## **BAB III**

# KONSEP UMUM CONTINUOUS INTEGRATION SECARA MANUAL DAN MENGGUNAKAN TOOLSET

Bab ini berisi penjelasan tentang analisis dari konsep umum pembangunan perangkat lunak dengan *continuous integration* yang dilakukan secara manual dan menggunakan *toolset*. Analisis tersebut dilakukan untuk menunjukkan perbedaan konsep dari keduanya. Konsep umum pembangunan perangkat lunak dengan *continuous integration* secara manual yaitu mencakup konsep penyimpanan versi secara manual, konsep pengujian kode program secara manual, konsep eksekusi *build* secara manual, dan konsep pengintegrasian modul secara manual. Sedangkan konsep umum dari pembangunan perangkat lunak dengan *continuous integration* menggunakan *toolset* yaitu mencakup konsep penyimpanan versi dengan *version control system tool*, konsep pengujian kode program dengan *automated testing tool*, konsep eksekusi *build* dengan *automated build tool*, dan konsep pengintegrasian modul dengan *automated continuous integration tool*.

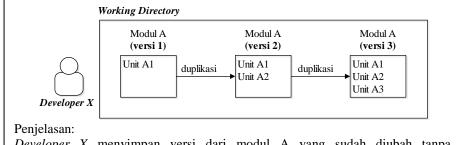
# 3.1. Konsep umum continuous integration secara manual

Continuous integration adalah praktik pembangunan perangkat lunak yang dilakukan secara tim dengan membagi pekerjaan berdasarkan modul pada perangkat lunak. Praktik tersebut mengharuskan setiap anggota tim untuk mengintegrasikan modul hasil pekerjaan mereka secara rutin. Tim yang membangun perangkat lunak dengan continuous integration secara manual, umumnya tidak menggunakan bantuan toolset. Kegiatan manual yang dilakukan tim tersebut mencakup penyimpanan versi, pengujian kode program, eksekusi build, dan pengintegrasian modul.

# 3.1.1. Konsep penyimpanan versi secara manual

Pada sub bab ini akan dijelaskan tentang konsep penyimpanan versi yang umum dilakukan tim pada praktik *continuous integration* tanpa menggunakan bantuan *tool* dari *version control system*.

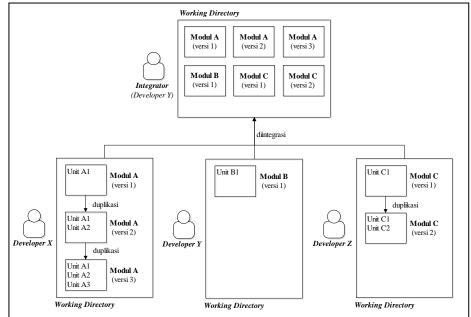
Penyimpanan versi dilakukan tim untuk menyimpan *history* dari setiap perubahan modul. Tim yang tidak menggunakan bantuan *tool* dari *version control system* umumnya akan menduplikasi modul sebelum mengubah modul tersebut. Hasil duplikasi modul digunakan tim sebagai *backup* untuk melakukan *rollback* terhadap modul yang belum diubah. Untuk membedakan hasil dari setiap duplikasi modul, tim perlu melakukan penamaan versi dan menambahkan informasi tentang detil perubahan yang telah dilakukan pada modul tersebut.



Developer X menyimpan versi dari modul A yang sudah diubah tanpa menggunakan bantuan version control system tool. Developer X menduplikasi modul A sebelum mengubahnya. Hasil duplikasi modul A digunakan developer X sebagai backup untuk melakukan rollback terhadap versi modul A sebelumnya yang belum diubah. Developer X melakukan penamaan versi untuk membedakan setiap hasil duplikasi dan mencatat detil isi perubahan setiap modul secara manual.

Gambar 3-1. Penyimpanan versi dengan cara manual

Setiap versi modul yang dibuat para anggota tim, umumnya akan disimpan di tempat penyimpanan versi terpusat. Kegiatan tersebut dilakukan agar mereka tidak salah dalam memahami versi modul yang telah mereka buat. Tim yang tidak menggunakan tool dari version control system, umumnya akan membutuhkan seorang integrator untuk mengelola semua versi modul di tempat penyimpanan versi terpusat. Integrator tersebut akan memilih versi dari setiap modul yang akan dijadikan paket aplikasi yang berisi file siap pakai.



Developer Y berperan sebagai integrator yang akan mengelola versi modul di tempat penyimpanan versi terpusat. Setiap versi modul yang dibuat para developer perlu disimpan di tempat penyimpanan versi terpusat, agar mereka tidak salah dalam memahami versi modul yang telah mereka buat. Developer yang membutuhkan modul hasil pekerjaaan developer lain, dapat mengambil versi modul tersebut dari tempat penyimpanan versi terpusat. Semua versi modul di tempat penyimpanan versi terpusat akan dikelola oleh integrator sebelum dijadikan paket aplikasi yang berisi file siap pakai. Ukuran kapasitas tempat penyimpanan terpusat akan semakin membesar seiring dengan bertambahnya jumlah versi modul yang telah dibuat para developer.

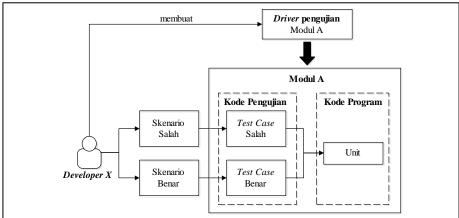
Gambar 3-2. Penggabungan versi modul secara manual

# 3.1.2. Konsep pengujian kode program secara manual

Modul yang dikerjakan setiap anggota tim akan ditambahi unitunit kode program. Setiap unit yang ditambahi ke dalam modul harus diuji. Pengujian unit dilakukan setiap anggota tim untuk memastikan bahwa *functional requirement* dari modul yang telah dibuat dapat dieksekusi serta minim dari kesalahan.

Untuk menguji setiap unit dari modul tersebut, tim memerlukan kode pengujian unit. Pada setiap kode pengujian, anggota tim akan menambahkan satu atau lebih kasus uji untuk menguji satu unit kode program. Umumnya, tim yang tidak menggunakan bantuan *tool* dari *automated testing* perlu membuat *driver* pengujian pada setiap kode pengujian. *Driver* pengujian digunakan setiap anggota tim untuk

mengeksekusi kode pengujian tersebut. Ketika terjadi kesalahan pada satu atau lebih hasil pengujian, anggota tim perlu memperbaikinya dan mengeksekusi kembali semua *driver* pengujian dari awal.



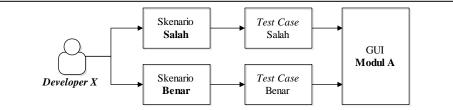
Penjelasan:

Developer X melakukan pengujian unit tanpa menggunakan bantuan *tool* dari automated testing. Developer X menguji unit pada modul A dengan membuat kode pengujian. Kode pengujian berisi test case berdasarkan skenario yang telah dibuat oleh developer X. Setiap unit kode program, dapat diuji dengan lebih dari satu test case. Untuk mengeksekusi kode pengujian unit pada modul A, developer X perlu membuat driver pengujian modul A.

Jika developer X membuat tiga modul, maka developer X perlu membuat tiga driver pengujian. Ketika terjadi kesalahan pada salah satu hasil pengujian, developer X perlu memperbaikinya dan mengeksekusi kembali ketiga driver pengujian tersebut satu per satu.

Gambar 3-3. Pengujian unit secara manual

Pada pengujian unit di komponen GUI (*Graphical User Interface*), anggota tim perlu membuat skenario salah dan benar terhadap komponen GUI pada modul tersebut. Umumnya, anggota tim yang tidak menggunakan bantuan *tool* dari *automated testing* akan melakukan skenario salah dan benar terhadap komponen GUI secara manual. Pengujian unit pada komponen GUI dilakukan tim untuk memastikan bahwa antarmuka modul dapat berfungsi seperti yang diharapkan serta dapat memenuhi spesifikasi dan persyaratan.



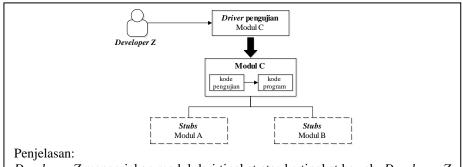
Developer X melakukan pengujian unit pada komponen GUI modul A tanpa bantuan tool dari automated testing. Developer X menguji unit pada komponen GUI dengan cara running antarmuka modul A kemudian melakukan skenario salah dan benar secara manual terhadap antarmuka modul A.

Setiap skenario benar dan salah dapat terdiri lebih dari satu *test case*. Ketika GUI modul A diubah, *developer X* perlu mengulangi kembali semua skenario salah dan benar dari awal untuk memastikan bahwa antarmuka modul tetap berfungsi seperti yang diharapkan.

Gambar 3-4. Pengujian unit pada GUI secara manual

Modul yang terdapat dependensi dengan modul yang lain perlu dilakukan pengujian integrasi. Pada pengujian integrasi, unit yang diuji berdependensi dengan unit dari modul yang lain. Pengujian integrasi dilakukan anggota tim untuk menguji kombinasi modul sebagai satu kesatuan modul perangkat lunak dan mengekspos kesalahan pada interaksi antar unit yang terintegrasi.

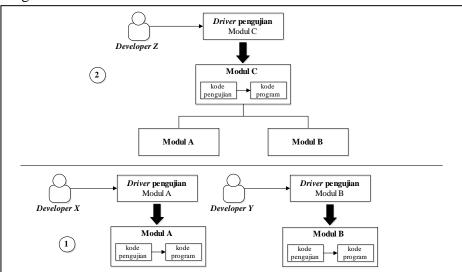
Untuk melakukan pengujian integrasi, anggota tim perlu menentukan strategi pengujian integrasi terlebih dahulu. Strategi pengujian integrasi dilakukan secara incremental, yang diklasifikasikan menjadi dua cara yaitu top-down dan bottom-up. Pada strategi top-down, anggota tim menguji modul dari tingkat atas ke modul tingkat bawah. Anggota tim yang menggunakan strategi topdown, perlu membuat stubs sebagai pengganti modul-modul tingkat bawah yang belum dibuat. Stubs tersebut akan digunakan anggota tim untuk menguji integrasi modul pada tingkat atas. Ketika anggota tim yang lain telah selesai membuat modul-modul pada tingkat bawah, stubs tersebut tidak akan digunakan kembali.



Developer Z mengerjakan modul dari tingkat atas ke tingkat bawah. Developer Z membuat modul C yang berdependensi dengan modul A dan modul B. Untuk menguji modul C, developer Z perlu membuat driver pengujian modul C dan stubs untuk menggantikan peran modul A dan modul B yang belum selesai dibuat. Ketika anggota tim yang lain telah menyelesaikan modul A dan modul B, stubs tersebut tidak digunakan kembali.

Gambar 3-5. Pengujian integrasi dengan strategi top-down

Pada strategi *bottom-up*, anggota tim menguji modul dari tingkat bawah ke modul tingkat atas. Anggota tim yang menggunakan strategi *bottom-up* tidak lagi memerlukan *stubs*, karena modul-modul pada tingkat bawah telah dibuat di awal.



### Penjelasan:

Tim terdiri dari tiga *developer* yang mengerjakan tiga modul yang berbeda. Salah satu *developer* akan menguji integrasi modul dengan strategi *bottom-up*. Pada strategi tersebut, tim akan membuat modul pada tingkat bawah terlebih dahulu, agar pengujian modul di tingkat atas tidak membutuhkan *stubs*.

Pada gambar **bagian 1**, dijelaskan bahwa *developer X* dan *developer Y* mengerjakan modul A dan modul B terlebih dahulu. Modul A dan modul B tidak terdapat dependensi modul satu sama lain.

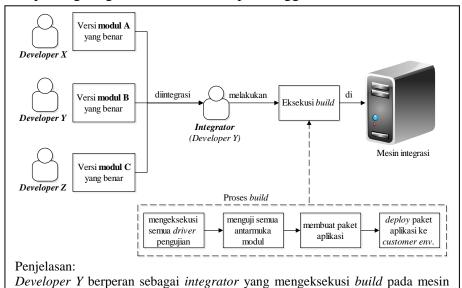
Pada gambar **bagian 2**, dijelaskan bahwa *developer Z* mengerjakan modul C yang berdependensi dengan modul A dan modul B. *Developer Z* tidak lagi memerlukan *stubs* untuk menguji modul C, karena modul A dan modul B telah dibuat di awal.

Gambar 3-6. Pengujian integrasi dengan strategi bottom-up

# 3.1.3. Konsep eksekusi build secara manual

Setelah para anggota tim menguji modul yang telah mereka buat, salah satu dari mereka akan berperan sebagai *integrator* untuk membuat paket aplikasi yang berisi *file* siap pakai. Untuk membuat paket aplikasi tersebut, seorang *integrator* perlu mengeksekusi *build*. Umumnya, eksekusi *build* dilakukan *integrator* pada mesin integrasi.

Untuk mengeksekusi *build*, seorang *integrator* perlu melengkapi semua versi modul yang benar dari para anggota tim dan menguji semua modul tersebut. *Integrator* yang tidak menggunakan bantuan *tool* dari *automated build* akan melakukan proses *build* secara manual. Proses *build* tersebut diantaranya eksekusi semua *driver* pengujian, pengujian semua antarmuka modul, pembuatan paket aplikasi, dan *deploy* paket aplikasi ke *customer environment*. Rangkaian proses *build* tersebut dilakukan *integrator* secara manual dan berulang kali setiap mengintegrasikan modul dari para anggota tim.



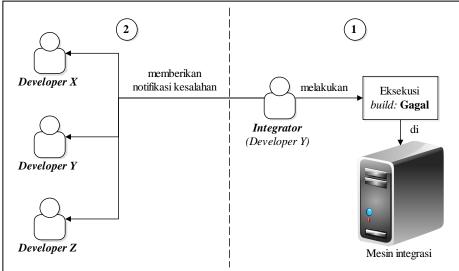
bergantung pada langkah proses yang sebelumnya.

Gambar 3-7. Eksekusi *build* dengan cara manual

integrasi tanpa menggunakan bantuan *automated build tool*. Sebelum *integrator* mengeksekusi *build*, *integrator* perlu melengkapi semua versi modul yang benar dari para anggota tim terlebih dahulu. Setiap langkah proses pada eksekusi *build*,

# 3.1.4. Konsep pengintegrasian modul secara manual

Tim yang mengintegrasikan modul tanpa bantuan *tool* dari automated continuous integration, umumnya akan membutuhkan seorang integrator untuk mengeksekusi build di mesin integrasi. Pada proses eksekusi build, integrator akan mengeksekusi semua driver pengujian dan menguji semua antarmuka modul yang telah dibuat para anggota tim. Pengujian tersebut dilakukan integrator untuk memastikan bahwa paket aplikasi yang akan dibuat dan di-deploy ke customer environment, telah sesuai dengan functional requirement serta minim dari kesalahan. Ketika terjadi kesalahan pada satu atau lebih hasil pengujian, integrator perlu menginformasikan kesalahan tersebut kepada para anggota tim untuk dapat segera diperbaiki.



#### Penjelasan:

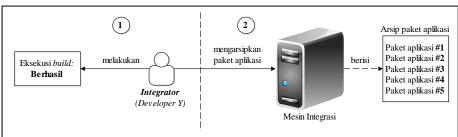
Tim mengintegrasikan modul tanpa menggunakan bantuan *automated continuous integration tool*. *Developer Y* berperan sebagai *integrator* yang akan mengeksekusi *build* di mesin integrasi. Sebelum membuat paket aplikasi yang berisi *file* siap pakai dan *deploy* paket aplikasi ke *customer environment*, *integrator* perlu mengintegrasi dan menguji semua versi modul yang benar dari para anggota tim.

Pada gambar **bagian 1**, dijelaskan bahwa *integrator* gagal melakukan eksekusi *build* karena terdapat kesalahan pada satu atau lebih hasil pengujian. *Integrator* perlu mencatat kesalahan tersebut secara manual.

Pada gambar **bagian 2**, dijelaskan bahwa *integrator* menginformasikan kesalahan tersebut secara manual kepada para *developer* untuk dapat segera diperbaiki.

Gambar 3-8. Pemberian notifikasi kesalahan secara manual oleh *integrator* 

Integrasi modul yang telah lulus dari pengujian, akan dijadikan paket aplikasi yang berisi *file* siap pakai dan di-*deploy* ke *customer environment*. Untuk mendapatkan *history* dari semua paket aplikasi yang telah dibuat, maka paket aplikasi perlu diarsipkan. Tim yang tidak menggunakan *automated continuous integration tool*, umumnya akan membutuhkan seorang *integrator* untuk mengarsipkan paket aplikasi tersebut di mesin integrasi.



#### Penjelasan:

Tim mengintegrasikan modul tanpa menggunakan bantuan *tool* dari *automated continuous integration*. *Developer Y* berperan sebagai *integrator* yang mengeksekusi *build* pada mesin integrasi.

Pada gambar **bagian 1**, dijelaskan bahwa *integrator* berhasil mengeksekusi *build* di mesin integrasi. Hasil dari eksekusi *build* tersebut adalah paket aplikasi yang berisi *file* siap pakai.

Pada gambar **bagian 2**, dijelaskan bahwa untuk menyimpan *hitory* dari perubahan paket aplikasi, maka paket aplikasi perlu diarsipkan. *Integrator* mengarsipkan apket aplikasi di mesin integrasi secara manual.

Gambar 3-9. Pengarsipan paket aplikasi secara manual oleh integrator

Arsip dari paket aplikasi tersebut, dapat dijadikan *milestone* dari kemajuan proses pembangunan perangkat lunak. Untuk mendapatkan informasi tentang kemajuan proses pembangunan perangkat lunak, tim yang mengintegrasikan modul secara manual umumnya akan memerlukan seorang *integrator* untuk membuat *report* kemajuan proses pembangunan perangkat lunak di mesin integrasi.

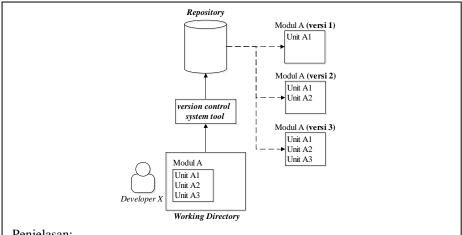
**Gambar 3-10.** Pembuatan *report* kemajuan proses pembangunan perangkat lunak oleh *integrator* 

# 3.2. Konsep umum continuous integration menggunakan toolset

Kegiatan-kegiatan yang dilakukan para anggota tim pada praktik continuous integration secara manual, membutuhkan effort yang besar. Selain itu, para anggota tim memiliki tingkat ketelitian yang terbatas, sehingga kegiatan manual tersebut sangat rentan terhadap kesalahan. Dengan menggunakan bantuan toolset, kegiatan-kegiatan manual yang mencakup penyimpanan versi, pengujian kode program, eksekusi build, dan pengintegrasian modul dapat diotomasi, sehingga praktik continuous integration dapat lebih efisien.

# 3.2.1. Konsep penyimpanan versi dengan version control system tool

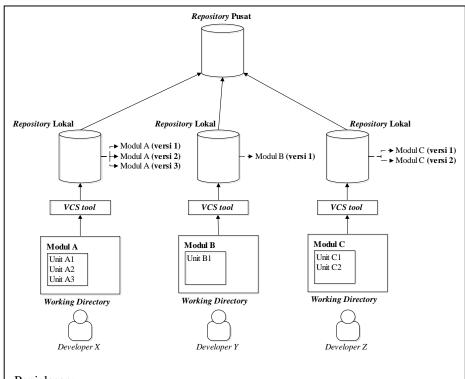
Pada sub bab ini akan dijelaskan tentang konsep penyimpanan versi pada praktik *continuous integration* dengan bantuan *tool* dari *version control system*. Tim yang telah menggunakan *version control system tool*, akan menyimpan semua versi modul yang sudah diubah ke dalam *repository*, sehingga mereka dapat melakukan *rollback* terhadap versi modul tanpa perlu menduplikasi modul terlebih dahulu. Para anggota tim tidak perlu lagi menambahkan informasi tentang detil perubahan yang dilakukan terhadap modul secara manual, karena *tool* dari *version control system* akan mencatat waktu dan detil isi perubahan secara otomatis ketika mereka menyimpan versi modul ke *repository*.



Developer X menyimpan versi dari modul A yang sudah diubah dengan menggunakan bantuan version control system tool. Versi dari modul A yang sudah diubah oleh developer X disimpan di dalam repository, sehingga developer X tidak perlu lagi menduplikasi modul A. Waktu dan detil isi perubahan modul A dicatat secara otomatis oleh version control system tool ketika developer X menyimpan versi modul A ke repository.

**Gambar 3-11.** Penyimpanan versi modul ke dalam *repository* 

Umumnya, cara penggunaan *repository* untuk menerapkan praktik *version control system* adalah *distributed*. Dengan menggunakan cara *distributed*, setiap anggota tim akan memiliki *repository* pada mesin lokal masing-masing. *Repository* dari setiap anggota tim tersebut, umumnya akan dihubungkan dengan sebuah *repository* pusat, agar para anggota tim tidak salah dalam memahami versi modul yang telah mereka simpan. Penggunaan *repository* dengan cara *distributed* dan dihubungkan pada sebuah *repository* pusat, disebut *centralized workflow*.

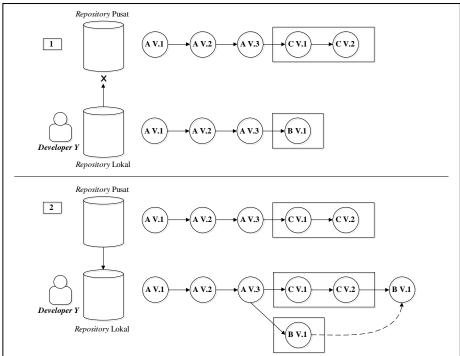


Developer X mengerjakan modul A, developer Y mengerjakan modul B, dan developer Z mengerjakan modul C. Setiap developer menyimpan versi dari modul yang sudah diubah ke dalam repository lokal masing-masing. Semua versi modul di dalam repository lokal akan disimpan ke repository pusat. Pekerjaan tersebut dilakukan agar setiap developer tidak salah dalam memahami versi modul yang telah mereka simpan.

Gambar 3-12. Centralized workflow

Setiap versi dari modul yang sudah diubah dan disimpan ke dalam repository lokal, selanjutnya akan disimpan ke dalam repository pusat. Anggota tim yang repository lokalnya belum ada versi terbaru dari modul di repository pusat, tidak dapat menyimpan versi modulnya ke repository pusat. Untuk mengatasi masalah tersebut, anggota tim hanya perlu mengambil versi terbaru dari modul di repository pusat terlebih dahulu. Setiap versi terbaru dari modul yang diambil dari repository pusat, akan digabungkan dengan versi modul yang ada di repository lokal secara otomatis. Dengan menggunakan tool dari version control system, penggabungan versi modul dapat dilakukan oleh setiap anggota tim, sehingga mereka tidak lagi memerlukan

seorang *integrator* untuk melengkapi semua versi modul yang benar dari setiap anggota tim.



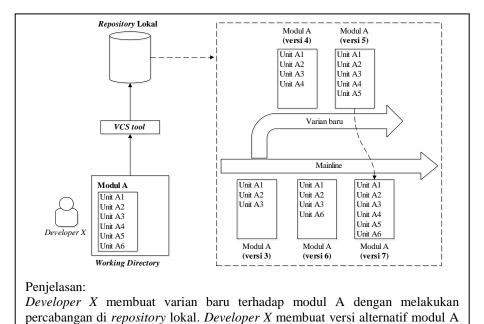
Penjelasan:

Pada gambar bagian 1, dijelaskan bahwa *developer Y* tidak dapat menyimpan versi modul yang ada di *repository* lokal ke *repository* pusat. Kegiatan tersebut tidak dapat dilakukan karena *repository* lokal *developer Y* belum ada versi terbaru dari modul di *repository* pusat.

Pada gambar bagian 2, dijelaskan bahwa *developer Y* telah mengambil versi terbaru dari modul di *repository* pusat. Versi terbaru dari modul yang diambil dari *respository* pusat, akan digabungkan dengan versi modul yang ada di *repository* lokal secara otomatis.

Gambar 3-13. Penggabungan versi modul

Pada proses penyimpanan versi secara manual, para anggota tim yang akan membuat varian baru terhadap modul, umumnya akan menduplikasi modul terlebih dahulu. Tetapi, para anggota tim yang telah menggunakan *tool* dari *version control system*, tidak lagi menduplikasi modul. Mereka dapat membuat varian baru terhadap modul dengan melakukan percabangan di setiap *repository* lokal masing-masing. Hasil dari percabangan tersebut dapat dijadikan versi alternatif modul tanpa harus mengubah kode program yang ada di *mainline repository*.



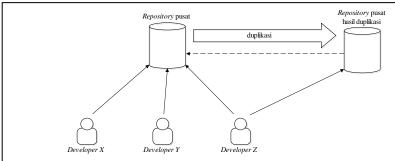
di cabang varian baru tanpa mengubah *mainline repository*. Versi alternatif modul tersebut dapat digabungkan dengan versi modul di *mainline repository* 

hingga menjadi versi terbaru dari modul A.

Gambar 3-14. Percabangan versi modul

Dengan menggunakan *tool* dari *version control system*, anggota tim yang akan membuat produk perangkat lunak yang berbeda dari perencanaan awal oleh tim, dapat menduplikasi *repository* pusat. Anggota tim tersebut dapat mengubah produk perangkat lunak pada *repository* hasil duplikasi, tanpa harus mengubah *repository* pusat.

14



Developer Z menduplikasi repository pusat karena akan membuat produk perangkat lunak yang berbeda dari perencanaan awal oleh tim. Developer Z dapat mengubah produk perangkat lunak di repository hasil duplikasi, tanpa harus mengubah repository pusat. Semua perubahan produk perangkat lunak yang dilakukan developer Z di repository hasil duplikasi tidak dapat disimpan ke repository pusat. Untuk dapat menyimpan perubahan tersebut ke repository pusat, developer Z perlu melakukan request terlebih dahulu kepada tim untuk mengambil perubahan yang telah dilakukan developer Z di repository hasil duplikasi.

Gambar 3-15. Penduplikasian repository pusat

# 3.2.2. Konsep pengujian kode program dengan automated testing tool

Pada sub bab ini akan dijelaskan tentang konsep pengujian kode program pada praktik *continuous integration* dengan bantuan *tool* dari *automated testing*. Praktik *automated testing* mencakup *tool* dari *unit testing* dan *integration testing*. Tim yang telah menggunakan *tool unit testing*, tidak perlu lagi membuat *driver* pada setiap kode pengujian. Selain itu, tim tidak perlu lagi mengeksekusi kode pengujian tersebut satu per satu, karena semua eksekusi kode pengujian akan diotomasi oleh *tool unit testing*. Informasi hasil pengujian, akan diberikan kepada tim secara otomatis oleh *tool unit testing*, sehingga tim dapat memperoleh *feedback* terhadap pengujian unit dengan cepat. Ketika terjadi kesalahan pada satu atau lebih hasil pengujian unit, tim dapat segera memperbaiki kesalahan tersebut dan mengulang kembali semua eksekusi kode pengujian unit dari awal secara otomatis.

Gambar 3-16. Pengujian unit dengan bantuan tool unit testing

Pada pengujian integrasi dengan tool integration testing, pengujian integrasi akan dieksekusi secara otomatis, sehingga tim tidak lagi memerlukan seorang integrator untuk melakukan pengujian integrasi. Tim akan membuat kode pengujian yang berisi skenario salah dan skenario benar dari hasil integrasi unit atau modul. Tool integration testing akan melakukan semua skenario benar dan salah tersebut terhadap hasil integrasi secara otomatis, sehingga tim tidak mengeluarkan effort yang besar untuk melakukan pengujian integrasi. Informasi hasil pengujian integrasi akan diberikan secara otomatis oleh tool integration testing, sehingga tim dapat segera memperbaiki kesalahan tersebut dan mengulang kembali semua skenario salah dan benar terhadap hasil integrasi modul secara otomatis.

#### [GAMBAR]

Gambar 3-17. Pengujian integrasi dengan bantuan tool integration testing

## 3.2.3. Konsep eksekusi build dengan automated build tool

Dengan menggunakan tool dari automated build, semua proses pengujian unit dan integrasi, hingga penyimpanan versi modul yang sudah diubah ke repository lokal dapat diotomasi. Untuk mengotomasi semua proses tersebut, tim membutuhkan build script. Build script tersebut berisi beberapa target dan task yang akan dieksekusi oleh tool dari automated build. Umumnya, tim membuat build script untuk menyamakan proses alur kerja dari setiap anggota tim dan mengotomasikan proses yang akan dieksekusi oleh mesin integrasi.

Build script yang dieksekusi oleh tool dari automated build pada mesin lokal setiap anggota tim, disebut private build. Untuk menyamakan alur kerja setiap anggota tim, tim perlu menentukan target dan task yang harus dilakukan oleh tool dari automated build pada mesin lokal setiap anggota tim. Setiap target dapat terdiri dari beberapa task dan setiap target dapat bergantung pada target yang

lain. Umumnya, beberapa *target* yang ada pada *private build* mencakup pengujian kode program dan penyimpanan versi modul yang sudah diubah ke dalam *repository* lokal.

### [GAMBAR]

### Gambar 3-18. Eksekusi build pada mesin lokal

Untuk mengotomasikan semua proses pada mesin integrasi, tim memerlukan beberapa target dan task yang harus dieksekusi oleh tool dari automated build di mesin integrasi. Build script yang dieksekusi oleh tool dari automated build pada mesin integrasi, disebut integration build. Umumnya target pada integration build mencakup inisialisasi path kode program, penghapusan file hasil kompilasi, kompilasi kode program, pengujian unit dan integrasi, serta pembuatan paket aplikasi.

# [GAMBAR]

# Gambar 3-19. Eksekusi build pada mesin integrasi

# 3.2.4. Konsep pengintegrasian modul dengan automated continuous integration tool

Pada umumnya, tim yang tidak menggunakan tool dari automated continuous integration akan mengeksekusi build script pada mesin integrasi secara manual. Tim akan mengambil versi terbaru dari modul yang sudah diubah pada repository pusat, kemudian mengeksekusi integration build dan release build. Dengan menggunakan tool dari automated continuous integration pada mesin integrasi, tim tidak lagi mengeksekusi kedua proses tersebut secara manual, karena kedua proses tersebut dapat diotomasi dan dijadwalkan. Mesin integrasi akan mengambil versi terbaru dari modul yang sudah diubah pada repository pusat secara otomatis, berdasarkan jadwal tersebut. Umumnya, eksekusi integration build dan release build akan dijadwalkan oleh tim pada setiap malam atau setiap terjadi perubahan versi modul pada repository pusat.

## Gambar 3-20. Penjadwalan eksekusi build script pada mesin integrasi

Pada setiap eksekusi *integration build*, mesin integrasi akan melakukan pengujian terhadap modul secara otomatis. Pengujian tersebut dilakukan mesin integrasi berdasarkan kode pengujian yang telah disimpan oleh setiap anggota tim di dalam *repository* pusat. Dengan menggunakan *tool* dari *automated continuous integration* pada mesin integrasi, tim tidak lagi memerlukan seorang *integrator* pada mesin integrasi untuk menginformasikan kesalahan pada satu atau lebih hasil pengujian. Notifikasi kesalahan tersebut akan diinformasikan oleh *tool* tersebut kepada setiap anggota tim secara otomatis.

#### [GAMBAR]

Gambar 3-21. Notifikasi kesalahan secara otomatis dari mesin integrasi

Dengan menggunakan *tool* dari *automated continuous integration*, tim tidak lagi memerlukan seorang *integrator* untuk mengarsipkan paket aplikasi pada mesin integrasi. *Tool* tersebut akan mengarsipkan paket aplikasi secara otomatis, ketika mesin intrgrasi berhasil mengeksekusi *integration build*.

## [GAMBAR]

Gambar 3-22. Pengarsipan paket aplikasi oleh mesin integrasi secara otomatis

Tool dari automated continuous integration dapat memberikan report kemajuan proses pembangunan perangkat lunak kepada setiap anggota tim secara otomatis, sehingga tim tidak lagi memerlukan seorang integrator untuk membuat report tersebut.

**Gambar 3-23.** *Report* kemajuan proses pembangunan perangkat lunak secara otomatis