CSE403 ● Software engineering ● sp12

Week 7-10				
Monday	Tuesday	Wednesday	Thursday	Friday
• Joel test & interviewing • No reading	• Groups	• Reviews	• Section	• Progress report due • Readings out
• Reading due	• Groups • Beta due		• Section	• Progress report due • Readings out
• No reading due	• Groups	• Midterm II • Reading covered [Notkin gone]	• No section	• Progress report due
Memorial Day Holiday	• Groups	• Final release due • **Project Pres. I**	• **Project Pres. II**	• **Project Pres. III**

Beta

- Continue along ZFR, adding function as planned

The Joel Test

http://www.joelonsoftware.com/articles/fog0000000043.html

The Joel Test

In general, a score of <= 10 means you're in trouble.

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?

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”

The ZFR should indicate the state of your repository.

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 and #ifdef combinations
  - [creates the installation package]
  - [creates the final media - CDROM, web site, …]
- All steps are automated and exercised regularly
- So, why is this valuable?

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
  - 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 - 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

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

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?

- 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

Highlight any changes, and keep all documents up to date.

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 no commercial value

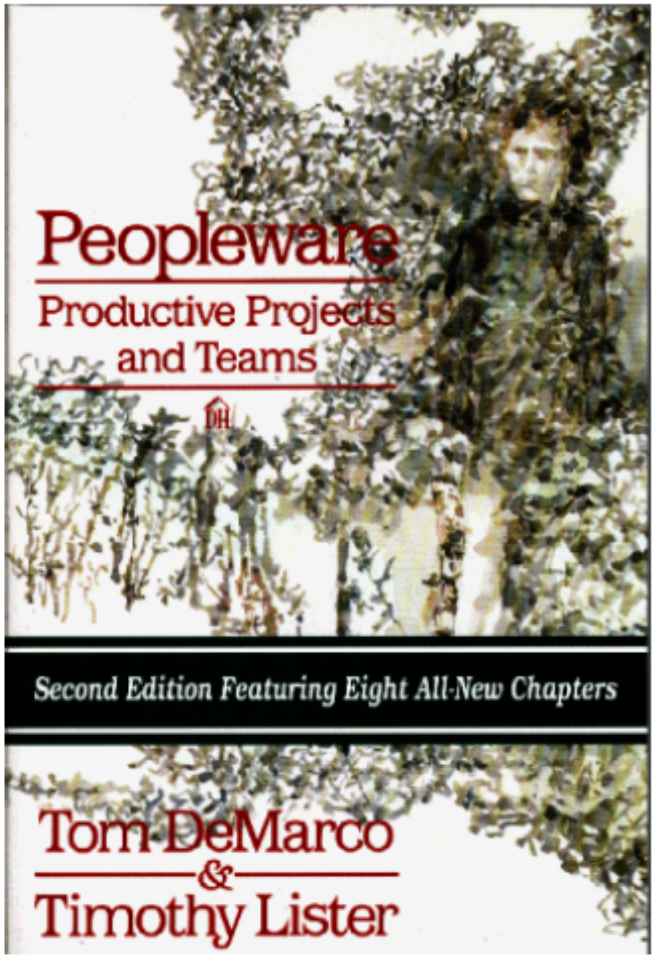
Do you do hallway usability testing?

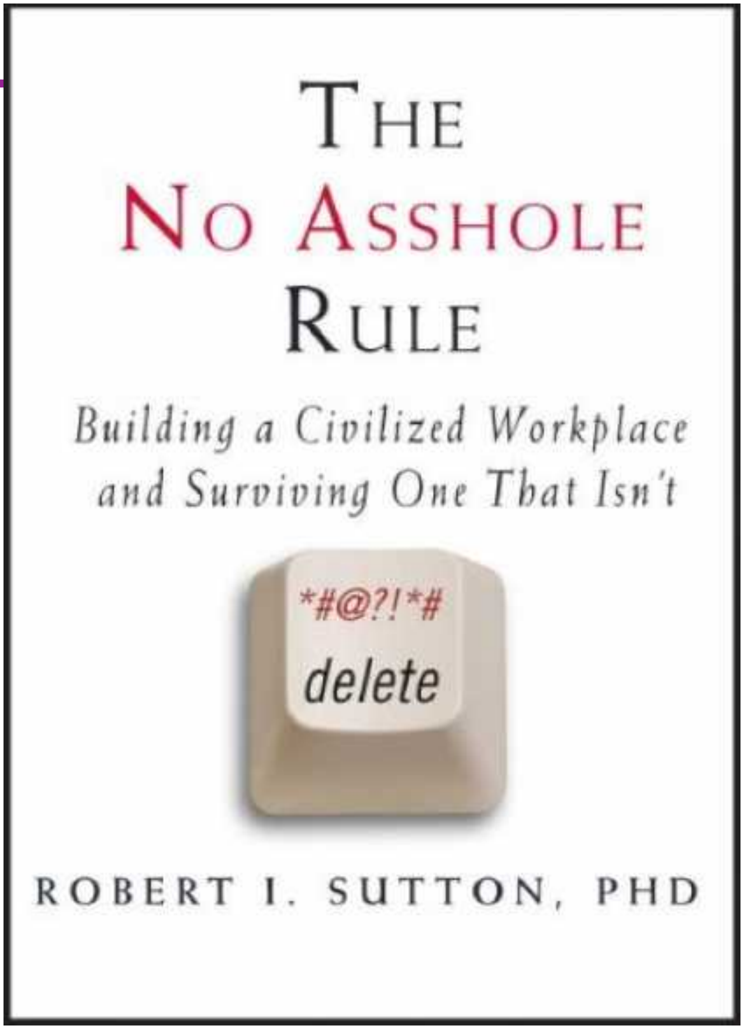
- Grab someone in the hallway and make them use your code
- Key idea: get feedback *fast*
- A little feedback now $\gg$ lots of feedback later
- You will get most of the valuable feedback from the first few users
  - Brooks: you get infinite utility from your first user

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

Other advice


Peopleware
Productive Projects and Teams
Second Edition Featuring Eight All-New Chapters
Tom DeMarco & Timothy Lister



Interviewing

"My short-term goal is to bluff my way through this job interview. My long-term goal is to invent a time machine so I can come back and change everything I've said so far."

The Basics

- Show up on time.
- Look professional.
  - Better than the average employee.
- Be awake.
- Plan ahead to make sure you know where you are going and who to ask for.

Before the Interview

- Research the position.
  - BAD: a ‘dev’ writes code, a ‘tester’ tests it, and a ‘manager’ keeps them on track.
  - Research exactly what that position means at that company.
- Study Computer Science basics
- Plan answers to ‘common questions’
- Practice!
  - It’s a lot more difficult than you might think.
  - Practice on paper and/or a white board. Don’t use an IDE.
- Do general practice problems.
  - Use online suggestion sites (glassdoor.com) and practice problems.

Computer Science ‘basics’

- Know the details about the language you will be using.
  - For Java: static, implements, extends, garbage collector, classes, methods, objects, etc.
- Know algorithms and data structures (you will be asked about these):
  - Quick sort, Merge sort, Tree traversals (BST especially for sorting)
  - Depth/breadth first search.
  - Arrays/ArrayLists/Linked Lists/Etc.
    - Insert/delete/search → efficiency of each?
  - HashMaps (or other Hash object)
    - Why it’s useful
    - Insert/delete/search → efficiency of each?
  - Properties of different variable types
    - Read the javadoc on Strings.

Technical problems to expect

- They’re meant to make you think (and talk).
- Rarely will the problem be obvious/blatant:
  - “Write me a program that returns the 5th character in a string.”
- The problems are designed to have more than one correct answer.
  - “If you have an array of integers, give me back a list of all integers which appear more than once.”

Answering a technical problem

- Do not obsess about finding the best answer right away. (they don't care)
  - Start with an obvious solution, write it out, and then improve on it.
- TALK!!!
  - You're code is *not* important.
- Plan it out first, don't just start coding.
- Ask tons of questions, even if they may be obvious.
  - Clarify the prompt.

Non-technical questions

- They will ask about you:
  - Experiences you've had.
  - Lessons you've learned.
  - Challenges you've overcome.
  - Etc.
- Assume they have read your resume, transcript, and application.
  - Reference to it if that will help your answer.
- Come prepared with a few 'stories' about your Computer Science history.

Now it’s your turn

- Come with questions.
- Ask for advice.
- Use this time to tell them anything extra about you that hasn’t come up.
  - If you have a great story, find a way to segue to it.
- This is their time to sell their company to you.

Final Notes

- Try to stay relaxed.
  - Be a nice and interesting person, beyond your code.
- One bad interview won't end your chances.
- They want to help you, let them do so!
  - Ask tons of questions.
  - Ask for advice.
- If you've gotten to the interview, you're already doing well!
- Look back at these slides, and find what you're missing.

After the Interview

- If you don’t hear back within a week or two, send them an email to remind them of you.
- Don’t take it personally if you don’t get an offer.
- You can ask for more time to make a decision (within reason)
  - You can also ask them to hurry up if you have another time constraint.
- If you don’t like the team they place you on, you can ask for another position.
  - BE POLITE!