COMS W4156 Advanced Software Engineering (ASE)

September 30, 2021

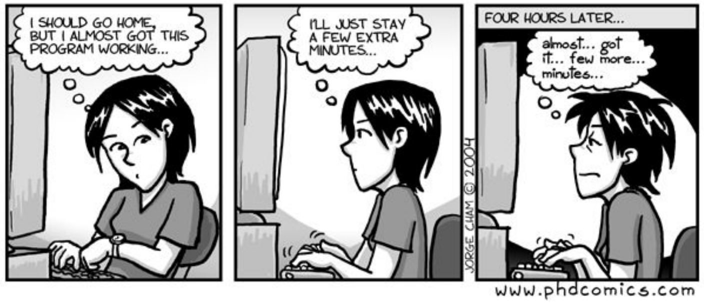
shared google doc for discussion during class

Individual Mini-Project part 2 and part 3 Deadlines Extended

Three parts:

1. Implementing a simple game graded - contact grader for regrade (or contact Head IA - Claire - if you don't know who graded)
2. Testing the game due yesterday, but you can (re)submit up to October 5 with no penalty
3. Saving game state nominally due October 6, but you can submit up to October 9 with no penalty

Use Cases


I SHOULD GO HOME, BUT I ALMOST GOT THIS PROGRAM WORKING...
I'LL JUST STAY A FEW EXTRA MINUTES...
FOUR HOURS LATER...
almost... got it... few more... minutes...
JORGE CHAM © 2004
www.phdcomics.com

Stories may not provide enough information to start coding

Use cases expand stories (or are written from scratch) to describe the step by step details of how the role interacts with the software to exercise the feature and achieve the goal, while trying to capture additional structure, contingencies and alternative flows, and
(in the extreme) anything/everything that ***CAN GO WRONG***

The term "*use case*" is often used by business to mean "a specific situation in which a product or service could potentially be used", but here we mean more specifically "description of how a user who actually uses that process or system will accomplish a goal"

The user does not need to be human, it is anyone or anything that uses the process or system

What is the Difference between User Story and Use Case?

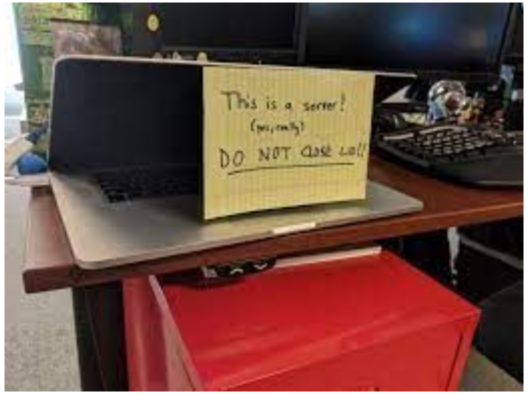
User stories are about needs. When you write a user story, what you’re describing is a “raw” user need. It’s something that the user needs to do in his day-to-day job. If you never build any software for him, then that need will still exist!

Use cases are about the behavior you’ll build into the software to meet those needs. A developer who needs to build working software should be able to read a use case and get a good sense of what the software needs to do. It typically has a lot of detail, and describes everything that the developer needs to build in order to meet the user’s need. That’s why it needs to have a lot more detail, and be clear and unambiguous.

What Could Possibly Go Wrong?

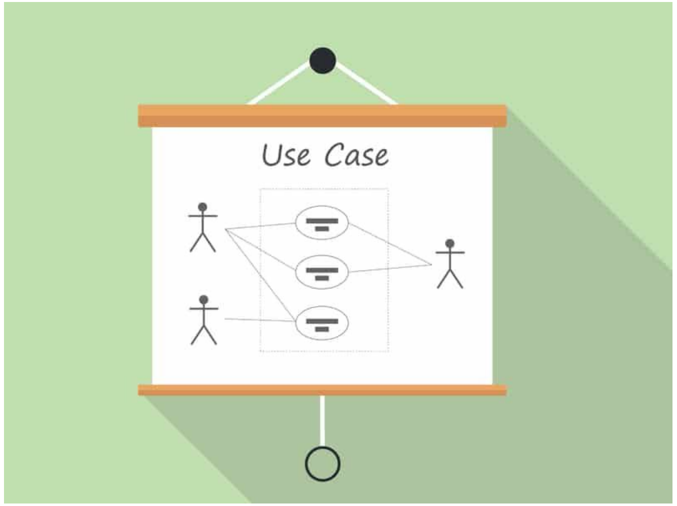
Another difference between user stories and use cases is use cases might not use customer language, at least for exception flows, since there are things that can go wrong that customer would not think about

- Invalid input data
- Invalid output data
- Something the feature relies on is down, busy, slow, …
- Requestor has exceeded limits
- ...

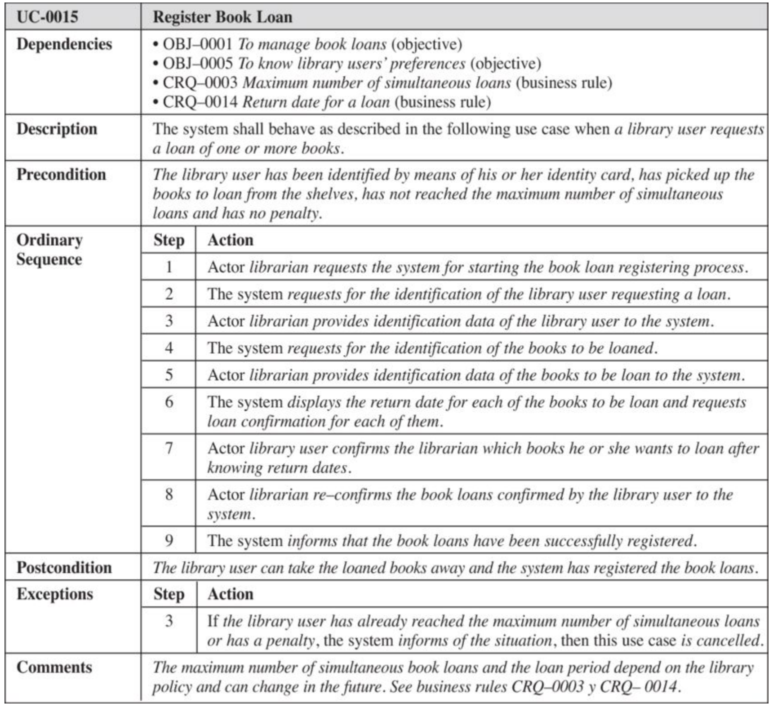


Not UML Use Case Diagrams

If you google for "use case", you may find something like this



We mean something more like this



UC-0015	Register Book Loan	
Dependencies	• OBJ–0001 *To manage book loans* (objective) • OBJ–0005 *To know library users' preferences* (objective) • CRQ–0003 *Maximum number of simultaneous loans* (business rule) • CRQ–0014 *Return date for a loan* (business rule)	
Description	The system shall behave as described in the following use case when *a library user requests a loan of one or more books.*	
Precondition	*The library user has been identified by means of his or her identity card, has picked up the books to loan from the shelves, has not reached the maximum number of simultaneous loans and has no penalty.*	
Ordinary Sequence	**Step**	**Action**
	1	Actor *librarian requests the system for starting the book loan registering process.*
	2	The system *requests for the identification of the library user requesting a loan.*
	3	Actor *librarian provides identification data of the library user to the system.*
	4	The system *requests for the identification of the books to be loaned.*
	5	Actor *librarian provides identification data of the books to be loan to the system.*
	6	The system *displays the return date for each of the books to be loan and requests loan confirmation for each of them.*
	7	Actor *library user confirms the librarian which books he or she wants to loan after knowing return dates.*
	8	Actor *librarian re–confirms the book loans confirmed by the library user to the system.*
	9	The system *informs that the book loans have been successfully registered.*
Postcondition	*The library user can take the loaned books away and the system has registered the book loans.*	
Exceptions	**Step**	**Action**
	3	If *the library user has already reached the maximum number of simultaneous loans or has a penalty*, the system *informs of the situation*, then this use case *is cancelled*.
Comments	*The maximum number of simultaneous book loans and the loan period depend on the library policy and can change in the future. See business rules CRQ–0003 y CRQ– 0014.*	

Use Cases are More Structured than Stories

Depending on how in-depth and complex you want or need to get, use cases describe a combination of the following elements:

- **Actor** – anyone or anything that performs a behavior (who is using the system)
- **Stakeholder** – someone or something with vested interests in the behavior of the system
- **Primary Actor** – stakeholder who initiates an interaction with the system to achieve a goal
- **Preconditions/Postconditions** – what must be true or happen before and after the use case runs
- **Triggers** – this is the event that causes the use case to be initiated
- **Main success scenario(s)** – aka basic flow or happy path (not necessarily literally a single execution path)
- **Alternative flows** – special cases
- **Exception flows** – alternative flows for error cases

Use case templates

Example

1. On Wednesdays, the housekeeper reports to the laundry room.
2. The housekeeper sorts the laundry that is there.
3. Then they wash each load.
4. They dry each load.
5. A. They fold the items that need folding. ⇐ basic flow

   B. They iron and hang the items that are wrinkled. ⇐ alternative flow

   C. They throw away any laundry item that is irrevocably shrunken, soiled or scorched. ⇐ alternative flow

Actors: Housekeeper, system that transports laundry to the laundry room.

Stakeholders: Owners/wearers of the laundry items.

Trigger: Dirty laundry has been transported to the laundry room.

Precondition: It is Wednesday.

Postcondition: Clean laundry is folded or hung. Damaged laundry is discarded.

Example - Further Details and Possible Exception Flows

1. A. The housekeeper reports to the laundry room.

   B. The housekeeper does not show up. ⇐ **exception flow**

   C. The laundry room is flooded (or other reasons the housekeeper cannot get to the laundry room). ⇐ **exception flow**
2. A. The housekeeper sorts the laundry that is there into hot, warm, cold wash (or other categories). ⇐ more detail needed

   B. The laundry is missing. ⇐ **exception flow**
3. A. Then they wash each load (in what order? what detergent, etc. is placed in washer?). ⇐ more detail needed

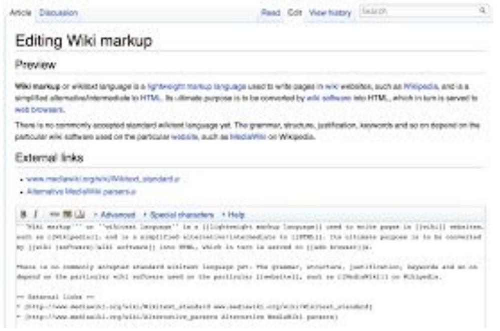
   B. The washing machine does not work. ⇐ **exception flow**

   C. The xxx load is too large for the washing machine and needs to be split (etc). ⇐ more detail needed / **exception**

   D. Washing machine takes too long to do all the loads before the housekeeper has to leave. ⇐ **exception flow**
4. And so on….

In extreme case, detail approaches an SRS or the actual program logic … but usually much simpler

Another Example



Article Discussion Read Edit View history Search

Editing Wiki markup

Preview

Wiki markup or wikitext language is a lightweight markup language used to write pages in wiki websites, such as Wikipedia, and is a simplified alternative/intermediate to HTML. Its ultimate purpose is to be converted by wiki software into HTML, which in turn is served to web browsers.

There is no commonly accepted standard wikitext language yet. The grammar, structure, justification, keywords and so on depend on the particular wiki software used on the particular website, such as MediaWiki on Wikipedia.

External links

- www.mediawiki.org/wiki/Wikitext_standard
- Alternative MediaWiki parsers

B I Advanced Special characters Help

```
'''Wiki markup''' or ''wikitext language'' is a [[lightweight markup language]] used to write pages in [[wiki]] websites,
such as [[Wikipedia]], and is a simplified alternative/intermediate to [[HTML]]. Its ultimate purpose is to be converted
by [[wiki [illegible]|wiki software]] into HTML, which in turn is served to [[web browser]]s.

There is no commonly accepted standard wikitext language yet. The grammar, structure, justification, keywords and so on
depend on the particular wiki software used on the particular [[website]], such as [[MediaWiki]] on Wikipedia.

== External links ==
* [http://www.mediawiki.org/wiki/Wikitext_standard www.mediawiki.org/wiki/Wikitext_standard]
* [http://www.mediawiki.org/wiki/Alternative_parsers Alternative MediaWiki parsers]
```

Title: Edit an article

Actor: Member *(Registered User)*

Description: Part of a Wiki system. The member edits any part (the entire article or just a section) of an article he/she is reading. Preview and changes comparison are allowed during the editing.

Preconditions: The article with editing enabled is presented to the member.

Triggers: The member invokes an edit request (for the full article or just one section) on the article.

Basic flow:

1. The system provides a new editor area/box filled with all the article's relevant content with an informative edit summary for the member to edit. If the member just wants to edit a section of the article, only the original content of the section is shown, with the section title automatically filled out in the edit summary.
2. The member modifies the article's content till satisfied.
3. The member fills out the edit summary, tells the system if he/she wants to watch this article, and submits the edit.
4. The system saves the article, logs the edit event and finishes any necessary post processing.
5. The system presents the updated view of the article to the member.

Postconditions (minimal guarantees):

- The article is saved and an updated view is shown.
- An edit record for the article is created by the system, so watchers of the article can be informed of the update later.

Alternate Flows (extensions)

2-3.

a. Show preview:

1. The member selects *Show preview* which submits the modified content.
2. The system reruns step 1 with addition of the rendered updated content for preview, and informs the member that his/her edits have not been saved yet, then continues.

b. Show changes:

1. The member selects *Show changes* which submits the modified content.
2. The system reruns step 1 with addition of showing the results of comparing the differences between the current edits by the member and the most recent saved version of the article, then continues.

c. Cancel the edit:

1. The member selects *Cancel*.
2. The system discards any change the member has made, then goes to step 5.

4a. Timeout: ...

…

In-Class Exercise

Again your goal is to develop an app (or service) called CONTAIN, for **CON**tact **T**racing for instruction **A**ss**I**sta**N**ts, which does "contact tracing" based on IA (instruction assistant) connections. An IA for xxx course is a contact of all the other IAs for xxx as well as for every student who takes xxx. Since an IA is also a student, they are a contact for all other students in every course they take as well as for all the IAs of those courses

The CANVAS API enables your app to find out which students are IAs for which courses. The SSOL API has an ID photo for most students (note to self: show courseworks photo roster). Your CONTAIN app wants to know which IAs and students actually attended a given class session, to limit false positives (when they weren't there)

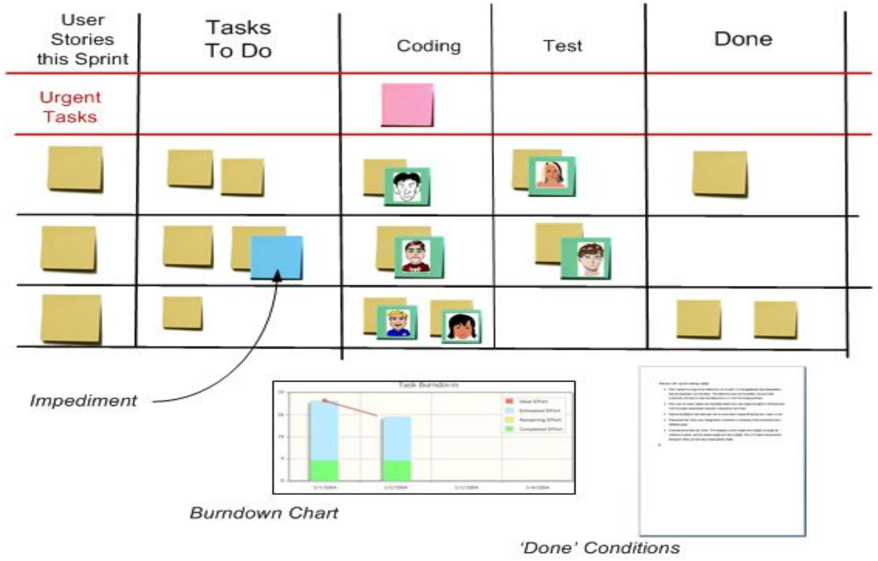
Invent an API for a drone-based attendance-recording service that can be used by CONTAIN. The drone flies around the room and records images (from multiple angles) of every person present in the classroom, or entering or leaving the classroom, throughout the class period. The service has access to the Canvas and SSOL APIs and includes facial recognition software

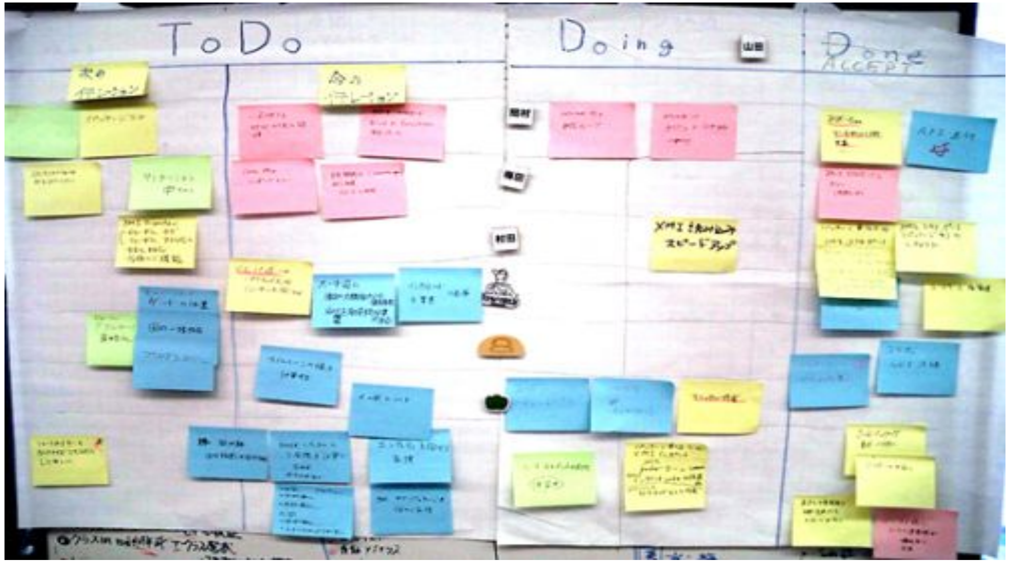
Write some use cases (not stories) for the API for this drone-based attendance recording service

Blank paper available

timer..

User Stories More Practical Granularity than Use Cases for Software Project Planning (because they fit on task boards)





Software Planning

Software planning ~= which features are developed in what order by whom

(At least) three kinds of software processes support software planning:

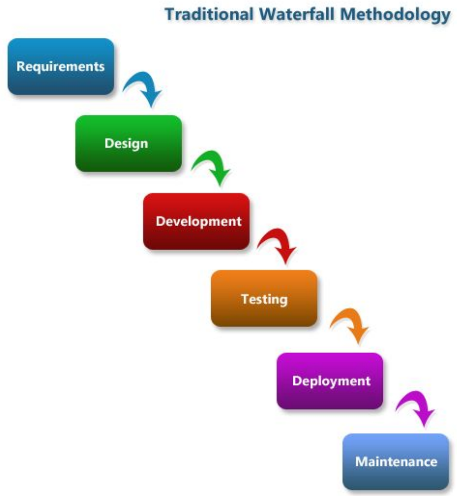
Ad hoc - start with null program, debug until it works ⇐ from my ugrad CS1 instructor

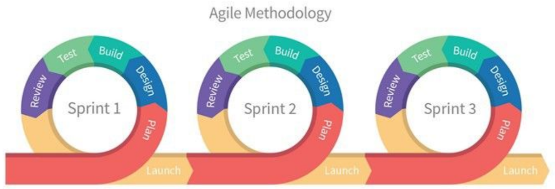
Sequential - requirements analysis (determine complete set of features); develop (design, code, test complete set of features); deploy; maintain (version 2.0)

Iterative - { requirements, develop, deploy } for small set of features, repeat with another small set of features, repeat again with more features, etc.

Waterfall (sequential) vs. Agile (iterative)

Waterfall ~= ! agile, agile ~= ! waterfall



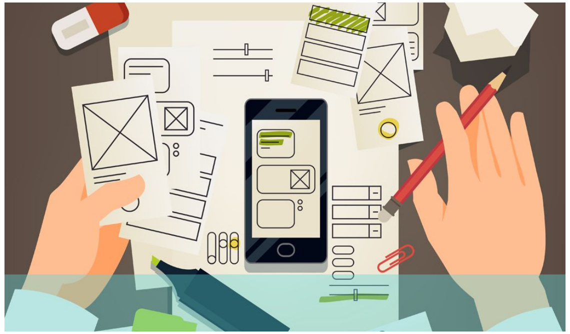


Waterfall (phase-driven)

Easy to explain to CS1 students and government procurement agents

Do the entire first phase, do the entire second phase, etc.

Optional prototyping during requirements phase



Then requirements frozen -
if requirements change, need to start over (with some reuse)

Design, code, test usually treated as separate phases,
with very limited feedback to each previous phase

Deploy all-at-once, everything after deployment is maintenance

Agile (adaptive)

Each sprint (iteration), typically two weeks,
is a tiny parallelized waterfall for a bite-sized chunk of the project

Deploy increments, typically quarterly,
possibly after every sprint

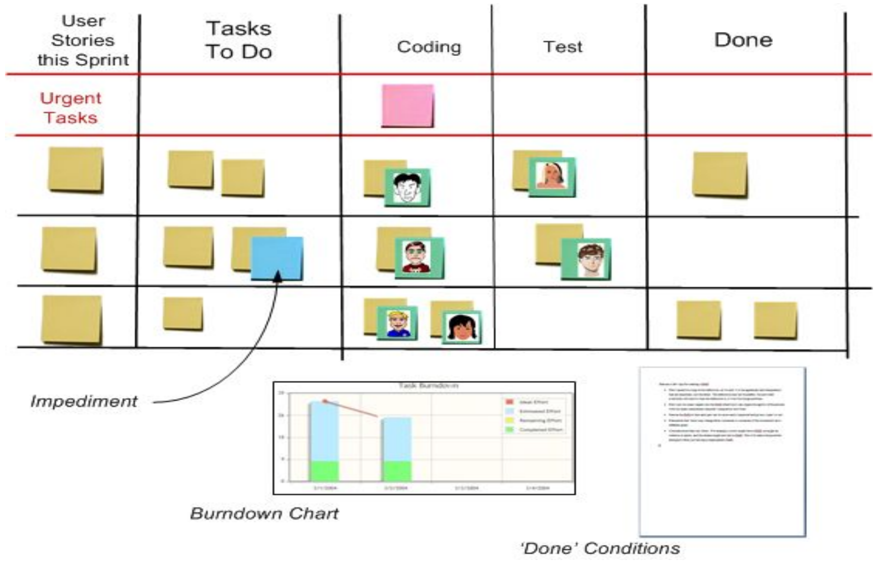
Often a “hardening” sprint as last iteration
before release, with only testing and
bug fixes (no new/changed features)

“Tiny Parallelized Waterfall” ???

This is my wording, not a technical term 😎

$$\begin{tikzcd}
\text{User Story} \arrow[dr] & & & \\
& \text{Tasks To Do} \arrow[dr] & & \\
& & \text{Coding} \arrow[dr] & \\
& & & \text{Test}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{User Story} \arrow[dr] & & & \\
& \text{Tasks To Do} \arrow[dr] & & \\
& & \text{Coding} \arrow[dr] & \\
& & & \text{Test}
\end{tikzcd}$$



Single user story often split into multiple tasks (tickets), e.g., corresponding to multiple personas

Coding typically includes developer testing, Test typically includes Code Review (before commit) followed by separate testing team (after commit)

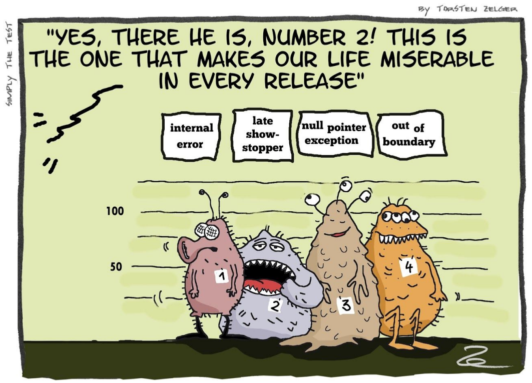
How Does Bug-Fixing Fit Into Sprints?

Small bugs found by developer fixed as part of their coding task, before sending for review

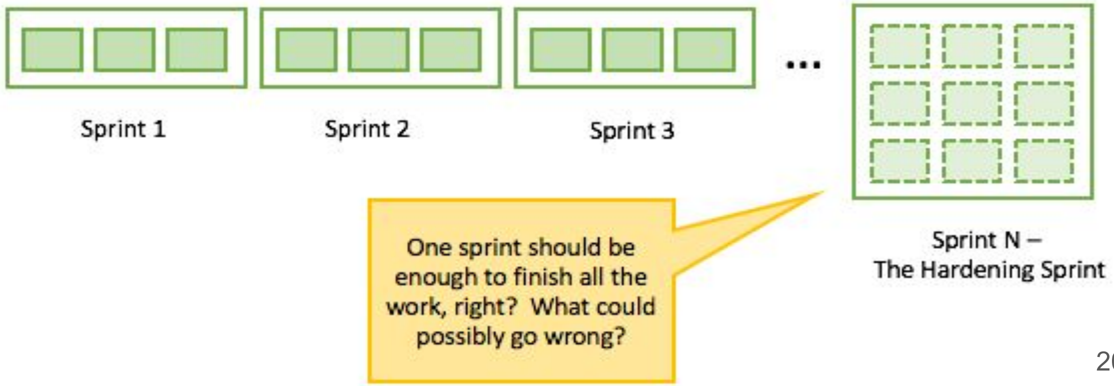
Large bugs, whether found by developer, separate testing team or user, often become their own user story ("bug story")

To add to current iteration need to remove something else

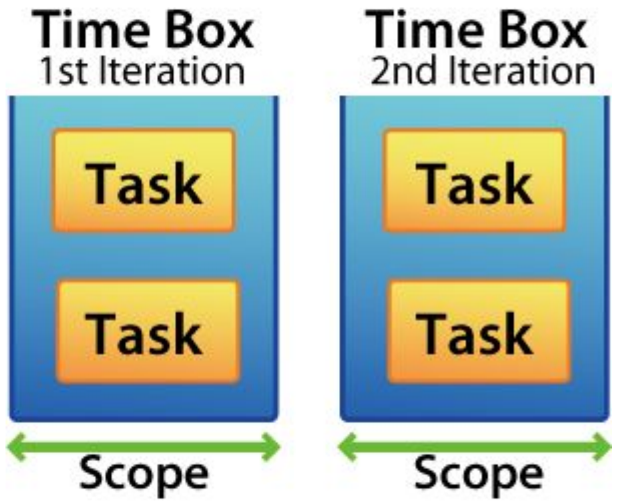
Could include in next iteration - or push off to hardening sprint



Lining up for the aftermath session


Sprint 1
Sprint 2
Sprint 3
...
Sprint N – The Hardening Sprint
One sprint should be enough to finish all the work, right? What could possibly go wrong?

Agile Concepts - Time Boxing vs. Scope Boxing



Fixed length iteration means scope is limited to what can be accomplished during that time

If more features (scope) are added to a iteration, then the length of the iteration necessarily increases

If less time is allocated to an iteration, then fewer features (scope) can be completed

If it turns out that the originally planned features will take longer than scheduled, then the scope must be reduced with some features pushed off to later sprint (or dropped)

- ➢ This course (and probably every regular course) is time-boxed

Agile Concepts - Daily Standup

- What did you accomplish yesterday?
- What will you do today?
- Are there any impediments in your way?

Why standup? To keep it short

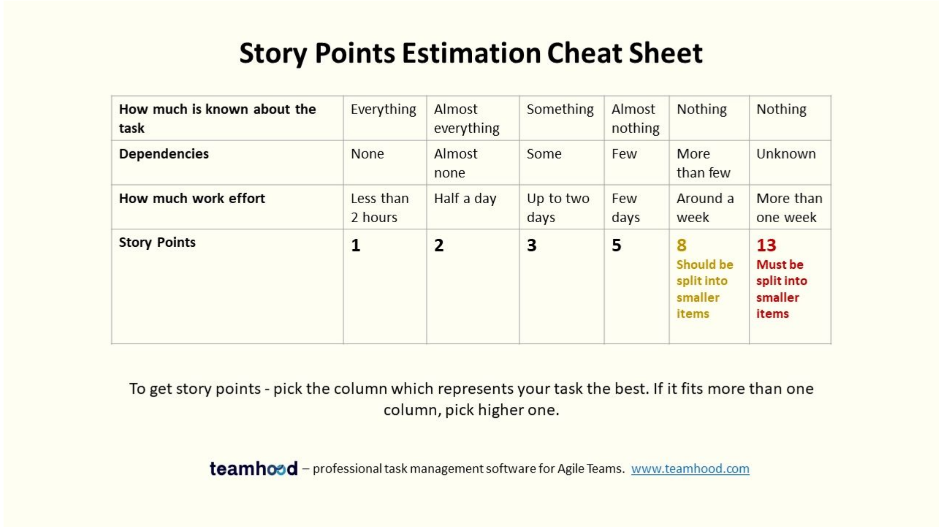
Agile Concepts - Project Velocity

Since sprints are fixed-length, there are a maximum number of hours/days available to get work done =
# developers (or pairs) x 10 working days x 8 hours/day x 70%

Why 70% instead of 100%? ⇐ project velocity

Computers have lives (hardware and software maintenance and downtime)
Developers have lives (personal days, sick days, unproductive days)
Projects have lives (training, impromptu meetings, customer interruptions)
Organizations have lives (CTO unexpectedly schedules full-day all-hands meeting)

Agile Concepts - Story Points



Story Points Estimation Cheat Sheet

How much is known about the task	Everything	Almost everything	Something	Almost nothing	Nothing	Nothing
Dependencies	None	Almost none	Some	Few	More than few	Unknown
How much work effort	Less than 2 hours	Half a day	Up to two days	Few days	Around a week	More than one week
Story Points	1	2	3	5	8 Should be split into smaller items	13 Must be split into smaller items

To get story points - pick the column which represents your task the best. If it fits more than one column, pick higher one.

teamhood – professional task management software for Agile Teams. www.teamhood.com

How do you calculate project velocity?

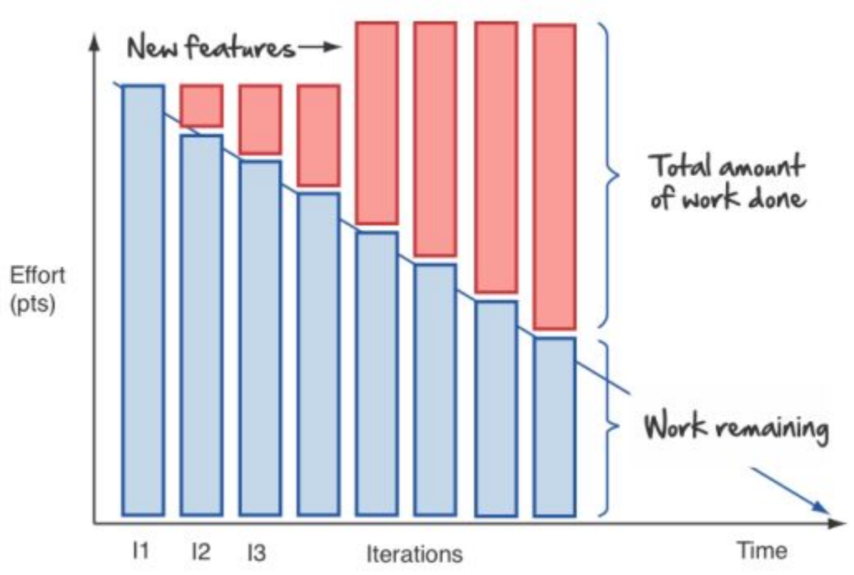
Each task estimated as some number of hours, "story points", or some other time/effort unit

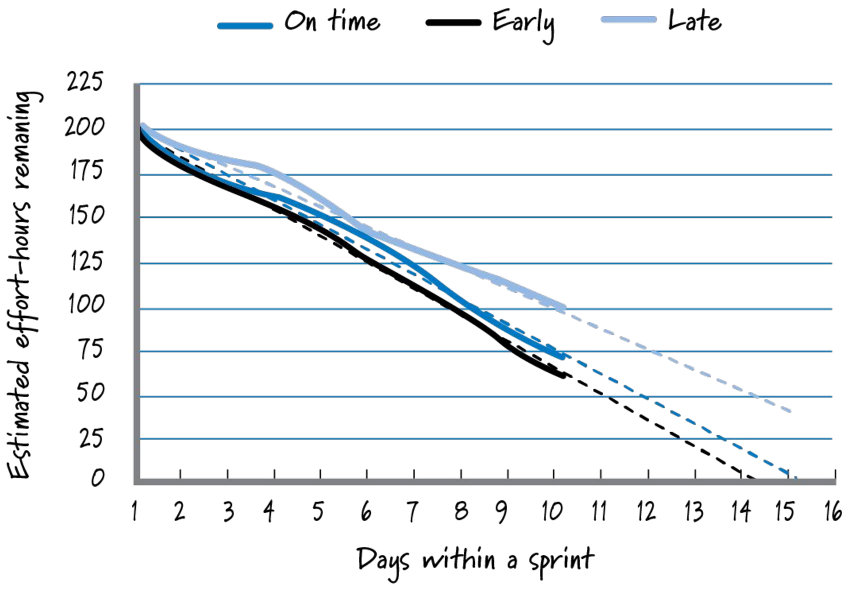
Estimates come fromexperience with similar tasks, relative "size" of different tasks (baseline, ½ x baseline, 3 x baseline, 5 x baseline), guess

PV = Actual / Ideal

Agile Concepts - Burndown Chart

Augments task board to visualize how much has been done and how much there is left to do





Individual Mini-Project part 2 and part 3 Deadlines Extended

Three parts:

1. Implementing a simple game graded - contact grader for regrade (or contact Head IA - Claire - if you don't know who graded)
2. Testing the game due yesterday, but you can (re)submit up to October 5 with no penalty
3. Saving game state nominally due October 6, but you can submit up to October 9 with no penalty

Team Project

Preliminary Project Proposal due October 13