**Background**

The growth of using software and the competition between IT companies in order to fast deployment of software to the market has make companies pay great attention to relocates their old unsuitable approach to new and more sophisticated approach that provide much fast release of software with respect to its quality .Fast release of software can get high profit and also provide early feedback from customer which guided the company to spend it’s time in building valuable software(Lianping Chen , 2016) .

**Introduction**

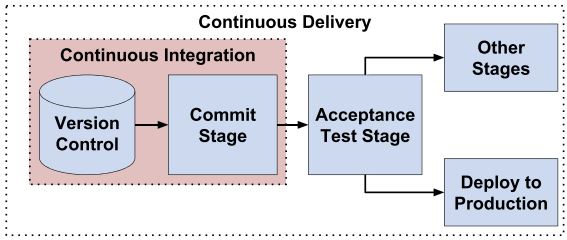
1. Continuous integration :

“is a software development practice where software is integrated continuously during development”( M.Fowler, 2006) .In constant , some project have integrate the work of developers after amount of time in fact when the integration is late it possible to have a lot of line conflict in the code which increase time by fixing this conflicts (Eero Laukkanen et al.,2015) .It’s better to integrate the code several time in a day by using share repository .In additional each time a code is integrated to the repository it auto built and tested to ensure that the code is still do what is intended from it while it’s changed and as soon as test fail it reported and fixed.

1. Continuous Delivery(CD) :

“is a software engineering approach in which teams keep producing valuable software in short cycles and ensure that the software can be reliably released at anytime”( Chen , 2015a). Companies that have adopted CD have reported huge benefits , such as significant improvements in time to deployment , customer satisfaction , product quality , release reliability , productivity , efficiency and the ability to build the right product through rapid experiments (Chen , 2015a ; Leppanen et al , 2015). These benefits have motivated many companies to adopt CD (Lianping Chen, 2016). However, implementing CD can be quite challenging (Chen, 2015a; Leppanen et al., 2015; Claps et al., 2015).

The following figure show the different between the CI and CD , which indicate that CI is a part of CD , when a commit into CI happen it trigger an auto build and test to code in the repository if the test is pass deploy the release into production otherwise the developer recommended to fix the code .

 Fig 1: different between CI and CD

**Challenges and Strategies to overcome CD adoption:**

1. **challenges**

* Acquiring buy-in from a wide range of stakeholders whose goals May seemingly be different from those of the team driving the CD implementation (Lianping Chen, 2016).
* Gaining sustained support in a dynamic complex enterprise environment (Lianping Chen, 2016).
* Maintaining an application development team’s momentum when their application’s migration to CD requires an additional strenuous effort over a long period of time (Lianping Chen, 2016).

1. **Strategies**
   1. Selling CD as a painkiller: adopting CD requires making changes in many areas. Apart from automating build, test, and deployment activities, introducing CD also requires changes to architecture activities (Chen, 2015b), software development practices, organizational structure and so forth (Lianping Chen, 2016). To make such substantial changes, we need support from a wide range of stakeholders. However, gaining buy-in is a enormous challenge because all of them have their own goals an can seemingly be different from or even conflict with the CD’s team(Lianping Chen, 2016).this strategies help in solving this problem by identify each stakeholder’s pain points and identify the ones that CD can help to solve (Lianping Chen, 2016). When we express the CD we focus on how CD helps to solve his pain.

**Problems and solutions when adopting CD**

1. **problem**
   1. Related to integration :-
      1. Large commits: It may contain large number of change that is conflict with other (Eero Laukkanen et al., 2015), thus solving this conflict may take a lot of time to solve which increases the time for development.
      2. Long running branches: easily lead to merge conflicts, and developing code in branches slows the frequency of integration (Eero Laukkanen et al., 2015).
      3. Slow integration: because it leads to larger commits and delays testing feedback which may cause wasting time in building unreliable code. Code review processes should be designed so that they do not cause extensive delays during integration (Eero Laukkanen et al., 2015).
   2. Related to testing :-
      1. Ambiguous test result: an ambiguous test result means that the test result does not guide the developer to action.
      2. Time-consuming testing: getting feedback from the tests can take too long (Eero Laukkanen et al., 2015). This make developers using large commit and motivates slow integration.
2. **Solution** 
   1. Reject bad commits**:** Reject bad commits is a practice where a bad commit is detected automatically (fails some tests) form the mainline. Thus, the mainline is always functional, builds are not broken and discipline is enforced (Eero Laukkanen et al., 2015).
   2. No branch: no merge conflict (Eero Laukkanen et al., 2015).
   3. Monitor build length: monitor the length and take action when the length is much large which guide to solve time consuming testing (Eero Laukkanen et al., 2015).
   4. Test segmentation: category tests in test suit based on functionality and speed. This way, the most critical tests can be run first and other and slower tests later. Developers get fast feedback from the critical and fast tests (Eero Laukkanen et al., 2015).
   5. Comprehensive testing and commit-by-commit tests: test each small pace of code which minimize the scope of debugging and ensure testing completeness and granularity (Eero Laukkanen et al., 2015).