4.3. Otomasi Build dan strategi build

Tujuan Build

Proses *build* pada perangkat lunak adalah proses penting. Hasil produk yang dihasilkan dari proses *build* adalah *package* aplikasi (*file* *executable*). File *executable* ini yang akan digunakan oleh anggota tim untuk sebagai aplikasi yang mengotomasi pekerjaan *user*. *User* dapat menjalankan paket aplikasi tanpa harus menjalankan *source* *code* perangkat lunak.

Manfaat otomasi build

Proses *build* terdiri dari serangkaian proses, antara lain: *compile*, *testing*, dan *packaging*. Apabila proses *build* tersebut dijalankan secara manual (seperti yang dijelaskan pada sub bab sebelumnya) maka pekerjaan tersebut tidak efisien dan rentan terhadap masalah. Masalah tersebut dapat diotomasi dengan *build* *tool* yang dapat mengotomasi *proses* *build*. Salah satu *build* *tool* adalah Ant. Dengan Ant, semua proses *build* diotomasi dengan menulis *build* *script*.

Strategi yang dilakukan tim untuk melakukan build

Tool build menggunakan Ant

Pada pengotomasian *build* diperlukan strategi *build*. Strategi build yang dilakukan pada saat pembangunan perangkat lunak adalah setiap kali integrasi terjadi. Ant *script* akan dijalankan setiap kali integrasi dilakukan oleh anggota tim. Sebelum anggota tim melakukan pengintegrasian ke *version* *control* *repository* , maka anggota tim harus melakukan menjalankan *build* di lokal direktori. Pekerjaan tersebut bertujuan untuk mendeteksi kesalahan sedini mungkin sebelum digabungkan digabungkan dengan *source* *code* dari anggota tim lainnya.

Gambaran umum *build* dapat dilihat pada gambar berikut ini:



4.4. Strategi dan *tool* *Continuous* *Integration*

#strategi

Pembuatan paket aplikasi dapat diotomasi dengan *tool*. Keseluruhan strategi dan *tools* yang mencakup VCS, *testing* dan *build* dapat diimplementasikan dengan praktik CI. Untuk menerapkan praktik CI, anggota tim membutuhkan strategi. Strategi yang akan digunakan anggota tim untuk mengimplementasikan CI adalah mengotomasi *build* dengan *trigger* dari perubahan *repository* pusat. Anggota tim dapat mengintegrasikan kode program secara rutin tanpa harus mengeluarkan *effort* yang besar karena *testing* dan *build* telah diotomasi.



Gambar 2. Pemilihan strategi dan *tool* pada praktik CI (*Continuous Integration*)

#gambar konsep CI



Gambar 3. Konsep CI

Pada gambar konsep CI dijelaskan bahwa terdapat sebuah tim yang terdiri dari 2 *developer* yang ingin menggabungkan hasil pekerjaan dengan menerapkan konsep *Continuous Integration*. Modul yang dikerjakan oleh anggota tim terdiri dari *source* *code* yang mencakup *test* *code*, *library* dan *ant* *script*. Anggota tim telah membuat kesepakatan awal bahwa sebelum mengirimkan hasil pekerjaan ke *repository* pusat harus menjalankan *build* terhadap *source code* modul yang terdapat pada *repository* lokal. Anggota tim bekerja secara paralel. Pada *server* CI telah diatur *trigger* *build* (ketika hasil pekerjaan anggota tim diintegrasikan) yang digunakan adalah memeriksa perubahan yang terjadi pada *repository* pusat. *Server* CI juga memberikan *feedback* terhadap *build* perangkat lunak yang gagal dilakukan kepada anggota tim.

Pertama sekali *developer* X akan membuat sebuah modul, yaitu modul A dan menyimpannya ke *repository* *lokal* kemudian modul tersebut dikirim ke *repository* *pusat*. Begitu juga dengan anggota tim lainnya, *developer* Y akan membuat modul B dan menyimpannya di *repository* lokal. Karena *developer* X pertama sekali mengirim hasil pekerjaanya, yaitu modul A ke *repository* pusat. Maka, ketika *developer* Y ingin mengirim perubahannya ke *repository* pusat, harus terlebih dahulu mengambil perubahan yang ada di *repository* pusat, yaitu modul A. *Server* CI (mesin pengintegrasi hasil pekerjaan *developer*) akan memerika perubahan yang terjadi di *repository* pusat. Jika ditemukan perubahan pada *repository* pusat, maka *server* CI akan melakukan *build* secara otomatis. *Build* yang berhasil dilakukan oleh *server* CI akan menghasilkan paket aplikasi *executable*. Sedangkan *build* yang gagal tidak akan akan menghasilkan paket aplikasi namun memberikan notifikasi kepada anggota tim sehingga anggota tim dapat langsung memperbaiki kesalahan pada perangkat lunak yang dibangun dan menyimpannya kembali ke *repository* pusat.

#tool CI

Untuk mengimplementasikan praktik CI, anggota tim membutuhkan sebuah *tool* CI. Salah satu *tool* CI yang dapat mengimplementasikan praktik tersebut adalah Jenkins. Dengan Jenkins, strategi dan *tools* yang mencakup VCS, *testing* dan *build* dapat diintegrasikan. Selain itu, Jenkins dapat memberikan notifikasi kesalahan kepada anggota tim apabila *build* perangkat lunak gagal dilakukan.

#gambar praktik CI (Bob, Alice, dan Charlie)



**Gambar 4. Praktik CI**

Pada gambar Praktik CI dijelaskan bahwa terdapat anggota tim yang terdiri dari Alice, Bob dan Charlie menerapkan praktik CI untuk mengintegrasikan hasil pekerjaannya. Anggota tim bekerja secara paralel. Anggota tim telah membuat kesepakatan bahwa sebelum mengintegrasikan hasil pekerjaan ke *repository* pusat harus terlebih dahulu menjalankan *build* terhadap *source code* modul yang terdapat pada *repository* lokal. Alice mengerjakan modul Jaminan, Bob mengerjakan modul Staf sedangkan Charlie mengerjakan modul Spesialis. Modul yang dikerjakan anggota tim terdiri dari *source code* yang mencakup *test code*, *library* dan *ant script*. Pada *server* CI telah diatur *trigger build* (ketika pekerjaan anggota tim diintegrasikan) yang digunakan adalah memeriksa perubahan pada repository pusat. Ketika ditemui perubahan pada *repository* pusat, maka *server* CI akan otomatis menjalankan *build*. Jika *build* perangkat lunak gagal, maka anggota tim akan memperoleh notifikasi via *email*.

Pertama sekali Alice akan membuat modul Jaminan, kemudian *source code* modul tersebut di-*build*. Jika *source* *code* tersebut telah berhasil di-*build* maka di-*commit* ke *repository* lokal. Alice akan menge-*push* modul Jaminan ke *repository* pusat. Jenkins akan memeriksa perubahan yang terjadi pada Github, ketika ditemukan perubahan pada *repository* pusat Jenkins akan melakukan *build* secara otomatis. Jika proses *build* berhasil, maka akan dihasilkan paket aplikasi. Akan tetapi *build* yang gagal dilakukan pada Jenkins akan mengirimkan notifikasi via melalui *email* kepada anggota tim.

Bob dan Charlie juga akan melakukan hal yang sama untuk modul yang dikerjakan. Ketika Bob ingin menge-*push* hasil pekerjaannya, maka Bob harus mengambil perubahan yang ada pada *repository* pusat terlebih dahulu dan melakukan *build* terhadap *source* *code* yang ada di *repository* lokal. Jika *build* tersebut telah berhasil dilakukan maka Bob akan meng-*commit* perubahan modul tersebut dan menge-*push* hasil pekerjaannya ke Github. Jenkins akan langsung melakukan *build* jika melihat ada perubahan pada *repository* pusat. Jika *build* berhasil dilakukan maka akan dihasilkan paket aplikasi. Sedangkan *build* yang gagal akan mengirimkan notifikasi via *email* kepada anggota tim

4.5 Validasi CI

Setelah anggota tim melakukan implementasi prakti CI dalam pembangunan perangkat lunak, maka anggota tim melakukan validasi praktik CI. Validasi tersebut dilakukan untuk memastikan manfaat yang diperoleh oleh anggota tim. Parameter yang digunakan untuk melakukan validasi praktik CI, antara lain:

1. Pengurangan resiko pembangunan perangkat lunak

Resiko dari pembangunan perangkat lunak yang diperoleh anggota tim, salah satunya adalah *effort* untuk perbaikan perangkat lunak. Semakin tinggi tingkat kesalahan yang ditemukan pada perangkat lunak, maka semakin tinggi pula effort yang dikeluarkan untuk perbaikan. Dengan pengimplementasian praktik CI, pengujian akan selalu dilakukan setiap kali anggota tim mengintegrasikan kode program, sehingga kesalahan perangkat lunak pada level unit dan integrasi dapat diminimalisasi.

2. Pengurangan proses manual yang berulang

Sebelum anggota tim mengimplementasikan praktik CI, anggota tim sering melakukan aktivitas pembangunan perangkat lunak yang berulang secara manual. Misalnya *import* *database*, *compile*, *testing*, *drop* *database* dan *packaging*. Aktivitas tersebut mengakibatkan anggota tim mengeluarkan *effort* yang besar. Dengan pengimplementasian praktik CI, aktivitas manual yang berulang tersebut dapat diotomasi.

3. Visibilitas proyek yang lebih baik

Pada proses pembangunan perangkat lunak yang cepat, anggota tim dituntut untuk selalu mempersiapkan semua paket aplikasi yang telah berhasil di-*build*. Dengan pengimplementasian praktik CI, semua paket aplikasi hasil *build* dapat tersimpan secara otomatis, sehingga anggota tim dapat me-*monitoring* *history* dari paket aplikasi. *History* tersebut akan membantu anggota tim dalam menentukan kualitas setiap paket aplikasi yang dihasilkan.

4. Peningkatan kepercayaan diri anggota tim terhadap produk perangkat lunak

Pembangunan perangkat lunak yang dilakukan oleh anggota tim harus minim dari kesalahan. Untuk meminimalisasi kesalahan tersebut anggota tim melakukan pengujian setiap kali melakukan *build* perangkat lunak. Dengan pengimplementasian praktik CI, pengujian dapat diotomasi pada setiap pembuatan paket aplikasi, sehingga anggota tim dapat memastikan perangkat lunak yang di-*build* minim dari kesalahan.

5. Kesimpulan

#setelah mengimplementasikan praktik CI, bagaimana? Manfaat

Setelah mengimplementasikan praktik CI dengan studi kasus MedRecApp dapat disimpulkan bahwa CI memiliki manfaat sebagai berikut:

1. Pengurangan resiko pembangunan perangkat lunak
2. Pengurangan proses manual yang berulang
3. Visibilitas proyek yang lebih baik
4. Peningkatan kepercayaan diri anggota tim terhadap produk perangkat lunak

#jelaskan prosedur, teknik dan toolset yang digunakan anggota tim

Manfaat tersebut dicapai dengan menggunakan prosedur, teknik dan toolset yang mendukung CI. Adapun prosedur, teknik dan *toolset* yang digunakan dalam pengimplementasian CI adalah:

1. Melakukan kesepakatan awal sebelum melakukan CI, seperti aturan modul dan standar modul yang ingin dibangun
2. *Tool* VCS yang sesuai kebutuhan anggota tim. Karena anggota tim bekerja secara terdistribusi, maka dipilih *tool* Git untuk mengelola versi kode program. Sedangkan *software* *hosting* yang digunakan adalah Github.
3. *Tool* *testing* yang digunakan untuk level unit dan integrasi. Pada level unit, pengujian dilakukan langsung oleh anggota tim dengan JUnit. Sedangkan pengujian integrasi dilakukan oleh anggota tim dengan menggunakan FEST
4. *Tool* *build* yang digunakan adalah Ant, build akan dilakukan setiap kali anggota tim melakukan integrasi kode program
5. *Tool* CI yang digunakan adalah Jenkins. *Tool* tersebut berperan untuk melakukan pengaturan alamat penyimpanan versi, *script* *build*, menentukan *trigger* *build*, yaitu ketika melakukan *push* ke Github dan melakukan *poll* SCM setiap satu jam sekali. Sehingga ketika Jenkins memeriksa Github dan menemukan perubahan, maka *build* akan dijalankan.