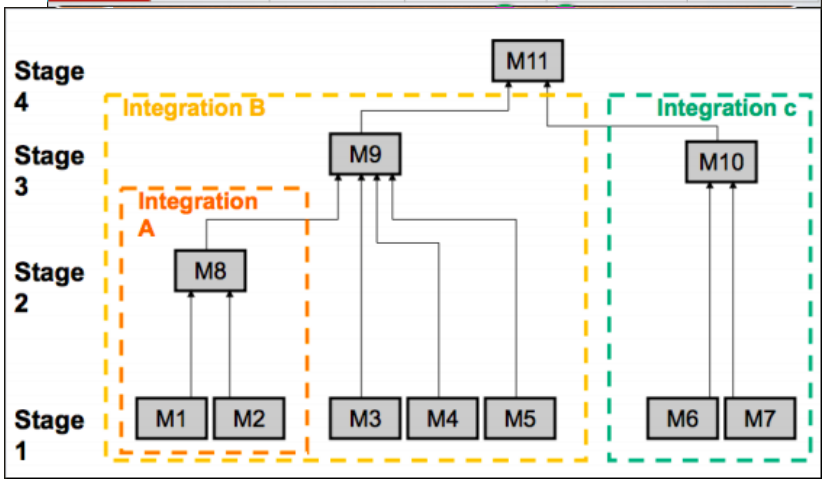
- Unit Testing was to test the individual pieces
- Where integration testing is when you test combinations of pieces
  - It is harder, because you are testing how classes get used
  - So it's hard to get comprehensive testing involved.

	Unit Testing	Integration Testing	Release Testing	Acceptance Testing	User Testing
Input	Functional Spec of Unit	Functional Spec of Product	Requirements of Product	Real Use Cases & Real Data	Actual use
Output	Pass/Fail	Bug Reports	1) Bug Reports 2) Sign off for Acceptance Tests	Bug Reports Sign off for End of Contract	Bug Reports
Performed by	Developer	Development Team	Testing Team	QA Team & Customer	Real Users
Frequency	Many per day	Periodically e.g. End of a sprint	Prior to Acceptance Testing	Prior to Use	After Release

Release Testing

- After Unit testing and system integration testing
  - The system is complete is ready
- Release testing is testing the whole system works together
  - To meet the overall Requirements, not just the Specs.
  - Includes the non-functional requirements
- This process is concerned with finding errors that result from unanticipated interactions between components.



Release vs Integration testing

- A separate team that has not been involved in the system development should be responsible for release testing
- Rather than finding integration bugs, The objective of release testing is to check that the system meets its requirements, and is good enough for external use.

Release testing

- **Primary goal** : convince the company the the software is good enough to give to the customer. The end is a sign-off approval that it is ready.
- **Team role** : not let the software go to acceptance test until it is ready
- **Way to do this** : show that it meets all of the requirements
  - functional and non-functional
  - does not fail during the normal use
  - helps if the **requirements** are properly documented.

When you integrate components to create a system, you get emergent behaviour.

- Some elements of the system functionality only become obvious when you put the components together.
- Sometimes components can only be tested when all of them are combined.
- It is nearly impossible to test all interactions comprehensively, especially because its hard to automate Release Testing

Main strategies

- Requirements driven tests
- Scenario driven tests
- Performance driven tests

If requirements have been documented **properly**, you can design test-cases that show that the requirement has been met.

- This is validation testing, rather than defect testing
  - Black-box rather than white-box
  - I do the normal actions and the right results happen

Scenario Strategy

- Utilizes scenarios.
  - You identified and modelled a standard customer scenario, now need to show that this process can be **satisfied.**
  - A scenario test may test several individual requirements
- As a release tester.
  - You play the role of the scenario character
  - Make both : deliberate mistakes and the right actions.

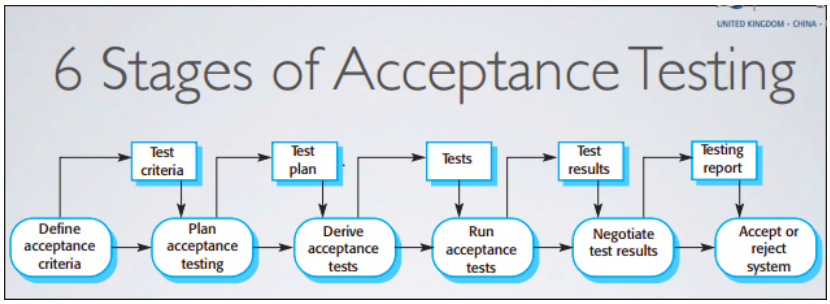
Performance strategy

- Once the system is all complete and integrated, only now you can test some cases
  - Non-functional requirements, like performance.
  - Does it work with 50,000 records?
  - Can it handle 1000 users connected at once?
- Particularly useful for distributed systems

Acceptance testing

- Takes place before the system is accepted for **active** users
- A formal test by the customer to agree that it is ready
  - Tested with real data supplied by the customer
  - Rather than example test case data
- Acceptance implies that payment should be made for the system.
- Some systems do not have a client so they can't do
  - Release testing
  - Acceptance testing
- Acceptance testing may reveal errors and omissions in the requirements definition
- May also reveal requirements problems where the **system does not really meet the user's needs,** or the system **performance is unacceptable.**

Stages of acceptance testing



- **Define acceptance criteria**
  - This stage should, ideally, take place early in the process before the *original* contract for the system is signed.
  - In practice, it's hard to do this at the start of the project, but It has to be agreed for acceptance testing.
- **Plan acceptance testing**
  - Acceptance testing might mean importing real data, setting up test machines, etc.
  - In general you should be planning to show all the ways that is passes the tests
- **Derive acceptance tests**
  - After the criteria and the plan have been designed.
  - Define the actual tests that will be performed
  - Functional and non-functional
  - Cover (at some stage) all aspects of the system
- **Run tests**
  - Agree the day, run all the tests, etc.
  - Ideally in the actual environment of use, if possible
  - May need to do some training / explaining before each test.

- **Negotiate test results**
  - Its unlikely that all test will be passed.
  - Or that the client wont identify some problem / issue
  - For each problem negotiate whether it is good enough for use, or whether more development must be done.
- **Reject / accept system**
  - If the system is good enough for use, then it is accepted, if not, more work needs to be done
  - After the improvements are complete, the acceptance testing phase is repeated.

You have to begin by documenting what you agree to build

  - This is why we build requirements document
  - And have a customer agree to what functionality will be
  - But you also have to be accommodating of **reasonable** things that were not predicted.
    - And can be easily fixed, rather than months of work.
- The project does not end when you finished system testing
  - You might have to do another 10 weeks of coding
  - And then testing and do more acceptance testing
- When is the best time to do acceptance testing
  - **Not at the very end**

In the end acceptance testing goes most successfully if :

  - Customer has seen it before and given you feedback
- There is less room for surprises all round.
- Ideally : they've **already accepted**
  - the requirements
  - the lo-fidelity prototype
  - a high-fidelity design
  - a mid-fidelity functioning prototype
  - and then the **final system**

Although it sounds clear cut it never is that easy

- Customer might need new software ASAP
- They might be willing to accept the software, irrespective of problems, because the costs of not using the software are greater than the costs of working around the problems.
- The agreement might be that we provide an updated version later.

User testing

- Focused on real use with users, rather than specified test with customer data
- Allows for problems to emerge from the working environment, that were unforeseen non-functional requirements.
  - Any simulated version of real-life is inherently incomplete
- Two types : Alpha & Beta testing
- A specified amount of UT may be agreed in the contract

Alpha user testing

- A few specific user trials the software for real work tasks
- A lot like acceptance testing, except
  - Real users doing everyday tasks
  - Rather than perform specified tests.
- Also important before selling a product on wider release.
  - You publicly release a convincingly alpha-tested version.

Beta user testing

- Where the software is released for limited general use
  - A limited group is allowed to use a a release candidate
  - Users feedback problems as bug reports
- Often used as a form of marketing
- Used when the cost of fixing the problems is low
  - Web systems, which the development company hosts
  - Conversely deployed software requires updates to be installed