Upcoming assignments:

Tonight – black-box unit testing and another informal demo for your mentor, this time your mentor should try to "break" your system

Tuesday November 15 – new (ungraded) assignment to come to class prepared for in-class team meetings, with tentative user stories and wireframes ready

Thursday November 17 – second/final iteration plan due, note this includes class diagrams as well as user stories and wireframes

Second 360-degree Feedback originally scheduled for Tuesday has been pushed to following week, due November 22nd

Tuesday November 22nd class will consist of a "panel" discussion by real-world software engineers, come with your questions!

Second Iteration Development assignment has been posted, due December 1st

Ending an iteration:

Every iteration should end with demo or delivery, there may also be formal release iterations beyond just delivery (and eventually there will be an end-of-project iteration)

For demo/delivery iterations, need to include demo/delivery activities in iteration plan

These are more tasks for the task board, could be described as user stories, but with fixed pre-determined scheduling (e.g., the morning of the last day of the iteration)

Example: "As the team lead, I want to prepare and practice a demo script for the customer so he/she can see that we've fulfilled his/her requirements"

Also need to plan for iteration post-mortem tasks, also with fixed pre-determined scheduling (e.g., an hour in the afternoon of the last day of the iteration)

Note this shortens the effective development time for the iteration

What is included in demo activities?

Show the demo to the customer, feedback from customer

May include customer's introduction of new user stories, changes and/or reprioritization of known user stories scheduled for future iterations

What is included in delivery activities?

In addition to demo activities, need to package for deployment, conduct full system tests of delivery download on pristine hardware/software configuration(s)

Feedback from customer may be delayed until after some period of use

What is included in release activities?

There may be one or more entire iterations devoted to formal release

Emphasis on acceptance testing by customer in customer's environment (alpha/beta users), with corresponding bug fixes as needed, documentation writing, training of end-users and/or technical support staff

In addition to demo/delivery, iteration should end with post-mortem or retrospective:

Sit-down meeting (without customer), time-boxed to max one hour

Did the team meet the iteration's goals? Yes/no

What's working well, what isn't, and what the team can do better next time (qualitative)

Root cause analysis and potential corrective actions = "improvement stories", review improvement stories from the previous retrospective

Example: Every developer sets up their own local build/test environment, new developer breaks the build by pushing local configuration files that haven't been explicitly excluded, build engineer has trouble figuring out what went wrong - oops

Example: History of "flaky tests", company culture ignores system test failure if only happened once and unit tests passed, but new feature had recently been pushed to system testing so had only been system-tested the one time, major bug is deployed to customers - oops

Calculate project metrics (quantitative)

Known bugs – defect count per kloc, both fixed and unfixed, increasing/decreasing/steady? Ideally, few if any new bugs are being found and the unfixed defect count is zero or nearly zero prior to release, may need to extend the project with a bugfix iteration

Testing coverage – probably not 100% but can we get closer to 100%? Measure number of new test cases, number of refactors. Refactors can sometimes result in increasing (or decreasing) coverage from existing test cases.

"Project Velocity" – There were N days (or points, etc.) of work scheduled for the iteration, M days were actually completed

Within the pre-defined length L work days (e.g., 20 days per calendar month) by D developers (e.g., four) = L x D

Thus, for team of four developers, schedule at most 80 days of work for a one-month iteration, but it's unlikely that 80 days of work will really be completed

Velocity = M actual work / N scheduled work, so ?? / 80

Rule of thumb for first iteration: assume velocity 0.7 (70%)

So for a team of four developers, 0.7 x 80 = only 56 days of actual work!

If we instead assume velocity 0.8 (80%), then 0.8 x 80 = 64

If we instead assume velocity 0.6 (60%), then 0.6 x 80 = 48

Calculate what was the actual velocity at the end of each iteration, use this velocity for planning the next iteration - Don't schedule more work than can realistically be expected, you will not be able to "catch up"

Some organizations allow one day per week, or one half-day per week, for "pet projects", reducing time that can be scheduled for iteration, so this is not where the 30% comes from

Where does the time go?

Interviewing, on-boarding new hires, staff training – these could be planned as tasks and included in the schedule, but other lost time is harder to plan for

Overly optimistic estimates for user stories and tasks, particularly in early iterations, although this should be accounted for by velocity

Developers have lives – personal days, sick days, doctor/dentist appointments, traffic and other commuting issues, someone in another company in a shared building pulls the smoke alarm, etc

Computers have lives too – software installation, patches and upgrades, network issues, hardware issues

Should we hire more people, so we have more person-days per calendar day?

Fred Brooks book on mythical person-month, see suggested reading (available free online)

Brooks was the manager of a huge project at IBM in the 1960s, so many of the details are extremely dated and may not make much sense to modern readers, but the gist of his argument remains the same

The person-month theory is that is you have X days of work for one person to do, then if you hire another person the work will take X/2 days, if you hire two additional people the work will take X/4 days, and so on

Brooks' law: "Adding manpower to a late software project makes it later."

First, the person-month theory ignores the problem of finding and hiring these people, but let's assume more developers are available in-house and can be re-assigned to this project (the case at IBM then)

Second it ignores training – the newly assigned developers typically have little or no prior knowledge of this project, a big problem if they're joining midstream, not a big problem if brought on at or near beginning of project

Training consumes both the time of the new developers **and** the time of the original developers who are now less productive because they need to do the training

Third it ignores communication overhead – two developers have one communication path, three developers have three communication paths, four developers have six communication paths, five developers have ten communication paths

N developers have N*(N-1)/2, blows up (image)

Another problem is that some work isn't divisible and there are dependencies among tasks - critical path

Example: If using a relational database, the schema should be designed before or in tandem with the code that queries or updates the database

Example: The basic UI has to be constructed before other code can fill in

This does **not** mean you should never hire more developers!

Early in the project may work fine, since the new developer plus the developer who trained him/her will together be more productive over a longer term, and work can often be organized into small more-or-less independent teams where only the team leads need to communicate (but beware deep hierarchies)

But after the project is already running late, and nearing the originally planned completion (of a release if not the entire project), that it's probably too late to catch up

Law of diminishing returns

What if there is time left over at the end of an iteration?

Could start user stories/tasks intended for the next iteration, particularly fixing known defects or refactoring

Could investigate/learn new technologies and tools

If one developer (or pair) is done early, can help others who aren't

Hard to schedule specific activities in advance since may not know until day or two before that work will be completed early, unless team historically estimates pessimistically – in which case should revise velocity based on time left over, and schedule more work for the next iteration

Does the next iteration always start immediately after the previous iteration?

Not always, some organizations schedule a "bug backlog" week or training or some other activity between iterations

Note this allows more time for feedback from customer before starting next iteration

Also may be used for "design sprints", particularly after a release towards the next release or towards a new project

Starting the next iteration

Start with a sit-down planning meeting (with customer)

New task board, new burndown chart

Choose new/revised user stories with new/revised priorities that fit into the iteration schedule, considering the actual project velocity at the end of the previous iteration

How does testing fit into iterations?

Unit testing is part of developing each unit, and the code unit is not “complete” until its unit tests pass (possibly with an acceptable coverage level), so the time estimate for coding a unit should include the time expected to test the unit

A user story is not “complete” until all its unit **and** system tests pass, so the time estimate for implementing a user story should include the time expected to test the user story

This assumes the developers and the testers are the same people, which is almost always the case for unit testing, but sometimes there’s a separate team for system testing. (The testing team might test multiple projects, not just yours.)

For separate system testing teams, we need parallel iterations and parallel task boards

Testing by a separate testing team also starts with the customer’s set of user stories, because those define the features they need to test, but their tasks are to design, code and run the test cases rather than to implement the stories

Testing tasks take time, where do the time estimates come from? Guestimate from previous experience, planning poker, or “spike” (explained shortly)

The system testing team may report bugs to the development team either as the bugs are found, at arbitrary points in the development iteration - particularly for show-stopper bugs, or only at the end of an iteration

New bugs are just new stories/tasks to fix the bug, either added to that iteration - pushing off other tasks to “overflow”, or scheduled for the next iteration

Example: “As a tester, I want to fix the concurrency error between multiple client threads, so we can support multiple simultaneous users”.

Bug-fixing stories need time estimates, but where do these estimates come from? Guestimate from previous experience, planning poker, or “spike”

What is a “spike”?

Time-limited (e.g., one developer for five working days) devotion to developing test cases, fixing bugs, or some other activity where time estimates are needed.

For bug case, pick a set of bugs to try to fix in the time allotted

Choose even mix of bugs deemed easy, medium, hard, and assign each proportional “points” (or some other unit that isn’t explicitly time, since its time that you’re trying to derive) - Easy might be 1 point, medium 3 points, hard 5 points.

Round-robin between the types and spend full-time doing nothing else for the “spike” period

At the end, use the number of points completed in the number of days to determine how long a “point” takes.

Example: If completed 10 points in 5 days, then it takes 1 day to complete 2 points of debugging

Debugging includes reproducing the bug, fault localization, changing the code, testing that the bug is indeed removed and no new bugs introduced.

(debugging notes if time permits)