

Ending an Iteration

Kenneth M. Anderson

University of Colorado, Boulder

CSCI 5828 — Lecture 24 — 04/08/2010

© University of Colorado, 2010

Goals

2

- ▶ Review material from Chapter 9 of Pilone & Miles
 - ▶ Ending an Iteration
 - ▶ System Testing
 - ▶ Bug Reports
 - ▶ Iteration Review

Ending an Iteration

3

- ▶ Following the agile practices we've covered each iteration will have
 - ▶ customer-driven functionality (user stories; feedback)
 - ▶ compiling code & monitored builds (continuous integration)
 - ▶ solid test coverage and continuously tested code (TDD)
 - ▶ reliable progress tracking (burn-down chart)
 - ▶ pacing that adapts to the team (iteration plan; velocity)
- ▶ and with this you may find yourself with spare time at the end of an iteration

What else can be done?

4

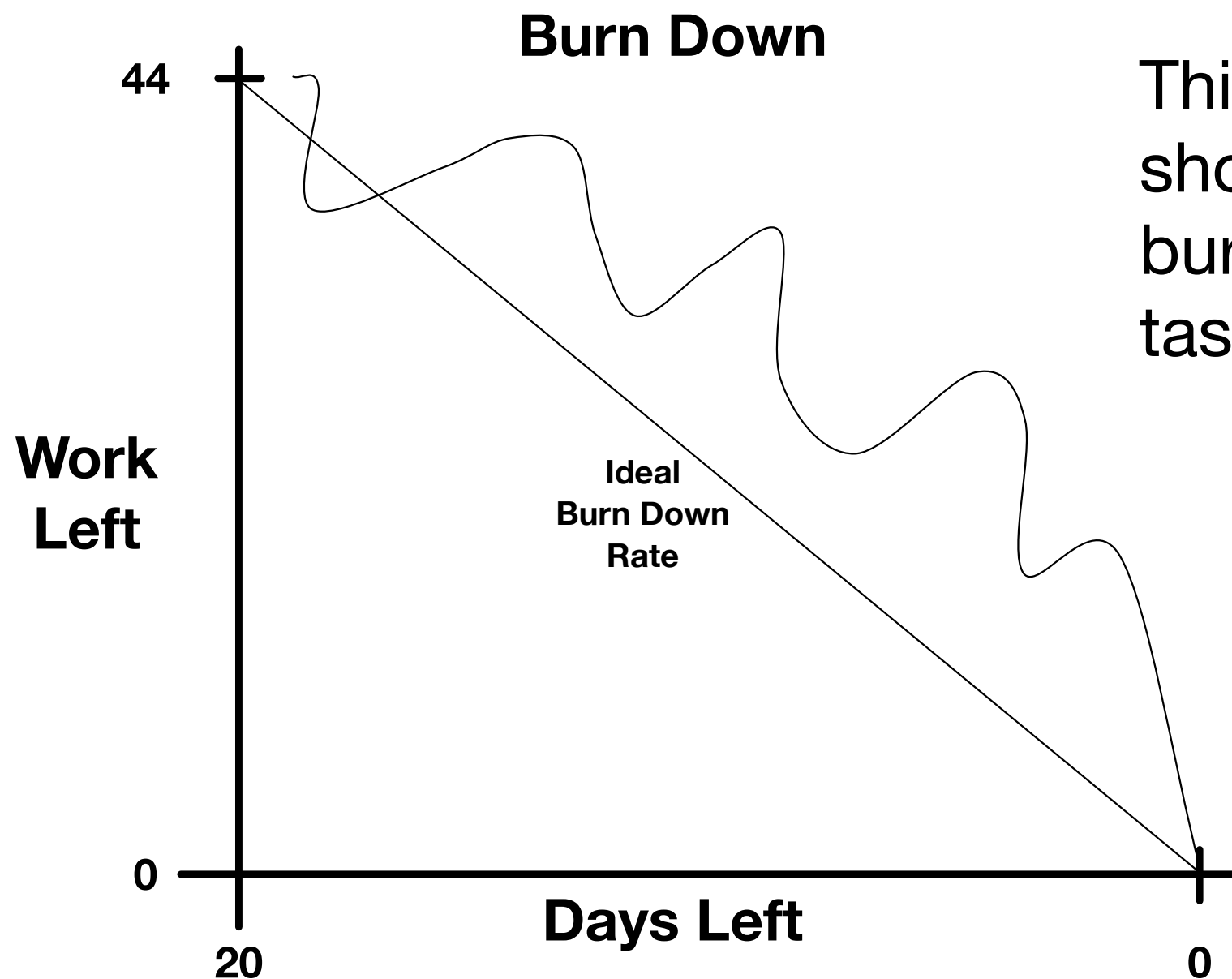
- ▶ At the end of an iteration, you can reflect on things you'd like to add to daily work practice to provide additional benefits
- ▶ Such as
 - ▶ process improvements (what's not working?)
 - ▶ system testing (we've got unit tests and integration tests)
 - ▶ refactoring of code based on lessons learned
 - ▶ code cleanup and documentation updates
 - ▶ design patterns
 - ▶ environment updates, R&D, personal development time

Data in Burn Down

5

- ▶ One way to reflect on the iteration is to look at the burn-down chart
 - ▶ It can provide insight into the effectiveness of the team
- ▶ Were we ahead, were we always behind?
 - ▶ Are we good at adapting to change?

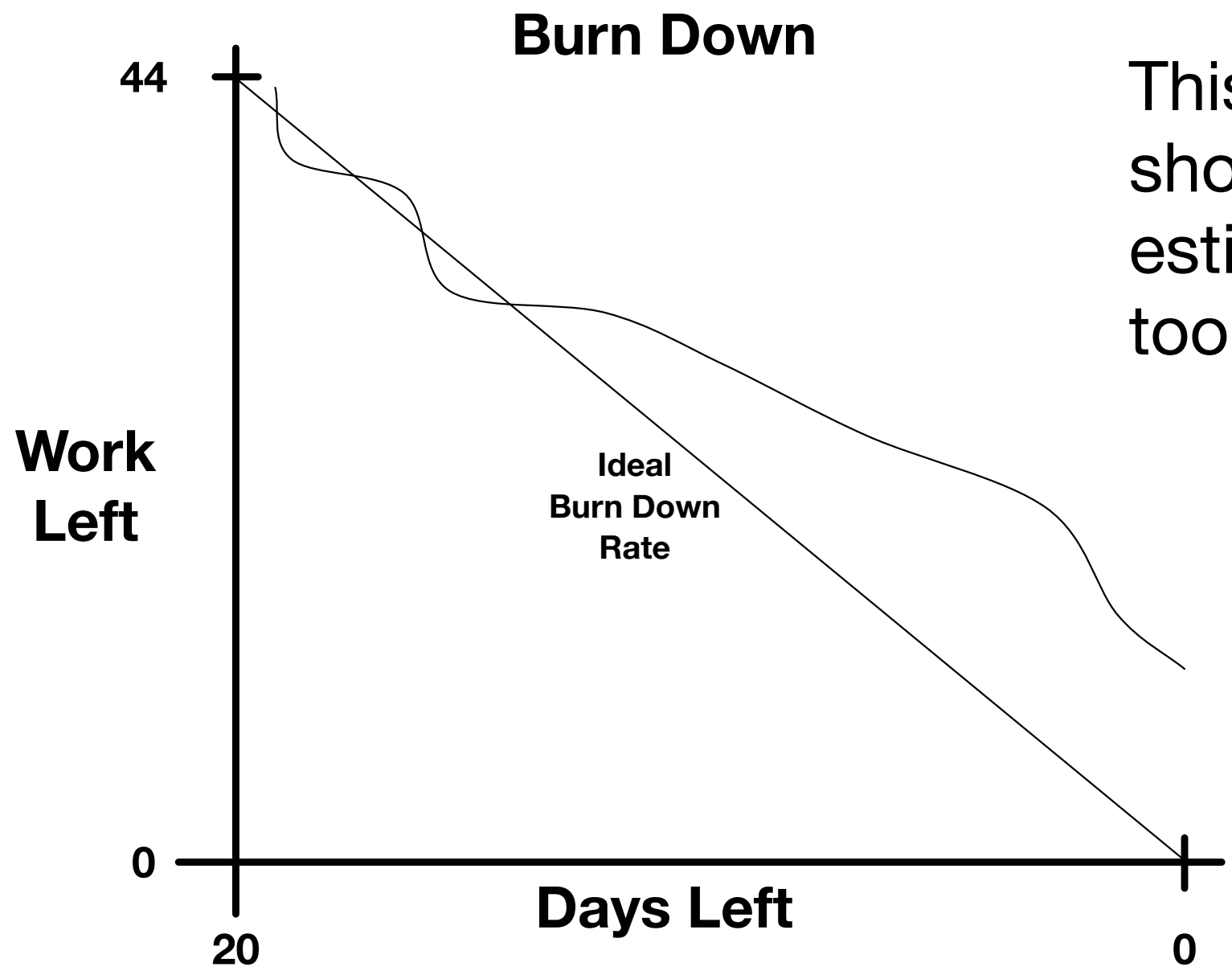
Unplanned Tasks



This burn-down chart shows the team getting burned with unplanned tasks and/or user stories

The drop to zero at the end is NOT the result of some heroic effort on the part of the team; likely it is simply a result of scoping down

Bad Estimates



This burn-down chart shows the team had bad estimates; everything took longer than planned

Not getting to zero means the team needs to learn how to re-scope: delaying tasks and stories to subsequent iterations

Integrating System Tests

- ▶ Our techniques do not provide time for system testing
 - ▶ A system test exercises the functionality of the system from “front to back” (UI to persistence layer) in real-world black-box scenarios
- ▶ Developers are too biased to do system testing, they know the code too well and do not necessarily have access to realistic test data
- ▶ Your end users should be the ones performing system tests on real data
 - ▶ If that’s not possible, you need a testing team!

Off by One

- ▶ In each iteration, the developers are concerned with the current set of user stories
 - ▶ They test constantly but those are unit/integration tests
- ▶ A test team, then, can perform system testing on system $n-1$ during iteration n
 - ▶ During iteration 1, the test team gets ready for iteration 2
 - ▶ Reviewing stories, writing tests, installing tools, etc.
- ▶ This leads to more being done in each iteration
 - ▶ and the book views them as separate iteration cycles
 - ▶ that is, more iterations

More iterations, more problems

10

- ▶ Running two iteration cycles means
 - ▶ LOTS more coordination which requires LOTS more communication
 - ▶ Will require “cross pollination” of standup meetings
 - ▶ Forces testing into a “box”: fixed time step
 - ▶ May not be able to cover all functionality within a single iteration
 - ▶ Bug fixing mixes in with new work
 - ▶ If the testing team is finding bugs, guess who has to fix them?
 - ▶ Tests are written against a moving target

More problems, more talk

11

- ▶ In order to deal with these problems, you just need
 - ▶ MORE communication to enable better coordination
 - ▶ and remember, in agile approaches, we value direct communication
 - ▶ We do have to worry about this on one level (Mythical Man Month) but remember that agile approaches avoid a lot of the documentation that slow traditional SE approaches down

Effective System Testing

12

- ▶ Good, frequent communication (devs., test team, customer)
- ▶ Known starting and ending state of system
- ▶ Document your test suites
- ▶ Establish clear success criteria (when can we go live?)
- ▶ Automate your tests
- ▶ Devs and test team work together (avoid fights!)
- ▶ Test team understands big picture view of system
- ▶ Accurate system documentation

Test results?

13

- ▶ We eventually want to see all tests pass
 - ▶ but before we do, the results of testing are bug reports
- ▶ Bug life cycle
 - ▶ Tester finds bug
 - ▶ Creates a bug report and submits it to issue tracking system
 - ▶ Developers create a story or task to fix the bug
 - ▶ Enters iteration plan and handled as normal
 - ▶ Developers fix the bug
 - ▶ Tester checks the fix and verifies the bug is gone
 - ▶ Tester updates the bug report (sets status to closed/resolved)

Bug Trackers

14

- ▶ Plenty of systems out there to do bug tracking
 - ▶ FogBugz, Bugzilla, Mantis, TestTrackPro, ClearQuest
- ▶ Important because they
 - ▶ let you prioritize bug reports
 - ▶ related to success criteria “go live when only priority 4 bugs remain”
 - ▶ let you keep track of everything related to a bug fix
 - ▶ let you generate important metrics related to life cycle quality
 - ▶ bug submission rate? location of bugs? bugs outstanding?

Bug Reports

15

- ▶ Good bug reports contain
 - ▶ A summary that describes the bug in 1-2 sentences
 - ▶ The steps needed to reproduce the bug (see it in action)
 - ▶ Expected Output vs. Actual Output
 - ▶ Configuration Information: Platform, version, etc.
 - ▶ Severity: how bad is the impact of this bug?
 - ▶ Priority: how quickly do we need to fix this bug?
 - ▶ Current Status

Iteration Review

16

- ▶ At the end of an iteration, take time to reflect and identify how the process can change to make things run smoothly
- ▶ A good iteration review requires that you
 - ▶ prepare ahead of time: bring a list of things to discuss
 - ▶ be forward-looking: what should we do to improve the next iteration?
 - ▶ calculate your metrics: velocity, burn-down rate, etc.
 - ▶ review a standard set of questions that helps the team look for opportunities to improve

Review Questions

17

- ▶ Was the quality of our work acceptable?
- ▶ Was the pace acceptable?
- ▶ Are you comfortable with your current work assignments?
- ▶ Are our tools getting in the way? Are there new tools to consider?
- ▶ Was our process effective? Does something need to change?
- ▶ Performance problems? Bugs to discuss?
- ▶ Testing effective? “Bad smells” to get rid of

If you have extra time

18

- ▶ If you have “free” days at the end of an iteration
 - ▶ Fix bugs and/or refactor and/or update documentation
 - ▶ Tackle a user story from the next iteration
 - ▶ Prototype solutions needed in the next iteration
 - ▶ Training or Learning Time: Google’s “20% time” practice

Wrapping Up

19

- ▶ The end of an iteration is a time for reflection
 - ▶ What should we change to make the next iteration better?
- ▶ It is also a time for catching up or getting ahead
- ▶ Learn to use iterations well
 - ▶ Pay attention to burn-down rates and what they tell you about the team
 - ▶ Pace the iteration ; if you have too much to do, scope the iteration down
 - ▶ Review each iteration to continuously improve your process

Coming Up

20

- ▶ Lectures 25 and 26:
 - ▶ Not being held; be on the lookout for homework 4 instead
- ▶ Lecture 27: The Next Iteration
 - ▶ Read Chapter 10 of Head First Software Development
- ▶ Lecture 28: Model-Driven Design
 - ▶ Last chapter from optional textbook
 - ▶ No reading assignment