COMP33711

Agile Software Engineering

(Agile Methods Theme)

How Do We Know What to Build Next?

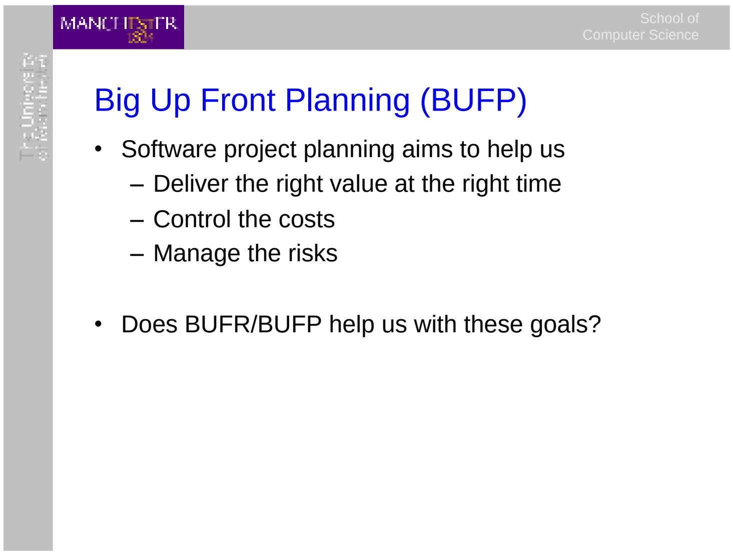
2017/2018 Academic Session

Suzanne M. Embury
Room KB 2.105
School of Computer Science
University of Manchester
Oxford Road
Manchester M13 9PL
U.K.

Suzanne.M.Embury@manchester.ac.uk
Telephone: (0161) 275 6128

Sessions 4 & 5: How Do We Know What to Build Next?

Having seen how we can gather and expand on requirements in a lightweight, just-in-time/just-enough fashion, we now turn our attention to the question of how software project planning can be carried out in the same way. We'll try our hands at a simple agile planning process, to show how user stories (describing value-rich thin end-to-end slices of functionality) allow us to plan our work in a way that delivers value early, while also allowing the team to cope gracefully with change and to gradually evolve the means to deliver on a predictable schedule.



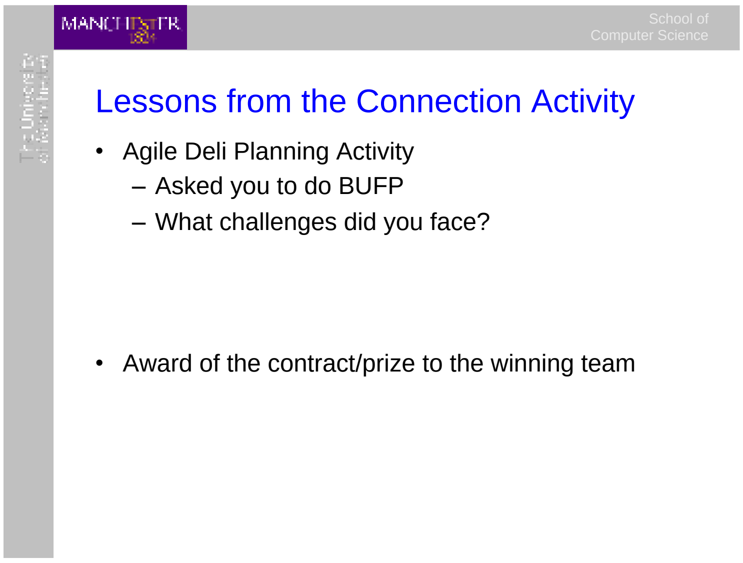
School of Computer Science

Big Up Front Planning (BUFP)

- Software project planning aims to help us
  - Deliver the right value at the right time
  - Control the costs
  - Manage the risks

- Does BUFR/BUFP help us with these goals?

Notes



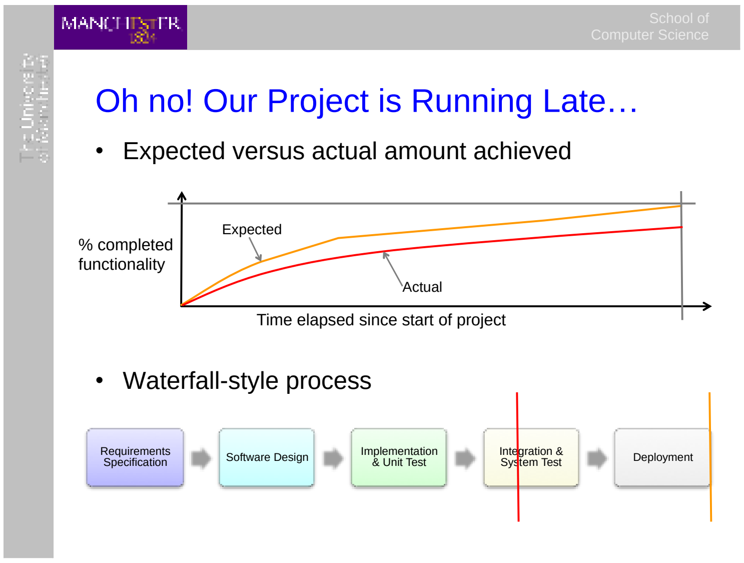
School of Computer Science

Lessons from the Connection Activity

- Agile Deli Planning Activity
  - Asked you to do BUFP
  - What challenges did you face?

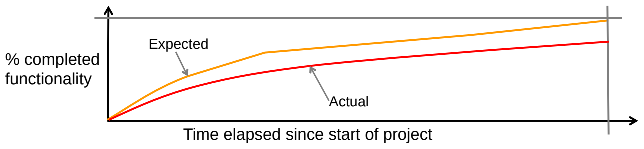
- Award of the contract/prize to the winning team

Notes

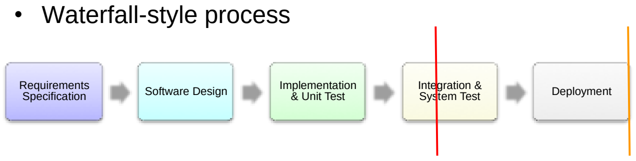


Oh no! Our Project is Running Late…

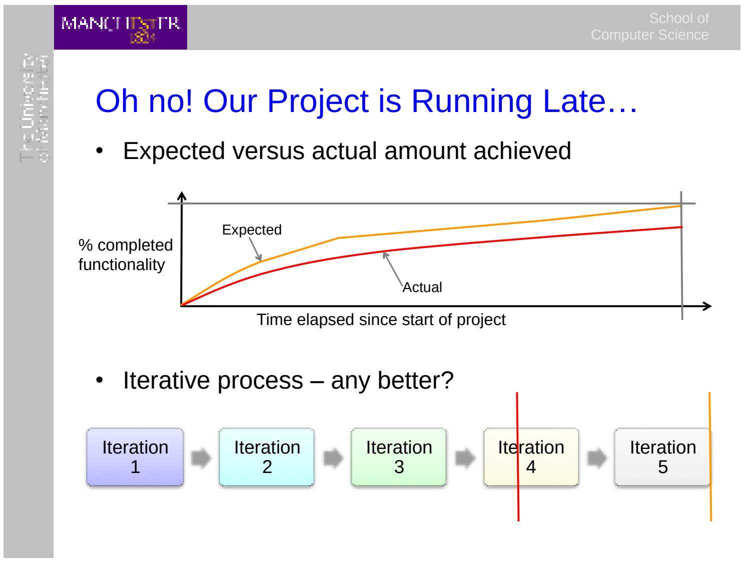
- Expected versus actual amount achieved



- Waterfall-style process

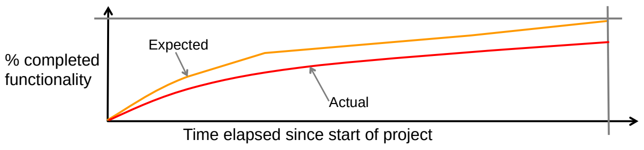


Requirements Specification
Software Design
Implementation & Unit Test
Integration & System Test
Deployment

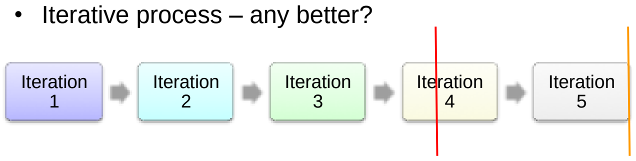


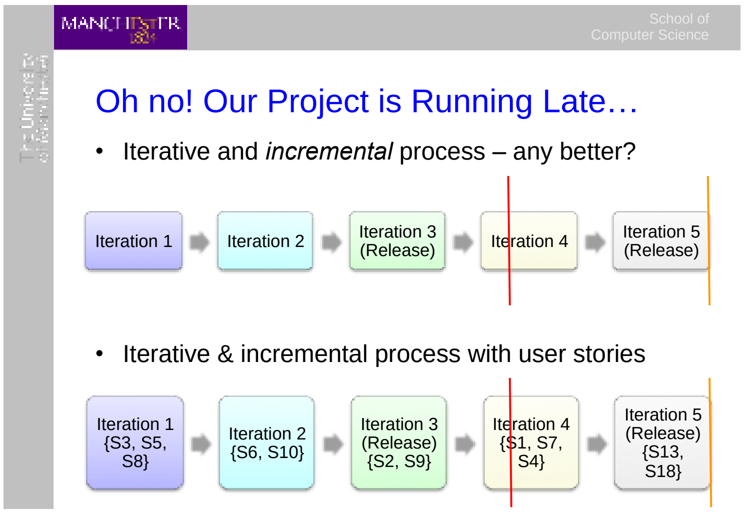
Oh no! Our Project is Running Late…

- Expected versus actual amount achieved



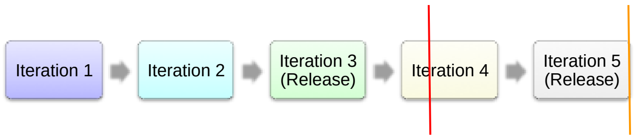
- Iterative process – any better?



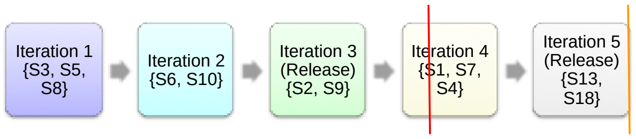


Oh no! Our Project is Running Late…

- Iterative and *incremental* process – any better?



- Iterative & incremental process with user stories



Oh No! Our Project Has Been Cancelled!

On some projects, the problem is not that the work is running late, but that the funds are running out. It can sometimes be the case that a project is cancelled before it was originally planned to end. This might not be due to any problem with the project itself, but might be caused by other difficulties experienced by the customer organisation, like the sudden closing down of a major market, the sudden withdrawal of a key product, or even a change of personnel in top management. When organisations have to tighten their belts, even a successful project might need to be cancelled part way through.

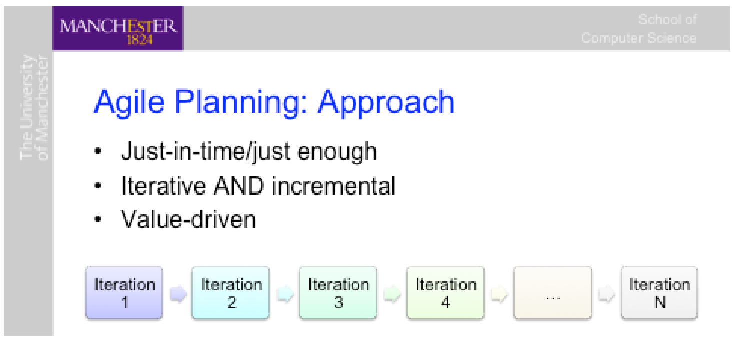
If this happens to a waterfall project, then it is possible that nothing useful will come out of the project. All the months of work on requirements documents, plans and the coding of individual components may turn out to have been completed wasted. If the team is lucky, then more or less complete versions of the core components might be shippable in some form by the time of the cancellation. But this will be a matter of luck, rather than judgment.

Imagine how it must feel when you've been working on a project for months, or even years, to have it be cancelled and deliver nothing… Cancellation for these teams is a disaster.

On an agile project, on the other hand, the team will have the work of any completed iterations in a form that is ready to deploy and to deliver value, no matter when the project is cancelled. Indeed, the team may already have deployed some useful part of the code. Rather than feeling that all their time has been wasted, the team feels they have used the time available to good effect, and are ready to move on to start work on the next impact needed by the organisation.

I recently heard about an agile team whose project was cancelled and then refunded, a few months later. The team just stuck their story cards back up on the wall, reprioritised them and picked up where they left off, working to build the next most useful slice of functionality.

The greater resilience of the agile team to this kind of sudden change is due to the emphasis on *delivering early and often* and on *delivering real value* to the organisation.


MANCHESTER 1824

School of Computer Science

The University of Manchester

Agile Planning: Approach

- Just-in-time/just enough
- Iterative AND incremental
- Value-driven

Iteration 1 → Iteration 2 → Iteration 3 → Iteration 4 → … → Iteration N

Notes

A Typical Agile Planning Process

- Walkthrough to illustrate basic concepts
  - One variant among many possible processes
- By the end, you should have an idea of
  - Basic iterative planning approach
  - Agile approaches to estimation
  - Basic use of story/task boards in agile planning
  - How this approach can deliver value early & often
  - How this approach can reduce waste
- We ignore testing for now (despite its importance)

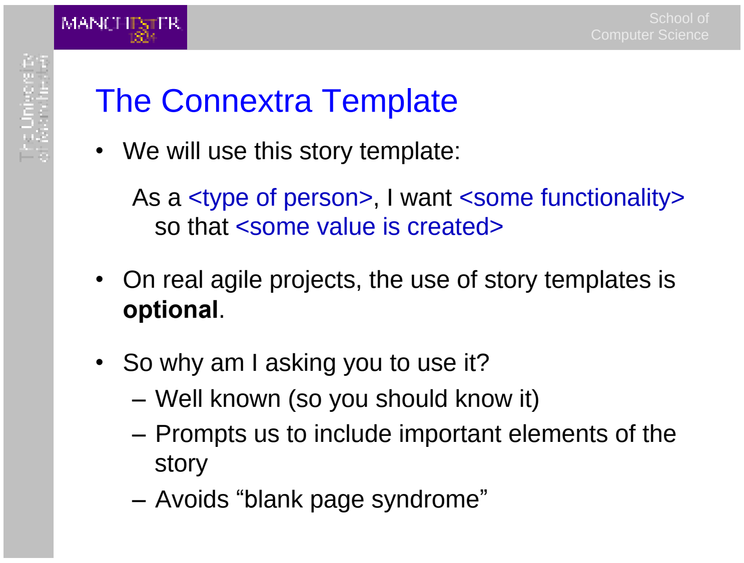
Step 1: Organise your Team

- Divide your team into:
  - A customer (the person with the domain knowledge)
  - A business analyst
  - An even number of developers
- Odd sized team?
  - Export one developer to another odd-sized team
  - Import a developer from another odd-sized team
  - Have an agile coach role? QA role? Two customers?

Step 2: Hold a Story Writing Workshop

- The first step is to gain an understanding of the high-level requirements for the project
- The customer, business analyst, developers and QA all bring their expertise to the task
- We'll try our hands at it now
  - Stories that will have impact
  - Thin end-to-end slices
  - Using the Connextra story template

Notes


School of
Computer Science

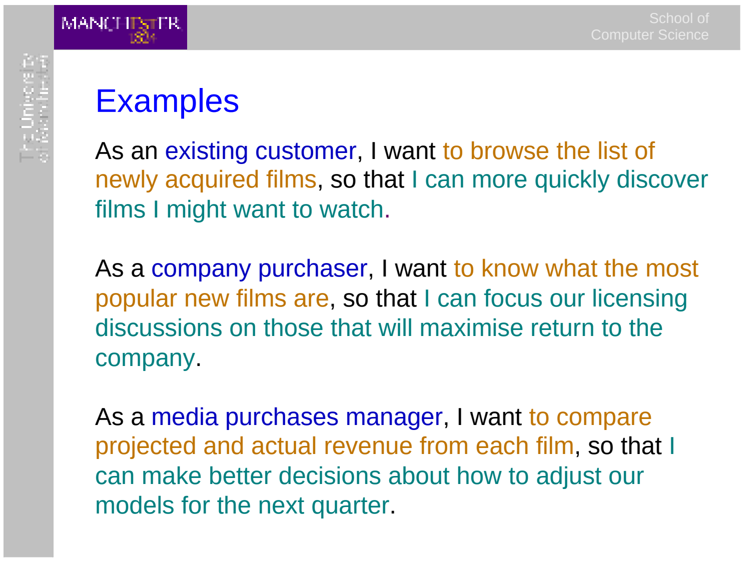
The Connextra Template

- We will use this story template:

  As a <type of person>, I want <some functionality>
  so that <some value is created>

- On real agile projects, the use of story templates is **optional**.

- So why am I asking you to use it?
  - Well known (so you should know it)
  - Prompts us to include important elements of the story
  - Avoids “blank page syndrome”

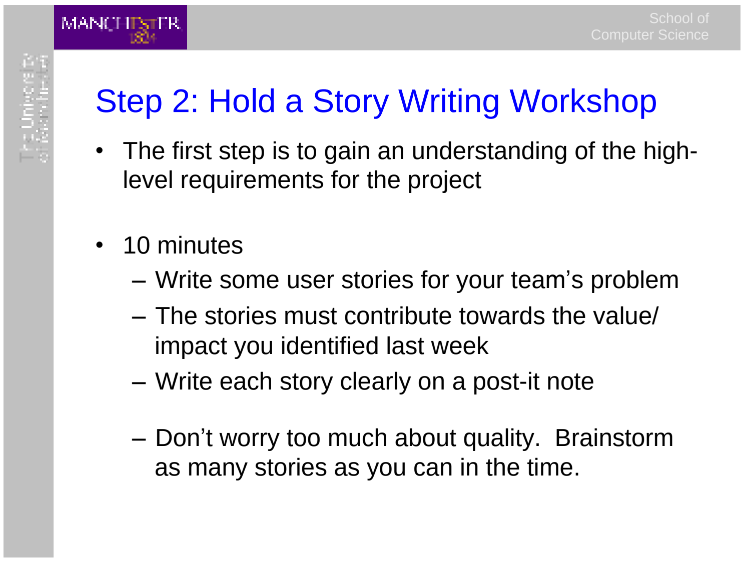

School of
Computer Science

Examples

As an existing customer, I want to browse the list of newly acquired films, so that I can more quickly discover films I might want to watch.

As a company purchaser, I want to know what the most popular new films are, so that I can focus our licensing discussions on those that will maximise return to the company.

As a media purchases manager, I want to compare projected and actual revenue from each film, so that I can make better decisions about how to adjust our models for the next quarter.


School of
Computer Science

Step 2: Hold a Story Writing Workshop

- The first step is to gain an understanding of the high-level requirements for the project

- 10 minutes
  - Write some user stories for your team’s problem
  - The stories must contribute towards the value/ impact you identified last week
  - Write each story clearly on a post-it note

  - Don’t worry too much about quality. Brainstorm as many stories as you can in the time.

Getting Our Priorities Right

Software requirements prioritisation is an important task for a software engineering team. We have already seen in the YAGNI connection exercise, and in the stories from placement students in the classes, how easy it is for teams to build software that no one wants. By carefully prioritising our requirements, we can hopefully avoid this kind of waste.

But prioritising requirements turns out not to be that easy, especially if we wish to do it in a defensible and objective way. Different stakeholders for a new software development will typically have different goals for the system, as well as their own preferred ways of doing things, and this means they would prioritise the requirements in a different order. A number of increasingly sophisticated methods for requirements prioritisation have been proposed over the years, aiming to make this task easier and more effective, involving more complex ways of comparing requirements against one another and of eliciting real user needs through interactions with stakeholders. But, requirements prioritisation remains a difficult and subjective task. It can take a long time to perform in a defensible manner, and as soon as we are done the requirements change, so our work is invalidated.

In agile approaches, when some task is costly and time-consuming to do, and when the results will quickly become out of date, we ask ourselves whether we really need to do that task at all. This is definitely true of requirements prioritisation, and therefore on an agile project we try to spend as little time on this as we can sensibly get away with.

One way to simplify the process is to concentrate on assigning relative priorities between stories, rather than giving each story its own absolute priority score. This means we don't need to spend time discussing whether story X is a 4 or a 5 or a 5.5. We just have to decide whether story X is more important than story Y or Z.

Since we should have a clear idea of what the value of each of our user stories is, the process of prioritising stories is really a process of prioritising values. When we have decided which of our values we should try to meet first, we then have to order the stories that deliver that value, relative to each other.

Step 3: Release Planning

- Aim: group stories into meaningful releases
  - Get high level picture of what the overall project must develop
  - Seek to identify "minimum marketable features"
    - What are the smallest increments of functionality (i.e. subsets of stories) that customers will pay for/consider as delivering real value?
  - Seek to meet external deadlines
- Importance of "story independence"

Story Dependence

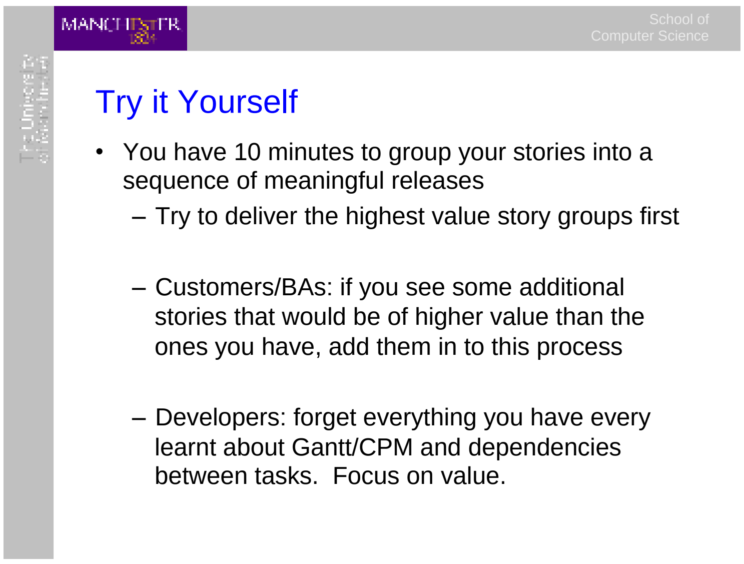
Which of these stories offers the highest value?

1. Browse the list of currently enrolled students
2. Create a new student
3. Create a new course unit
4. Unenrol students from a course unit
5. Enrol a student on a course unit
6. Modify student name, registration number and contact details
7. Enter marks for student for some assignment

Story Independence Means…

- We can choose to work on the highest value stories first
  - Technical independence
  - Value independence
- This is the first of several benefits we get from describing the required functionality in thin end-to-end slices.

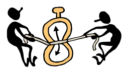
Notes



Try it Yourself

- You have 10 minutes to group your stories into a sequence of meaningful releases
  - Try to deliver the highest value story groups first
  - Customers/BAs: if you see some additional stories that would be of higher value than the ones you have, add them in to this process
  - Developers: forget everything you have every learnt about Gantt/CPM and dependencies between tasks. Focus on value.

Notes



Release Planning in the Wild

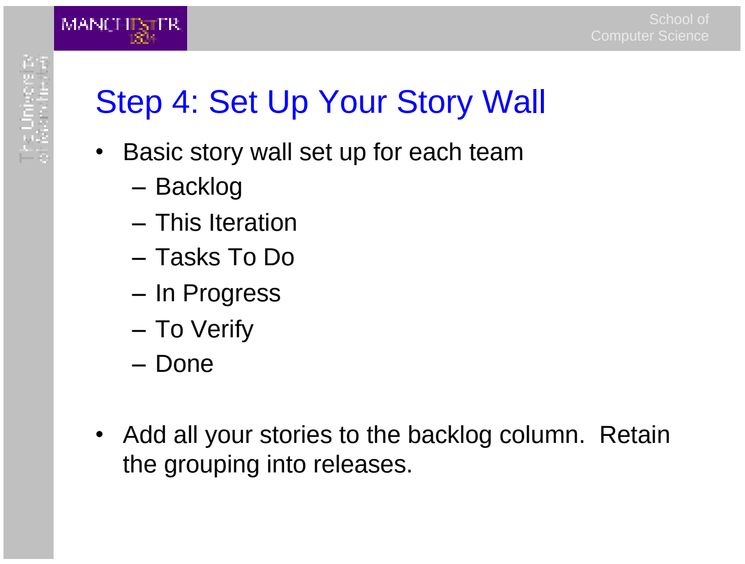
The approach we have practiced in this class is a very simple release planning process, just to get the basic ideas across. In practice, we also have to think about when the release plan will be delivered and therefore when the project will start to create value for the organisation. We have also to balance competing interests in the project, as well as liaising with other parts of the organisation involved in the release of new software, such as those involved with training, technical writing, marketing and technical support. (On my first development project, for example, we had to hold meetings with the local trade union, to check that the changes in working practices our software was to support were acceptable to the workforce.)

There are two basic approaches to release planning: date-based and story-based. In date-based release planning, we have some deadlines (possibly hard), by which we must deliver a solution. Our task in release planning is to work out what functionality we can deliver before the deadline that solves the problem we have been set. To do this, we try to group stories into sets that each correspond to a release by some important deadline. The deadline may be too soon for a full solution to be delivered, in which case some hard decisions have to be made. Is there a subset of the full solution that *can* be delivered by the deadline, and that will deliver the core parts of the solution? Solving this kind of problem requires flexibility on both the technical *and* the business sides of the issue.

In story-based release planning, we have a set of stories that we hope to deliver, and we want to know how long it will take to complete them. We concentrate on identifying stories that complement each other to deliver some important benefit or facilitate some important behaviour change.

As with the detail of user stories, in release planning, we care more about getting the commitments that we must implement in the near term right, than we do about commitments that are further away from implementation. Nothing that we decide in release planning is binding. When we roughly assign stories to iterations that are a couple of weeks away, we do so only to see what the future might look like (to see what one plausible development future looks like), and not as a firm commitment that those stories will definitely be implemented in those specific iterations.

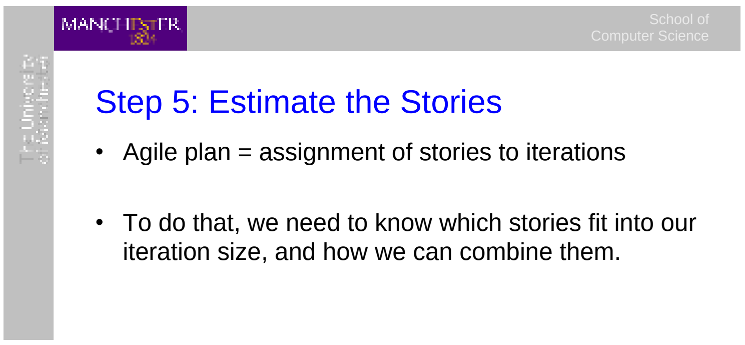
Notes


School of Computer Science

Step 4: Set Up Your Story Wall

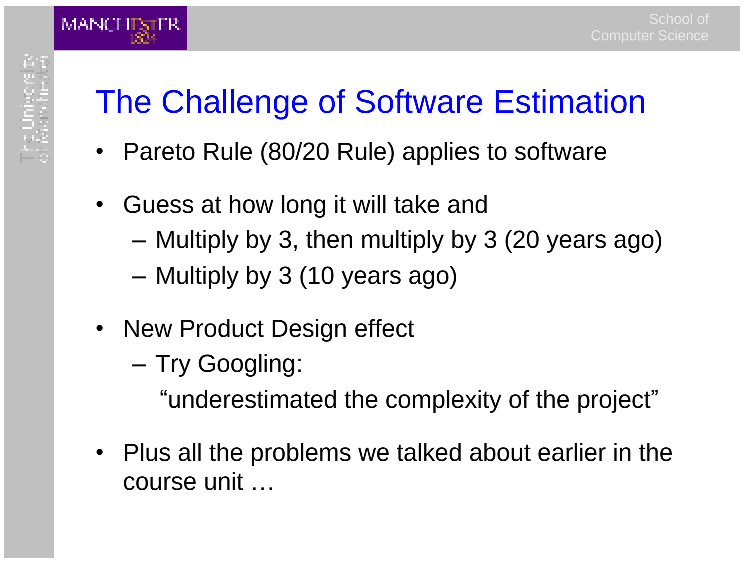
- Basic story wall set up for each team
  - Backlog
  - This Iteration
  - Tasks To Do
  - In Progress
  - To Verify
  - Done
- Add all your stories to the backlog column. Retain the grouping into releases.

A story wall is an example of the agile practice of using Big Visible Charts. The idea is that rather than having loads of project progress information more or less invisible inside a project management tool, it is better to have a couple of key metrics shown visually on a large chart, placed right beside where the development team work. This practice is also sometimes called the Information Radiator, since the idea is to present the key information in such a way that it can quickly and easily transmit its message to those nearby.


School of Computer Science

Step 5: Estimate the Stories

- Agile plan = assignment of stories to iterations
- To do that, we need to know which stories fit into our iteration size, and how we can combine them.

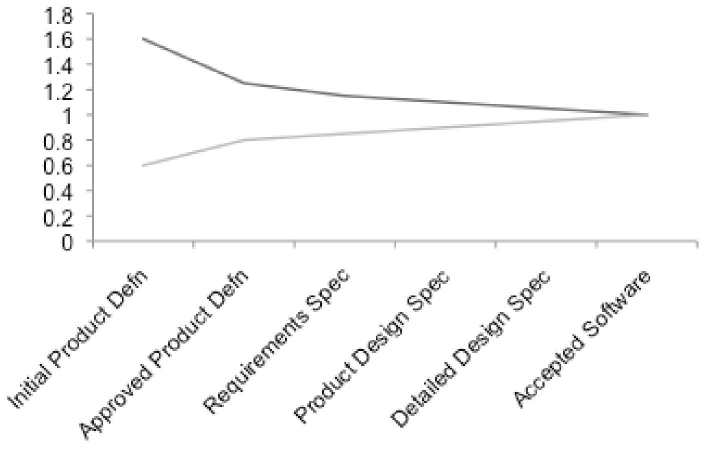

School of Computer Science

The Challenge of Software Estimation

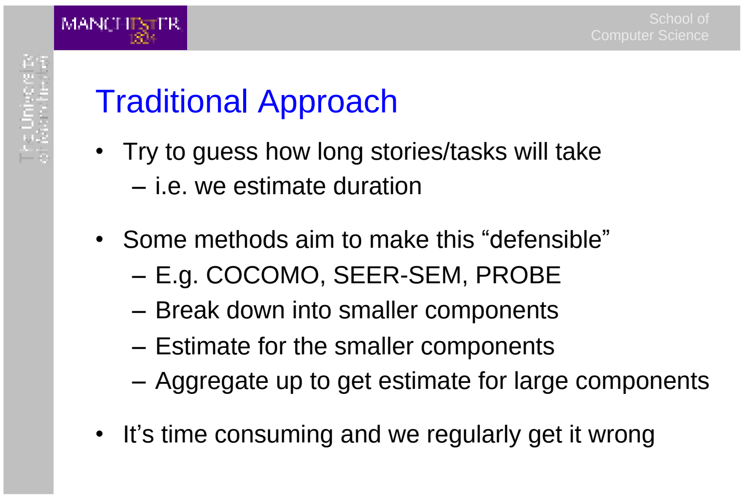
- Pareto Rule (80/20 Rule) applies to software
- Guess at how long it will take and
  - Multiply by 3, then multiply by 3 (20 years ago)
  - Multiply by 3 (10 years ago)
- New Product Design effect
  - Try Googling:
    “underestimated the complexity of the project”
- Plus all the problems we talked about earlier in the course unit …

Boehm's Cone of Uncertainty

The figure below is taken from studies by Barry Boehm of how real software projects behave [Boehm 81]. It shows the errors in our estimates at different points in the waterfall lifecycle. So, at the beginning of the project, our estimates might be as high as 1.6 times the final, actual duration, or they could be as low as 0.6 times the actual duration. As we progress through the project, we gain knowledge, which allows us to make better estimates. By the end of the project, when we have near complete knowledge, we can produce estimates that are 100% accurate. Unfortunately, the time when we most need the estimates is at the start of the project, when they are at their least accurate.



The solution to this problem is to plan and replan at regular intervals throughout the project, so that errors can be corrected and are limited in their scope (just until we create the next version of the plan). This would be a lot of work if we were creating a full software development plan every time, but we are not doing that. We are just making a tentative assignment of stories to releases and iterations, to give a sanity check on what we think we can achieve with the resources given to us.



School of
Computer Science

Traditional Approach

- Try to guess how long stories/tasks will take
  - i.e. we estimate duration
- Some methods aim to make this "defensible"
  - E.g. COCOMO, SEER-SEM, PROBE
  - Break down into smaller components
  - Estimate for the smaller components
  - Aggregate up to get estimate for large components
- It's time consuming and we regularly get it wrong

Agile Approach

- Key idea #1: estimate effort, derive duration

  Duration = Effort / Number of People

- Key idea #2: estimate effort in terms of “story size”
- Key idea #3: estimate *relative* size, not absolute size

Notes

Estimating Relative Size of Stories

- Which story is bigger?

As a fresher, I want to be e-mailed 15 minutes before each class in the first week, so that there’s less chance I’ll miss something important.

As a lecturer, I want to track the progress of printing requests for teaching materials such as handouts, so that I can save time having to go into ACSO to check on them.

- Not a trivial decision, but much easier than saying exactly how long each one will take to implement.

Nebulous Units of Time (NUTs)

- Use **story points** as the unit of story size
  - Also called Jelly Beans/Gummi Bears/…
- If story A estimated at 2 story points and story B at 4, that means (only) that B is about twice the size of A.
- The number of story points a team delivers in an iteration is called the **project velocity**.
- Teams use the early iterations of a project to learn how many story points they can deliver in one iteration.
  - This **predicted velocity** is then used for planning

Project Velocity Example

- A team commits to deliver 4 stories in the next iteration:
  - Story A is estimated at 2 story points
  - Story B at 5 story points
  - Story C at 5 story points
  - Story D at 8 story points
- During the iteration, the team delivers stories A, B and D, but not C
- Their velocity for the iteration was 15 story points.
- In the next iteration, they commit to 15 story points worth of stories

Notes

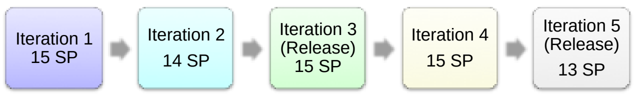
From Story Points to Durations

- So, once we have:
  - Consistent relative story point estimates for our stories
  - A predicted project velocity in story points
  - An iteration length (e.g. 2 weeks, 1 month)

  we can derive durations for our project.

Example

- Iteration length = 2 weeks
- Predicted project velocity = 15 story points
- Total story points assigned to iterations = 72
- Requires 5 iterations to complete
- So duration is 5x2 weeks = 10 weeks

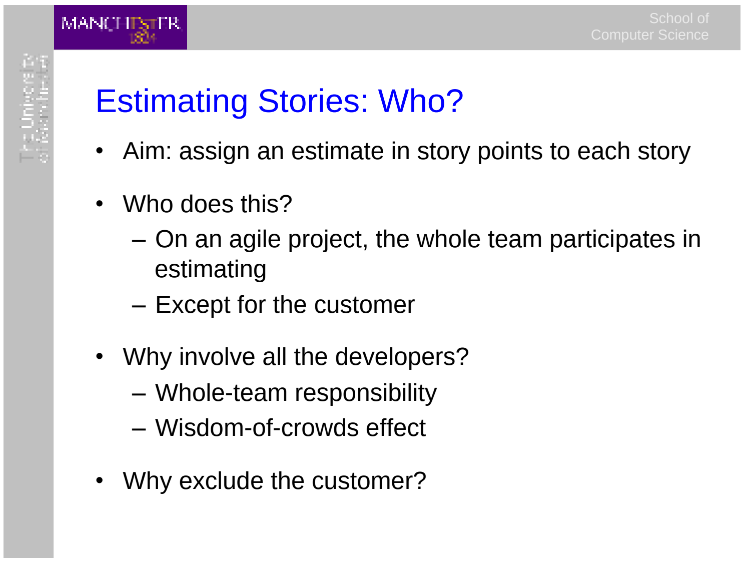


Story Points Are Not the Only Unit

An alternative to story points as a unit for estimating stories is the *ideal day* [Cohn 2006]. Traditionally, we have estimated the durations of software tasks in hours or days (or weeks). But, these units are somewhat unhelpful, because we do not normally have the luxury or the stamina to concentrate on just one task for a complete working day. We also have to factor in attending meetings, arranging meetings, answering e-mail, dealing with queries that come up, etc, etc. We just don't know how much time we will get to work on the assigned software task in any one day, so estimating its duration in these (so-called) *real days* is unlikely to give a true picture. The alternative is to use the notion of an ideal day, which really does correspond to a day of work on the software task. But, having estimated a story at 4 ideal days, for example, we do not then map that on to 4 real days. We know that it will take somewhat longer than that.

Other members of the agile community prefer to avoid using numbers as a measure of story size, because of the difficulty of being consistent with anything more than 3-4 points on the scale, and the ease with which those numbers can be misused. Instead, they propose the use of size categories. They warn that giving sizes as numbers leads us to do questionable things, such as adding together all the story points for an iteration. Does it make sense to add story points, they ask, when these numbers describe only relative sizing and not any real quantity? A popular set of categories for use in story sizing is: extra small (XS), small (S), medium (M), large (L) and extra large (XL). For obvious reasons, this approach is called "*Tee-Shirt Sizing*". We used this kind of size category for the stories in the Agile Lego game at the start of the course unit.

There is no reason why these different approaches cannot be combined on a project, if that is appropriate. For example, we might use tee-shirt sizing at the very start of a project, when too few stories have been estimated to allow detailed relative estimating using story points, and then switch to story points later.



School of
Computer Science

The University of Manchester

Estimating Stories: Who?

- Aim: assign an estimate in story points to each story
- Who does this?
  - On an agile project, the whole team participates in estimating
  - Except for the customer
- Why involve all the developers?
  - Whole-team responsibility
  - Wisdom-of-crowds effect
- Why exclude the customer?

Wisdom of Crowds

- Proposition: a group of people (including non-experts) can estimate more accurately than an individual (even an expert individual)
- Let’s test this out…
  1. How many sweets in this box?
  2. How many steps does Iliada take when walking from the back of the room to the front?
  3. How many pages in this book?
- Write your guesses on an index card
- Hand it to a teaching assistant

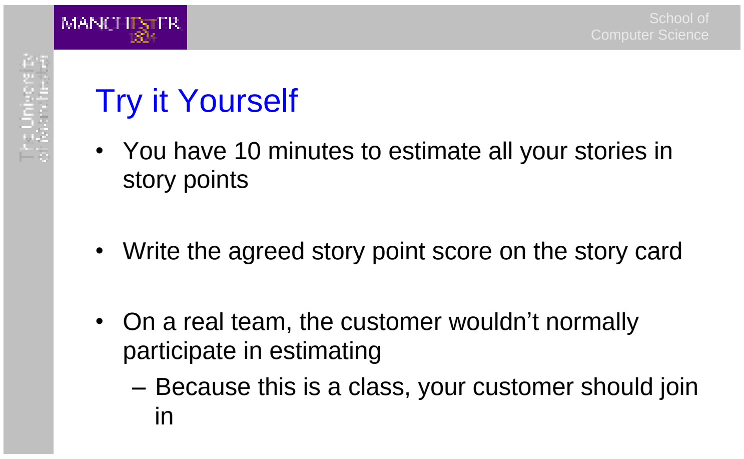
Estimating Stories: How?

- We’ll use Planning Poker
  - Proposed by James Grenning
  - Aim: to get good quality estimates quickly, by encouraging whole team participation
- You will need:
  - Some stories to estimate
  - A set of planning poker cards for each team member
  - (Optional) an egg timer
  - (Really Optional) a poker table

Planning Poker: How to Play

- Choose a story to estimate
- Someone (product owner is usual) describes it briefly
- Team members decide on story points, and place the relevant card **face down** on the table
  - Product owner does not take part in this step
- When everyone has estimated, turn cards over
- High/low estimators explain their choice
- If consensus not achieved, take back cards and estimate again
- Egg timer can be deployed at any time, by anyone

Notes

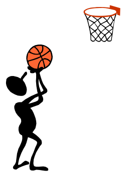


MANCHESTER 1824 — School of Computer Science — The University of Manchester

Try it Yourself

- You have 10 minutes to estimate all your stories in story points
- Write the agreed story point score on the story card
- On a real team, the customer wouldn't normally participate in estimating
  - Because this is a class, your customer should join in

Notes

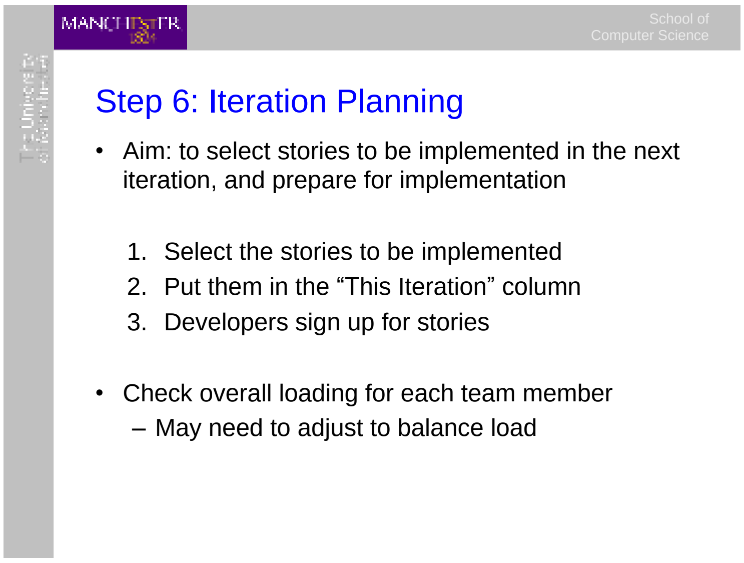


Games for Estimating Story Size

Several games have been proposed for story size estimation. These games are not "coaching games", like the Agile Lego Game, designed for helping to convey agile concepts. Instead, they are "development games"; they are games that we play while carrying out an agile software project, to help us get to a better outcome.

Games are particularly useful for story estimating, since they promote full participation from all developers, regardless of levels of expertise or seniority or personality. It is important that all perspectives on a story are heard when estimating, as it is hard to expect a single person to be able to consider all angles on a story. When we collaborate, we can get an all-round view of a story very quickly, which helps estimation to be both more accurate and more efficient.

Some other games that you might like to investigate are Paper-Scissors-Stone (which is based on the well known children's game of the same name) and White Elephant Sizing (in which we move stories between tee-shirt size categories until consensus is reached). Information on both these games can be found on TastyCupcake.org. A further famous example of a development game is the XP Planning Game, in which the entire planning process is converted into a set of game moves, and rules for who can play them.



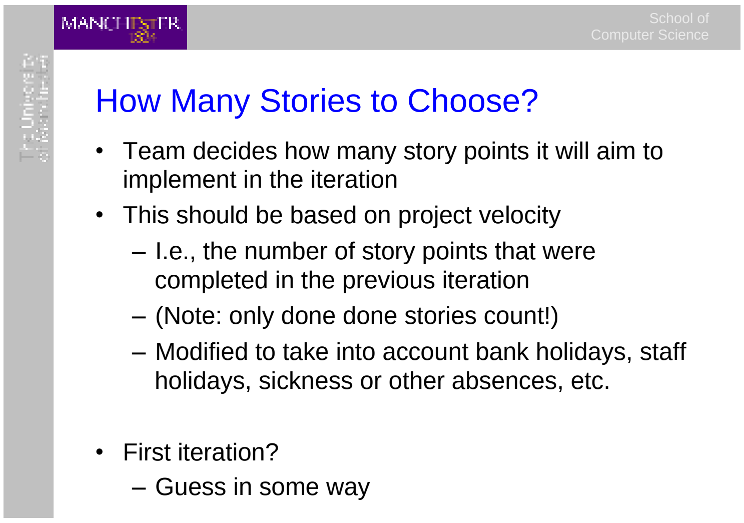
MANCHESTER 1824 — School of Computer Science — The University of Manchester

Step 6: Iteration Planning

- Aim: to select stories to be implemented in the next iteration, and prepare for implementation

  1. Select the stories to be implemented
  2. Put them in the "This Iteration" column
  3. Developers sign up for stories

- Check overall loading for each team member
  - May need to adjust to balance load


School of
Computer Science

How Many Stories to Choose?

- Team decides how many story points it will aim to implement in the iteration
- This should be based on project velocity
  – I.e., the number of story points that were completed in the previous iteration
  – (Note: only done done stories count!)
  – Modified to take into account bank holidays, staff holidays, sickness or other absences, etc.
- First iteration?
  – Guess in some way

Notes

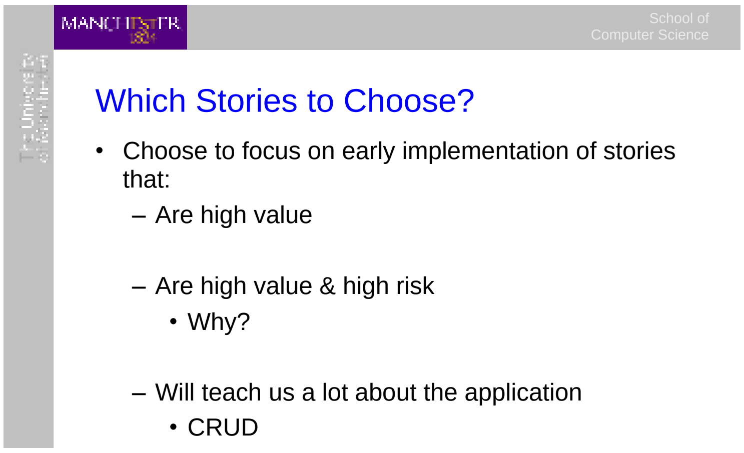
What Can We Commit to in this Iteration?

There have been a number of proposals for how teams can select a target number of story points for an upcoming iteration.

One approach, known as "Yesterday's Weather", is to use the velocity achieved in the most recently completed iteration. The basis for this is that the most recently completed iteration is likely to have been conducted under conditions similar to those we will experience on the next iteration. So, we should expect to achieve a similar amount. (Of course, the same adjustments for staff absence and other exceptional features should still be made.)

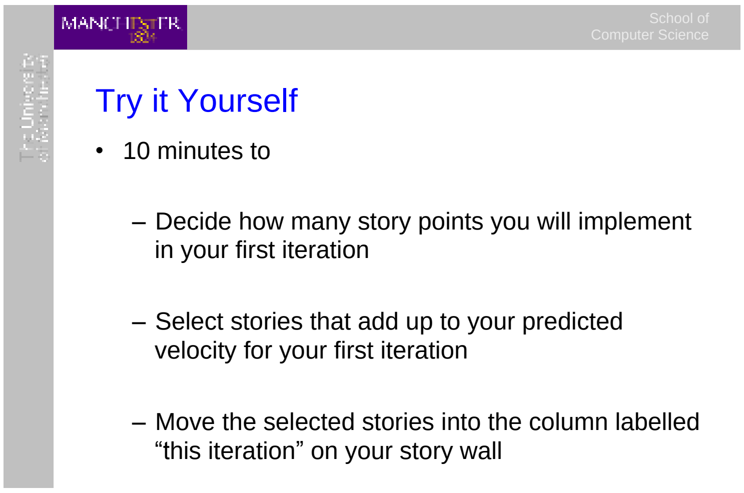
Some teams use a rolling average, based on the velocities achieved in the last 3 or 4 iterations. This avoids the team's planning being too skewed by an unusual iteration (perhaps taking place close to Christmas, or having a larger than normal number of staff absences).

Similarly, various approaches to selecting an initial target number of story points have been suggested. Some people recommend the best approach as being to run without details plans for a couple of iterations, just implementing as many stories as will fit into the working time available. This will allow the team to establish an idea of its natural velocity, without the need to guess at random. Another option is to ask an expert to guess, based on relevant past experience. A third approach is to use historical data.


School of
Computer Science

Which Stories to Choose?

- Choose to focus on early implementation of stories that:
  – Are high value
  – Are high value & high risk
    - Why?
  – Will teach us a lot about the application
    - CRUD



MANCHESTER 1824

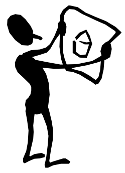
School of Computer Science

The University of Manchester

Try it Yourself

- 10 minutes to
  - Decide how many story points you will implement in your first iteration
  - Select stories that add up to your predicted velocity for your first iteration
  - Move the selected stories into the column labelled "this iteration" on your story wall

Notes



Iteration Planning in the Wild

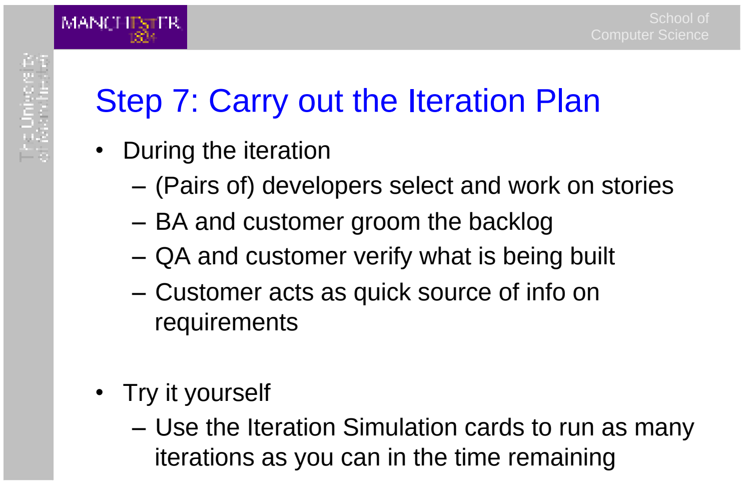
As with release planning, we have tried a somewhat simplified version of iteration planning in the lecture, just to get the basic idea. In practice, we would probably do a rough assignment of stories to iterations for the whole release, or for a sequence of upcoming releases, if this had not already been done during release planning.

It is also common to give estimates to tasks during iteration planning, so that a sanity check can be made of the amount of work the team is being asked to commit to. Tasks are usually estimated in ideal hours, rather than story points, and since they are concrete and fine-grained, we can add all the ideal hours up, and see if it is practical to expect this amount of work to be done in one iteration or not. This is especially useful in the earlier stages of a new project with a new team, when estimation errors are likely to be significant.

At the end of iteration planning, some agile proponents recommend getting each member of the team to commit verbally to implementing the selected stories. It is felt that saying "I can commit to this work" in front of the whole team helps to motivate team members. It can also help to bring any lingering worries with the plan to the surface, where they can be addressed, since any team member with concerns they have not yet voiced may be reluctant to make an open verbal commitment to implement something they don't think is realistic or important.

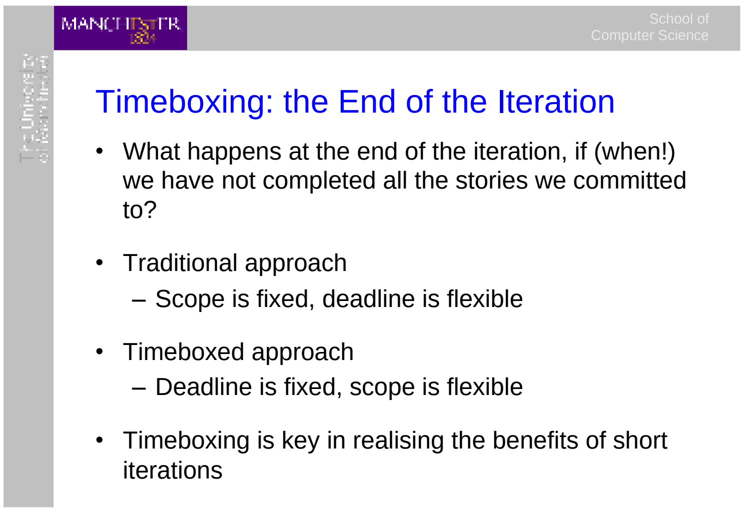
Note also the self-correcting nature of the iteration planning process. We have to make some initial decisions and create some initial estimates, because without these we can't make progress. But we know that these decisions and estimates will be wrong (in some cases, poor) because at the start of a project we know very little about it. On a waterfall style project, where one major plan is made at the start of everything, and then we genuinely try to follow that plan, we are in trouble when the plan contains large errors. On an agile project, we'll be making a new plan in a week or two's time, when we will have more information and can do a better job.

For example, suppose a new agile team has wildly over-estimated the number of story points they can implement in an iteration. After the first few iterations, they'll revise down their predicted velocity, as experience tells them how many story points the team can actually complete each iteration. They might at first go too far, and start to under-estimate their velocity, but again experience in the next few iterations will reveal that, and the team can self-correct. Suppose now a new team member joins, who has a quite different view of what makes a "big" story from the rest of the team. The input of this team member will skew the estimates of larger stories for a while within the team, but gradually, after an iteration or two, the team and the new member can gradually adapt their ideas of story size, until they are more consistent with one another. Again, the team have used the repeated planning sessions to learn from each other, and self-correct, creating a process that is personalised for that particular team on that particular project.


School of Computer Science

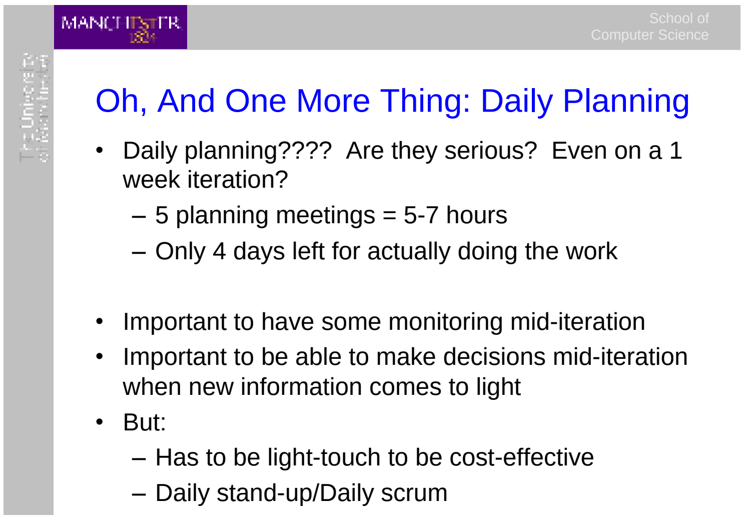
Step 7: Carry out the Iteration Plan

- During the iteration
  - (Pairs of) developers select and work on stories
  - BA and customer groom the backlog
  - QA and customer verify what is being built
  - Customer acts as quick source of info on requirements
- Try it yourself
  - Use the Iteration Simulation cards to run as many iterations as you can in the time remaining


School of Computer Science

Timeboxing: the End of the Iteration

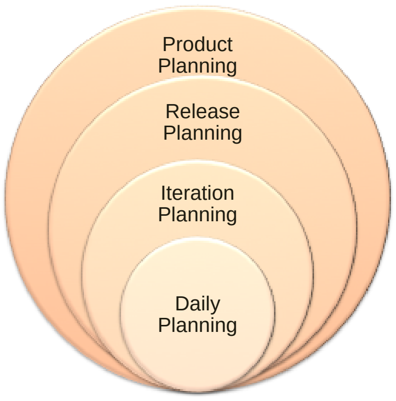
- What happens at the end of the iteration, if (when!) we have not completed all the stories we committed to?
- Traditional approach
  - Scope is fixed, deadline is flexible
- Timeboxed approach
  - Deadline is fixed, scope is flexible
- Timeboxing is key in realising the benefits of short iterations


School of Computer Science

Oh, And One More Thing: Daily Planning

- Daily planning???? Are they serious? Even on a 1 week iteration?
  - 5 planning meetings = 5-7 hours
  - Only 4 days left for actually doing the work
- Important to have some monitoring mid-iteration
- Important to be able to make decisions mid-iteration when new information comes to light
- But:
  - Has to be light-touch to be cost-effective
  - Daily stand-up/Daily scrum

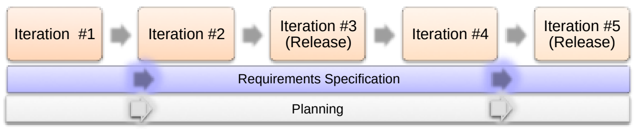
The Agile Planning Onion



Notes

No Planning in Agile: True or False?

- Critics of agile methods claim that they are chaotic and unpredictable because of the lack of BUFP
- What do you think?



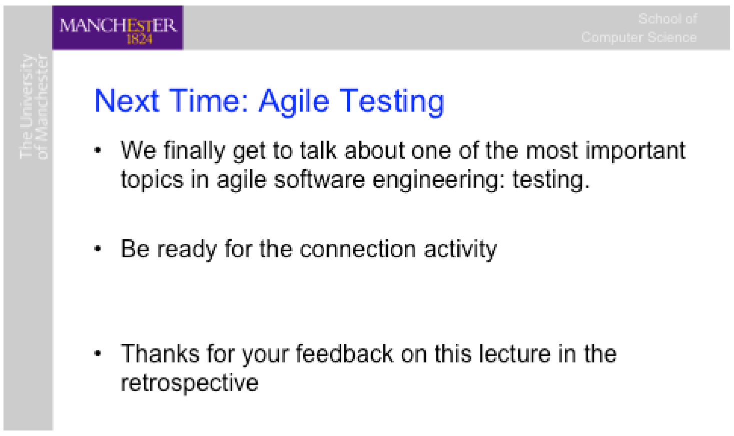
Reflection Exercise

- How does the agile approach to planning help us deal gracefully with:
  - Changes to requirements?
  - Incorrect requirements?
  - Inaccuracy in estimates?

Note that both the customer and the developers must bring their own special expertise to the process of planning for an agile project:

- The customer has knowledge of the domain, the problem to be solved, and the context in which the solution must operate. He has the final say on which stories will deliver value, on the prioritisation of stories, and (most importantly) on the selection of which stories will be implemented. He can say "sorry, that's not what we need."
- The developer has technical knowledge about the kinds of solution that are feasible, and what is involved in bringing them into being. She has the final say on how big stories are, and how many can be fitted into an iteration. She can say "sorry, that can't be achieved" or "that can't be achieved in that timescale".

Each party must respect and trust the other for this process to be successful. In particular, the developers must trust the customer enough to allow them into the team, where all the set backs, errors and delays that are normal on a software project will be visible to them. This allows the customer to be involved in the decisions about how to respond when things haven't gone as expected, and to ensure that the maximum value is delivered, despite the set backs. This contrasts with the approach taken by many non-collaborative teams, where problems are hidden from the customer until it is too late for the customer to have a say in how the team should respond.


MANCHESTER 1824
School of Computer Science
The University of Manchester

Next Time: Agile Testing

- We finally get to talk about one of the most important topics in agile software engineering: testing.
- Be ready for the connection activity
- Thanks for your feedback on this lecture in the retrospective

References

[Boehm 81]	*Software Engineering Economics*, Barry Boehm, Prentice-Hall, 1981, ISBN 0-13-822122-7.
[Cohn 2006]	Agile Estimating & Planning, Mike Cohn, Pearson Education, Inc., 2006. ISBN 0-13-147941-5.
[TCC 2011]	TastyCupCakes.org