**1.1 Introduction:**

Software engineers have introduce a lot of developing model with different process, workflow and roles in order to produce software with high quality. Generally quality here compose to two type: project’s quality which refer to issues that affect the project as whole such as (time management and cost management); product quality which refer to issues that is related with software product itself such as (high performance, reliable software, correctness and so). In balancing between project’s quality and product’s quality companies agree that improving the software processes is the solution for software against delivery delays, cost overriding, and quality flaws in the end products (R. A. Khan et al, 2000). These problems make software engineer starting to develop more flexible model with more flexible process which called Agile method, after the spread of software companies and the competition between its, each company wish to reach the market first so the demand on Agile methodologies is grown .The growth of using Agile placing on QA teams for faster delivery cycles is forcing many to replace the older monolithic development models with a more streamlined process (kristen Aebersold). However, pervasive method such as manual testing are keeping the teams for adopting flexible practices like continuous integration and continuous delivery and reaching true agility(kristen Aebersold) .Continuous integration is software developing practice in which small piece of code are verified using automated building and automated testing every time a team member make change in code repository .Continuous delivery is software approach in which software product’s is ready for deploying to the market at any times. In order to reach true agility we need to develop continuous integration continuous delivery pipeline to enable developing team to release constant flow of software updates into production to quicken release cycles, lower cost and reduce the risk associated with development (kristen Aebersold).

1.1.1 What is 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 development time in 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 trigger an auto building and testing 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.

1.1.2 Continuous Delivery (CD)?

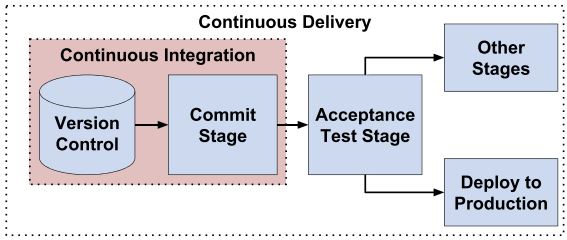
“is a software engineering approach in which teams keep producing valuable software in short cycles and ensure that the software can be reliably released at any time”( Chen , 2015a). Companies that have adopted CD have reported huge benefits , such as significant improvements in time to deployment, customer 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, but CD has much process. When a commit into CI happen it triggers an auto build and test to the code in the repository if the test is pass deploy release into production otherwise the developer recommended fixing it.

Figure 1: different between CI and CD

* 1. **Search problem:**

1.2.1 Current system example scenario:

In development phase a group of developer configure their development environment and setup repository for sharing their codes, after that they starting develop, each developer has its own task which may has some dependency with other developer task. Each developer start developing and committing code into the repository so that other developer can see and use it, after the development group finish their module they send it for testing and waiting for the result of testing. A tester run all the test and report bugs back to the developers, developers start debagging and fixing bugs and send module back to testing, after the tester report no more bugs developer start developing anther module.

1.2.1.1 Scenario problem:

1. Time to find and fix bugs are maximized: testing is performing per module so the scope of searching for each bugs is all the module. Early generate bugs stay without fixing until the module is finished and tested so that the cost and the effect of fixing it is increase. Also other developer may dependent on this ill piece of code.

2. Deploy software has not full tested: in some cases adding small change after testing process make the cost-benefits of run all tests again decreased resulting in deploy some piece of code has not been tested as stated in Continuous Delivery: Overcoming adoption challenges paper(Lianping Chen ,2016).

3. Testing environment: developers spend 20% of their time setting up or fixing their testing environment. In addition because the testing environment is shared among teams, a team may wait for one or two weeks for a testing environment to be released by other teams as stated in Continuous Delivery: Overcoming adoption challenges paper (Lianping Chen, 2016).

**1.3 Objective**

* Automate the process from development to delivery by Build an automated pipeline starting development small piece of code and commit to share repository which pipe auto building and auto testing processes.
* Early bug detecting by running the test per small piece of code’s change and send feedback to developer.
* Grantee that each piece of code has been verified by its targeted test.
* Performing the testing process in virtual environment that is setup to act as same as production environment.
* Increase productivity.
* Building valuable software.

**1.4 Expected outcome**

* Reliable software release of software products: by verifying each change has been tested.
* Increase number of release.
* There are no problem with software in production environment.

**1.5 Scope**

* Building automation.
* Testing automation.
* Integrated with GitHub as version control.
* Supports JavaSE as programming language and desktop applications only.
* Supports JUnit as java testing library.
* Generate JUnit core report.
* Integrated with Docker as container.

**Chapter 2**

**2.1 Background**

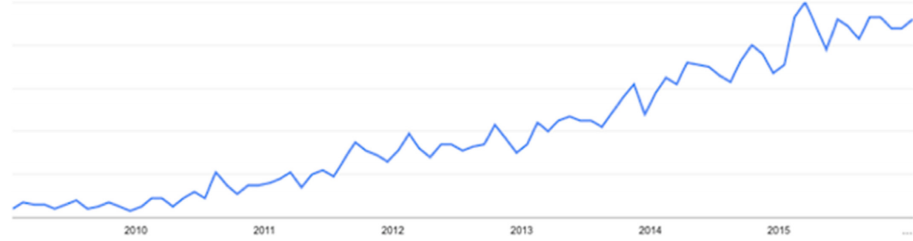
Continuous delivery (CD) approach is relatively new. Its start gaining attention in 2010. When Humble and Farley publish the book title “Continuous delivery: Reliable Software Release throw Build, Test and Deployment Automation”. However, the CD approach has become increasingly popular, as shown by Google search trends (Figure 2) (Lianping Chen, 2016).

Figure 2 : number of search for CD concept from 2010 to 2015

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

Although companies that adopt CD reports huge benefits which motivate other companies to adopt CD. However, implementing CD is quite challenge.

2.2.1 Challenges

* Implementing CD require buy-in from a wide range of stakeholders (tester, developer, manager and so) whose goals May seemingly be different from those teams 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).

2.2.2 Strategies

* + 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 an enormous challenge because all of them have their own goals and 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 introducing CD to any stockholder you should focus on how CD helps to solve his pain.
  + Continuous delivery of continuous delivery: study shown that it’s possible to decompose CD platform in small pieces and when you finish one pieces you can immediate achieving benefits from it (Lianping Chen, 2016). For example, you can apply CD on one module of your project on specific level of test and still able to delivering value to the company. Thus you can achieve early benefits by applying small feature of CD. These benefits can help in maintain sustained support, reduce the risk and difficulty of CD adoption by development teams and gradually align development teams to the new process and new tools in small steps (Lianping Chen, 2016). This gradual Shift makes the alignment easier (Lianping Chen, 2016). Behind that this strategy also provides with the opportunity to get feedback from the application development teams, which can be used to adjustments CD platform (Lianping Chen, 2016).

**2.3 Problems and solutions when adopting CD**

There are a lot of problem can happen when adopting CD. These problem may reduce CD’s benefit or even make applying CD hostile to the company. Problems is grouped depends on effecting software activates.

2.3.1 Problem

2.3.1.1 Related to integration:

* + - Large commits: it’s a commit that save large number of change to version control. This may contain large number of conflict line with other developers code (Eero Laukkanen et al., 2015), now instead of do productive work developer wasting time to this conflict. The reason of making large commit is after developer make small change he wait for testing process. If the test take much time developer feel that this waiting is wasting his time so he think that he shouldn’t wait for small change and start commit large numbers of change.
    - Long running branches: is using multiple way to build specific piece of code. Easily lead to merge conflicts, and developing code in branches slows the frequency of integration (Eero Laukkanen et al., 2015).
    - Slow integration: because of large commits which delays testing feedback. Its may waste developer 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).

2.3.1.2 Related to testing:

* + - Ambiguous test result: an ambiguous test result means that the test result does not guide the developer to action.
    - 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.

2.3.2 Solution

* + 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).
  + No branch: no merge conflict (Eero Laukkanen et al., 2015).
  + Monitor build length: monitor the length of building and take action when the length is much large which guide to solve time consuming testing (Eero Laukkanen et al., 2015).
  + Test segmentation: category tests in test suit based on functionality and speed. This way, the most critical tests can be run first and other slower tests later. Developers can early get feedback from the critical and fast tests (Eero Laukkanen et al., 2015).
  + 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).

**2.4 Continuous integration tools**

There are a lot of continuous integration continuous delivery tools. Here is two different type of tools and much popular.

2.4.1 Jenkins:

It is a server develop to automate build, run processes. With much advance feature which helping in get statistics about your project.” In a nutshell, Jenkins is the leading open-source automation server. Built with Java, it provides over 1000 plugins to support automating virtually anything, so that humans can actually spend their time doing things machines cannot” (kohsuke). Use Jenkins to automate your development workflow so you can focus on work that matters most. Jenkins is commonly used for:

* Building projects (EX: platform category).
* Running tests to detect bugs and other issues as soon as they are introduced.
* Static code analysis.
* Deployment (EX: deploy to container plugin) (Jenkins).

2.4.2 Travis:

 It is a hosted continuous integration and deployment system. There are two versions of it, [travis-ci.com](https://travis-ci.com/) for private repositories, and [travis-ci.org](https://travis-ci.org/) for public repositories. Its work with github only so you connect it with your github account and its detect your Git’s repository. Its use special (.travis.yml) file for configuration build and run processes.

2.4.3 Jenkins vs Travis:

This table show some different between Jenkins and Travis CI\Cd tools.

|  |  |
| --- | --- |
| Jenkins | Travis |
| Jenkins is a server setup on company local server. | Travis is a web site application |
| Plug-in architecture. | Doesn’t use plug-in. |
| Much complex in using. | More easy to use. |

Figure 3 different between Travis and Jenkins

**2.5 The context:**

As we listed above there are many problem arise when adopting CD. This research is focus in building CI/CD tools able to participate in adoption journey. By developing way to solving some of testing and integrations problem by the concept of micro-pipeline. A micro-pipeline is group of testing blocks each block is responsible for specific type of test for specific unit of code connecting together in such way that passing the block tests trigger the next test block in series. This micro-pipeline help in building more appointed tests which solve some of testing problem. As we see above testing problem is root for some integration problem so fixing the root will fix its children. A clear define micro-pipeline reducing testing time as you test affected part of change only which is recommended by Lianping Chen in “Continuous Delivery: Overcoming adoption challenges” paper as new area of search “Test execution optimization”. Also developing simple CD environment with much clear feature, easy to configure and easy to use helping startup companies adopting CD.