Computer Integrated Program Design

Markum Reed

Summer Semester

Introduction

- What is Agile?
- How does Agile work?
- How is Agile different?

Fundamentals

- User Stories/Backlog
- Estimation
- Iterations
- Planning
- Burndown Chart

Engineering

- Unit Testing
- Refactoring
- Continuous Integration
- Test Driven Development

What is Agile?

- Agile is a time boxed, iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver it all at once near the end.
- The project is completed incrementally, not all at once.
- It works by breaking projects down into little bits of user functionality called **user stories**, which creates your **backlog**
  - By prioritizing them,and continuously delivering them in short one/two week cycles called **iterations**.

How does Agile work?

- Agile does the same thing you and I do when faced with too much to do and not enough time:
  1. **Make a list**
  2. **Size things up**
  3. **Set priorities**
  4. **Start executing/doing**
  5. **Update as you go**

How does Agile work?

1. **Make a list:** Sitting down with your customer you make a list of features they would like to see in their software.
   - We call these things **user stories/backlog**; they become the To Do list for your project.
2. **Size things up:** Using estimation techniques, size your stories relatively to each other, coming up with a guess as to how long you think each user story will take.
3. **Set priorities:** Like most lists, there always seems to be more to do than time allows.
   - Prioritize your list so you get the most important stuff done first, and save the least important for last.

How does Agile work?

4. **Start executing/doing:** Then you start delivering some value. You start at the top. Work your way to the bottom. Building, iterating, and getting feedback as you go.
5. **Update as you go:** Then as you starting delivering one of two things is going to happen. You'll discover:
    a. You're going fast enough. All is good. Or,
    b. You have too much to do and not enough time.

- At this point you have two choices. You can either (a) do less and cut scope (recommended). Or you can (b) push out the date and ask for more money/time.

How is Agile different?

- **Analysis, design, coding, and testing are continuous activities:**
  - You are never done analysis, design, coding and testing. So long as there are features to build, and the means to deliver them, these activities continue for the duration of the project.
- **Development is iterative:**
  - Iterative development means starting with something really simple, and adding to it incrementally over time.
  - It means evolving the architecture, accepting that your requirements are going to change, and continuously refining and tweaking your product as you go.
- **Planning is adaptive:**
  - When reality disagrees with your plans, change your plans. This is **adaptive planning**.
  - And while there are many ways to changes plans, the preferred way is to flex on scope.

How is Agile different?

- **Roles blur:**
  - Roles really blur on Agile projects. When it's done right, joining an Agile team is a lot like working in a mini-startup. People pitch in and do whatever it takes to make the project successfulregardless of title or role.
  - Yes, people still have core competencies, and, yes, they generally stick to what they are good at. But on an agile project, narrowly defined roles like analyst, programmer, and tester don't really exist - at least not in the traditional sense.
- **Scope can vary:**
  - Agile deals with the age old problem of having too much to do and not enough time by doing less.
  - By fixing time, budget, and quality, and being flexing around scope, Agile team's maintain the integrity of their plans, work within their means, and avoid the burn out, drama, and dysfunction traditionally associated with our industry.

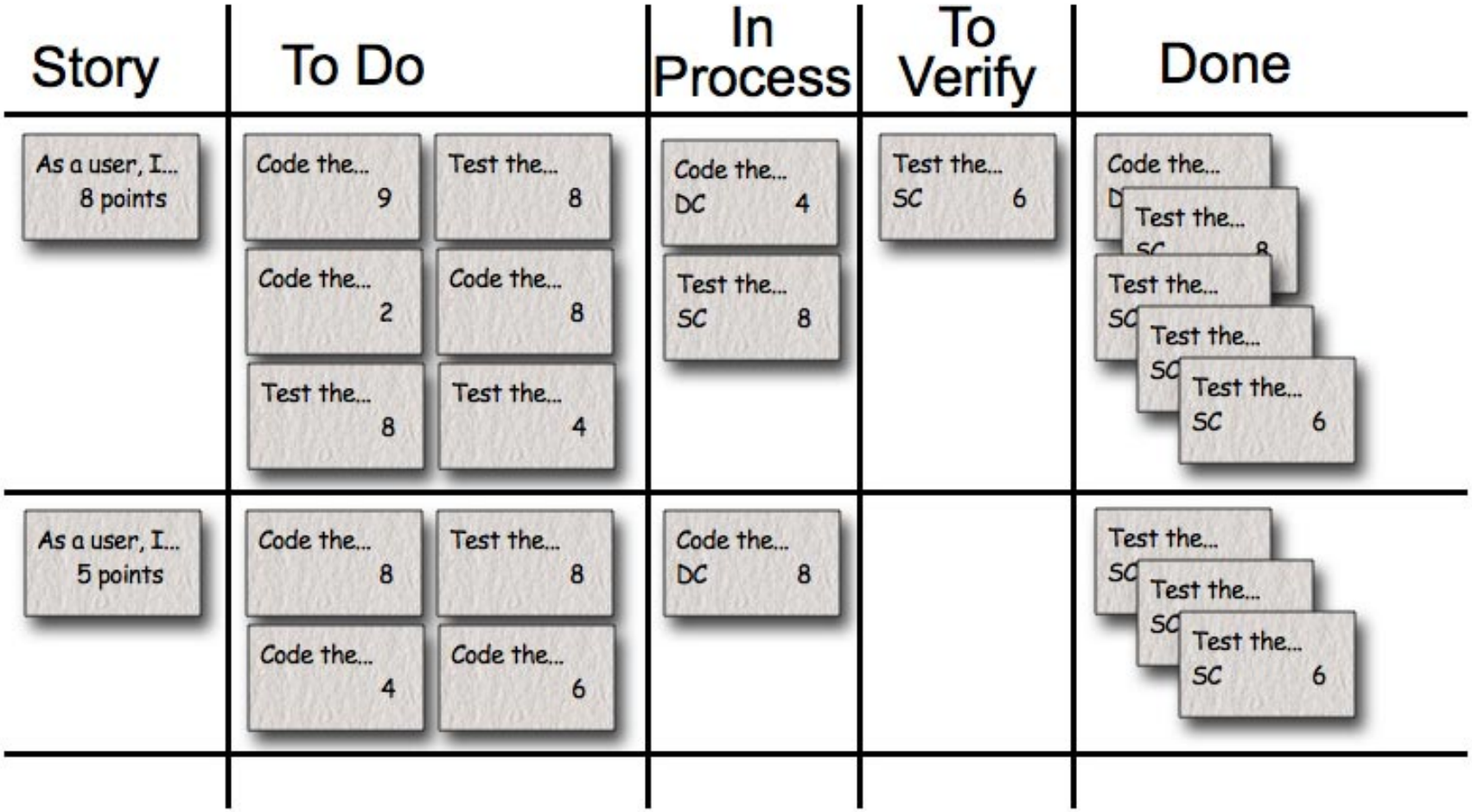
How is Agile different?

- **Requirements can change:**
  - Traditionally change has been shunned on software projects because of it's high perceived cost late in the game. Agile challenges this notion and believes the cost of change can be relatively flat.
  - Through a combination of modern software engineering practices, and open and honest planning, Agilsts accept and embrace change even late in delivery process.
- Working software is the primary measure of success:
  - The rate at which teams can turn their customer's wishes into working software is how Agilists measure productivity. Project plans, test plans, and analysis artifacts are all well and good but Agilists understand they in themselves are of no value to the end customer.

User Stories/Backlog

- **User stories are features our customers might one day like to see in their software.**
  - User stories are like requirements, but they aren't.
  - There's no guarantee all these features are going to make it into the final version of the software.
  - You know are going to change your mind - and that's OK. Because they weren't really requirements to begin with.
- **They are written on index cards/post-Its to encourage face-to-face communication.**
  - Words are slippery things. Get a comma wrong and it can cost you a million dollars.
  - That's why we love post-Its. They make it impossible to write everything down and instead force you to get off your butt and go talk to your customers about the features they'd like to see in their software.

Scrum Board


Story
To Do
In Process
To Verify
Done
As a user, I...
8 points
Code the... 9
Test the... 8
Code the... 2
Code the... 8
Test the... 8
Test the... 4
Code the... DC 4
Test the... SC 8
Test the... SC 6
Code the...
Test the...
Test the... SC 6
As a user, I...
5 points
Code the... 8
Test the... 8
Code the... 4
Code the... 6
Code the... DC 8
Test the... SC 6

Scrum Board

- Story: The story description shown on that row.
- To Do: Place for all cards that are not in the Done or In Process columns for the current iteration/sprint.
- In Process: Any card being worked on goes here.
- To Verify: A lot of tasks have corresponding test task.
- Done: Cards pile up over here when they're done. They're removed at the end of the sprint.

User Stories/Backlog

- **Typically no more than a couple days work, they form the basis of our Agile plans.**
  - User stories form the basis of the Agile plan.
  - They are sized and prioritized like any other wish list.
  - You simply start at the top and work your way down.
  - Nothing big or complex.
  - Just a prioritized todo list and a desire to get things done.

User Stories/Backlog

- **We get them by sitting down with our customers and asking lots of questions.**
  - Big rooms with lots of white space to draw are great for gathering user stories.
  - In these story gathering workshops we draw lots of pictures (flowcharts, screens, storyboards, mockups, anything that helps)
  - Break the functionality down into simple easy to understand words and phrases our customers understand.

Make Your User Stories

- Get into groups
  1. Make a list of what features you want in your program
  2. Make a sublist about how each feature needs to be done
- These are your user stories.

Estimation

- **The fine art of expectation guessing**
  - While we aren't very good at estimating things absolutely, it turns out we are pretty good at estimating things relatively.
  - Sizing stories relatively means not worrying about exactly how big a story is, and worrying more how this story's size compares to others.

Example

Small = 1pt (No sweat, easy), Medium = 3pts (Nothing we can't handle, not easy but not hard), Large = 5pts (This is going to take some time, Ahhh!)

- This style of estimation (relative over absolute) forms the corner stone of planning.
- By sizing our stories relatively, and feeding actuals back into our plan, you can make (relatively) accurate predictions about the future while based on what we've done in the past.

Rank your user stories

Example

Small = 1pt (No sweat, easy), Medium = 3pts (Nothing we can't handle, not easy but not hard), Large = 5pts (This is going to take some time, Ahhh!)

- Get into groups:
  - Size your stories: Either 1, 3, or 5

Iterations

- **Agile's engine for getting things done**
  - An Agile iteration is a short one to two week period where a team takes a couple of their customers most important user stories and builds them completely as running-tested-software.
  - This means everything happens during an iteration. Analysis, design, coding, testing. It all happens here.
  - The beauty of working this way, is every couple weeks the customer gets something of great value (working software)
  - It's also a great way to track progress (measuring the rate at which the team can turn user stories into production ready working software).

Planning

- **The fine art of expectation setting**
  - In its simplest form, agile planning is nothing more than measuring the speed a team can turn user stories into working, production-ready software and then using that to figure out when they'll be done.
  - Our to-do list on an agile project is called the master story list. It contains a list of all the features our customers would like to see in their software.
  - The speed at which we turn user stories into working software is called the team velocity. It's what we use for measuring our teams productivity and for setting expectations about delivery dates in the future.
  - The engine for getting things done is the agile iteration - one to two week sprints of work where we turn user stories into working, production-ready software.

Planning

- *To give us a rough idea about delivery dates, we take the total effort for the project, divide it by our estimated team velocity, and calculate how many iterations we think we'll require to deliver our project. This becomes our project plan.*

$$\textit{iterations} = \frac{\text{total effort}}{\text{estimated team velocity}} \tag{1}$$

Example

$$\textit{iterations} = \frac{100 \text{ pts}}{10 \text{ pts per iteration}} \implies 10 \text{ iterations}$$

- Now, as we start delivering, one of two things is going to happen. We are going to discover that (a) we are going faster than expected or (b) we are going slower than we originally thought.

Iterations Calculation:

- From before, take your User Stories Point Estimations
  - Calculate your iterations.
  - If your Iterations are above 3, you need to slim down your User Stories
  - Final due date for your project is **9/15/2016**

Planning

- Faster than expected means you and your team are ahead of schedule. Slower than expected (more the norm) means you have too much to do and not enough time.
- When faced with too much to do, agile teams will do less (kind of like what you and I do when faced with a really busy weekend). They will keep the most important stories, and drop the least important. This is called adaptive planning and it's how Agile teams work within their budgets and keep their projects real.
- For your projects you only have three/four iterations

Burndown Chart

- The burndown is a chart that shows how quickly you and your team are burning through your backlog/user stories.
- It shows the total effort against the amount of work delivered each iteration.

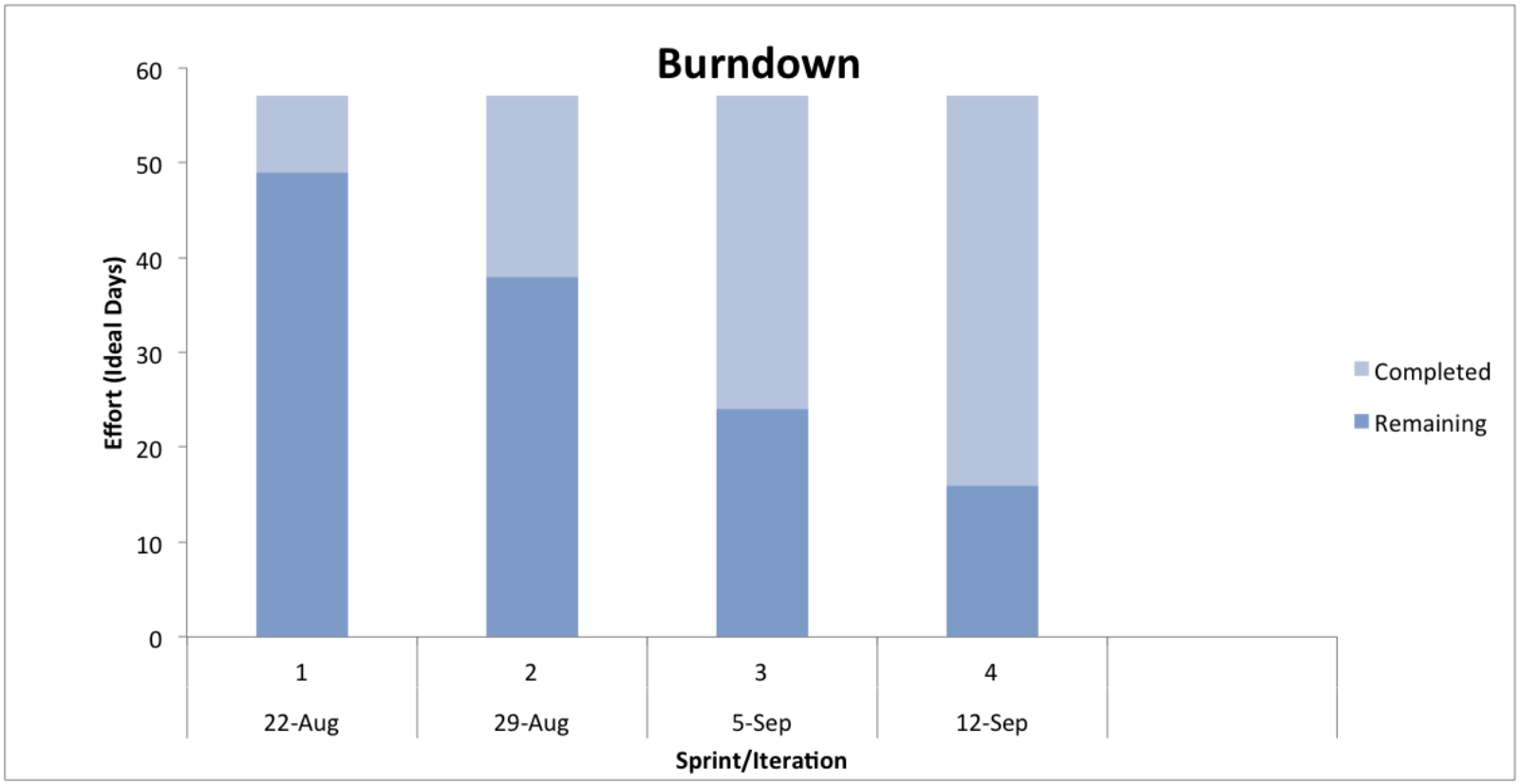


Figure: Burndown Chart

Burndown Chart

- We see the total effort on the left, our team velocity on the right. But look what else this simple graphs gives us.
  - Work done each iteration
  - Work remaining
  - Work done so far
  - When we can expect to be done
- The burndown above displays total work done each iteration. This let's us look at the chart, and immediately get a sense of whether we are a quarter, a third, or 1/2 way done the project.

Burndown

- Burndowns are great because they:
  1. Make the reality of the project clear.
  2. Show the impact of decisions.
  3. Warn you early if things aren't going according to plan.
  4. Get rid of all the wishful thinking around dates

Burndown Chart: Estimation, Iterations, Planning

- Use the excel template for the Burndown to estimate your completeness.

Unit Testing

- Unit tests are snippets of test code developers write to prove to themselves that what they are developing actually works. Think of them as codified requirements.
- They are powerful because when combined with a continuous integration process they enable us to make changes to our software with confidence.

Refactoring

- As we add more functionality to the system we need a way of maintaining our design and keeping our house in order. In Agile we call this refactoring.
- For example, instead of copying and pasting code every every time we need some functionality, it's much easier to maintain if we extract that code into a one place and call it from where ever we need it.

Continuous Integration

- Continuous integration is about keeping it all together. On a team of more than one, you are going to have people checking code in all the time.
- We need a way to make sure that all the code integrates, all the unit tests pass, and a warning if anything goes wrong.

Test Driven Development

- Test Driven Development is about writing the test first before adding new functionality to the system. This seems backwards as first, but doing this:
  - Defines success up front.
  - Helps break our design down into little pieces, and
  - Leaves us with a nice suite of unit tests proving our stuff works.
- Agile developers work in this circle of life when adding new code.
- Write the test first. Make it pass. Then refactor.