putnam and Ware Myers discuss the specific topic of defect modeling at some length in *Measures for Excellence* (Yourdon Press, 1992). ◆