Ending an Iteration & On to the Next One

Content from Chapters 9 & 10 of "Head First Software Development", Pilone et al.

Miami University Software Technology & Analysis Group (MUSTANG)
Computer Science & Software Engineering
Miami University, Oxford, Ohio, USA

Questions From Last Class?

- Bugs
  - Definition
- Bug fixing as an activity
  - What do we fix?
  - What do we not?
  - Spike testing?
- Abuse cases vs. Use cases
  - Abuse case purpose
  - Example

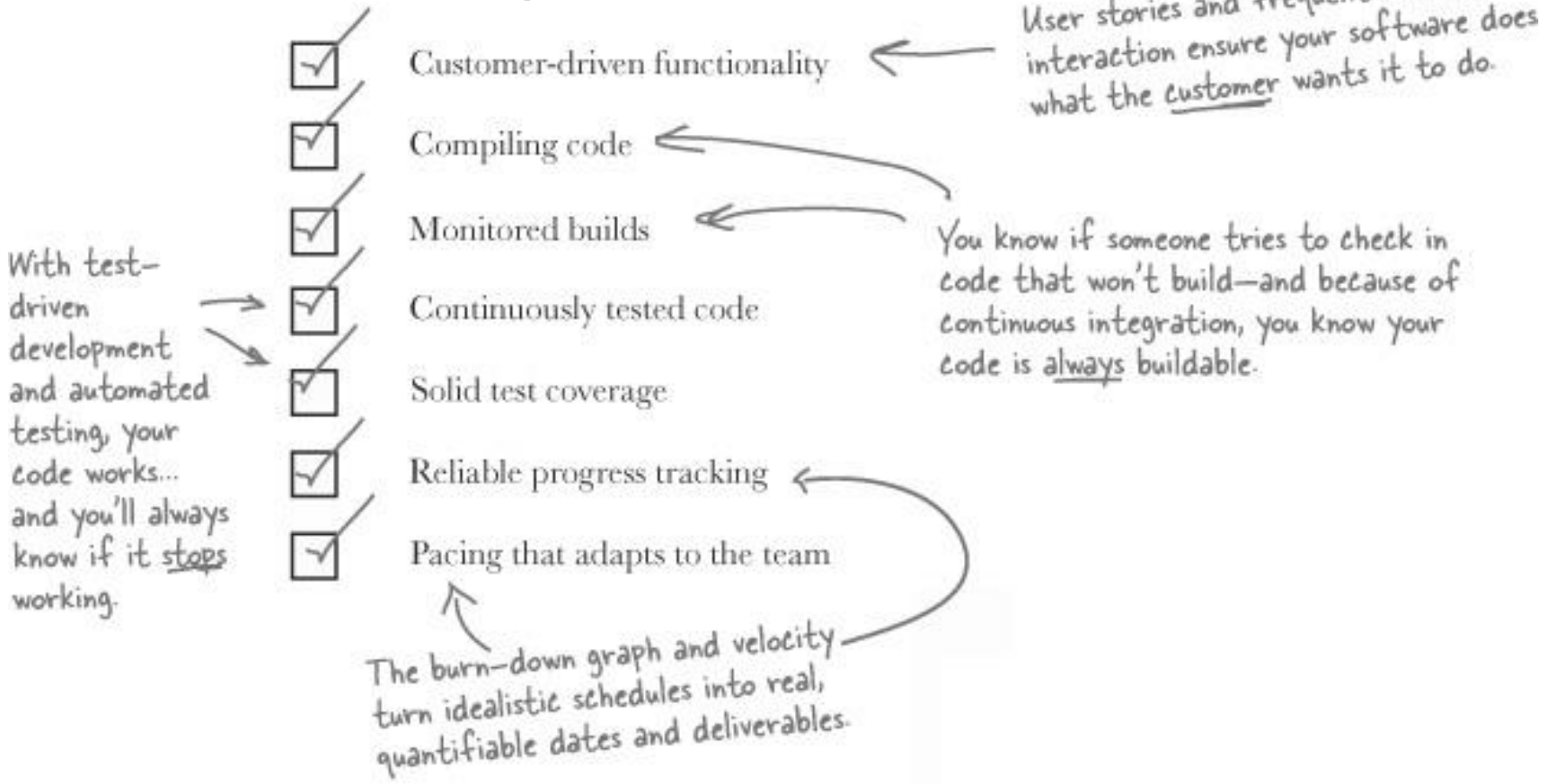
PROJECT'S DUE DATES ?

- ~~April 2nd → 5th week deliverables~~
- ~~April 6th → Iteration 1 presentation → weeks 3 + 4 → 90 mints~~
- ~~April 16th → 6th week deliverables~~
- ~~April 20th → Iteration 2 presentation → weeks 5 + 6 → 90 mints~~
- ~~April 23rd → 7th week deliverables~~
- April 30th → 8th week deliverables
- May 4th → Iteration 3 (final) presentation → weeks 7 + 8 → 90 mints

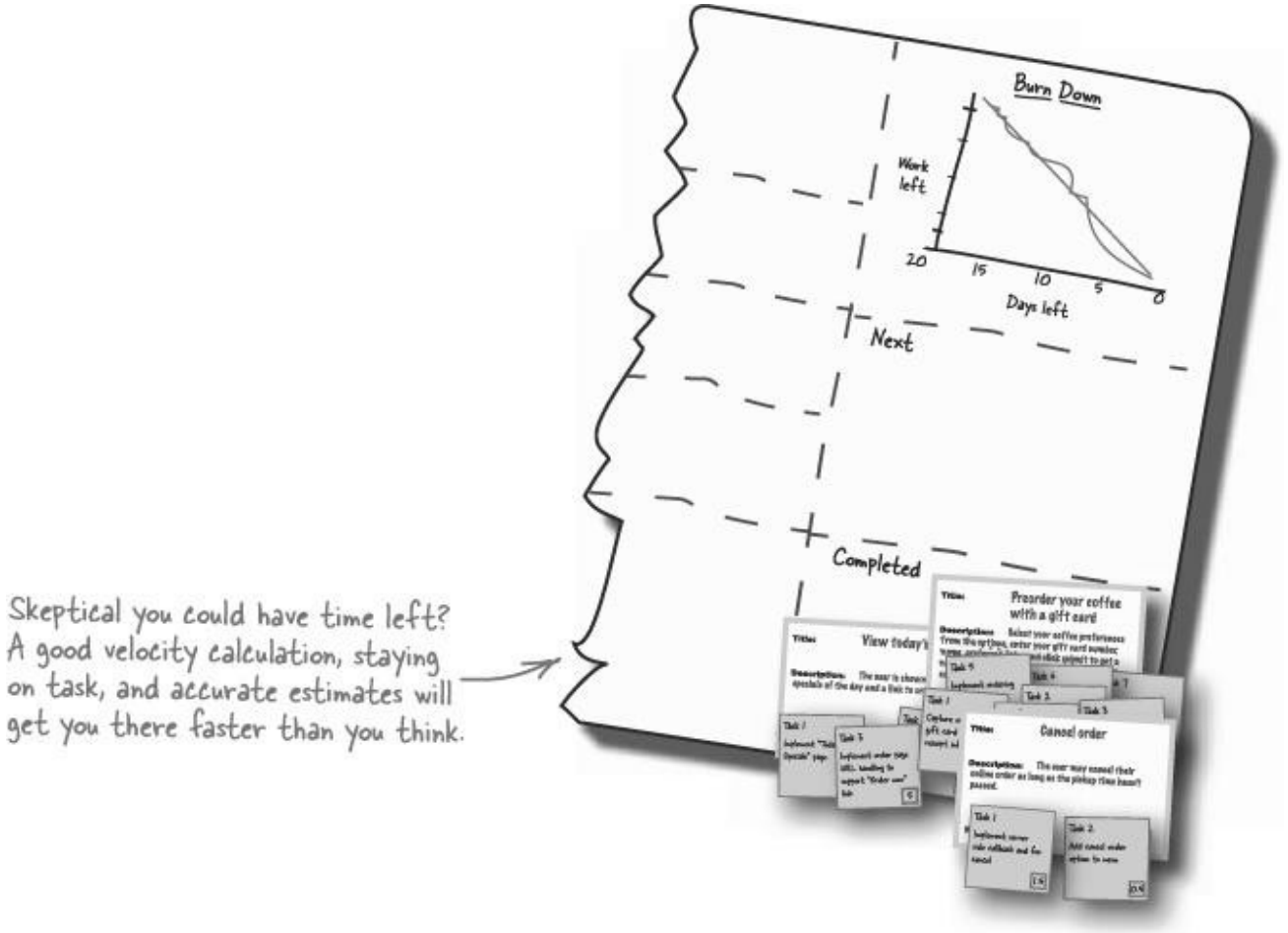
Ending An Iteration?

- You're almost finished!
- The team's been working hard, and things are wrapping up.
- Your tasks and user stories are complete, but what's the best way to spend that extra day you ended up with?
  - Where does user testing fit in?
  - Can you squeeze in one more round of refactoring and redesign?
  - And there sure are a lot of lingering bugs...when do those get fixed?
- It's all part of the end of an iteration

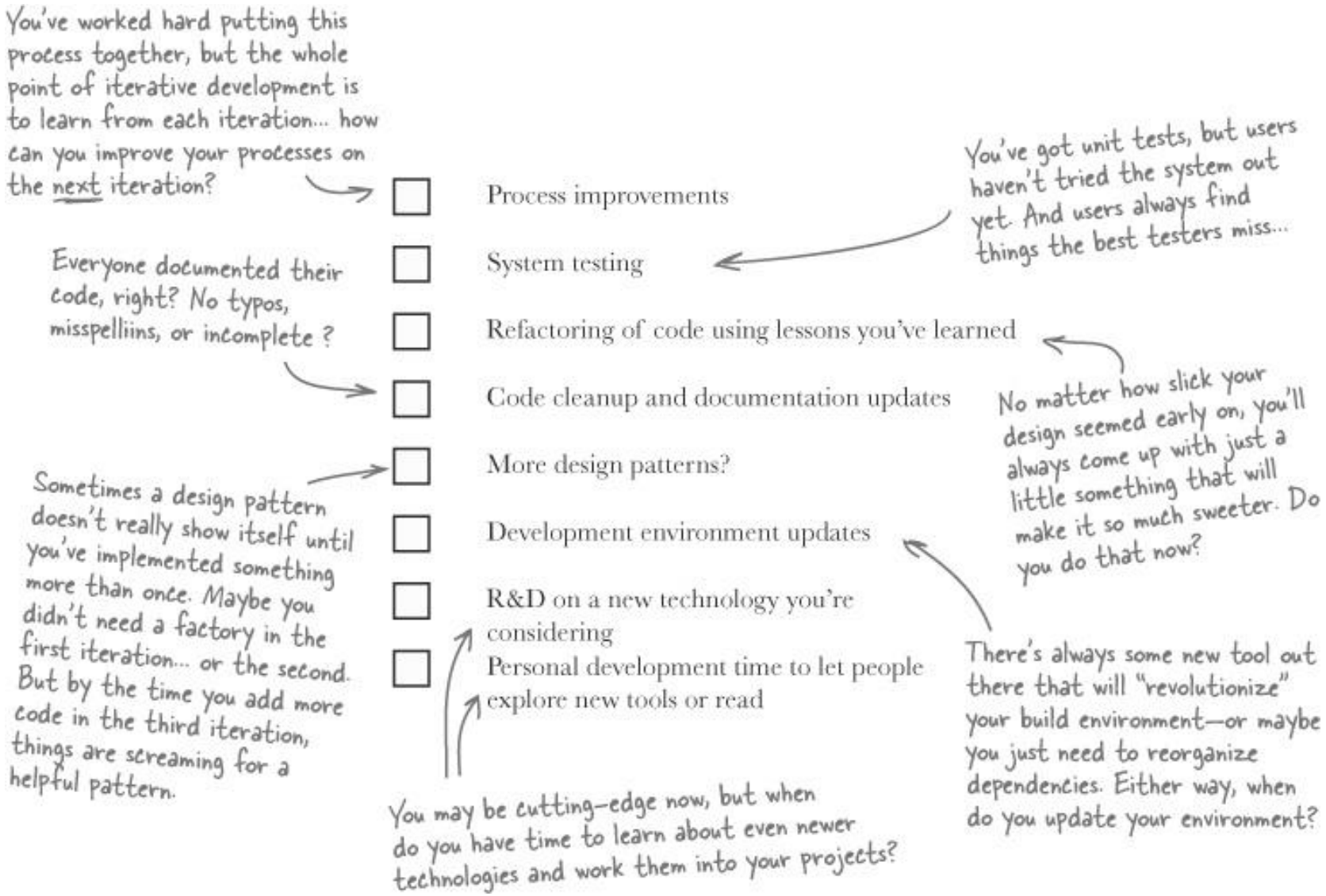
ITERATION IS ALMOST COMPLETE



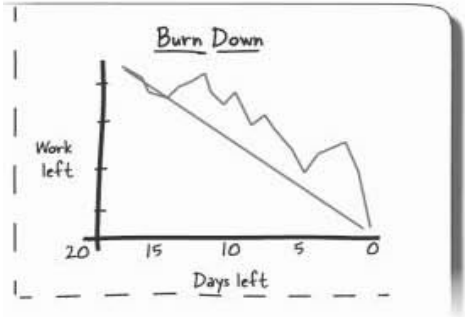
LEFTOVER TIME? … IT CAN HAPPEN BECAUSE...

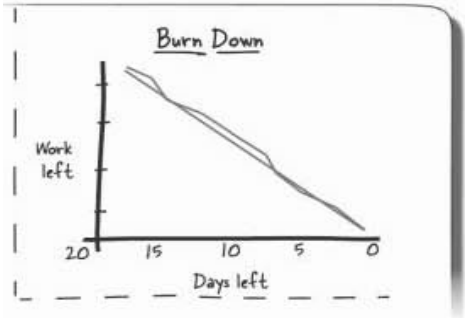


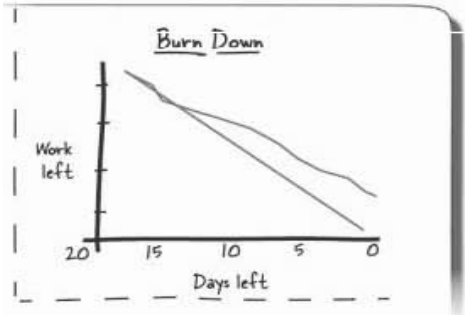
Always Stuff To Do

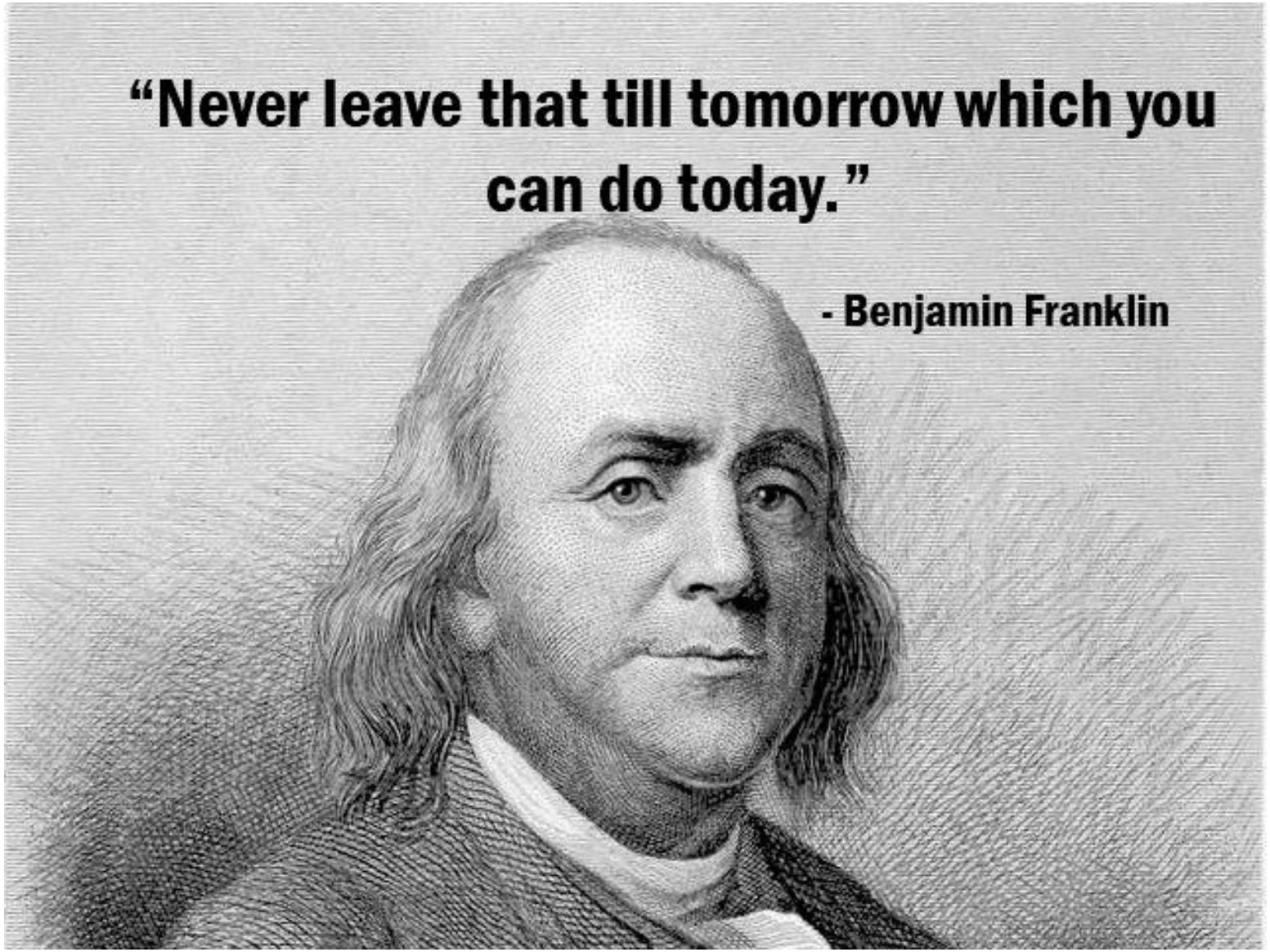


EXPLAIN THE SITUATION IN EACH GRAPH







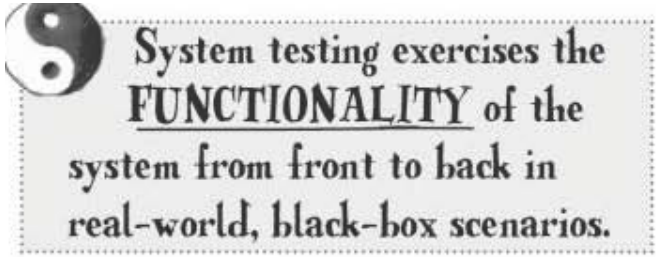

“Never leave that till tomorrow which you can do today.”
- Benjamin Franklin

System Testing

- Your system has to work, and that means using the system.
- So you've got to either have
  - a dedicated end-to-end system testing period,
  - or you actually let the real users work on the system (even if it's on a beta release).
- No matter which route you go, you've got to test the system in a situation that's as close to real-world as you can manage.
- That's called **system testing**, and it's all about reality, and the system as a whole, rather than all its individual parts.

What About Our Unit Tests? Aren't Those Enough?

- So far, we've been **unit testing**.
  - Our tests focus on small pieces of code, one at a time, and deliberately try to isolate components from each other to minimize dependencies.
  - This works great for automated test suites, but can potentially miss bugs that only show up when components interact, or when real, live users start banging on your system.
- And that's where **system testing** comes in: hooking everything together and treating the system like a black box.



Who Does System Testing?

- You should try your best to have a **different set of people** doing your system testing.
- It's not that developers aren't really bright people; it's just that dedicated testers bring a testing mentality to your project.
- Never system test your own code.
  - You know it too well to be unbiased.

Testing Done By Developers

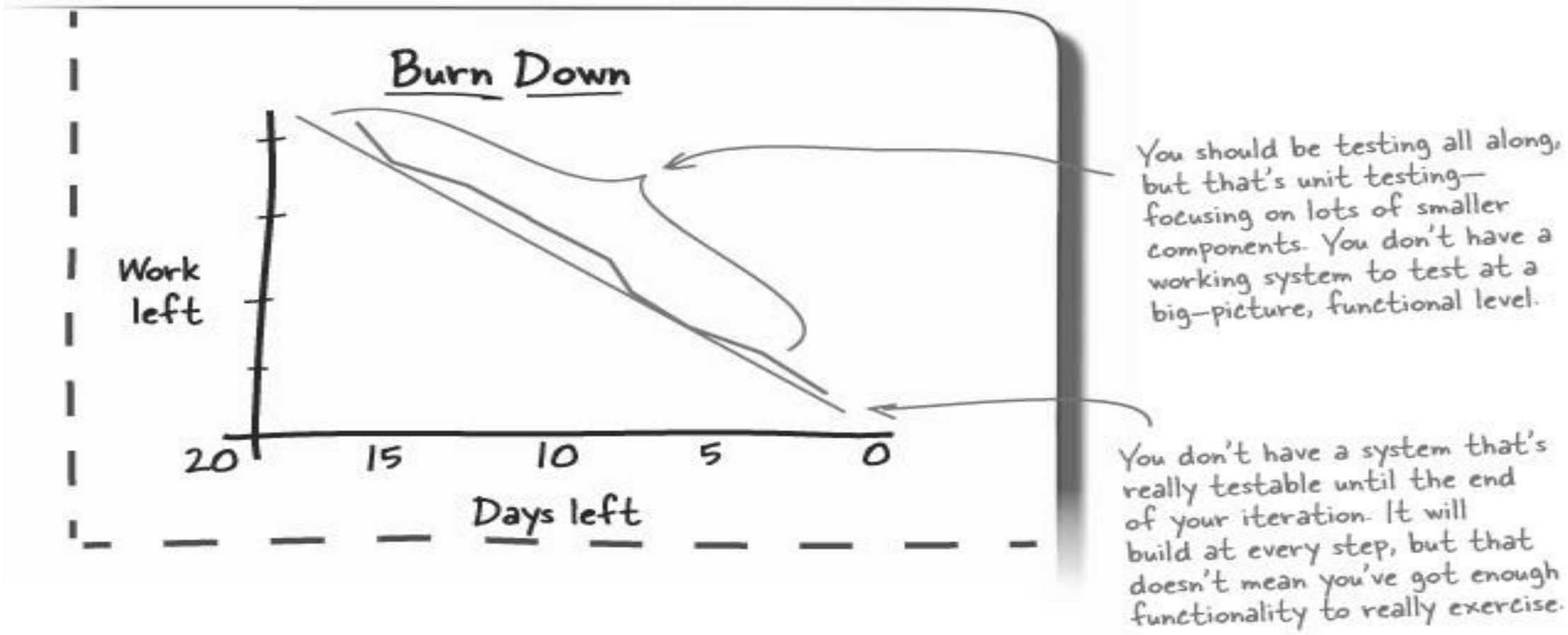
- Developers come preloaded with lots of knowledge about the system and how things work underneath.

- No matter how hard they try, it's really tough for developers to put themselves in the shoes of end users when they use the system.

Testing Done By Testers

- Testers can often bring a fresh perspective to the project. They approach the system with a fundamentally different view.
- They're trying to find bugs. They don't care how slick your multithreaded, templated, massively parallel configuration file parser is.
- They just want the system to work.

YOU STILL NEED A "SYSTEM" TO TEST



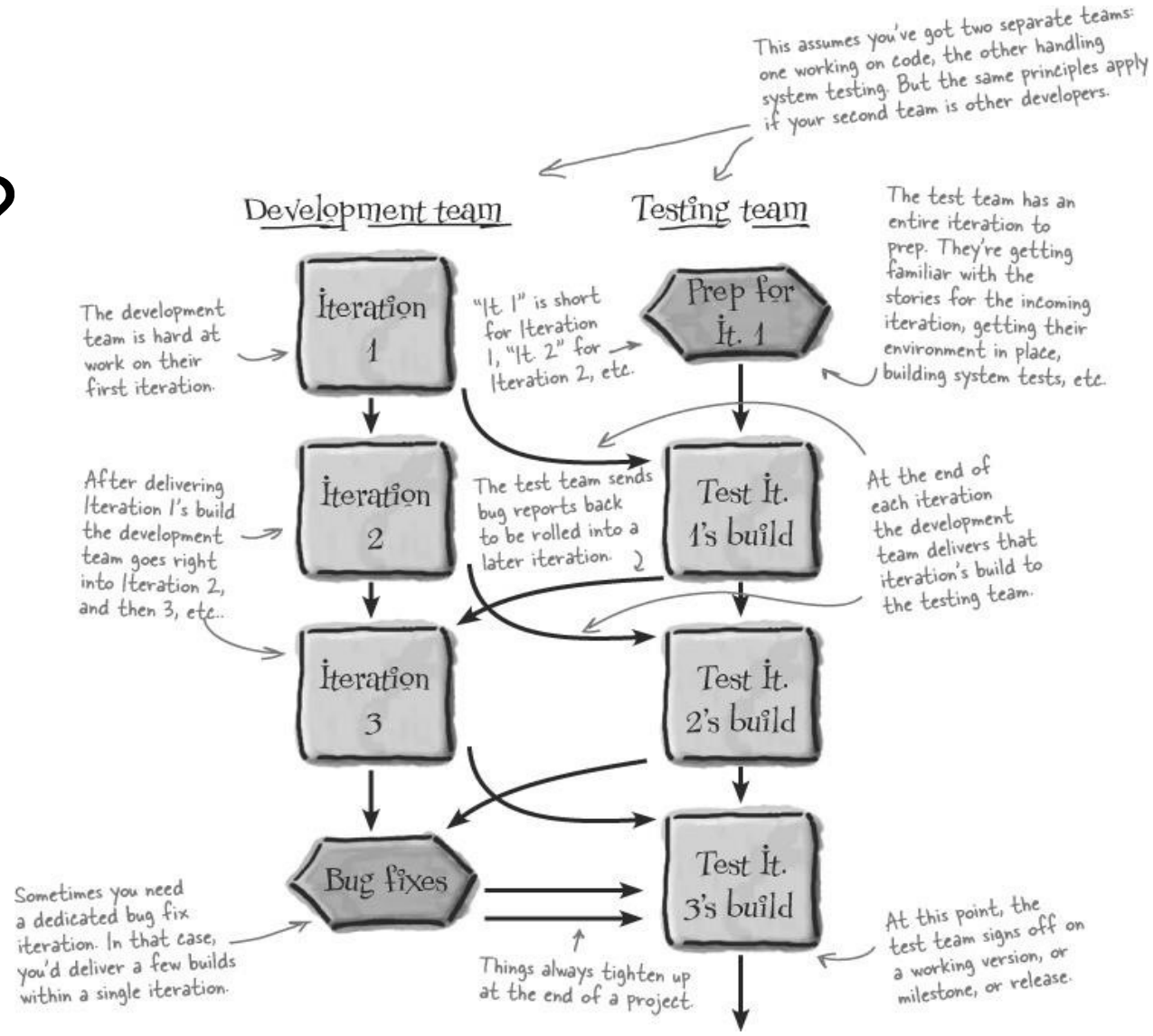
WHO'S DOING WHAT?

- Unit testing? And when?
- System testing? And when?

How Many Cycles For System Testing?

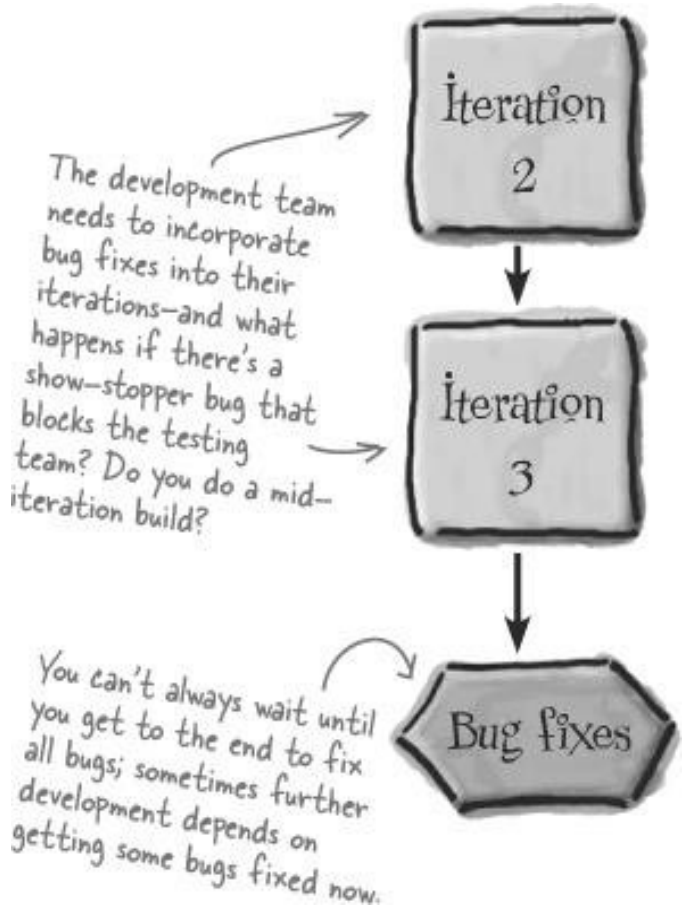
- **Good system testing requires TWO iteration cycles**
- Iterations help
  - your team stay focused,
  - deal with just a manageable amount of user stories, and
  - keep you from getting too far ahead without built-in checkpoints with your customer.
- But you need all the same things for good system testing.
  - So what if you had two cycles of iterations going on?

How Many Cycles For System Testing?



Fix Bugs While You Work

- Development team may find out about bugs a few iterations later.
- Determine if it's important enough to fix now, or put off till later.
- Treat bugs like stories.
  - Prioritize and bump lower-priorities.

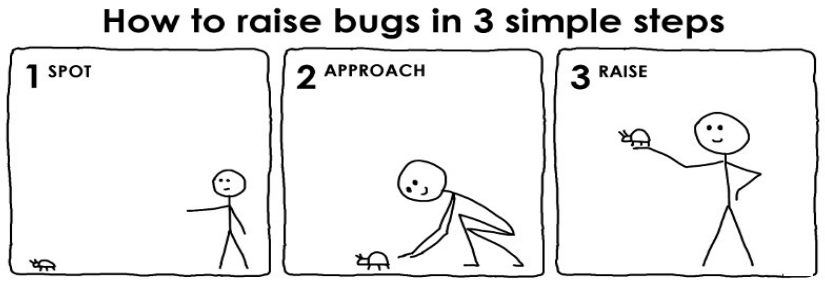


Could Dedicate Some Time

Another approach is to carve off a portion of time every week that you'll dedicate to bug fixes.

Take this off of the available hours when you do your iteration planning, so you don't need to worry about it affecting your velocity.

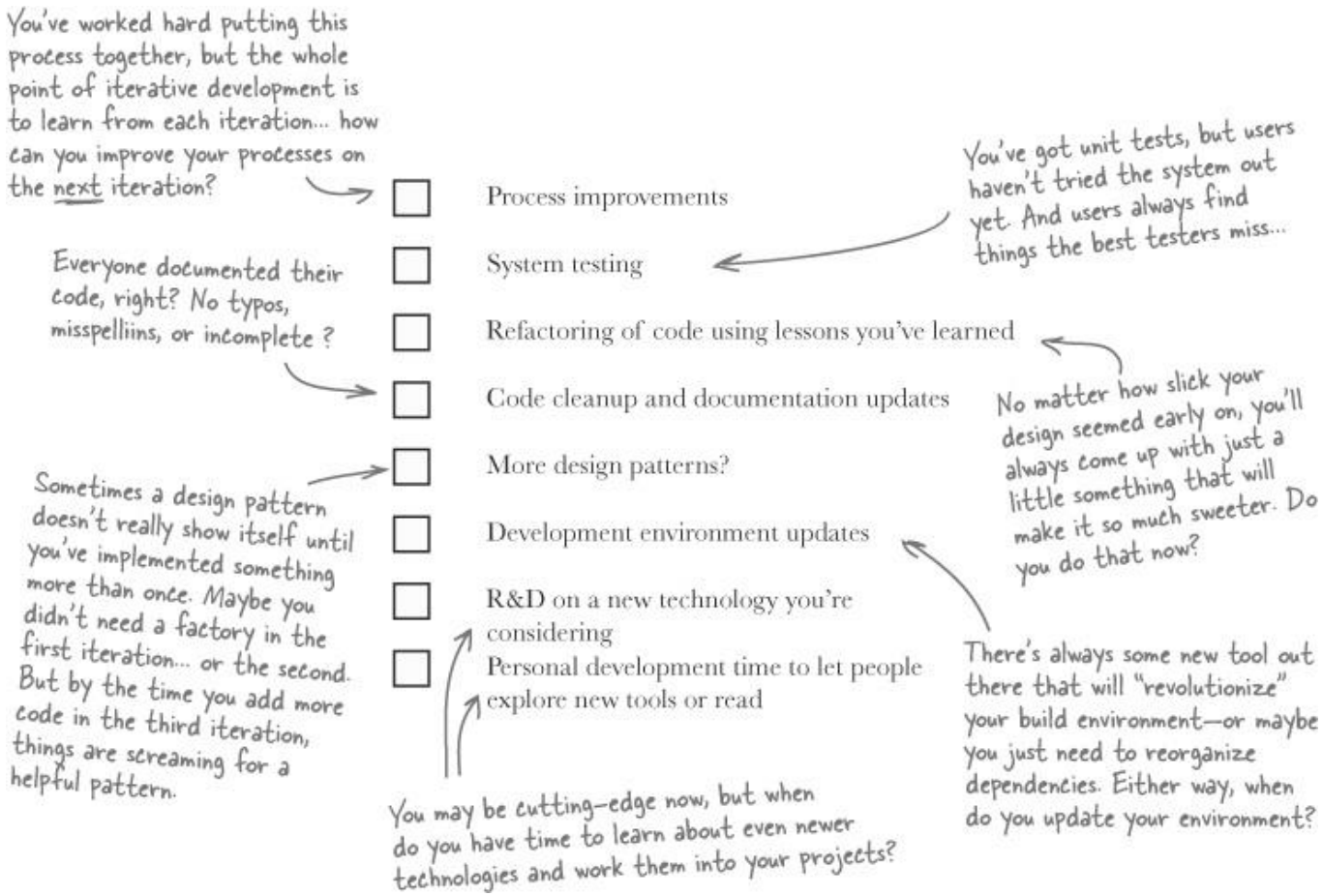
For example, you could have everyone spend one day a week on bug fixes—about 20% of their time



COMMUNICATION IS KEY!

- We now have more of the same issues
  - multiple team members
  - your customer changing requirements and user stories
  - priorities of different pieces of functionality
  - having to guess at what you're going to build before your requirements are complete
- The key to most problems you'll run into in software development is **COMMUNICATION**. When in doubt, TALK to your team, other teams, and your customer.

STILL MORE THINGS TO DO



Iteration Review

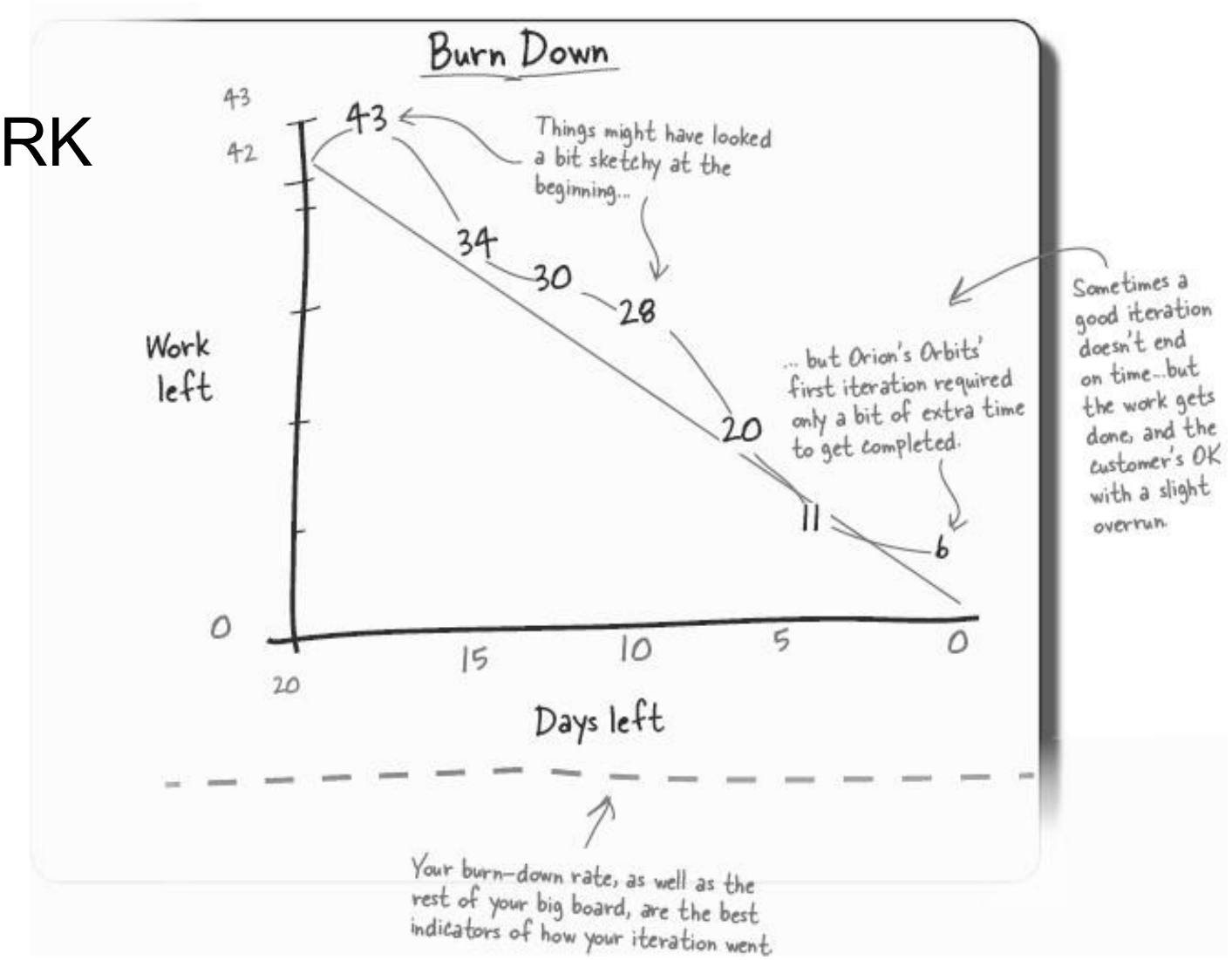
- You're remaining work is at zero, you've hit the last day of the iteration, and it's time to start getting ready for the next iteration.
- But, before you prioritize your next stories, remember:
  - it's not just software we're developing iteratively.
    - You should develop and define your process iteratively, too.
- At the end of each iteration, take some time to do an iteration review with your team.
  - Everyone has to live with this process so make sure you incorporate their opinions.

Starting The Next Iteration

- Software development is all about change, and moving to your next iteration is no exception.
- For example,
  - You've got to rebuild your board and adjust your stories and expectations based on what the customer wants NOW, not a month ago.

- Reminder
  - What is working software?
    - It's more than just code.

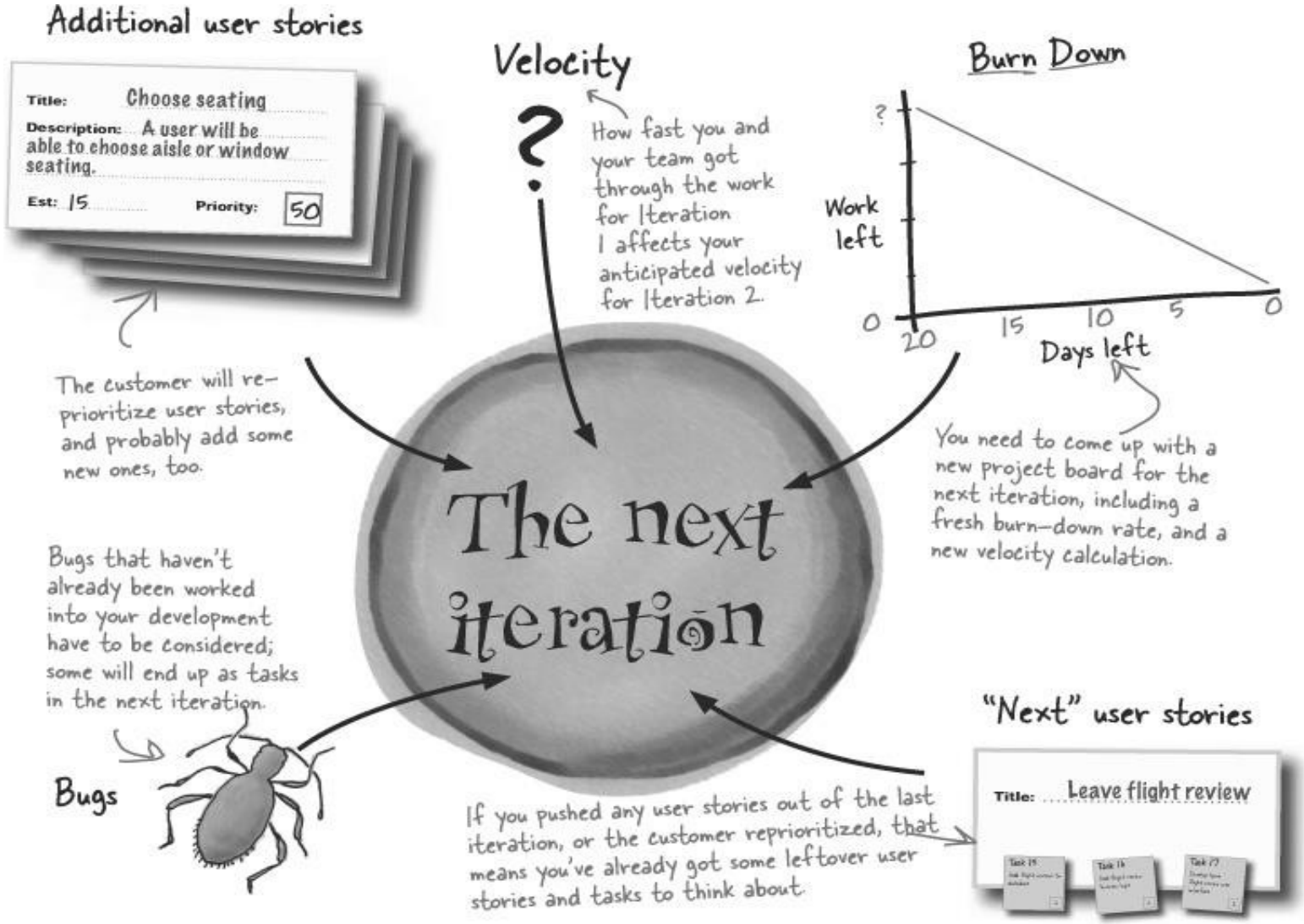
It's Completing Your Iteration's Work



PASSING ALL YOUR TESTS

SATISFYING YOUR CUSTOMER

TIME TO PLAN!


Additional user stories
Title: Choose seating
Description: A user will be able to choose aisle or window seating.
Est: 15
Priority: 50
Velocity
?
How fast you and your team got through the work for Iteration 1 affects your anticipated velocity for Iteration 2.
Burn Down
Work left
Days left
20
15
10
5
0
The customer will re-prioritize user stories, and probably add some new ones, too.
You need to come up with a new project board for the next iteration, including a fresh burn-down rate, and a new velocity calculation.
The next iteration
Bugs that haven't already been worked into your development have to be considered; some will end up as tasks in the next iteration.
Bugs
"Next" user stories
Title: Leave flight review
If you pushed any user stories out of the last iteration, or the customer reprioritized, that means you've already got some leftover user stories and tasks to think about.

QUESTIONS YOU MAY BE ASKING IN YOUR PROJECT

ADJUST

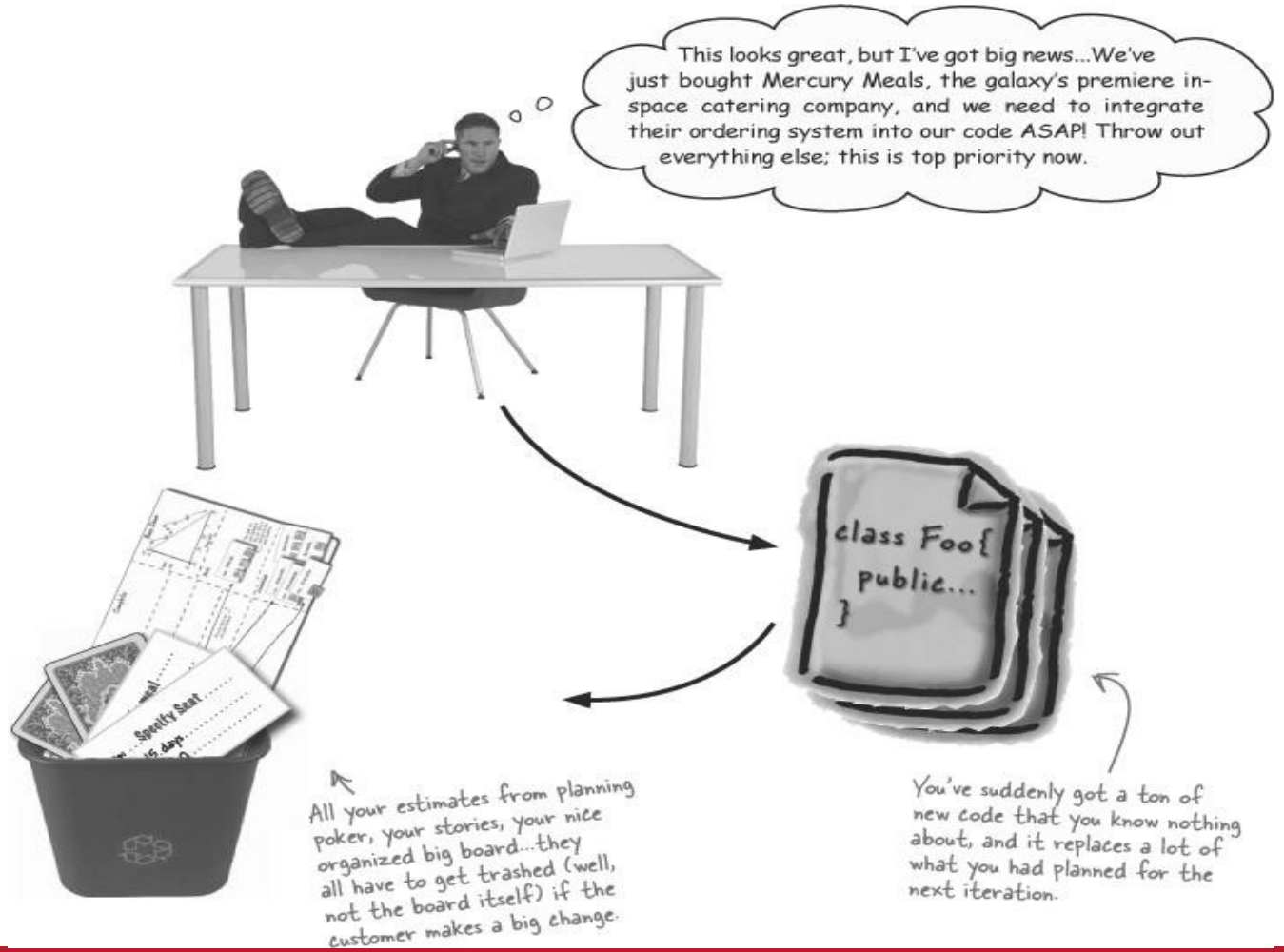
- You've got hard data you can use for recalculating velocity, and getting a more accurate figure.
- Revise both your estimates and your velocity at every iteration.
- Recalculate your estimates and velocity at each iteration, applying the things you learned from the previous iteration.

More On Velocity

- Velocity = REAL WORLD
- Velocity is a key part of planning
  - Looking for a value that corresponds to how fast your team actually works, in reality, based on how they've worked in the past.
- Only factor in the work that has been completed in your last iteration.
  - Any work that got put off doesn't matter; you want to know what was done, and how long it took to get done.
  - That's the key information that tells you what to expect in the next iteration.
- Velocity tells you what your team can expect to get done in the NEXT iteration
- By calculating velocity, you take into account the REALITY of how you and your team develop, so that you can plan your next iteration for SUCCESS.

DON'T FORGET THE CUSTOMER

- You still have to go back and get your customer's approval on your overall plan. And that's when things can go really wrong...



Software Engineering Is About Change!

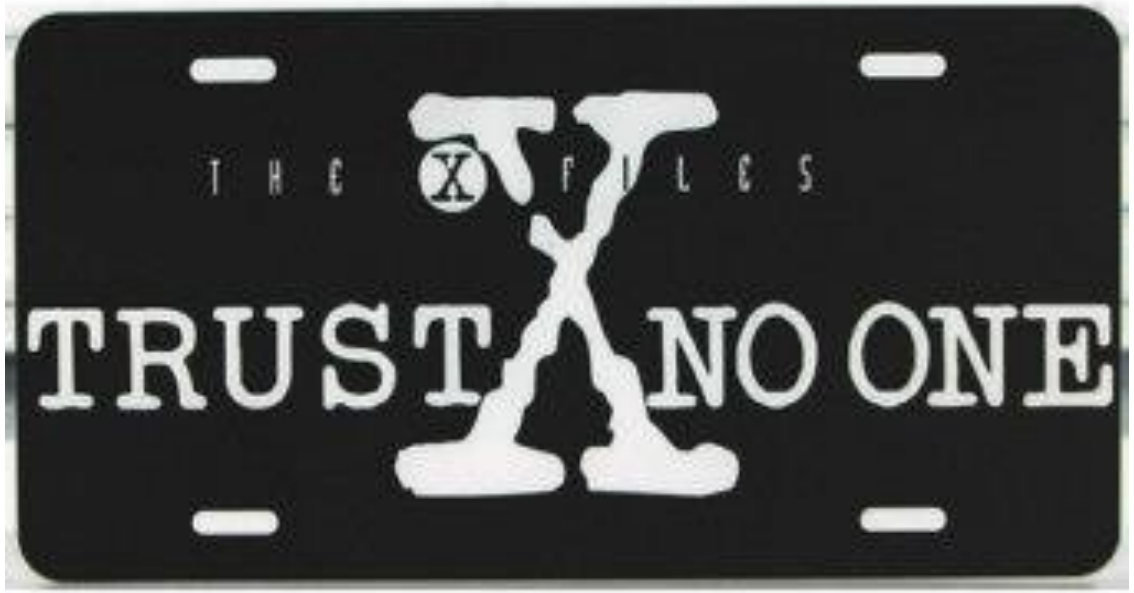
- Expect the unexpected
- (Stuff) happens
  - Customer is going to

  come up with a big change at the last minute.
  - Star programmer takes a new job.
  - The company you're working

  for lays off an entire department.
- Don't panic, build new user stories, reprioritize, and re-plan.

"Before you begin, I'd like to thank you for coming in early to do this on such short notice."

TRUST NO ONE (ELSE'S CODE)



Trust No One

- Unless you've seen a piece of code running, or run your own tests against it, someone else's code could be a ticking time bomb that's just waiting to explode.
- When you take on code from a third party, you are relying on that code to work. It's no good complaining that it doesn't work if you never tried to use it, and until you have seen it running, it's best to assume that third-party code doesn't really work at all.
- It doesn't matter who wrote the code. If it's in YOUR software, it's YOUR responsibility.
- TEST IT!

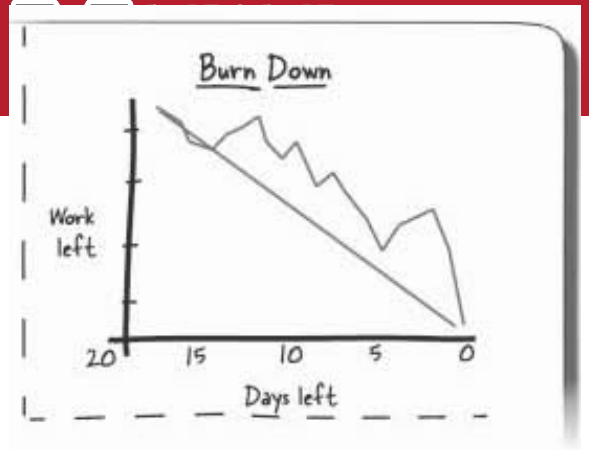
CUSTOMER APPROVAL

- Shouldn't be any surprises

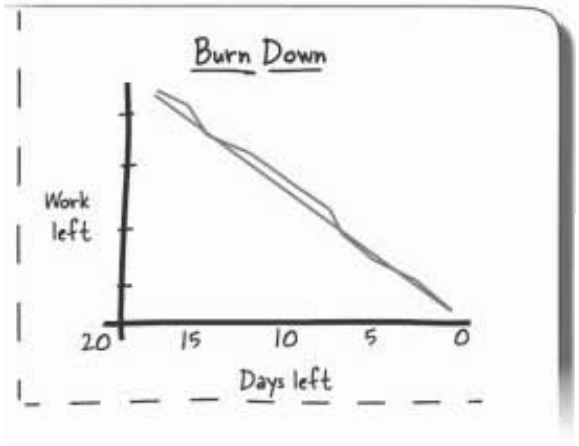
WITHOUT A PROCESS …

With A Process

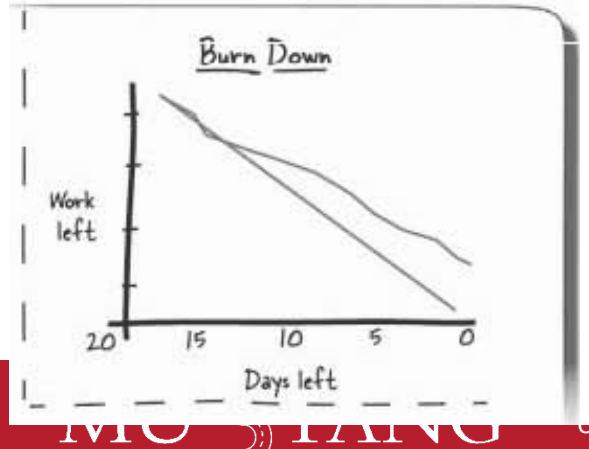
- It's not a perfect world, and there's no perfect process.
- When those bad things are happening,
  - That's the best time for a good process!
- Estimate
- Recalculate
- Adjust
  - Be flexible.



In this graph, the work remaining kept increasing as the iteration progressed. The team probably missed some things in their user stories: maybe lots of unplanned tasks—remember, red stickies are great for those—or bad estimates that got uncovered when user stories were broken down into tasks. Note the steep drop at the end—odds are that the team had to cut out things or drop stories altogether as deadlines started creeping up.



In this graph, the work left just keeps drifting to the right of the ideal burn-down rate. Chances are this is an estimate problem. There aren't any real spikes in the work left, so it's not likely that there were too many things the team didn't account for, but they just severely underestimated how long things would take. Notice they didn't make it to zero here. The team probably should have dropped a few stories to end the iteration on time.



This is a perfect graph—what every team wants. The team probably had a good idea what they were getting into, their estimates were pretty close at both the user story and task level, and they moved through tasks and stories at a nice predictable pace. Remember, a good iteration doesn't have lots of time at the end—it ends right when it's scheduled to.

Retrospective Questions

- What are some things you can do with leftover time?
- System testing? Contrast to Integration?
- Who should do system testing? How many cycles behind are they?
- What is working software?

Final Examination Information

- Check Banner for Times
  - Thursday 8:00 AM - 10:00 AM
- Cumulative (?)
- Same format as Midterm
- Online – 20% 100 pts out of 500

Study Tips

- Allocate study time based on time spent in class (# of lectures) and Course Outcomes + Week allocation
  - That is what the exam question load reflects
  - http://www.miamioh.edu/cec/academics/departments/cse/academics/course-descriptions/cse-201/index.html
- Take it seriously
  - Some of you getting > 90% in course, but don't be over confident!
  - The examination is tough as it is the complete evaluation of everything you learned this semester.
- This is an introduction course
  - We learned a little (introduction level) about many things!
  - Much breadth, little depth

Study Tips I Would Do

1. Read and comprehend the lecture notes
2. Read the textbook, especially for sections I'm struggling with
3. Repeat step 1
4. Complete all posted exercises on Canvas
5. Review Midterm

PROJECT

- Peer evaluations are mandatory
  - Very meaningful (score modifier)
- Reminder about “total” score distribution
  - (max 10 points per team member)
- Modifiers (penalty or not) released during finals week

Course Evaluation

- Final Course Evaluations -Up to 2% Final Grade Bump;
- Will give you 2% * (% of respondents in class) bump towards final grade
- So if half the class fills it out 0.5% bump
- Positive feedback and constructive criticism
- Used for promotion and evaluation of instructors.

GOOD STUDYING, AND THANK YOU!