# BAB I PENDAHULUAN

1. Latar Belakang

Siklus hidup pengembangan perangkat lunak adalah suatu strategi pengembangan perangkat lunak yang melingkupi proses, metode, dan *tools[ref?]*. Siklus hidup pengembangan perangkat lunak terdiri tahapan *requirement*, *design*, *code*, *testing*, *deployment* dan *maintenance*[2]. Pada pembangunan perangkat lunak skala besar, setiap penambahan fitur akan dilakukan secara *increment*. Penambahan fitur tersebut akan melewati siklus hidup pengembangan perangkat lunak beberapa kali hingga produk versi tertentu dirilis. Berdasarkan survei pengembangan perangkat lunak perusahaan mengeluarkan biaya paling besar untuk *maintenance* [2].

Metodologi pengembangan perangkat lunak yang sering digunakan adalah metodologi pengembangan perangkat lunak tradisional. Metodologi ini lebih menekankan urutan tahapan pada siklus hidup pengembangan perangkat lunak. Jenis metodologi ini memiliki sifat kaku karena tidak dapat menerima perubahan kebutuhan *customer* di tengah pengembangan perangkat lunak.

Fakta yang terjadi pada saat membangun perangkat lunak adalah kebutuhan *customer* terhadap perangkat lunak yang sering berubah dan *customer* tidak ingin menunggu lebih lama lagi untuk memperoleh perangkat lunak. Sifat kaku yang dimiliki metodologi pengembangan perangkat lunak tradisional menyebabkan *customer* tidak merasa puas dengan hasil perangkat lunak.

Metodologi pengembangan perangkat lunak yang mampu menerima perubahan dari kebutuhan *customer* dan dapat memenuhi kebutuhan *customer* dengan cepat adalah metodologi pengembangan perangkat lunak *agile*. Prinsip Agile Manifesto adalah sebagai berikut[4]: (1)Interaksi antar individu selama proses dan pemakaian alat . (2)*Software* bekerja dengan dokumentasi yang komprehensif. (3)Kolaborasi Pelanggan atas negosiasi kontrak. (4)Selalu siap menerima perubahan terhadap perencanaan.

Pada dasarnya sebuah perangkat lunak dibangun dengan memperhatikan dua fokus utama [3]. Di antaranya: (1)Seberapa besar biaya yang dibutuhkan untuk membangun perangkat lunak? (2)Berapa lama waktu yang dibutuhkan untuk membangun perangkat lunak? Kedua hal tersebut memiliki keterikatan satu sama lain. Permasalahan yang sering terjadi pada pembangunan perangkat lunak adalah produk *error* pada tahap integrasi. Jika proses penanganan *error* tersebut adalah manual maka dibutuhkan *effort* maksimal dan waktu jangka panjang untuk memperbaiki *product* tersebut.

Otomatisasi adalah kunci untuk melakukan proses yang sama berulang kali. Dengan penerapan otomatisasi maka proses *build*, *deploy*, *testing* dan rilis dapat dilakukan dengan cepat. Pengintegrasian perangkat lunak juga harus dilakukan berulang kali agar dapat mengurangi resiko dan memperbaiki kualitas perangkat lunak[6].

CI adalah praktik pengembangan perangkat lunak yang mengintegrasikan hasil pekerjaan anggota tim secara rutin untuk menemukan kesalahan pada proses integrasi secepat mungkin[5]. Dengan penerapan *continuous integration* maka perangkat lunak *customer* dipastikan dapat bekerja (dengan asumsi satu set pengujian otomatis yang lengkap) terhadap setiap perubahan baru. Jika ada kesalahan pada proses integrasi, maka tim dapat memperbaikinya dengan cepat. Tim yang menggunakan *continuous integration* secara efektif mampu mendeteksi *bug* lebih awal, menghasilkan perangkat lunak lebih cepat dengan sedikit *bug*, mengurangi biaya dan jangka waktu perbaikan perangkat lunak pada proses *delivery* dibandingkan tim yang tidak menggunakan *continuous integration*[1].

Perangkat lunak membutuhkan jangka waktu tertentu untuk melakukan perubahan pada tahap integrasi. Waktu yang dibutuhkan untuk melakukan perubahan pada tahap rilis dinamakan *cycle time*. Rilis perangkat lunak tidak dapat diulang pada satu *cycle time*. Pengamatan dilakukan pada aplikasi MedRecApp yang dibangun oleh sebuah tim. Tidak adanya *automated* *testing* pada tahap integrasi mengakibatkan sulitnya memperoleh produk versi tertentu tanpa cacat. Meskipun tim telah menggunakan *version control* namun pengintegrasian aplikasimasih dilakukan manual. Dari segi kegunaan, aplikasi tersebut memiliki potensi besar untuk dapat dikembangkan dengan fitur-fitur baru yang sesuai dengan kebutuhan rumah sakit X.

Berdasarkan uraian tersebut, maka judul tugas akhir yang diangkat adalah ”Penerapan *Continuous* *Integration* pada aplikasi MedrecApp”.

1. Tujuan

Berdasarkan latar belakang yang telah diuraikan sebelumnya, Tugas Akhir ini bertujuan untuk membentuk kerangka kerja yang meliputi prosedur, teknik, *toolset* yang mendukung *continuous integration* aplikasi MedrecApp berbasis Java *desktop*.

1. Rumusan Masalah

Pada tugas akhir ini, rumusan masalah yang diangkat adalah bagaimana membentuk kerangka kerja yang meliputi prosedur, teknik, dan *toolset* yang mendukung *continuous integration* pada aplikasi MedrecApp berbasis Java *desktop*?

1. Ruang Lingkup Masalah

Ada beberapa ruang lingkup pada *Continuous integration* yang dilakukan, diantaranya:

1. Mencakup *Java desktop application* yang sudah ada dengan seolah-olah membangun aplikasi tersebut dari awal yang mencakup modul master, modul *front office*, modul poliklinik dan modul rekam medis. Proses *development* mencakup *compile source program*, membuat dan menjalankan *unit test* dan *integration* *test*
2. Prosedur yang digunakan pada CI mencakup:
   1. Penentuan komitmen kesepatan awal
   2. Penggunaan VCS
   3. Penggunaan *automated build*
3. Teknik yang digunakan untuk *version control system*, *automated testing*, *continuous* *integration* adalah:
   1. VCS : *distributed* *version control system* dengan alur kerja *Gitflow*
   2. *Unit test* dan *Integration testing*: *Automated testing* dan *Buttom-up integration*
   3. *Build tool* : *scripting* yang dihasilkan dari Netbeans dan dimodifikasi
   4. *Continuous Integration* : menggabungkan teknik dari VCS, *testing* dan *build*
4. *Tools* yang digunakan untuk *version control system*, *automated testing*, *continuous* *integration* adalah *open source* dan *free software* dengan rincian sebagai berikut:
   1. *Version Control* dan *Software hosting* : Git dan Github
   2. *Unit Testing* dan *Integration* *testing*: Junit dan Fest
   3. *Automated build* dan *CI :* Ant dan Jenkins
5. Metodologi

Pada bagian ini diuraikan mengenai metodologi penelitian sebagai berikut:

1. Studi Literatur

Hal yang perlu dilakukan untuk melakukan studi literatur sebagai berikut:

1. Mengumpulkan, mempelajari dan memahami bahan dan konsep *continuous* *integration*
2. Memilih *sample* aplikasi MedRecApp berbasis Java Desktop yang telah dikembangkan oleh sebuah tim sebagai studi kasus penerapan *continuous integration*
3. Mempelajari, memahami dan menentukan prosedur, teknik dan *toolset* yang mendukung *continuous integration* pada aplikasi MedRecApp
4. Analisis
5. Menentukan ide pemikiran kerangka kerja yang mendukung *continuous integration* pada aplikasi MedRecApp
6. Implementasi dan Pengujian
7. Menerapkan prosedur dan teknik dengan menggunakan *toolset* yang ada pada VCS, *testing*, *build* yang mendukung *continuos integration*
8. Menguji alur *continuous integration* pada aplikasi MedRecApp
9. Kendala yang dihadapi pada saat melakukan *continuous integration*
10. Penutup

Berisi kesimpulan yang dapat diperoleh dari hasil implementasi dan pengujian CI sehingga membentuk kerangka kerja yang meliputi prosedur, teknik dan *toolset* yang mendukung *continuous integration* aplikasi MedrecApp berbasis Java *desktop* dan saran untuk perbaikan penelitian selanjutnya.

Daftar Pustaka

1. Jez Humble, David Farley, 2010, Continuous Delivery: Reliable Software Releases through Build, Test and Deployment Automation (Addison-Wesley Signature Series), USA
2. Kaner, Falk, Nguyen, 1999, Testing Computer Software, Second Edition (Wiley Series), USA
3. Pilone, Dan., Russ Miles., 2007, Head First Software Development, USA
4. Stober, Thomas., Uwe Hansman, 2010, Agile Software Development: Best Practices for Large Software Development Projects, German, Springer
5. Fowler Martin, 2006, *Continuous Integration*, [http://martinfowler.com/articles/continuousIntegration.html diakses 10 Oktober 2013](http://martinfowler.com/articles/continuousIntegration.html%20diakses%2010%20Oktober%202013)
6. Duvall, Paul M., Steve Matyas dan Andrew Glover, 2007, *Continuous Integration*: *Improving software quality and reducing risk* (Addison-Wesley Signature Series), USA