CSE 403: Software Engineering, Spring 2015

courses.cs.washington.edu/courses/cse403/15sp/

The Joel Test: 12 Steps to Better Code

Emina Torlak
emina@cs.washington.edu

Outline

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 code?
6. Do you have an up-to-date schedule?
7. Do you have a spec?
8. Do you have quiet working conditions?
9. Do you use the best tools money can buy?
10. Do you have testers as part of the team?
11. Do you have interview candidates write code?
12. Do you do hallway usability testing?

www.joelonsoftware.com/articles/fog0000000043.html

Outline

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 code?
6. Do you have an up-to-date schedule?
7. Do you have a spec?
8. Do you have quiet working conditions?
9. Do you use the best tools money can buy?
10. Do you have testers as part of the team?
11. Do you have interview candidates write code?
12. Do you do hallway usability testing?

Score <= 10 usually means your team is in trouble!

Outline

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 code?
6. Do you have an up-to-date schedule?
7. Do you have a spec?

12. Do you do hallway usability testing?

Score <= 8 usually means your team is in trouble!

Do you use source control?

Do you use source control?

- Source control …
  - allows multiple developers to collaborate
  - keeps project in consistent state
  - tracks changes and enable roll-back
  - manages multiple versions
  - saves data in case of a disaster
  - is the authoritative source for “daily build”

Do you use source control?

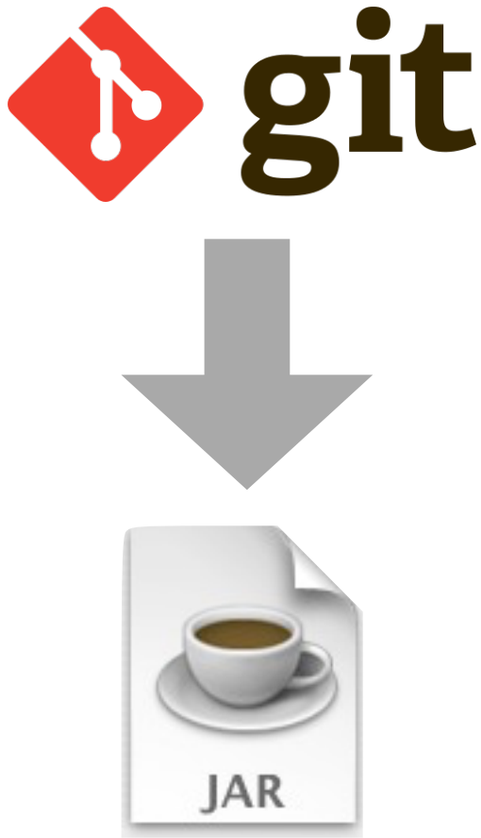
- Source control …
  - allows multiple developers to collaborate
  - keeps project in consistent state
  - tracks changes and enable roll-back
  - manages multiple versions
  - saves data in case of a disaster
  - is the authoritative source for “daily build”

The ZFR should indicate the state of your repository.

Can you make a build in one step?

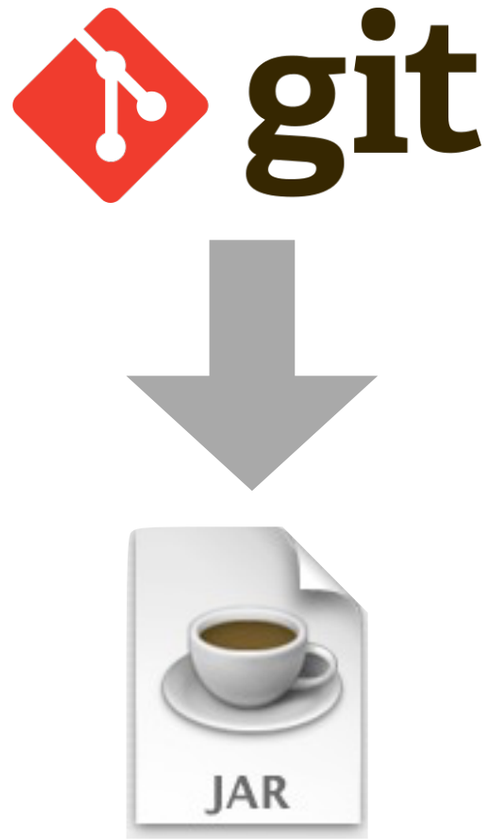
Can you make a build in one step?

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



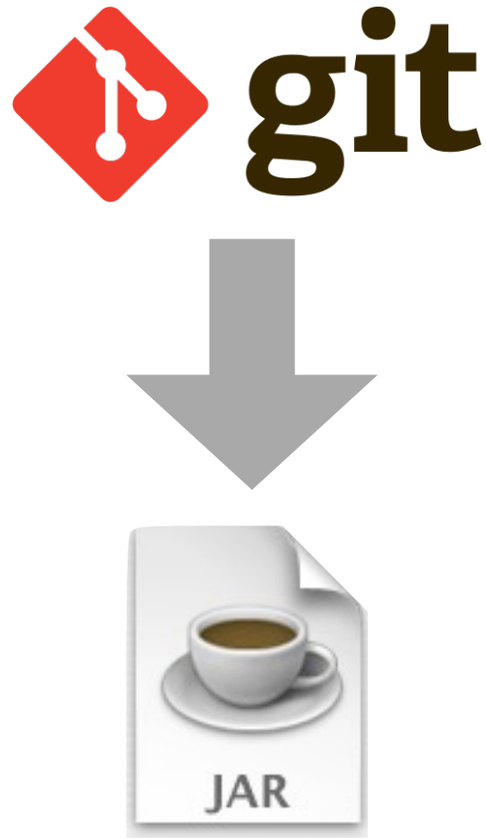
Can you make a build in one step?

- A single script that
  - [does a full checkout from scratch]
  - rebuilds every line of code
  - makes the binary executable files in all versions, languages and #ifdef combinations
  - [creates the installation package]
  - [creates the final media, web site, …]
- All steps are automated and exercised regularly



Can you make a build in one step?

- A single script that
  - [does a full checkout from scratch]
  - rebuilds every line of code
  - makes the binary executable files in all versions, languages and #ifdef combinations
  - [creates the installation package]
  - [creates the final media, web site, …]
- All steps are automated and exercised regularly
- So, why is this valuable?



Do you make daily builds?

Do you make daily builds?

- Build the entire product every day and run a good test suite against the new version
    - build from checked in sources
    - automatic and frequent

Do you make daily builds?

- 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: find out early that you've got problems and fix them before disaster strikes

Do you make daily builds?

- 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: find out early that you've got problems and fix them before disaster strikes

- Benefits
    - minimizes integration risk
    - reduces risk of low quality
    - supports easier defect diagnosis
    - improves morale for developers, managers, customers

Do you make daily builds?

- 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: find out early that you've got problems and fix them before disaster strikes

- Benefits
    - minimizes integration risk
    - reduces risk of low quality
    - supports easier defect diagnosis
    - improves morale for developers, managers, customers

The ZFR must include your build script or sequence.

Do you have a bug database?

Do you have a bug database?

- You can’t keep the bug list in your head
    - especially with multiple developers and multiple customers

Do you have 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!

Do you have 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

Do you have 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.

Do you have 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.

For the beta release assignment, we'll be asking to see a log of your bugs.

Do you fix bugs before writing new code?

Do you fix bugs before writing new code?

- Why not fix them later?

Do you fix bugs before writing new code?

- Why not fix them later?
    - Familiar with the code now

Do you fix bugs before writing new code?

- Why not fix them later?
    - Familiar with the code now
    - Harder to find (and fix) later

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…)

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

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

Do you have an up-to-date schedule?

Do you have an up-to-date schedule?

- Keeps expectations realistic
    - For the team, customers, stakeholders

Do you have an up-to-date schedule?

- Keeps expectations realistic
    - For the team, customers, stakeholders
- Allows for more accuracy
    - Use experience to improve estimates

Do you have an up-to-date schedule?

- Keeps expectations realistic
    - For the team, customers, stakeholders
- Allows for more accuracy
    - Use experience to improve estimates
- Helps prevent feature creep
    - Don’t take on anything without checking the schedule first

Do you have an up-to-date schedule?

- Keeps expectations realistic
    - For the team, customers, stakeholders
- Allows for more accuracy
    - Use experience to improve estimates
- Helps prevent feature creep
    - Don’t take on anything without checking the schedule first

For the SDS, we asked for a schedule. For later releases, we ask you to highlight any changes, and keep all documents up to date.

Do you have a spec?

Do you have a spec?

- Easier to fix problems at the design stage

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

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

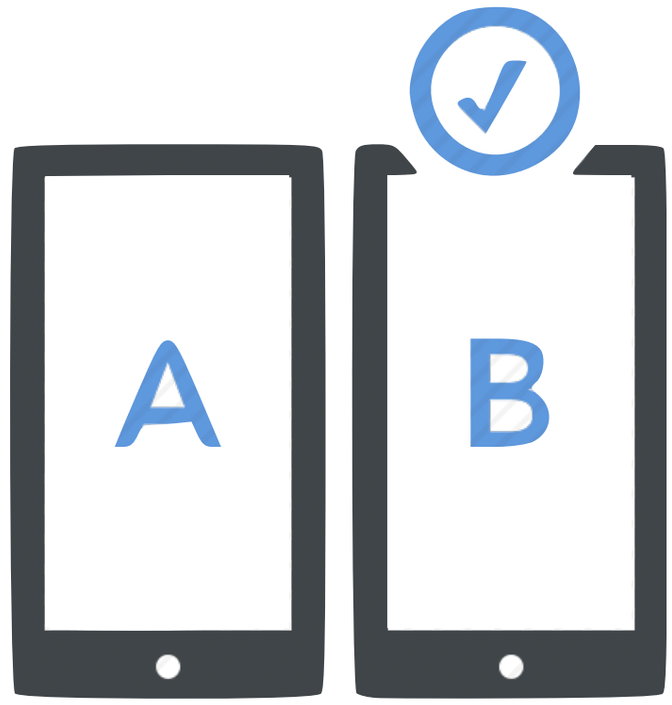
Do you have a spec?

- 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

Do you have a spec?

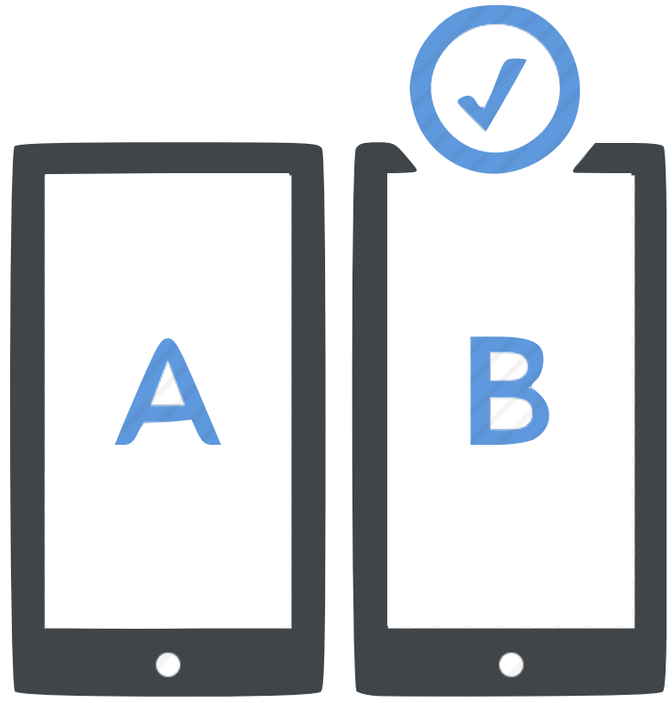
- Easier to fix problems at the design stage
- You know what you are trying to build
    - So do your teammates and customer
- More likely that you build the right thing
    - Pieces fit together
    - Customer is satisfied
- Conceptual integrity for your project
- Undocumented code has low value
    - Hard to maintain and to extend
    - Hard to bring new developers on board

Do you have hallway usability testing?



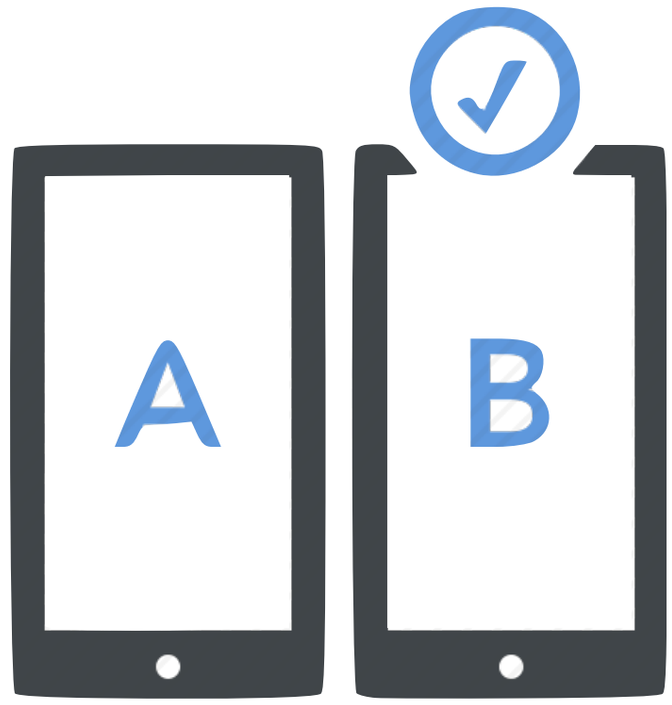
Do you have hallway usability testing?

- Grab someone in the hallway and make them use your code



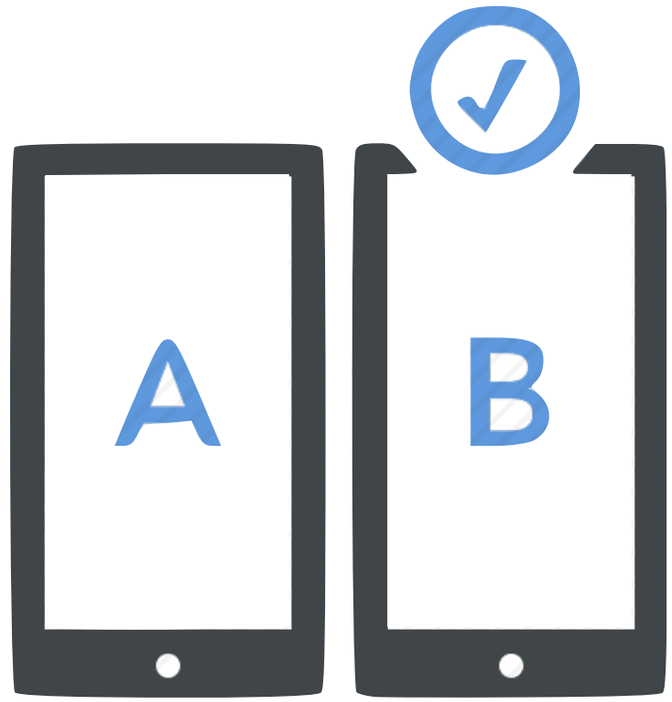
Do you have hallway usability testing?

- Grab someone in the hallway and make them use your code
- Key idea: get feedback fast



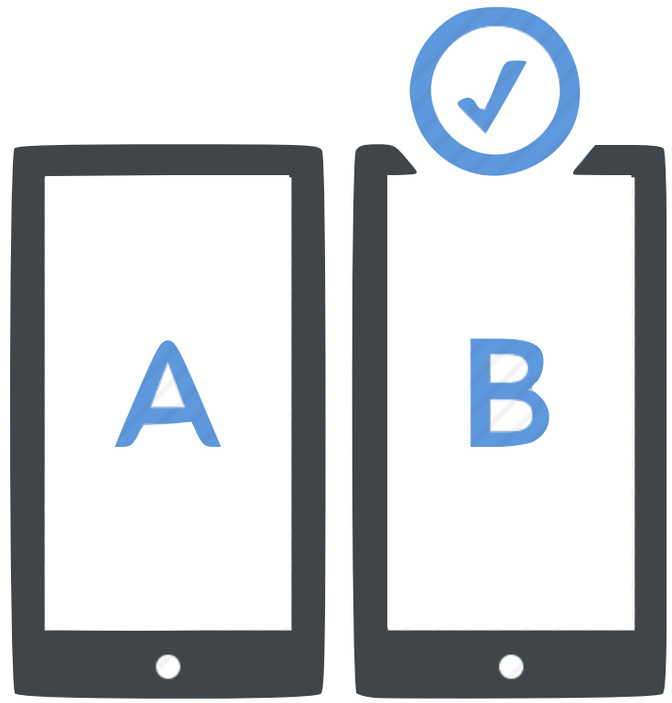
Do you have hallway usability testing?

- Grab someone in the hallway and make them use your code
- Key idea: get feedback fast
- A little feedback now ≫ lots of feedback later



Do you have 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



Joel's disclaimer

Joel's disclaimer

- These are not the only factors that determine success or failure
    - A great team will not help if you are building a product no one wants
    - An incredibly talented team might produce an incredible product without these guidelines

Joel's disclaimer

- These are not the only factors that determine success or failure
    - A great team will not help if you are building a product no one wants
    - An incredibly talented team might produce an incredible product without these guidelines
- But all things being equal, these factors indicate a disciplined team that can consistently deliver

Joel's disclaimer

- These are not the only factors that determine success or failure
    - A great team will not help if you are building a product no one wants
    - An incredibly talented team might produce an incredible product without these guidelines
- But all things being equal, these factors indicate a disciplined team that can consistently deliver

> "The bummer about The Joel Test is that you really shouldn't use it to make sure that your nuclear power plant software is safe."

Making mission-critical software safe ...

Gerard J. Holzmann.
Mars code, 2014.

Making mission-critical software safe ...

- First, standard precautions for reducing risk in complex software systems:

Gerard J. Holzmann.
Mars code, 2014.

Making mission-critical software safe …

- First, standard precautions for reducing risk in complex software systems:
    - A good software architecture with a clean separation of concerns, data hiding, modularity, well-defined interfaces, and strong fault-protection mechanisms.

Gerard J. Holzmann.
Mars code, 2014.

Making mission-critical software safe …

- First, standard precautions for reducing risk in complex software systems:
    - A good software architecture with a clean separation of concerns, data hiding, modularity, well-defined interfaces, and strong fault-protection mechanisms.
    - A good development process, with clearly stated requirements, requirements tracking, daily integration builds, rigorous unit and integration testing, and extensive simulation.

Gerard J. Holzmann.
Mars code, 2014.

Making mission-critical software safe ...

Gerard J. Holzmann.
Mars code, 2014.

Making mission-critical software safe …

- Risk-based coding rules
  - Six compliance levels.
  - Level 3: "We require that the flight software as a whole, and each module within it, had to reach a minimal assertion density of 2%."

Gerard J. Holzmann.
Mars code, 2014.

Making mission-critical software safe …

- Risk-based coding rules
  - Six compliance levels.
  - Level 3: "We require that the flight software as a whole, and each module within it, had to reach a minimal assertion density of 2%."

Gerard J. Holzmann.
Mars code, 2014.

- Tool-based code review
  - Peer review great at discovering design flaws.
  - But tools are better at discover coding flaws.

Making mission-critical software safe …

- Risk-based coding rules
  - Six compliance levels.
  - Level 3: "We require that the flight software as a whole, and each module within it, had to reach a minimal assertion density of 2%."

Gerard J. Holzmann.
Mars code, 2014.

- Tool-based code review
  - Peer review great at discovering design flaws.
  - But tools are better at discover coding flaws.
- Formal methods
  - Used for critical software and hardware components.
  - Provides high assurance but requires expertise, time.

Summary

The Joel Test for 403:

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 code?
6. Do you have an up-to-date schedule?
7. Do you have a spec?
8. Do you do hallway usability testing?

www.joelonsoftware.com/articles/fog0000000043.html