The Joel Test

CSE 403 Software Engineering

What is the Joel Test?

The Joel Test is:

- A checklist of 12 best practices good software teams do
- A blog post 20 (!) years ago
  - https://www.joelonsoftware.com/2000/08/09/the-joel-test-12-steps-to-better-code/
  - https://dev.to/checkgit/the-joel-test-20-years-later-1kjk
- By Joel Spolsky (founder of Stack Overflow and Trello)
- These factors indicate a disciplined team that can consistently deliver

Score out of 12:

- 12 is good
- 11 is OK
- 10 or fewer is bad

So… Is the test still relevant today?

Today's Outline

1. Overview the 12 best practices [Time check – leave 10 min 2 and 3]
2. Discuss practices of some realistic software teams
3. Which team/company has the best chance of success?

Do you use source control?

What are the benefits?

- Allows multiple developers
- Keep project in consistent state
- Track changes and enable roll-back
- Manage multiple versions
- Save data in case of a disaster
- Authoritative source for “daily build”

Do you have a one-step build?

A single script that

- [does a full checkout from scratch]
- rebuilds every line of code
- makes the binary executable files in all versions, languages, OSes, and #ifdef combinations
- [creates the installation package]
- [creates the final media – CDROM, web site, …]

All steps are automated and exercised regularly

Do you do a daily build and test?

Build the entire product every day and run a good test suite against the new version

- build from checked in sources
- automatic and frequent

Goal: discover problems early, and fix them before disaster strikes

Benefits

- Minimizes integration risk
- Reduces risk of low quality
- Supports easier defect diagnosis
- Improves morale – developers, managers, customers

Do you use a bug database?

You can't keep the bug list in your head

- Especially with multiple developers and multiple customers
- Moreover, looking at the history of bugs can be insightful!

To characterize a bug consider:

- how to reproduce it
- expected behavior, actual behavior
- responsible party, status, priority

Best to use what is integrated with your code hosting

Alternatives: JIRA, Trac, Bugzilla, text file (☹)

Do you fix bugs before writing new code?

Why not fix them later?

- Familiar with the code now
- Harder to find (and fix) later
- Later code may depend on this code (try building on quicksand…)
- Bugs may reveal fundamental problems
- Leaving all bugs to the end will make it harder to understand and keep the schedule

"Technical debt"

Do you have an up-to-date schedule?

Keeps expectations realistic

- For the team, customers, stakeholders

Allows for more accurate predictions

- Use experience to improve estimates

Helps prevent feature creep

- Don’t take on anything without checking the schedule first

Do you have a spec?

- Easier to fix problems at the design stage
- You know what you are trying to build

  So do your teammates and customer
- More likely that you build the right thing

  Pieces fit together

  Customer is satisfied
- Conceptual integrity for your project
- The manual has similar benefits, & is part of the spec
- Undocumented code has no commercial value

  Joel's example: Netscape Navigator

  Other examples: Viaweb (Paul Graham, Robert Morris), Vanu

Do you do hallway usability testing?

Grab someone in the hallway and make them use your code

Key idea: get feedback fast

A little feedback now ≫ lots of feedback later

You will get most of the valuable feedback from the first few users

The Joel Test – how does 403 stack up?

1. ✅ Do you use source control?
2. ✅ Can you make a build [+ release] in one step?
3. ✅ Do you ~~make daily builds~~ use CI (Continuous Integration)?
4. ✅ Do you have a bug database?
5. ❓ Do you fix bugs before writing new code?
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 buy?
10. ✅ Do you have ~~testers~~ automated testing and monitor coverage?
11. ❓ Do new candidates write code during their interview?
12. ❓ Do you do hallway usability testing?

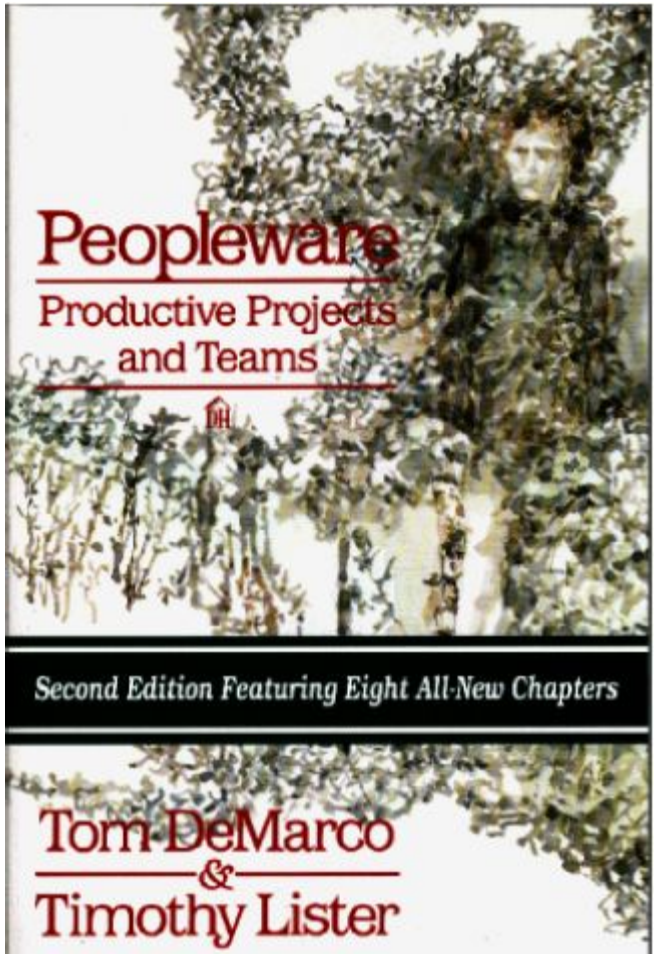
More advice: automation

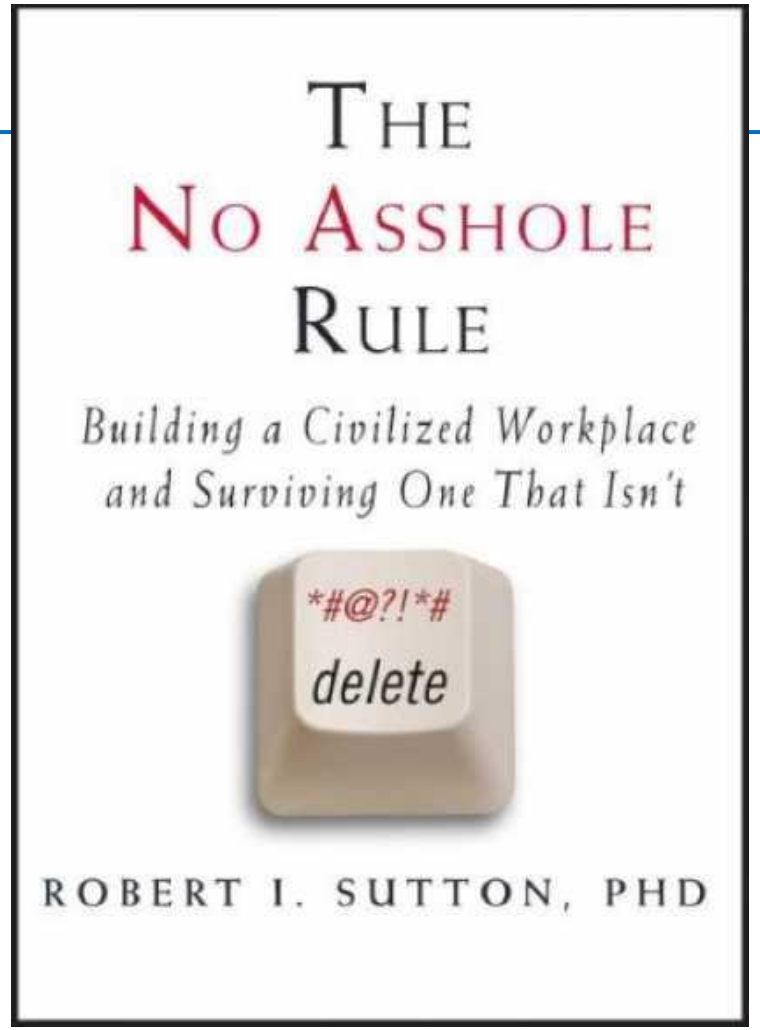
- Automated testing

- Automate everything (e.g., formatting)
  Nothing that requires human interaction is acceptable

- Use automated tools: linting, verification

Other advice





Let's try out the test

1. 8 teams/companies
2. Hypothetical but plausible – largely based on experience
3. Only some Joel Test questions are highlighted – assume others are covered adequately
4. Assess the scenarios as we go
   - How successful will they be in their scenario with their practices?
   - How much would you like to work in such an environment?

The Startup Incubator team

You work for an early-stage tech startup in an incubator. Things move fast around here.

(2.) One-step builds: Your team uses GitHub's continuous integration tools.

(8.) Loud conditions: You work in an incubator – so you share your cubicle with three other people, and you share your open floor with other companies. It can get pretty loud on a regular basis.

(9.) On a shoestring budget: Everyone works on their own laptop, partially from home (different OSes, etc), and you mainly avoid paid software – compatibility issues and some wasted time result.

(12.) Hallway usability testing: As a team you're constantly pinging ideas back and forth and demoing new features. As a result your UI is great, and you tend to only build useful features.

The Joel Test

1. Do you use source control?
2. Can you make a build in one step?
3. Do you use CI?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
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 buy?
10. Do you use automated testing?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

The Not-For-Profit Company team

Your team works for a mission-driven not-for-profit. You care a lot about the company, really get along with your co-worker, but some of the engineering practices are … questionable.

(1.) No source "control": Although you have your code in BitBucket, there is not a good process/effort to integrate upgrades from collaborators.

(5.) Lower bug priority: The company has little resources to keep up with new requirements. Bugs are only tackled only when somethings breaks really bad.

(8.) Quiet work conditions: you don't have offices, but your working spaces are fairly quiet, not like the cacophony of an incubator.

(12.) Hallway testing: you also do a good deal of hallway usability testing.

The Joel Test

1. Do you use source control?
2. Can you make a build in one step?
3. Do you do CI?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
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 buy?
10. Do you do automated testing?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

The Big Tech Company team

You work on a team at one of the big tech companies.

(1.) Source control: not only do you use source control, your company has its own suite of internal tools for code reviews, etc., increasing productivity a lot.

(2.) No one-step build: you cannot make the build in one step – in fact you have a "build manager" rotation which consumes an engineer's whole week.

(8.) Open floor plan: you have your own desk, thankfully, but it's on a floor with a few dozen desks and it's often a little busy.

(11.) Coding in interviews: coding is the biggest part of your company's notoriously difficult interview process. As a result, not only can you rely on your coworkers to be technically solid, you frequently learn from them.

The Joel Test

1. Do you use source control?
2. Can you make a build in one step?
3. Do you do CI?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
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 buy?
10. Do you do automated testing?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

The Investment Firm team

You work for a big bank or investment firm. Your team does in-house modeling and tooling for its investors.

(7.) No spec: leadership is pretty unclear on what they want you to do, and the software engineers hate writing documentation, so you frustratingly spend more time than you'd like working on projects that are ultimately dropped, or dealing with requirement churn

(8). Quiet work space: everyone has an office. In fact, maybe you have too much time away from your team.

(9.) Best tools money can buy: you have your own office and nice hardware. Cost is not a barrier to access any software or computing resources.

(10.) Do you have testers: Yes, but they are mostly focused on higher level issues, like the results of analysis. Tests aren't automated.

The Joel Test

1. Do you use source control?
2. Can you make a build in one step?
3. Do you do CI?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
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 buy?
10. Do you do automated testing?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

The Enterprise Company team

You work for a big enterprise software company. You have quarterly scheduled build releases, follow the Waterfall method, all that.

(3.) No daily builds: Every couple of weeks your team gets blocked on the build being broken by some bug a dozen commits ago. You can imagine a lot of time is lost at the whole company this way…

(6.) Up-to-date schedule: thanks to the company's structured releases, your team always knows what to have done, when. Other teams can count on yours to always hit your deadlines.

(7.) There are specs: Your team is careful to write specs.

(9.) Best tools available: Not really. Because of the companies' partnerships, you have to stick with the provided tools and it is really hard to try new ones.

The Joel Test

1. Do you use source control?
2. Can you make a build in one step?
3. Do you do CI?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
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 buy?
10. Do you do automated testing?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

The Trendy Startup team

You work for a trendy startup working on something to do with deep learning, or maybe blockchain.

(2.) One-click builds and **(3.) at-least daily builds**: both use standard continuous integration, resulting in little to no time wasted on fixing broken builds.

(5.) Your team doesn't prioritize fixing bugs and regularly **(6.) doesn't stick to a set schedule**. You're frequently meeting with and demoing the product for series A investors. Management prioritizes new feature launches ahead of fixing known bugs.

The Joel Test

1. Do you use source control?
2. Can you make a build in one step?
3. Do you do CI?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
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 buy?
10. Do you do automated testing?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

The Research Lab team

Your team works for a government-contracted research lab. Your engineering tasks encompass things like big-data biology, rocket engine simulations, etc.

(4.) No bug database – Your company's engineering developed to supplement code written by a principal researcher without software training, and not tracking bugs is one result of the lack of formality. You frequently encounter buggy code but have difficulty institutionally learning from any of these mistakes.

(7.) Your team uses specs which helps give direction to the team's efforts and avoid wasting time and **(8.) things are pretty quiet** – you work in a lab, and there aren't many distractions.

(11.) No coding in interviews – the company prioritizes other technical skills, so while some of your coworkers are very experienced engineers, others on your team (who write code) are researchers without a lot of programming experience.

The Joel Test

1. Do you use source control?
2. Can you make a build in one step?
3. Do you do CI?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
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 buy?
10. Do you do automated testing?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

The Big Non-Tech Company team

You work as part of the software team for a big non-tech company (like a hospital, a retail store chain, etc.)
You have quarterly deadlines for projects, and generally follow a more traditional business schedule.

(3) No daily builds: you're on quarterly cycles so you don't test the build on any regular schedule.

(7.) Your team works from a spec.

(8.) Has your own offices.

(10) No automated testing: Your company is not software focused so you don't have dedicated testers – but you *do* have stringent correctness requirements. As a result you have to spend a lot of time manually testing new features.

The Joel Test

1. Do you use source control?
2. Can you make a build in one step?
3. Do you do CI?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
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 buy?
10. Do you do automated testing?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

Wrapup

1. Are these tests still valid?
2. Which are most/least important?
3. Are some situational?

The Joel Test

1. Do you use source control?
2. Can you make a build in one step?
3. Do you use CI?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
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 buy?
10. Do you do automated testing?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?