Joel on Software

The Joel Test: 12 Steps t Code

by Joel Spolsky

Wednesday, August 09, 2000

Have you ever heard of SEMA? It's a fairly esoteric s

measuring how good a software team is. No, *wait! D*
link! It will take you about six years just to *understa*
I've come up with my own, highly irresponsible, slop
quality of a software team. The great part about it is
3 minutes. With all the time you save, you can go to

The Joel Test

1. Do you use source control?
2. Can you make a build in one step?
3. Do you make daily builds?
4. Do you have a bug database?
5. Do you fix bugs before writing new co
6. Do you have an up-to-date schedule?
7. Do you have a spec?
8. Do programmers have quiet working conditions?
9. Do you use the best tools money can b
10. Do you have testers?
11. Do new candidates write code during t interview?
12. Do you do hallway usability testing?

The neat thing about The Joel Test is that it's easy to
or **no** to each question. You don't have to figure out
day or average-bugs-per-inflection-point. Give your

each "yes" answer. The bummer about The Joel Test *shouldn't* use it to make sure that your nuclear powe safe.

A score of 12 is perfect, 11 is tolerable, but 10 or lowe serious problems. The truth is that most software or running with a score of 2 or 3, and they need *serious* companies like Microsoft run at 12 full-time.

Of course, these are not the only factors that determi failure: in particular, if you have a great software tea product that nobody wants, well, people aren't going it's possible to imagine a team of "gunslingers" that this stuff that still manages to produce incredible so changes the world. But, all else being equal, if you ge right, you'll have a disciplined team that can consiste

1. Do you use source control?

I've used commercial source control packages, and I which is free, and let me tell you, CVS is *fine*. But if y source control, you're going to stress out trying to ge work together. Programmers have no way to know w did. Mistakes can't be rolled back easily. The other n source control systems is that the source code itself i every programmer's hard drive -- I've never heard of source control that lost a lot of code.

2. Can you make a build in one step?

By this I mean: how many steps does it take to make

from the latest source snapshot? On good teams, the you can run that does a full checkout from scratch, r of code, makes the EXEs, in all their various version #ifdef combinations, creates the installation package final media -- CDROM layout, download website, wh

If the process takes any more than one step, it is pro when you get closer to shipping, you want to have a fixing the "last" bug, making the final EXEs, etc. If it compile the code, run the installation builder, etc., y crazy and you're going to make silly mistakes.

For this very reason, the last company I worked at sw WISE to InstallShield: we *required* that the installat able to run, from a script, automatically, overnight, u scheduler, and WISE couldn't run from the schedule threw it out. (The kind folks at WISE assure me that version does support nightly builds.)

3. Do you make daily builds?

When you're using source control, sometimes one pr accidentally checks in something that breaks the bui they've added a new source file, and everything com machine, but they forgot to add the source file to the So they lock their machine and go home, oblivious a nobody else can work, so they have to go home too, u

Breaking the build is so bad (and so common) that it daily builds, to insure that no breakage goes unnotic

teams, one good way to insure that breakages are fix
do the daily build every afternoon at, say, lunchtime
many checkins as possible before lunch. When they
build is done. If it worked, great! Everybody checks
version of the source and goes on working. If the bui
it, but everybody can keep on working with the pre-b
version of the source.

On the Excel team we had a rule that whoever broke
"punishment", was responsible for babysitting the bu
someone else broke it. This was a good incentive not
build, and a good way to rotate everyone through the
that everyone learned how it worked.

Read more about daily builds in my article Daily Bui
Friend.

4. Do you have a bug database?

I don't care what you say. If you are developing code
of one, without an organized database listing all kno
code, you are going to ship low quality code. Lots of
think they can hold the bug list in their heads. Nonse
remember more than two or three bugs at a time, an
morning, or in the rush of shipping, they are forgotte
have to keep track of bugs formally.

Bug databases can be complicated or simple. A mini
database must include the following data for every b

- complete steps to reproduce the bug
- expected behavior
- observed (buggy) behavior
- who it's assigned to
- whether it has been fixed or not

If the complexity of bug tracking software is the only you from tracking your bugs, just make a simple 5 co these crucial fields and *start using it*.

For more on bug tracking, read Painless Bug Trackir

5. Do you fix bugs before writing new code

The very first version of Microsoft Word for Windows a "death march" project. It took forever. It kept slipp team was working ridiculous hours, the project was again, and again, and the stress was incredible. Whe finally shipped, years late, Microsoft sent the whole t Cancun for a vacation, then sat down for some seriou

What they realized was that the project managers ha insistent on keeping to the "schedule" that programm rushed through the coding process, writing extremel because the bug fixing phase was not a part of the fo There was no attempt to keep the bug-count down. (The story goes that one programmer, who had to wri calculate the height of a line of text, simply wrote "re waited for the bug report to come in about how his fu always correct. The schedule was merely a checklist

to be turned into bugs. In the post-mortem, this was
"infinite defects methodology".

To correct the problem, Microsoft universally adopte
called a "zero defects methodology". Many of the pro
company giggled, since it sounded like management
could reduce the bug count by executive fiat. Actuall
meant that at any given time, the highest priority is t
before writing any new code. Here's why.

In general, the longer you wait before fixing a bug, th
and money) it is to fix.

For example, when you make a typo or syntax error t
catches, fixing it is basically trivial.

When you have a bug in your code that you see the fi
run it, you will be able to fix it in no time at all, becau
still fresh in your mind.

If you find a bug in some code that you wrote a few d
take you a while to hunt it down, but when you rerea
wrote, you'll remember everything and you'll be able
reasonable amount of time.

But if you find a bug in code that you wrote a few *mo*
probably have forgotten a lot of things about that co
harder to fix. By that time you may be fixing somebo
and they may be in Aruba on vacation, in which case

like science: you have to be slow, methodical, and m
can't be sure how long it will take to discover the cur

And if you find a bug in code that has *already shipp*
incur incredible expense getting it fixed.

That's one reason to fix bugs right away: because it t
There's another reason, which relates to the fact that
predict how long it will take to write new code than t
bug. For example, if I asked you to predict how long
write the code to sort a list, you could give me a pret
But if I asked you how to predict how long it would t
where your code doesn't work if Internet Explorer 5.
can't even *guess*, because you don't know (by definit
causing the bug. It could take 3 days to track it down
2 minutes.

What this means is that if you have a schedule with a
remaining to be fixed, the schedule is unreliable. But
the *known* bugs, and all that's left is new code, then
be stunningly more accurate.

Another great thing about keeping the bug count at z
can respond much faster to competition. Some progr
this as keeping the product *ready to ship* at all times
competitor introduces a killer new feature that is ste
customers, you can implement just that feature and
without having to fix a large number of accumulated

6. Do you have an up-to-date schedule?

Which brings us to schedules. If your code is at all in business, there are lots of reasons why it's important know when the code is going to be done. Programme crabby about making schedules. "It will be done whe scream at the business people.

Unfortunately, that just doesn't cut it. There are too decisions that the business needs to make well in ad the code: demos, trade shows, advertising, etc. And t this is to have a schedule, and to keep it up to date.

The other crucial thing about having a schedule is th decide what features you are going to do, and then it the least important features and *cut them* rather tha featuritis (a.k.a. scope creep).

Keeping schedules does not have to be hard. Read m Software Schedules, which describes a simple way to schedules.

7. Do you have a spec?

Writing specs is like flossing: everybody agrees that but nobody does it.

I'm not sure why this is, but it's probably because m hate writing documents. As a result, when teams cor programmers attack a problem, they prefer to expres code, rather than in documents. They would much r

write code than produce a spec first.

At the design stage, when you discover problems, yo
easily by editing a few lines of text. Once the code is
of fixing problems is dramatically higher, both emot
hate to throw away code) and in terms of time, so th
actually fixing the problems. Software that wasn't bu
usually winds up badly designed and the schedule ge
This seems to have been the problem at Netscape, w
versions grew into such a mess that management stu
throw out the code and start over. And then they ma
over again with Mozilla, creating a monster that spu
and took *several years* to get to alpha stage.

My pet theory is that this problem can be fixed by te
programmers to be less reluctant writers by sending
an intensive course in writing. Another solution is to
program managers who produce the written spec. In
should enforce the simple rule "no code without spe

Learn all about writing specs by reading my 4-part s

8. Do programmers have quiet working co

There are extensively documented productivity gain
giving knowledge workers space, quiet, and privacy.
software management book Peopleware documents
benefits extensively.

Here's the trouble. We all know that knowledge worl

getting into "flow", also known as being "in the zone" fully concentrated on their work and fully tuned out environment. They lose track of time and produce gr absolute concentration. This is when they get all of t work done. Writers, programmers, scientists, and ev players will tell you about being in the zone.

The trouble is, getting into "the zone" is not easy. Wh measure it, it looks like it takes an average of 15 min working at maximum productivity. Sometimes, if yo already done a lot of creative work that day, you just zone and you spend the rest of your work day fiddlin the web, playing Tetris.

The other trouble is that it's so easy to get knocked *o* Noise, phone calls, going out for lunch, having to dri Starbucks for coffee, and interruptions by coworkers interruptions by coworkers -- all knock you out of th coworker asks you a question, causing a 1 minute int this knocks you out of the zone badly enough that it hour to get productive again, your overall productivi trouble. If you're in a noisy bullpen environment like caffeinated dotcoms love to create, with marketing g the phone next to programmers, your productivity w knowledge workers get interrupted time after time a the zone.

With programmers, it's especially hard. Productivity able to juggle a lot of little details in short term mem

Any kind of interruption can cause these details to c
down. When you resume work, you can't remember
(like local variable names you were using, or where y
implementing that search algorithm) and you have t
these things up, which slows you down a lot until yo
speed.

Here's the simple algebra. Let's say (as the evidence
that if we interrupt a programmer, even for a minute
blowing away 15 minutes of productivity. For this ex
two programmers, Jeff and Mutt, in open cubicles n
in a standard Dilbert veal-fattening farm. Mutt can't
name of the Unicode version of the strcpy function.
up, which takes 30 seconds, or he could ask Jeff, wh
seconds. Since he's sitting right next to Jeff, he asks
distracted and loses 15 minutes of productivity (to sa
seconds).

Now let's move them into separate offices with walls
when Mutt can't remember the name of that functio
up, which still takes 30 seconds, or he could ask Jeff
45 seconds and involves standing up (not an easy tas
average physical fitness of programmers!). So he loo
Mutt loses 30 seconds of productivity, but we save 1
Ahhh!

9. Do you use the best tools money can bu

Writing code in a compiled language is one of the las
can't be done instantly on a garden variety home cor

compilation process takes more than a few seconds, and greatest computer is going to save you time. If c even 15 seconds, programmers will get bored while t and switch over to reading The Onion, which will su hours of productivity.

Debugging GUI code with a single monitor system is impossible. If you're writing GUI code, two monitors much easier.

Most programmers eventually have to manipulate bi toolbars, and most programmers don't have a good l available. Trying to use Microsoft Paint to manipula joke, but that's what most programmers have to do.

At my last job, the system administrator kept sendin spam complaining that I was using more than ... get megabytes of hard drive space on the server. I pointe the price of hard drives these days, the cost of this sp significantly less than the cost of the *toilet paper* I us even 10 minutes cleaning up my directory would be of productivity.

Top notch development teams don't torture t programmers. Even minor frustrations caused by underpowered tools add up, making programmers g unhappy. And a grumpy programmer is an unprodu

To add to all this... programmers are easily bribed by

coolest, latest stuff. This is a far cheaper way to get t
you than actually paying competitive salaries!

10. Do you have testers?

If your team doesn't have dedicated testers, at least
or three programmers, you are either shipping bugg
you're wasting money by having $100/hour progra
that can be done by $30/hour testers. Skimping on t
outrageous false economy that I'm simply blown aw
people don't recognize it.

Read Top Five (Wrong) Reasons You Don't Have Te
wrote about this subject.

11. Do new candidates write code during t interview?

Would you hire a magician without asking them to s
magic tricks? Of course not.

Would you hire a caterer for your wedding without t
I doubt it. (Unless it's Aunt Marge, and she would h
you didn't let her make her "famous" chopped liver

Yet, every day, programmers are hired on the basis
resumé or because the interviewer enjoyed chatting
they are asked trivia questions ("what's the differen
CreateDialog() and DialogBox()?") which could be a
looking at the documentation. You don't care if they
thousands of trivia about programming, you care if t

produce code. Or, even worse, they are asked "AHA! kind of questions that seem easy when you know the you don't know the answer, they are impossible.

Please, just *stop doing this*. Do whatever you want d but make the candidate *write some code*. (For more Guerrilla Guide to Interviewing.)

12. Do you do hallway usability testing?

A *hallway usability test* is where you grab the next p by in the hallway and force them to try to use the coc If you do this to five people, you will learn 95% of wh learn about usability problems in your code.

Good user interface design is not as hard as you wou crucial if you want customers to love and buy your p read my free online book on UI design, a short prime programmers.

But the most important thing about user interfaces i your program to a handful of people, (in fact, five or will quickly discover the biggest problems people are Jakob Nielsen's article explaining why. Even if your are lacking, as long as you force yourself to do hallwa which cost nothing, your UI will be much, much bett

Next: Wasting Money on Cats

Want to know more? You're reading Joel on Softwa years and years of completely raving mad articles ab development, managing software teams, designing u running successful software companies, and rubber

About the author. I'm Joel Spolsky, co-founder of F Software, a New York company that proves that you programmers well and still be highly profitable. Prog private offices, free lunch, and work 40 hours a week pay for software if they're delighted. We make FogBu enlightened bug tracker designed to help great team software, Kiln, which simplifies source control and c Fog Creek Copilot, which makes remote desktop con the co-founder of Stack Overflow.

joel@joelonsoftware.com