MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA
UNIVERSITY OF NEW ZEALAND

Software Design and Construction
159.251

Continuous Testing, Integration and Deployment

Amjed Tahir

a.tahir@massey.ac.nz

Overview

- Software systems are evolving fast

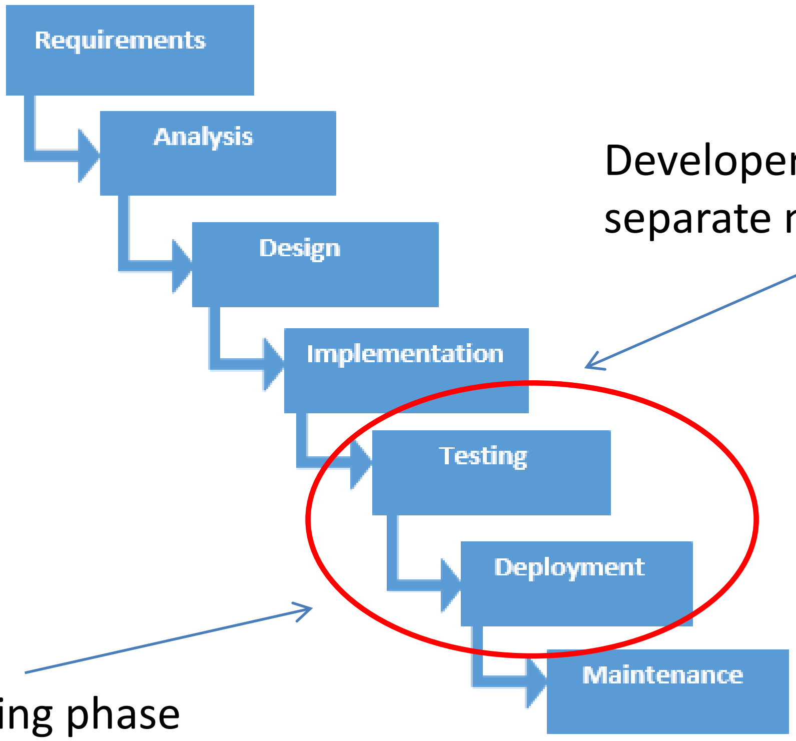
  ➔ require more flexible methods to cope with such evolution.
- Agile method are designed to build software systems *iteratively* and *incrementally*.
- This require dividing large components into smaller, more manageable pieces that integrate together as the software development progress.
- **Challenges**:
  - How to integrate those components?
  - How to manage the integration process?

Overview

- Continuous Integration (CI), Continuous Testing (CT) and Continuous Deployment (CD) are similar in nature, but (slightly) different in practice!
- They all work together …
- CI , CT and CD goals are similar
  - Speed up the development process.
  - Increase visibility which enables greater communication.
  - And most importantly, improve the quality!

Why do we need CI, CT and CD?

- Traditionally, integration, testing and deployment happen at the end of the development.



Developers may write code for separate modules/components

Integration and testing phase

Traditional Integration

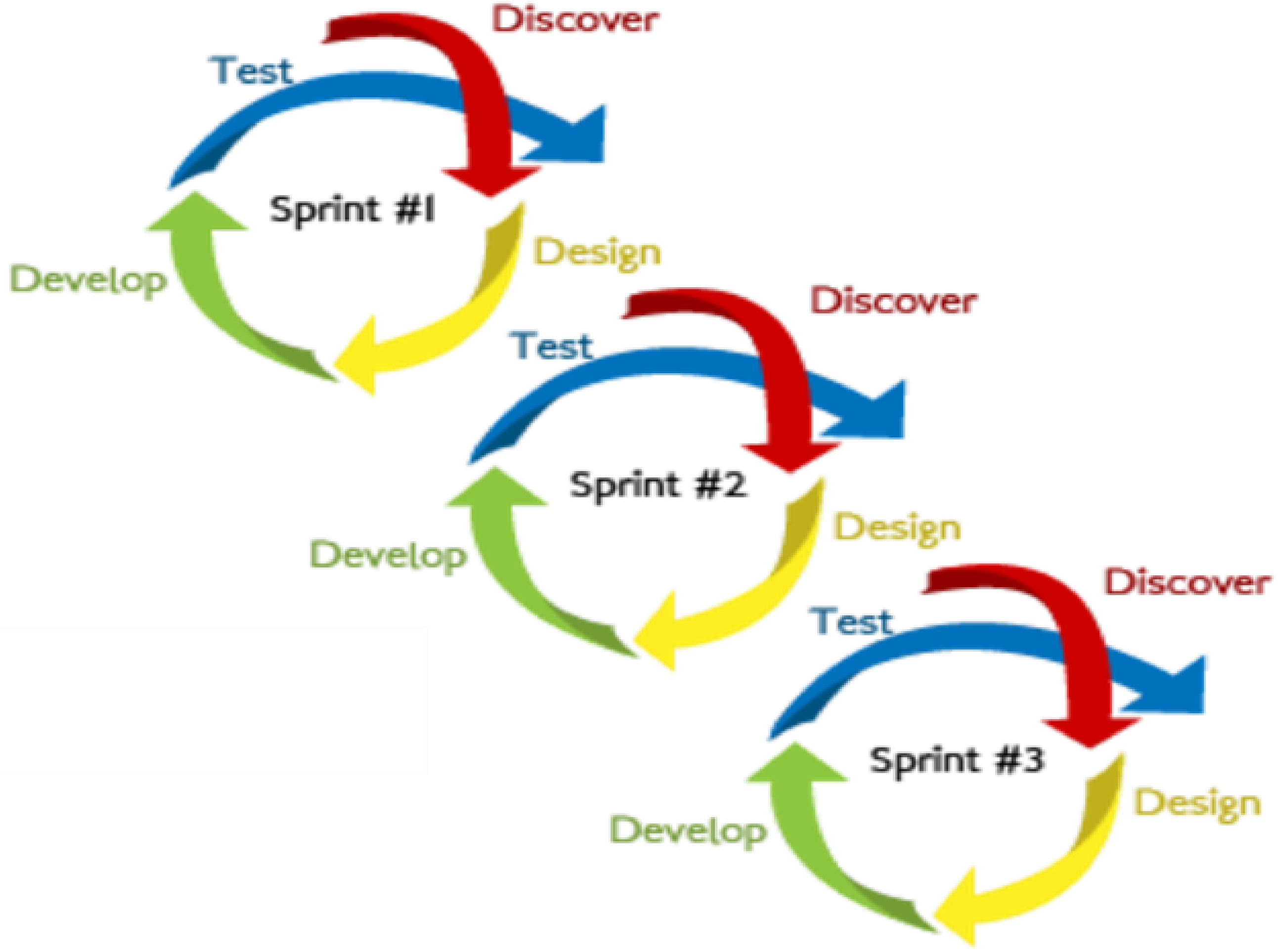
- All the features have to be completed before *Integration*, *Testing* and *Deployment* take place.
  - There are long periods between releases.
  - This can delay the process of finding and locating defects (late detected defects cost more to fix!).
  - Code review overhead – especially with large components.
  - Sometimes, poor control of the code when integrating large segments of the program (weeks/months of work).

Testing and Integration in Agile

- Agile value all form of testing.
- Great focus on unit, integration and user acceptance testing.
- Agile depends highly on *Testing* and *Integration.*
- *Most Agile methods consider Unit Testing and User Acceptance Test as key practices.*

Continuous Development

remember this Scrum example



Continues Integration (CI)

- *Continues Integration* is the practice where members of a team integrate their work frequently leading to multiple integrations per release or per a specific timeframe.

  Integrate -> build -> test

  - Integration can take place daily (or even in a shorter timeframe).
  - Each integrated unit is verified by an automated build (including test) to detect possible integration errors/defects, as quickly as possible.
  - When tests fail, the developer's priority would be to fix the bug first!

Caution!!

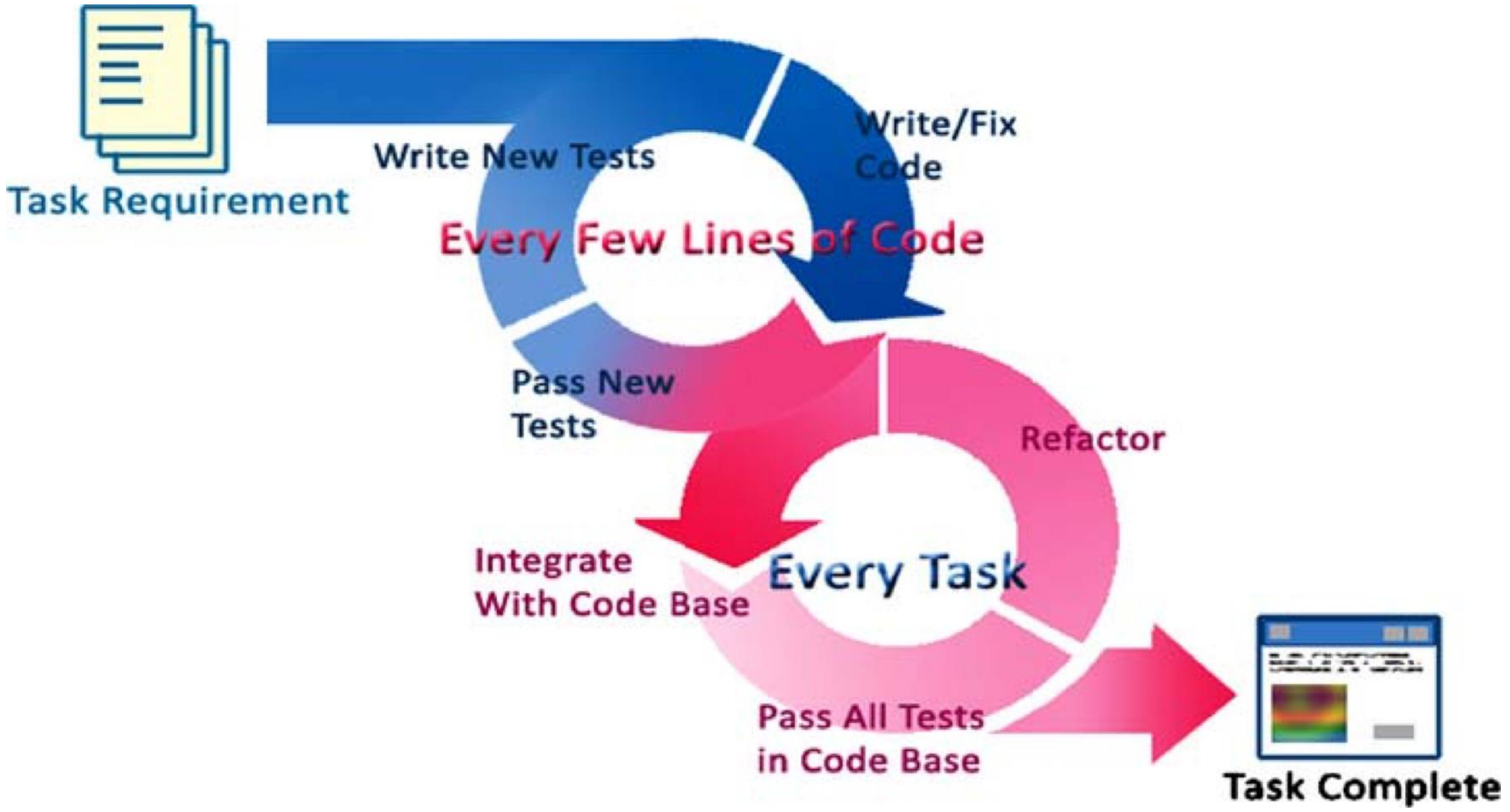
"Continuous Integration doesn't get rid of bugs, but it does make them dramatically easier to find and remove"

Martin Fowler

Integration workflow

- A typical continuous integration workflow will contain:
  - Setting up the automated build and unit testing.
  - Queuing the build and execute tests.
  - Configuring the whole work flow to execute on a continuous basis.

An example from Test Driven Development



Source: Nagappan et al. 2008

Continuous Deployment (CD)

- *CD is the deployment or release of code to Production as soon as it is ready.*

- Follows the testing that happens during CI, and pushes changes to production system.

- It aims at building, testing, and releasing software faster and more frequently which can reduce the time and cost of delivering changes by allowing for more incremental update to application in production.

Continuous Deployment cont’d

- CD helps to make sure that a version of your code is accessible at all times.
- Any testing is done prior to merging to the base code and is performed on Production-like environments.
- CD requires CI - otherwise, you will get errors in the release.

Continuous Deployment Process

- CD Process = CI + deploying the code to production.
- Every change made to the code (by ALL developers) has to be deployed to production.
- This can result in several deployments per day.
- The problem is with business decision – not allowing the code to be deployed (here comes CV!).

Benefits of CI and CD

- Frequent reviews and short feedback cycles.
- Frequent refactoring.
- Frequent testing in small chunks.
- Smooth integration process.
- Increase visibility which enables greater communication
- Catch bugs/defects faster.
- Spend less time debugging and more time adding features.
- Stop waiting to find out if your code's going to work.
- Reduce integration problems allowing you to deliver software more rapidly.

All related

- You usually starts with single *test* – then you *intergrade* separate units that have been individually tested and conduct an integration tests .. Then *deliver* and/or *deploy*.
- Checkout the source code -> write your code -> test -> build ➔ commit -> test -> build again! ➔ Test again! .

CI, CD and CT best practice

- ‘*Automation*’ is the key!

- Configuration Management in place!
  - Maintain a code repository (all the things you have learned about version control)
- CI doe not make sense without CT.
- Again, automate the testing (as much as possible).
- Everyone should commit to the base code, everyday
  - or preferably, multiple times a day!
- Make it accessible - everyone can see the results of the latest build - use a tracking system!

Automation

- Ideally, the entire process should be automated, as much as possible.

- Manual adjustment can also work, but it make the process harder.

As a developer……

- Developers play a central role in CI, CT and CD.
- Follow *best practice* to avoid problems.
- You need to make sure that
  - Check in the code as frequently as possible (long delays can be harmful!).
  - Do not check in broken or untested code.
  - Do not check in when the build is broken.

Tools that help the process

- Specific CI tools e.g., Hudson or Jenkins.
- Version control tools e.g. Git and SVN.
- Issue (change) tracking systems e.g. JIRA or Bugzilla.
- Build tools e.g. Maven or Ant.
- Continuous building e.g. CruiseControl.
- Unit testing e.g. XUnit framework and/or Selenium.

Moving towards Continuous Delivery (DV)

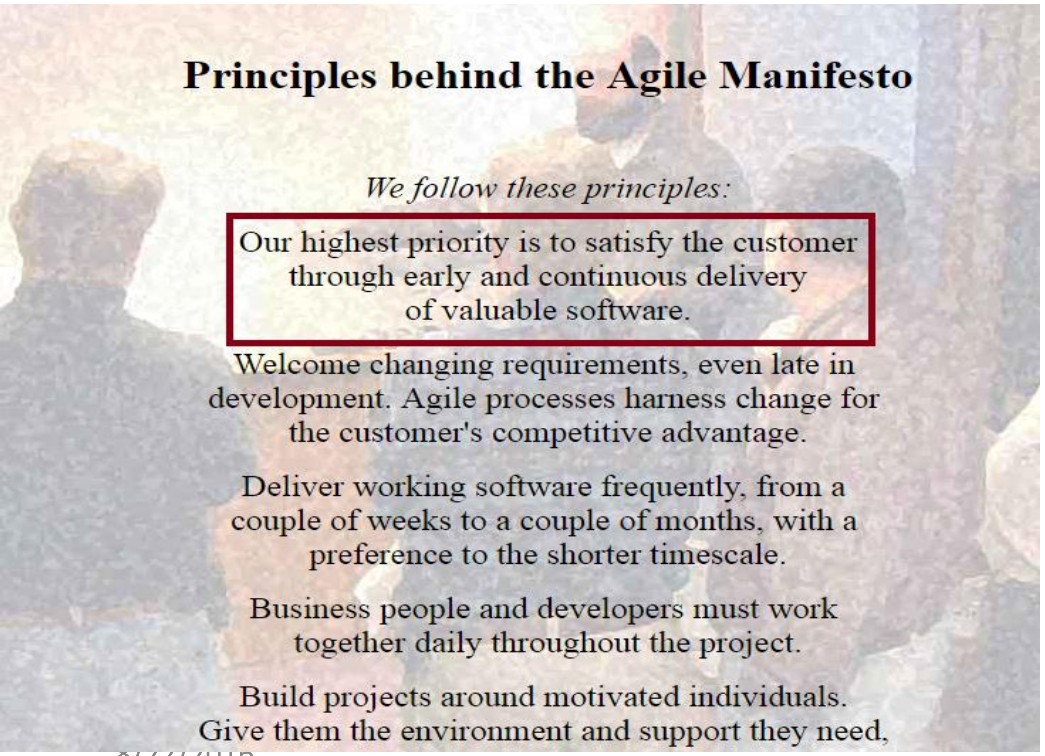
- Continuous Delivery is the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, *safely* and *quickly* in a *sustainable* way.

- It is not magic! But it works…

- DV means that you are able to do frequent deployments but may choose not to do it, usually due to businesses preferring a slower rate of deployment…

Continuous Delivery cont'd

- DV has been mentioned as the first principle in the Agile key principles!



Principles behind the Agile Manifesto

We follow these principles:

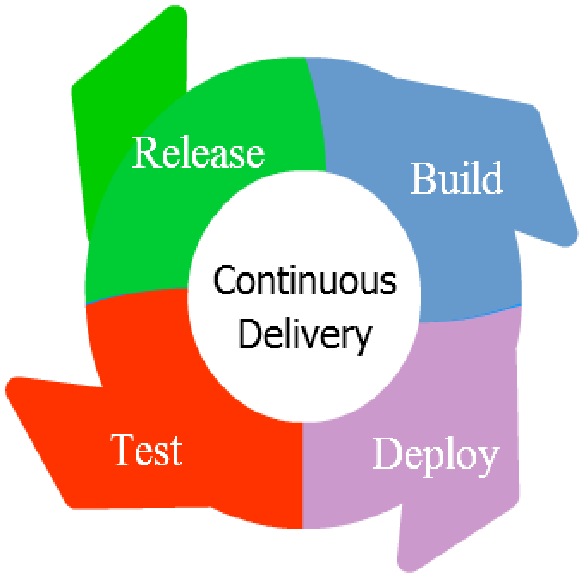
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need,



References and interesting reads

- Fowler, M . “Continuous Integration”. http://www.martinfowler.com/articles/continuousIntegration.html, Retrieved on 20/4/2016.
- Stolberg, S. "Enabling agile testing through continuous integration." *Agile Conference, 2009. AGILE'09.*. IEEE, 2009
- Richardson, J. “Continuous Integration: The Cornerstone of a Great Shop”. http://www.methodsandtools.com/archive/archive.php?id=42 Retrieved on 21/4/2016.
- Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education.