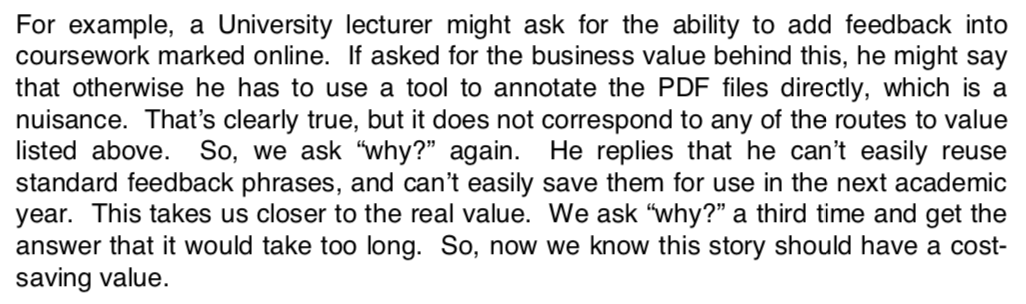
Requirements gathering: **“Just enough, just in time”.** This approach **saves time and effort** because if requirements are gathered for the future, then at the time of putting them into development, these requirements may have no longer be the right ones. As the customers and the developers with **time** may have learnt more about the application and the requirements. And the environment in which the software must be used may also change. The solution is to not stick to the later requirements rigidly. The decision to make regarding which requirements to use should be done at the latest possible time, when there cannot possibly be a later one. The **”Just enough, just in time”** approach is **incremental and iterative.**

Many software development processes are iterative. But agile processes differ in that they rely on **short incremental iterations.** The system to be built is **divided into units**. The units are **in themselves complete**. In each pass **work to create one of these units,** called **increments. End of pass** deliver a system that delivers **part of the full requirements.** System may be able to be **executed and deployed for use. Incremental design** allows to **fail fast** because customer feedback is used along the development process rather than only at the end. To **truly fail fast** increments should be as **small as possible. The iterations** **typically last 1 or 2 weeks stretching to one or two months.**

Principle #1

* Deliver value early and often.
* Satisfy the customer by delivering valuable software continuously.
* **Value oriented** agile practices**: impact mapping, user stories and roadmaps.**
* Need to know what value we are aiming for, for the customer. Usually related to time any money.
* Need to think about the impact we are aiming to achieve which is usually the value.
* Types of value:
  + 1. **Revenue generation**
    2. **Cost saving**
    3. **Competitive differentiation**
    4. **Brand projection**
    5. **Enhanced loyalty**
* Technique for eliciting business value information from customer and stakeholders is the **“5 why’s”** technique.

How do we know what to build?

Need an approach to requirements gathering that is:

* **Need to gather requirements “Just-in-time, just enough”**
* **Iterative and incremental**
* **Value driven.**

Iterative requirements gathering

* Get detailed requirements about features that are about to be developed
* Specify high level picture for features that will not be implemented for a while.
* Can be achieved using **user stories**.
* Useful because **user stories** can be of **different levels of abstraction.**
* **Grooming the backlog** AKA product backlog iceberg, is used to partition the **user stories** into **fine grained stories,** **stories planned for release** and **epics** for future releases**.** Can **adjust the level of abstraction** of the stories.
* Each **user story** should describe a (thin) **end-to-end slice of functionality.**
* **End-to-end slicing is a method of decomposing problems into smaller ones.**
* **Functional decomposition** and **object-oriented decomposition** are the other methods of decomposition.

Functional Decomposition

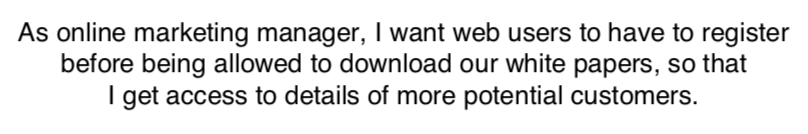
* Used since the 1960s
* Start with a high-level description of the overall functionality e.g bake a cake
* Successively refine it to reveal more detail.
* Look at bake a cake example in notes.

Object-Oriented decomposition

* Identify the actors in the system
* An actor is any component that has state and/or behavior
* An actor has a state if it knows something of importance to the desired functionality
* An actor has a behavior if it can do something that progresses us to the desired goal.
* Can decompose the behavior of each actor using a process of functional decomposition.
* Look at the bake a cake example in the notes.

Slide on pg 15

The Connextra Template

* Template for writing user stories.
* As a <type of person>, I want <some functionality> so that <some value is created>
* Well known
* Prompts us to include the important parts of the story.
* Avoids “blank page syndrome”
* The user is not the user of the system per se:

How to gather user stories

* Story writing workshops
* Impact Mapping
* Story Mapping

Story Mapping

* Tell the story of how the system will be used
* Add details of how the software will support the story.

Properties of user stories: **INVEST**

* **I**ndependent: The stories should be able to be implemented in any order.
* **N**egotiable: No aspect of the story is deemed unchangeable throughout the lifetime of the project.
* **V**aluable: The story should have real value to some customer/stakeholder.
* **E**stimable: It should be possible to estimate the size of the story.
* **S**mall/Appropriately sized: Should be size relative to the iteration slice being used.
* **T**estable: it should be possible to devise a concrete and unambiguous test that can tell us whether the story has been implemented or not.

Need to complete the practice for after lecture.