

Projects and Project Teams

CSE 403 Software Engineering

Autumn 2023

Some advice from past students:

- Foundation of the success of our team was communication and cooperation
- Well-run and consistently scheduled meetings helped a lot
- Working together (physically) was good
- Take time to break tasks down to manageable chunks (+/understanding and +/-estimation)
- Don't underestimate the time to learn new tools/APIs/languages
- Do small frequent updates and commits; failing to do this results in merges that can be a nightmare
- Need an upfront testing design – test well before checkin

Projects and Team assignments are done!

- Thank you for all the great pitches!
 - Projects and team assignments are done
 - Matched as best as possible against student preferences
 - 100% got one of their top 3 choices
- See Ed Announcement for link with Project teams
 - Staff will create a separate Ed Discussion channel for each team to use to privately communicate with staff
 - Consider creating a channel for your team members also

Considerations for your new teams

- The project belongs to the whole team – all members are created equal
- Embrace diverse ideas and opinions – make your discussion zones safe places for people to share their thoughts
- You now have opportunity to reshape/rescope your project given the team's collective input
- Teams and individuals must ensure everyone contributes

TEAM Expectations

Participate

Engage

Take initiative

Respectful

Responsible

Communicate

Reflect, improve

Deliver

Considerations for your project IP

- Your team owns its work's intellectual property (IP)
- **If you leverage 3rd party software, including APIs, make sure you understand the license agreement**
- UW's comotion can help guide you if you want to commercialize your project or patent a novel idea <https://comotion.uw.edu/>



Have an idea, invention, or discovery?

CoMotion partners with the UW community on their innovation journey, providing tools, connections, and acumen to transform ideas into economic and societal impact.

Key times in your typical week

Tuesday

- **Team meeting - 1:30pm**
- Final review of current assignment
- **Assignment due 11:59pm**

Wednesday

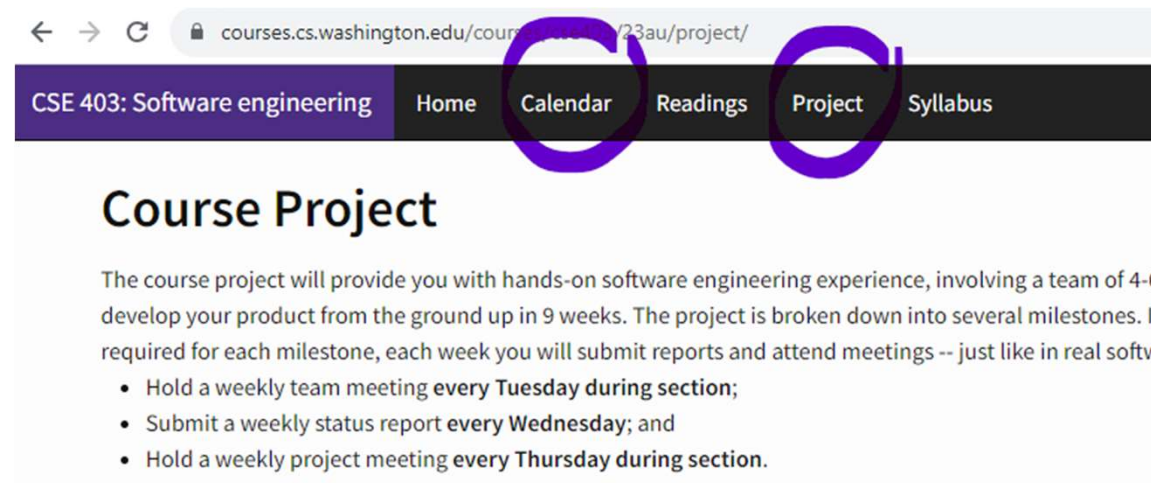
- Read next assignment and collect questions (leverage class discussion board or TA)
- Create status report and agenda
- **Status report and agenda due 8pm**

Thursday

- **Project meeting with product owner (TA) – 1:30pm**
- Get feedback on last assignment
- Resolve project-specific questions on current assignment
- Organize for next delivery

Time to roll!

- Project **weekly status report** info is posted on the Project tab
 - No report required this week (submissions start 10/18)
 - Use Thursday to discuss with your TA
- Project **requirements milestone** assignment is on the Calendar
 - Due Tuesday 10/17, 11:59pm



The screenshot shows a web browser window with the URL `courses.cs.washington.edu/course/23au/project/`. The navigation bar includes tabs for "CSE 403: Software engineering", "Home", "Calendar", "Readings", "Project", and "Syllabus". The "Calendar" and "Project" tabs are circled in purple. Below the navigation bar, the page title is "Course Project". The main content area contains the following text:

The course project will provide you with hands-on software engineering experience, involving a team of 4-6 develop your product from the ground up in 9 weeks. The project is broken down into several milestones. Ir required for each milestone, each week you will submit reports and attend meetings -- just like in real softw

- Hold a weekly team meeting every Tuesday during section;
- Submit a weekly status report every Wednesday; and
- Hold a weekly project meeting every Thursday during section.

The Joel Test

CSE 403 Software Engineering

Autumn 2023

Who is Joel?

Joel Spolsky

joelonsoftware.com

Avram Joel Spolsky is a software engineer and writer. He is the author of Joel on Software, a blog on software development, and the creator of the project management software Trello. He was a Program Manager on the Microsoft Excel team between 1991 and 1994. He later founded Fog Creek Software in 2000 and launched the Joel on Software blog. In 2008, he launched the Stack Overflow programmer Q&A site in collaboration with Jeff Atwood. Using the Stack Exchange software product which powers Stack Overflow, the Stack Exchange Network now hosts over 170 Q&A sites. [Wikipedia](#)



What is the Joel Test?

The Joel Test is:

- A checklist of 12 best practices good software teams do
- Written in a blog 20 (!) years ago by Joel Spolsky
- Overlaps with the SDLC concepts we discussed

Originally ...

- Scoring 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 some **hypothetical** software teams/companies and see how the practices played out in the real world
 3. Vote on which team/company has the best chance of success 😊
- Joel test references – quick reads:
 - <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>

The Joel Test

1. Do you use source control?

The Joel Test

1. Do you use source control?
2. **Can you make a build [+ release] in one step?**

The Joel Test

1. Do you use source control?
2. Can you make a build [+ release] in one step?
3. **Do you make daily builds?**

The Joel Test

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)?**

The Joel Test

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?**

The Joel Test

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?**

The Joel Test

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?**

The Joel Test

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?**

The Joel Test

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?**

The Joel Test

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?**

The Joel Test

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?**

The Joel Test

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?**

The Joel Test

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?**

The Joel Test

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?**

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?

Let's try out the test

1. 6 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?

<https://forms.gle/q5GZtMA6uy1qSpgd6>

CSE 403 Au23 - Joel's Test

alverson@cs.washington.edu [Switch account](#)

* Indicates required question

Startup Incubator team

How good are they doing regarding SW practices? *

Really bad 1 2 3 4 5 Really good

○ ○ ○ ○ ○

How much would you like to work in such an environment? *

Not at all 1 2 3 4 5 Top priority

○ ○ ○ ○ ○

[Back](#) [Next](#) [Clear form](#)

The Startup Incubator team (1/6)

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

(2.) One-step builds: Your team uses the 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 (2/6)

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 (3/6)

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 (4/6)

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 Trendy Startup team (5/6)

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, and management will prioritize new feature launches ahead of fixing

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 (6/6)

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 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?

And the survey says

So, what do we think of the Joel test?

1. Are these tests valid
2. Which are most/least important
3. Could we argue some are situational

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 (5/8)

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

(3.) No daily builds: and 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 Big Non-Tech Company team (8/8)

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?