**Agility**

Changes in the software.  
Changes in the team members.  
Change of technology.

**How do we respond/adapt to change**

Rapid delivery of work.

Make the customer a part of the project: Not us vs them. At the end of the day they will be validating the process. Including them mitigates the risk of them not approving the product.

Planning has its limits. Must be able to adapt to any problems with a plan.

**Cost of Change:** With any change, there is a cost associated with it.

**Difficult Requirements**

It is difficult to predict requirements, how long testing, design, construction ect will take.

What you can do is look at previous examples and how long they took. This is inaccurate though.

You must be able to manage and adapt to these incremental changes.

**Agility Principles (You don’t have to memorise these principles)**

At the end of the day, the customers satisfaction is the most important.

Accept changes.

Work consistently, not the day before.

When developing software, you must think of integration (how you software can integrate with something else eg Google Maps)

Motivation helps build good projects.

Communication in person with clients is key when developing software.

Keep things simple, don’t overcomplicate.

A good foundation/design helps build a good project.

**Agility, what is the best way to achieve it?**

**Scrum**

Product Owner  
Scrum Master  
Deliverable: Product  
Product Backlog  
Sprint Burnout  
Burndown Charts

Sprint: consists of 30 work units (about 30 days)  
Scrum: Short meetings (15ish min)

**Extreme Programming (This seems NB)**

This is a very agile, fast method of software development.

Start at Planning,  
  
**Planning** <-> **Design** <-> **coding** <-> **test** <-> **Planning**  
At **coding**, you can release candidate code instead of testing.  
At **test**, you can release product.  
At **Design** or **Coding**, you can go back to themselves to refactor.

**Planning (Extreme)**

Listen   
-> Create User Stories  
 -> Assign Values: How important achieving that requirement is  
-> Assign Cost : How costly that requirement is. (time, money ect)  
-> Commit

**Design**

**Coding**

Develop Unit Tests: developing each component on their own, the putting them together to make sure it works  
-> Pair Programming  
  
**Test Driven Development**

Testing allows us to see if the customer will be happy with the end product. Whether the product is going to be able to be finished in time.

New Requirement/Story  
-> Write Tests  
-> Tests Fail  
-> Write Code  
-> Tests Pass

**Pear Programming**

Driver: Person Coding  
Navigator: Person googling/researching, getting the things that must be tested.

**Kanban**

To break down the requirements into manageable task.

Backlog  
Selected  
Analysis (doing and done)  
Development (doing and done)  
Testing  
Done

**DevOps**

All about continuation and integration

**Recommended Process Model**

Look at slide.