

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*.

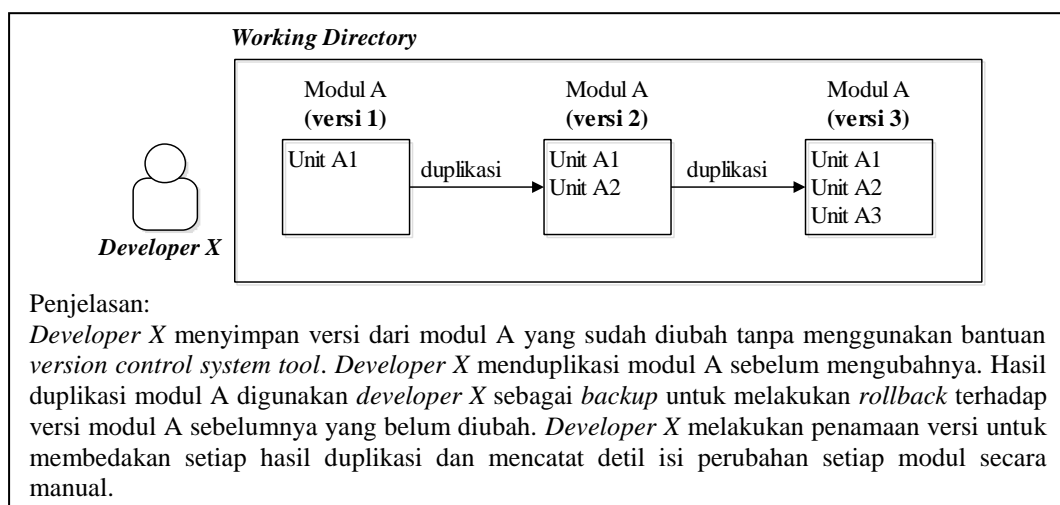
3.1. Konsep umum *continuous integration* secara manual

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

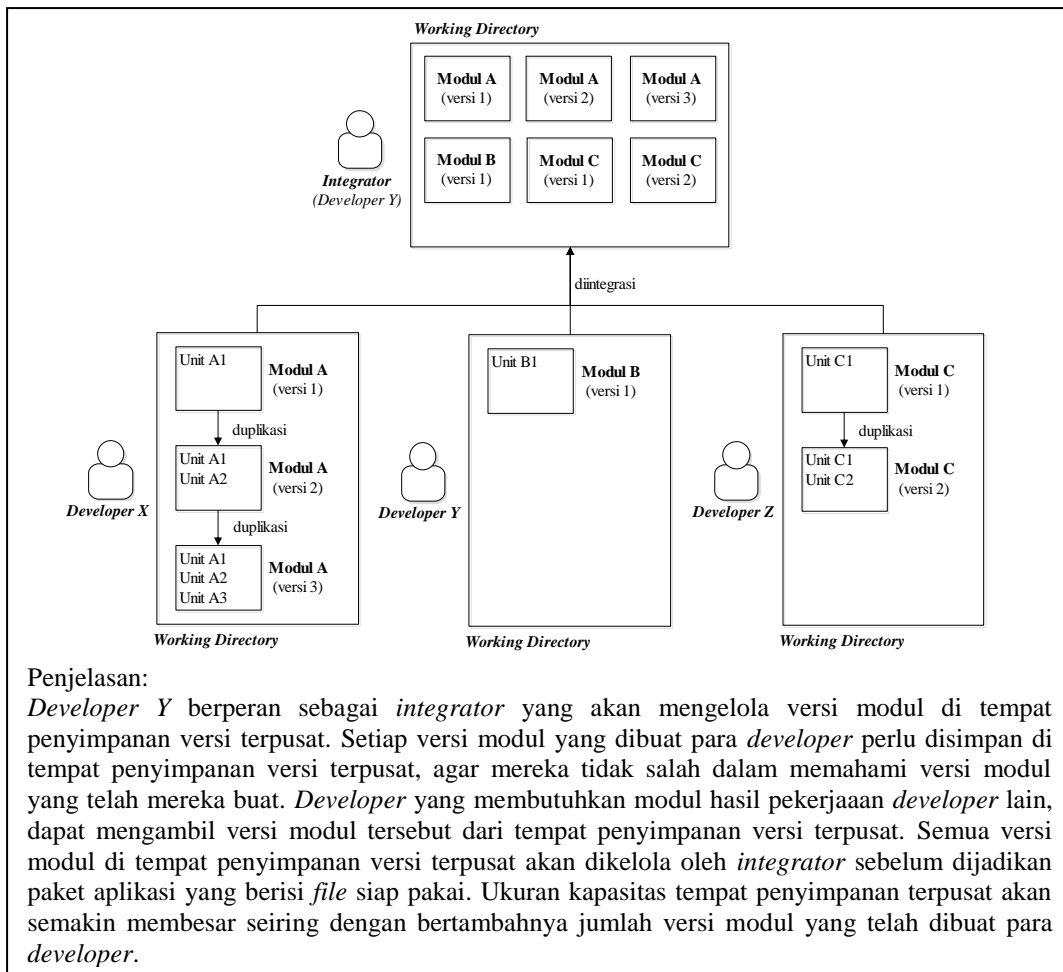
Pada sub bab ini akan dijelaskan tentang konsep penyimpanan versi yang umum dilakukan tim pada praktik *continuous integration* tanpa menggunakan

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 detail perubahan yang telah dilakukan pada modul tersebut.



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.



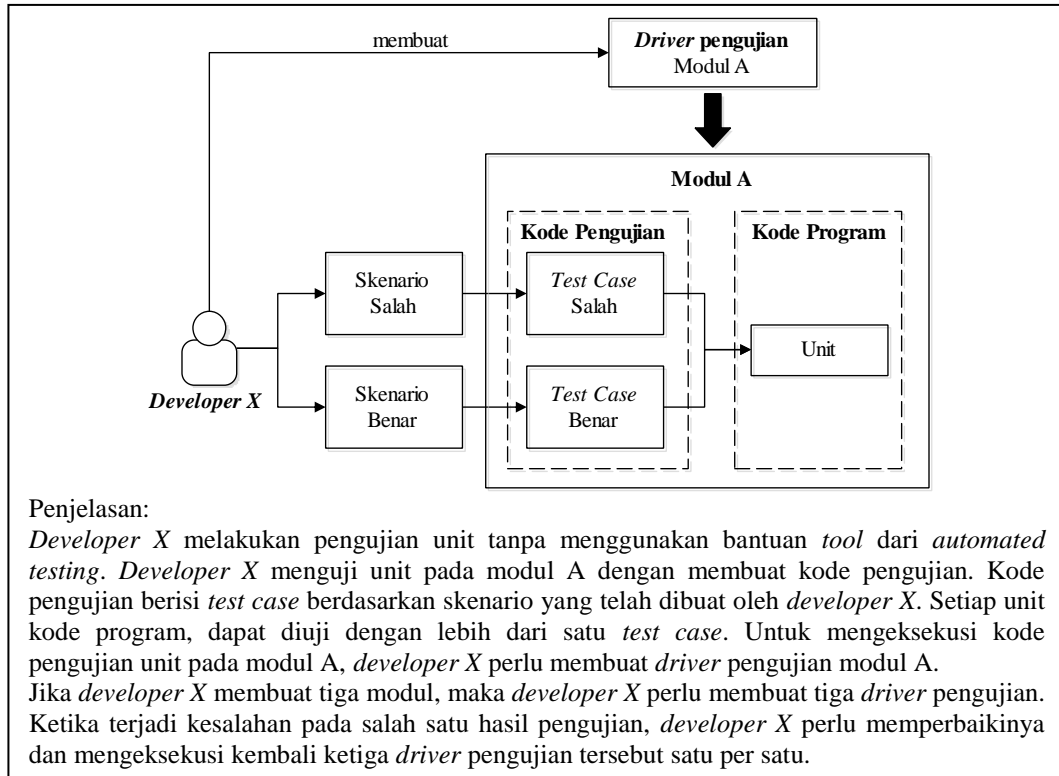
Gambar 3-2. Penggabungan versi modul secara manual

3.1.2. Konsep pengujian kode program secara manual

Modul yang dikerjakan setiap anggota tim akan ditambahi unit-unit 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.

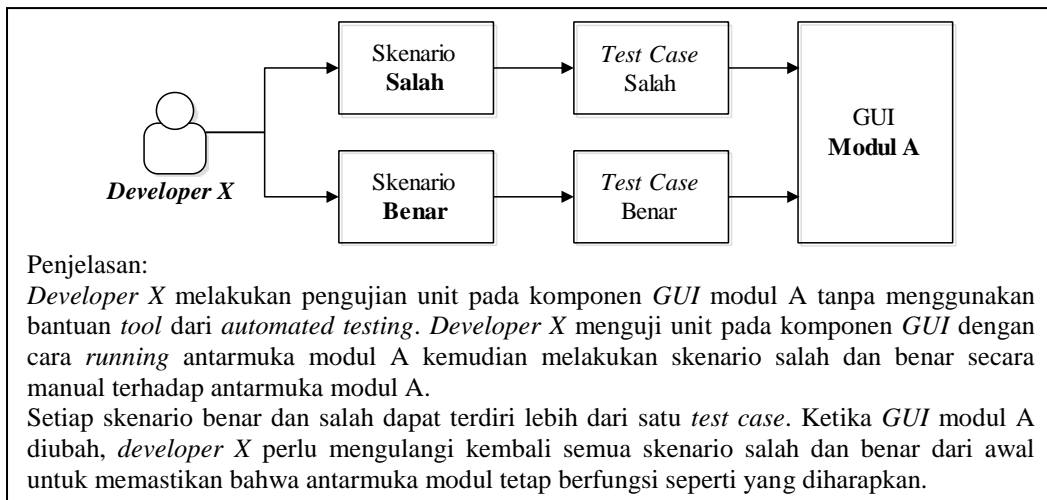
Untuk menguji setiap unit dari modul tersebut, tim memerlukan kode pengujian unit. Pada setiap kode pengujian, anggota tim akan menambahkan satu atau lebih kasus uji untuk menguji satu unit kode program. Umumnya, tim yang tidak menggunakan bantuan *tool* dari *automated testing* perlu membuat *driver*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



Gambar 3-3. Pengujian unit secara manual

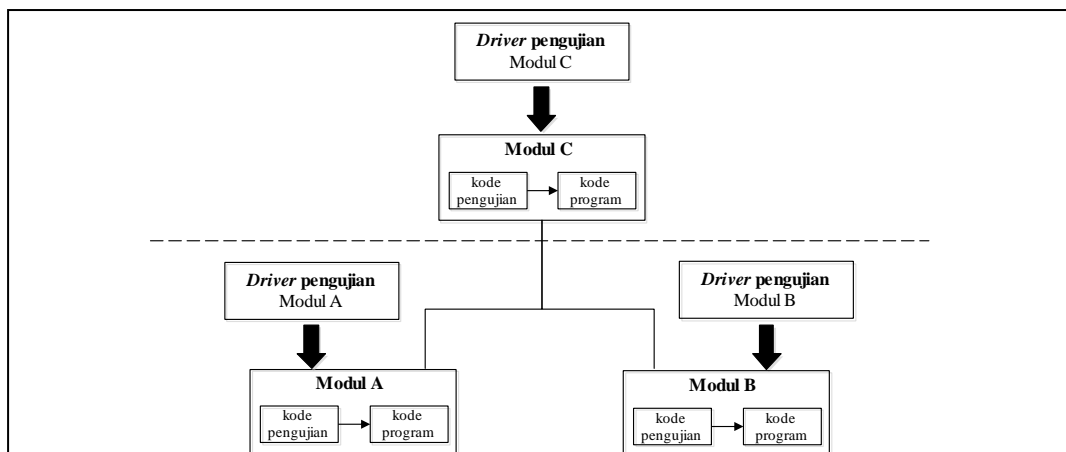
Pada pengujian unit di komponen *GUI* (*Graphical User Interface*), anggota tim perlu membuat skenario salah dan benar terhadap komponen *GUI* pada modul tersebut. Umumnya, anggota tim yang tidak menggunakan bantuan *tool* dari *automated testing* akan melakukan skenario salah dan benar terhadap komponen *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.



Gambar 3-4. Pengujian unit pada *GUI* secara manual

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

Untuk melakukan pengujian integrasi, tim perlu menentukan strategi pengujian integrasi terlebih dahulu. Strategi pengujian integrasi yang dilakukan secara *incremental*, diklasifikasikan menjadi dua cara yaitu *top-down* dan *bottom-up*. Pada strategi *bottom-up*, tim akan menguji integrasi modul dari tingkat bawah ke tingkat atas. Anggota tim yang menguji modul tingkat atas, perlu menguji modul tingkat bawah terlebih dahulu. Kegiatan tersebut dilakukan anggota tim agar dapat mengetahui penyebab pasti kesalahan pada modul tingkat atas. Umumnya, anggota tim yang tidak menggunakan bantuan *tool* dari *automated testing*, akan menguji integrasi modul dengan mengeksekusi *driver* pengujian modul dan menguji antarmuka modul pada modul tingkat bawah secara manual.

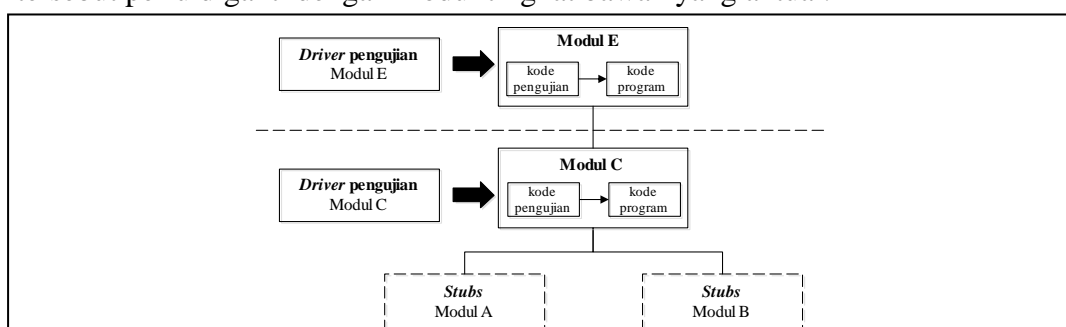


Penjelasan:

Terdapat tiga modul yang akan diintegrasikan yaitu modul A, B dan C. Modul C berdependensi dengan modul A dan B. Modul-modul tersebut diintegrasikan dengan strategi *bottom-up*. *Developer* yang mengerjakan modul C, perlu menguji integrasi modul C dengan modul A dan modul B. Sebelum menguji modul C, *developer* tersebut perlu untuk menguji modul tingkat bawah terlebih dahulu, agar *developer* tersebut dapat mengetahui penyebab kesalahan yang terjadi pada modul tingkat atas. *Developer* yang tidak menggunakan bantuan *automated testing tool* akan menguji modul tingkat bawah dengan mengeksekusi *driver* pengujian satu per satu dan menguji interaksi antarmuka modul pada modul tingkat bawah secara manual.

Gambar 3-5. Pengujian integrasi (*bottom-up*) secara manual

Pada strategi *top-down*, tim akan menguji integrasi modul dari tingkat atas ke tingkat bawah. Tim yang mengintegrasikan modul dari tingkat atas terlebih dahulu, perlu membuat *stubs* untuk menggantikan peran modul pada tingkat bawah yang belum selesai dibuat. Dengan *stubs* tersebut, anggota tim tetap dapat menguji modul tingkat atas walaupun modul tingkat bawah belum ada. Ketika anggota tim yang lain telah selesai membuat modul tingkat bawah, maka *stubs* tersebut perlu diganti dengan modul tingkat bawah yang aktual.

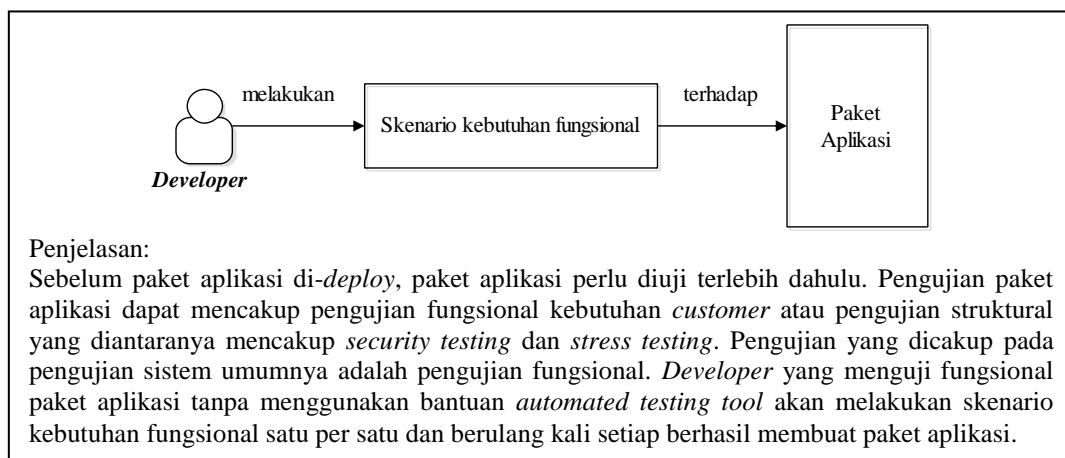


Penjelasan:

Terdapat empat modul yang akan diintegrasikan yaitu modul A, B, C dan E. Modul E berdependensi dengan modul C, modul C berdependensi dengan modul A dan B. Modul-modul tersebut akan diintegrasikan dengan dengan strategi *top-down*. Pada strategi *top-down*, modul dibuat dari tingkat atas ke tingkat bawah. Kebutuhan modul di tingkat bawah akan digantikan sementara dengan *stubs*. *Developer* yang mengerjakan modul E, perlu menguji modul di tingkat bawah terlebih dahulu. *Developer* yang menguji integrasi tanpa menggunakan bantuan *automated testing tool*, akan mengeksekusi *driver* pengujian modul di tingkat bawah hingga ke atas secara berurutan dan manual.

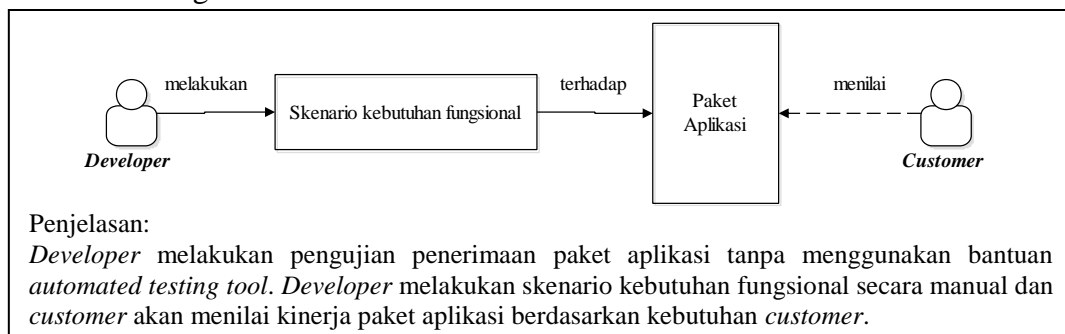
Gambar 3-6. Pengujian integrasi (*top-down*) secara manual

Setelah modul-modul diuji oleh para anggota tim, modul-modul tersebut akan dijadikan paket aplikasi yang berisi *file* siap pakai. Untuk memastikan paket aplikasi tersebut minim dari kesalahan, maka paket aplikasi perlu diuji. Pengujian paket aplikasi yang melibatkan seluruh komponen pengujian, disebut pengujian sistem. Umumnya, pengujian yang dilibatkan dalam pengujian sistem adalah pengujian fungsional perangkat lunak terhadap kebutuhan *customer*. Tim yang tidak menggunakan bantuan *tool* dari *automated testing*, akan memerlukan *effort* yang besar untuk melakukan pengujian tersebut.



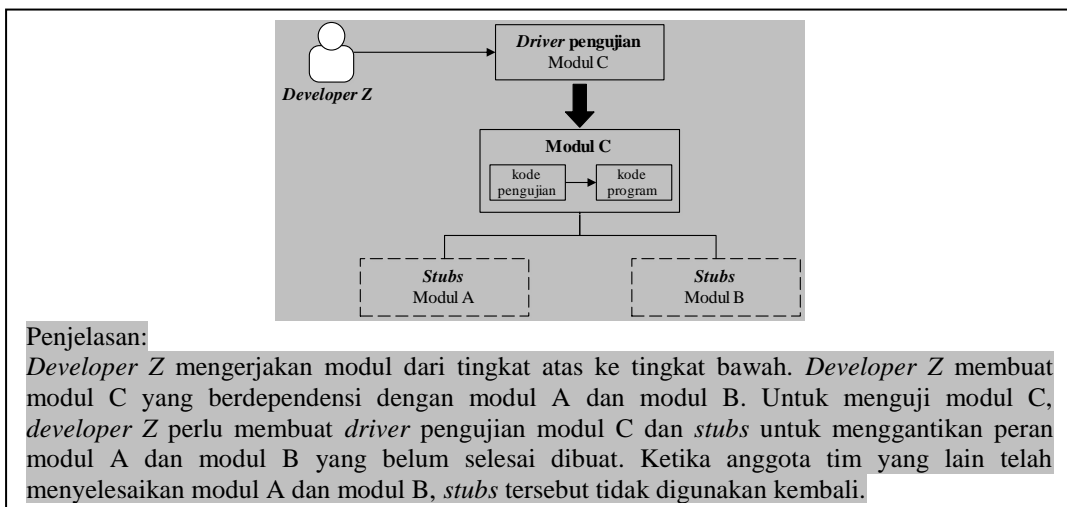
Gambar 3-7. Pengujian sistem secara manual

Paket aplikasi yang telah berhasil dilakukan pengujian sistem, selanjutnya akan di-*deploy* ke *customer environment*. Pada tahap ini, kelayakan paket aplikasi akan dinilai oleh *customer*. Pengujian paket aplikasi yang melibatkan *customer*, disebut pengujian penerimaan. Tim yang tidak menggunakan bantuan *tool* dari *automated testing*, perlu memperagakan pengujian paket aplikasi terhadap kebutuhan fungsional secara manual.



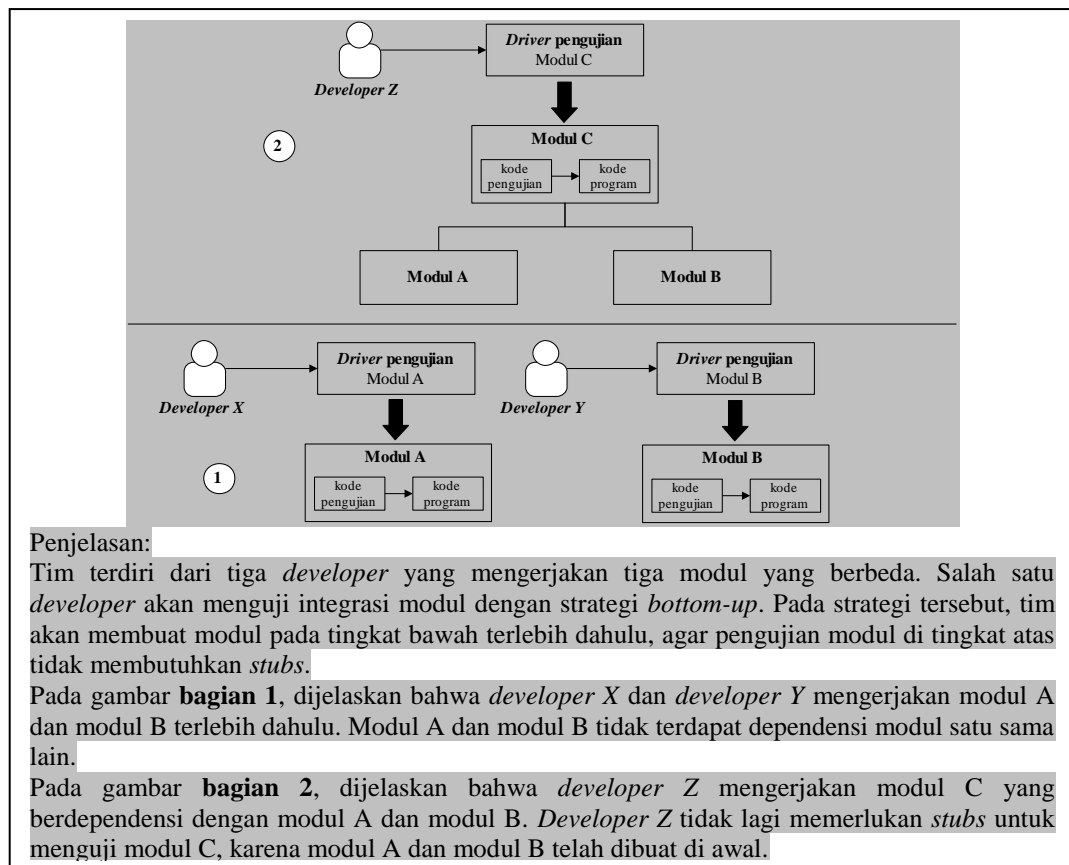
Gambar 3-8. Pengujian penerimaan secara manual

Pada strategi *top-down*, anggota tim menguji modul dari tingkat atas ke modul tingkat bawah. Anggota tim yang menggunakan strategi *top-down*, 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.



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.



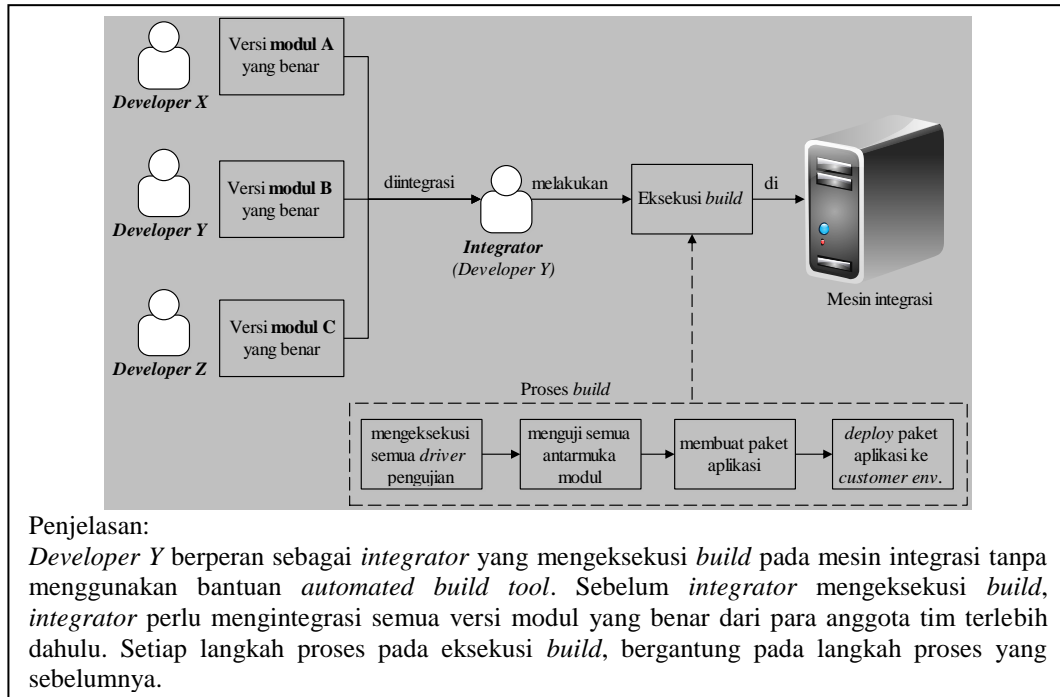
Gambar 3-6. Pengujian integrasi dengan strategi *bottom-up*

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 dan *deploy* paket aplikasi ke *customer environment*. Untuk melakukan kegiatan tersebut, seorang *integrator* perlu mengeksekusi *build*. Umumnya, eksekusi *build* dilakukan oleh *integrator* di mesin integrasi.

Sebelum membuat paket aplikasi, seorang *integrator* perlu mengintegrasikan dan menguji semua versi modul yang benar dari para anggota tim. *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, pengujian paket aplikasi dan *deploy* paket aplikasi ke *customer environment*.

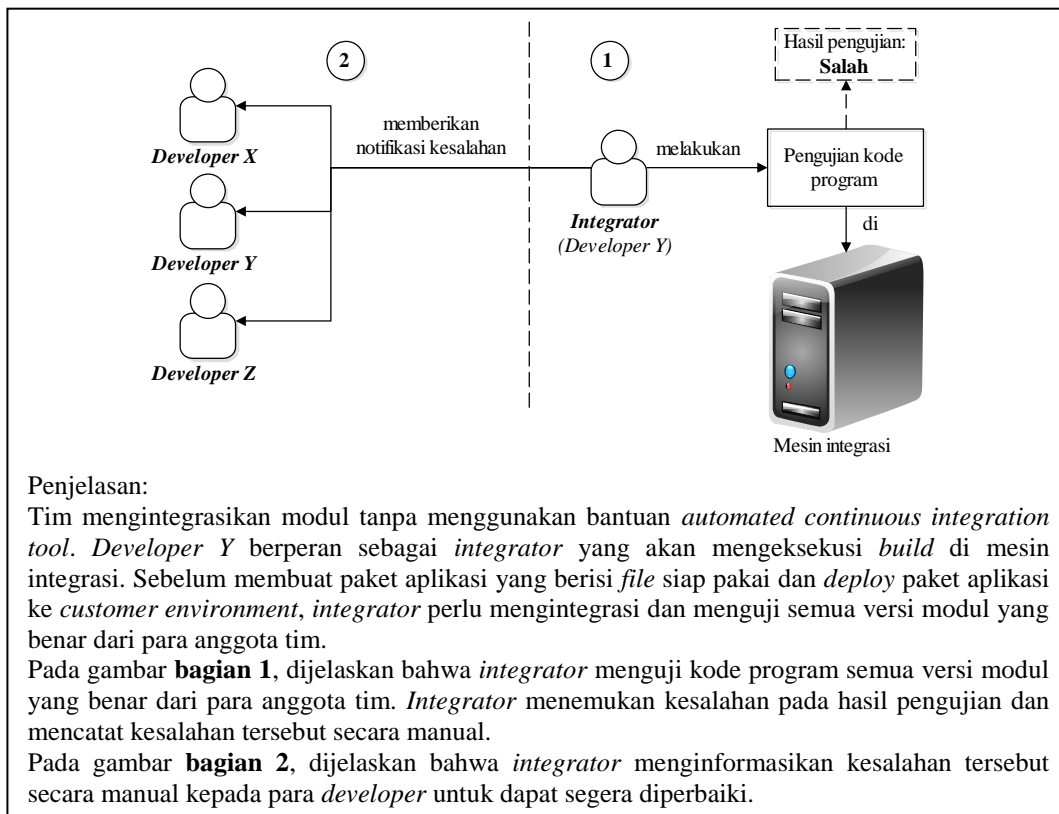
Rangkaian proses *build* tersebut dilakukan *integrator* secara manual dan berulang kali setiap mengintegrasikan modul dari para anggota tim.



Gambar 3-9. Eksekusi *build* dengan cara manual

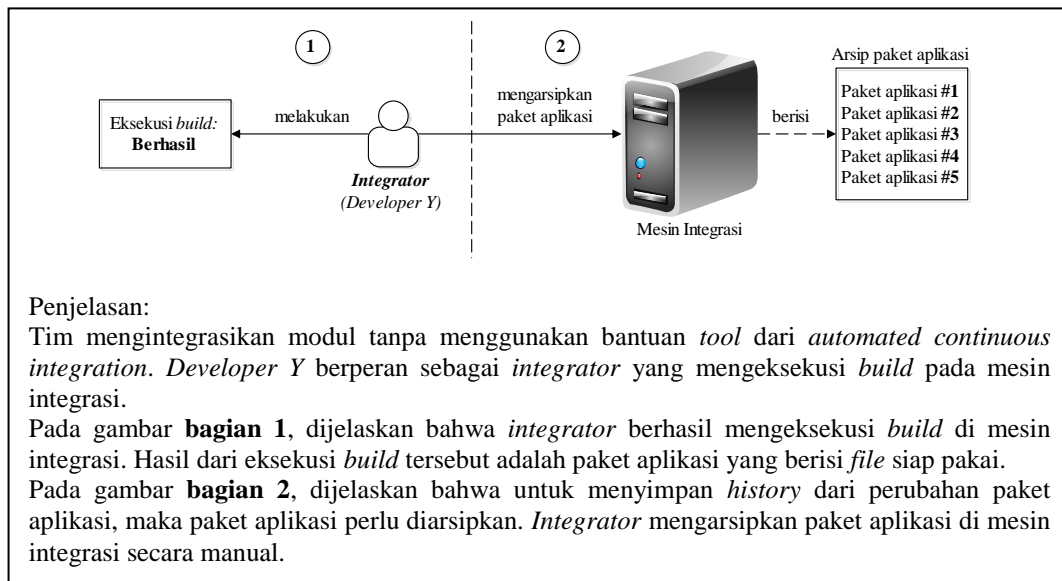
3.1.4. Konsep pengintegrasian modul secara manual

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, dapat minim dari kesalahan. Setelah *integrator* membuat paket aplikasi, *integrator* perlu menguji paket aplikasi tersebut sebelum di-deploy ke *customer environment*. Pengujian paket aplikasi dilakukan *integrator* untuk memastikan bahwa *functional requirement* pada paket aplikasi tersebut dapat dipenuhi. Ketika terjadi kesalahan pada satu atau lebih hasil pengujian, *integrator* perlu menginformasikan kesalahan tersebut kepada para anggota tim untuk dapat segera diperbaiki.



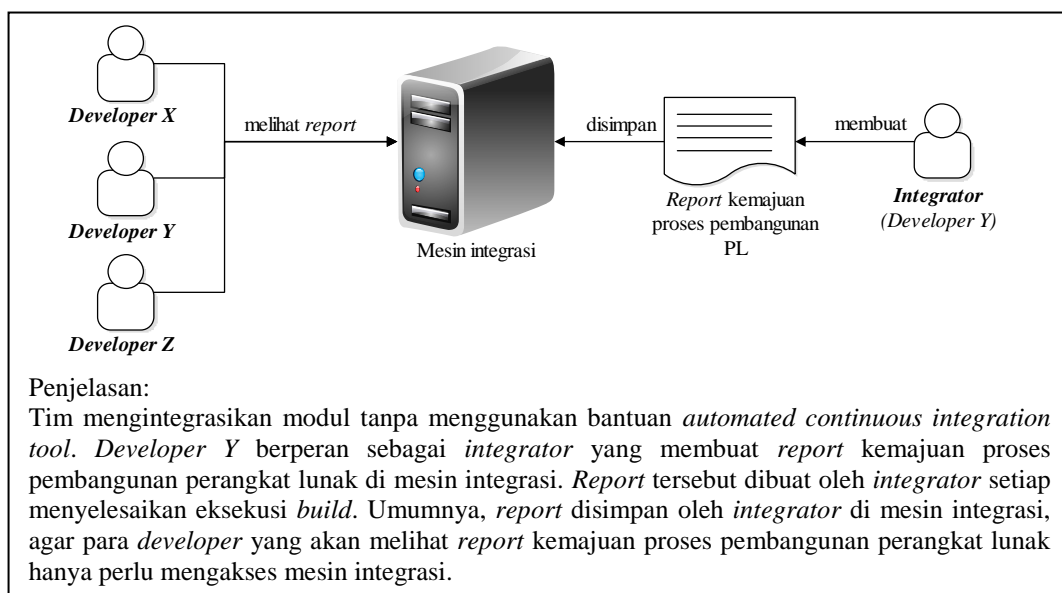
Gambar 3-10. Pemberian notifikasi kesalahan secara manual oleh *integrator*

Integrasi modul yang telah lulus dari pengujian, akan dijadikan paket aplikasi yang berisi *file* siap pakai, diuji kembali 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.



Gambar 3-11. 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.



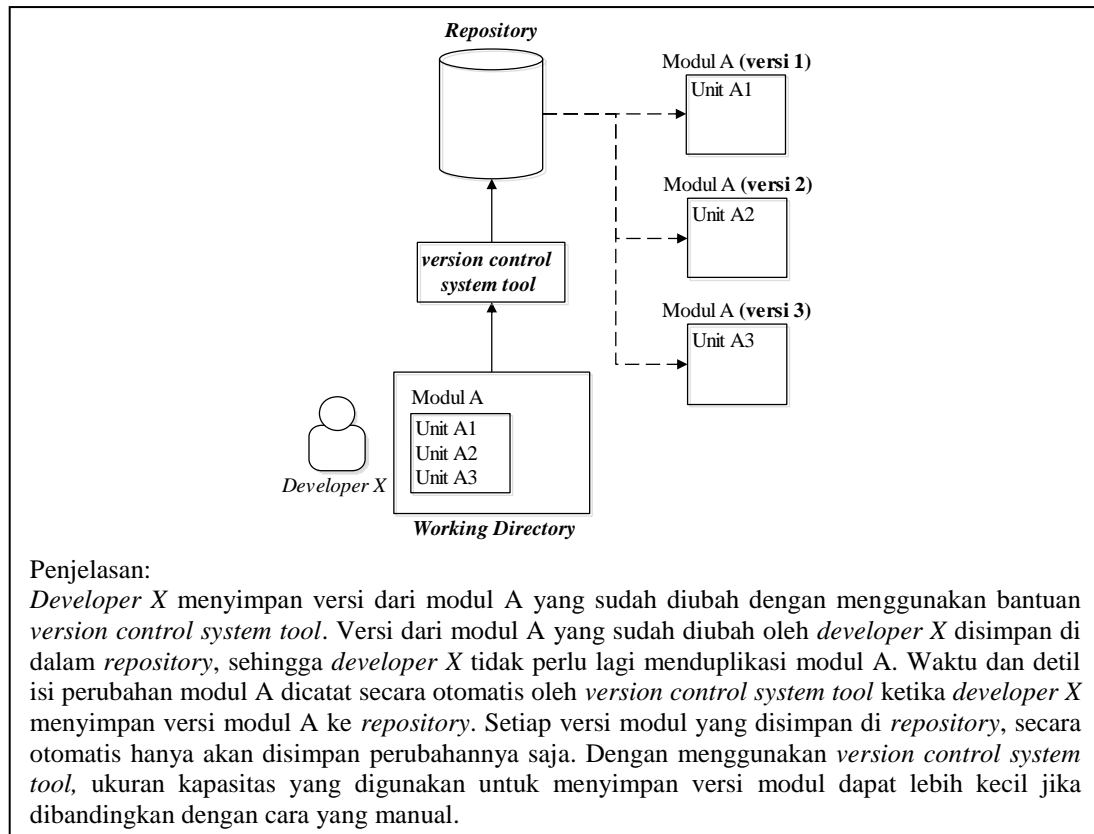
Gambar 3-12. Pembuatan *report* kemajuan proses pembangunan perangkat lunak oleh *integrator*

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.

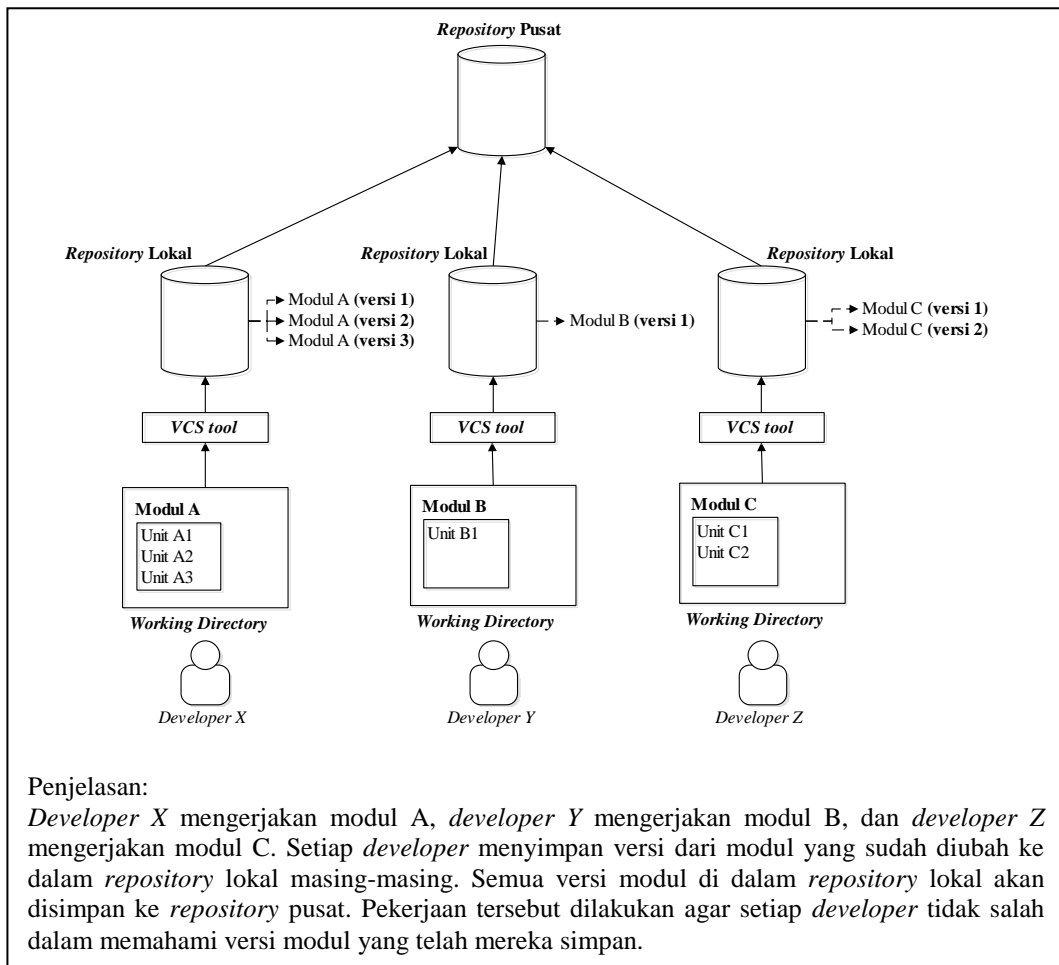
3.2.1. Konsep penyimpanan versi dengan *version control system tool*

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 detail perubahan yang dilakukan terhadap modul secara manual, karena *tool* dari *version control system* akan mencatat waktu dan detail isi perubahan secara otomatis ketika mereka menyimpan versi modul ke *repository*.



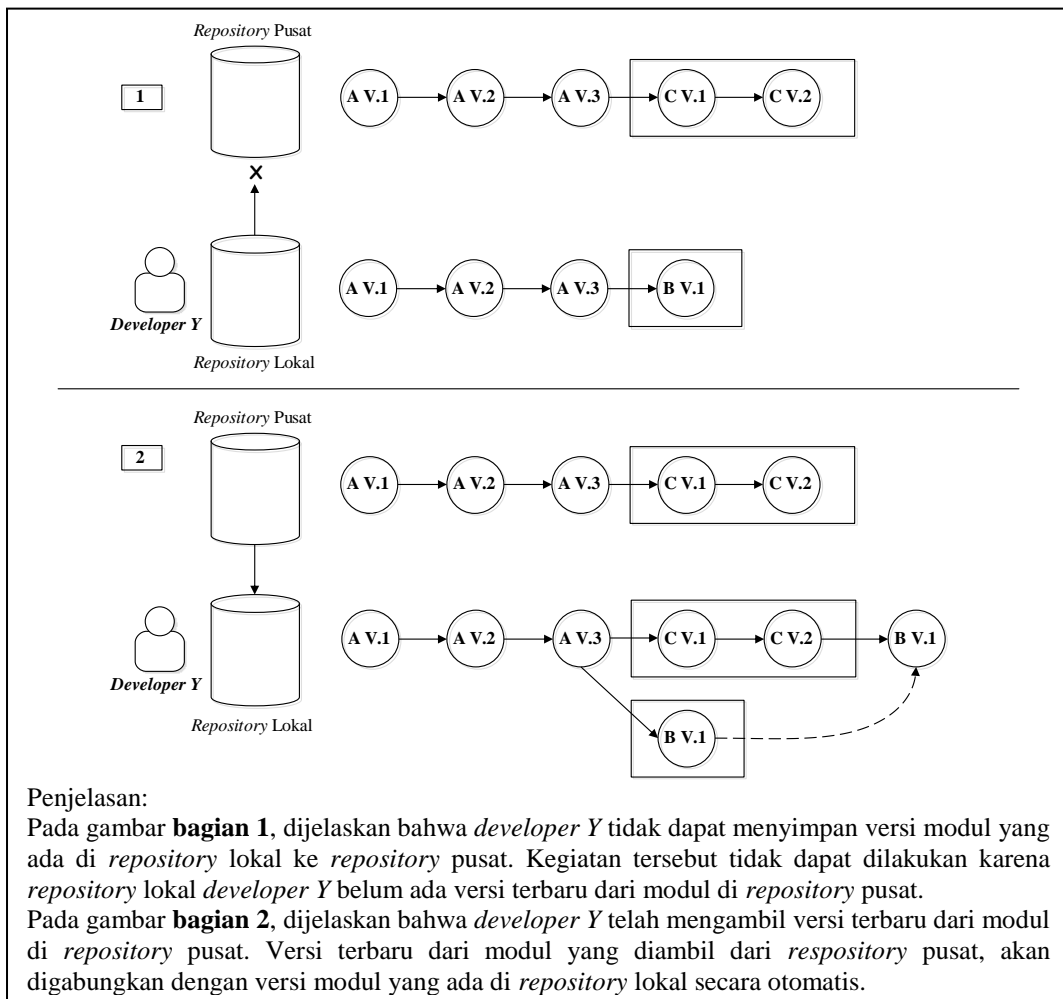
Gambar 3-13. Penyimpanan versi modul ke dalam *repository*

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*.



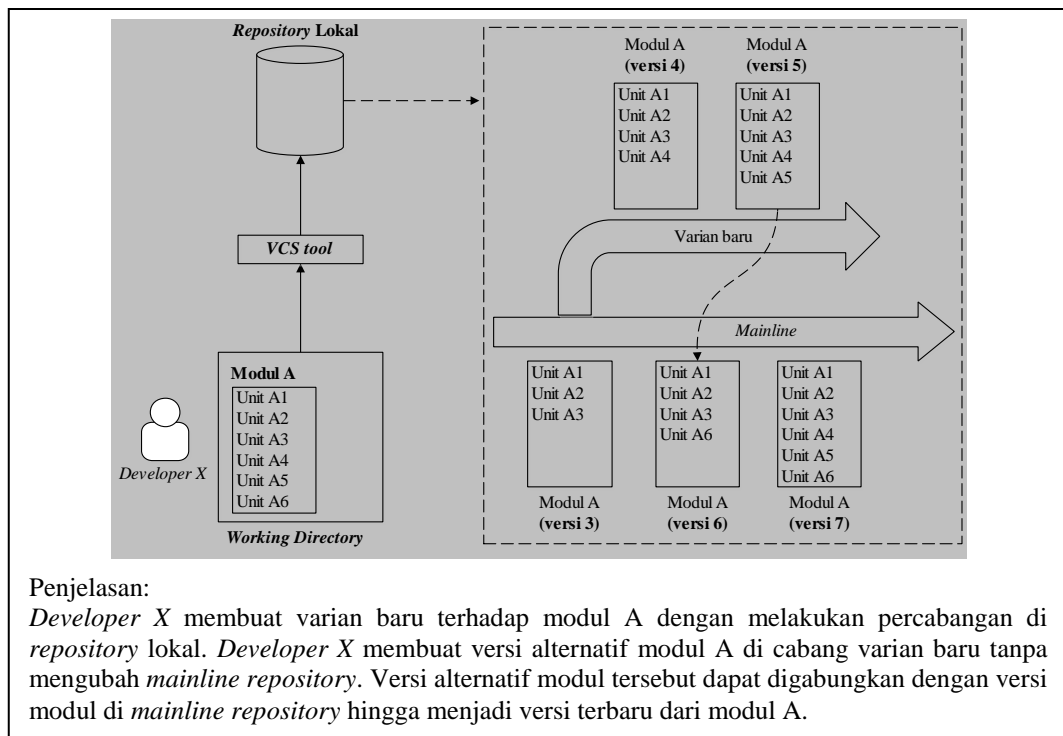
Gambar 3-14. *Centralized workflow*

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. Semua 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*, setiap anggota tim dapat selalu memperbarui semua versi modul dari anggota yang lain tanpa harus menyimpan duplikasi versi modul secara manual.



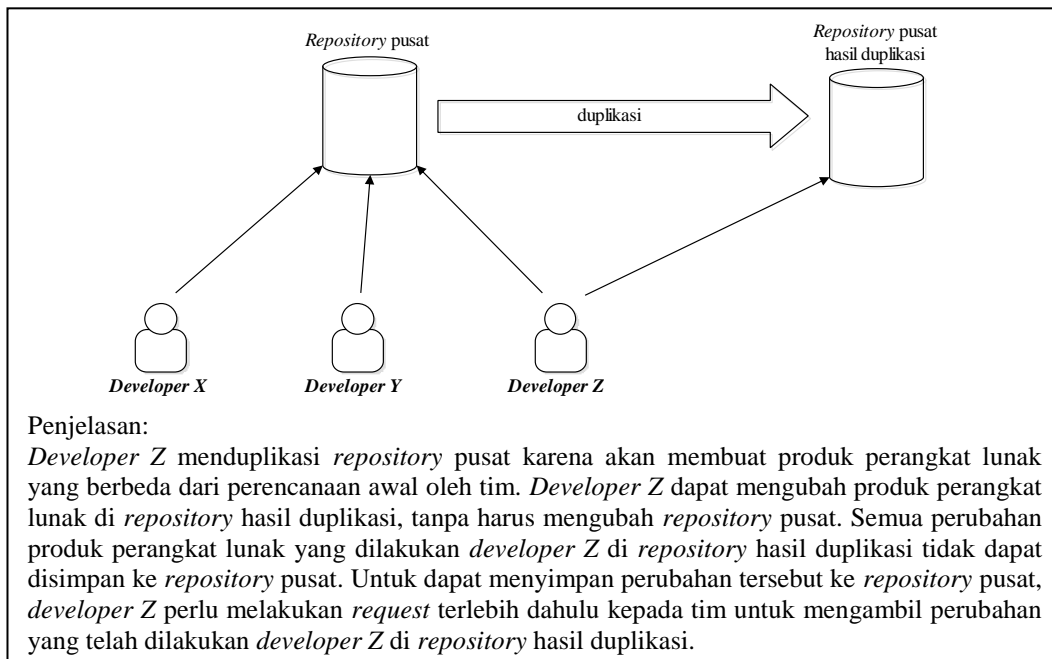
Gambar 3-15. Penggabungan versi modul

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*.



Gambar 3-16. Percabangan versi modul

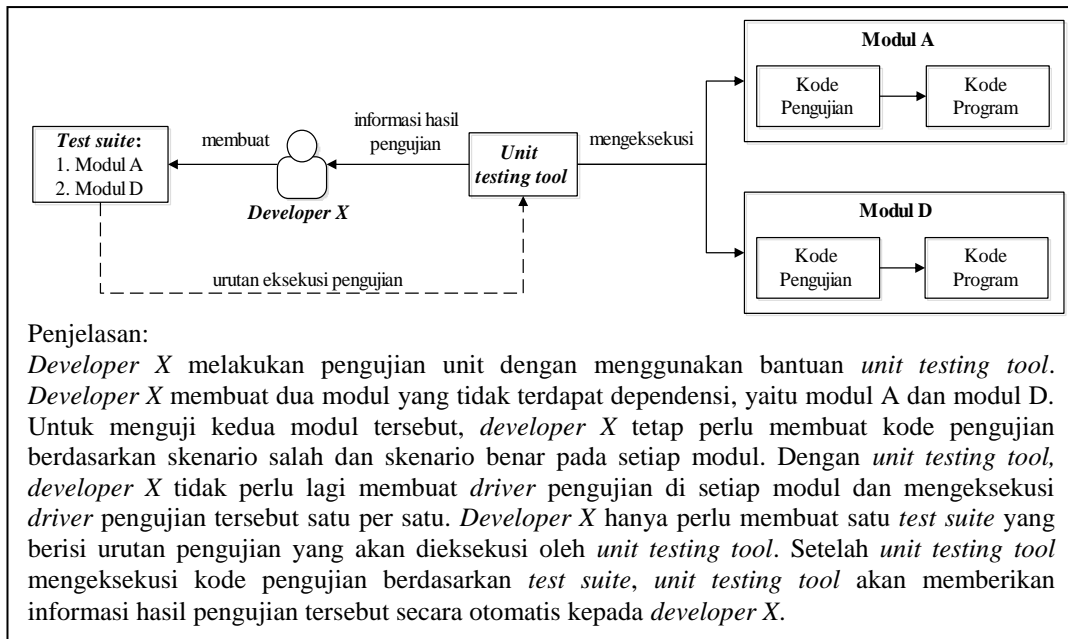
Dengan menggunakan *tool* dari *version control system*, anggota tim yang akan membuat produk perangkat lunak yang berbeda dari perencanaan awal oleh tim, dapat menduplikasi *repository* pusat. Anggota tim tersebut dapat mengubah produk perangkat lunak pada *repository* hasil duplikasi, tanpa harus mengubah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



Gambar 3-17. Penduplikasian *repository* pusat

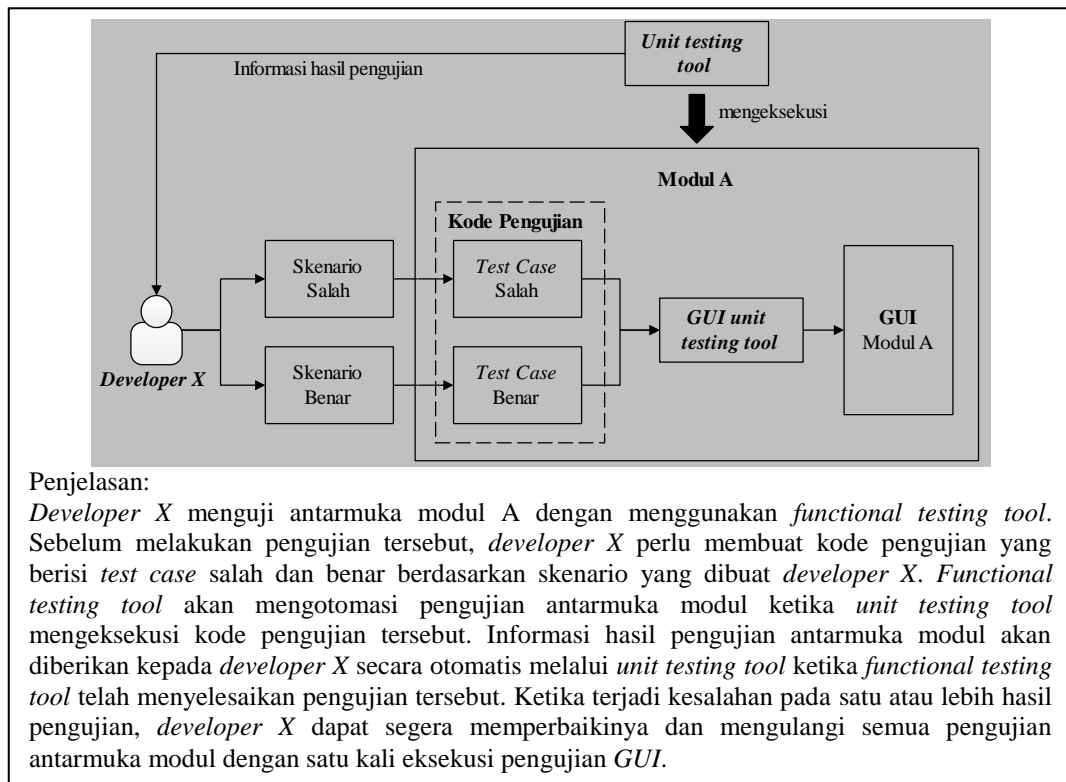
3.2.2. Konsep pengujian kode program dengan *automated testing tool*

Pada sub bab ini akan dijelaskan tentang konsep pengujian kode program pada praktik *continuous integration* dengan menggunakan bantuan *automated testing tool*. *Tool* yang digunakan pada praktik *automated testing* mencakup *unit testing tool* dan *functional testing tool*. Tim yang telah menggunakan *unit testing tool*, tidak perlu lagi membuat *driver* pengujian di setiap kode pengujian, karena *unit testing tool* secara otomatis dapat berperan sebagai *driver* pengujian. Dengan *unit testing tool*, para anggota tim dapat membuat *test suite* atau rangkaian pengujian yang akan dieksekusi oleh *unit testing tool* secara otomatis, sehingga mereka tidak perlu lagi mengeksekusi semua kode pengujian secara satu per satu. Selain itu, para anggota tim dapat memperoleh *feedback* terhadap pengujian unit dengan cepat, karena *unit testing tool* akan memberikan informasi hasil pengujian tersebut setelah mengeksekusi kode pengujian. Ketika terjadi kesalahan pada satu atau lebih hasil pengujian unit, mereka dapat segera memperbaiki kesalahan tersebut dan mengulangi semua eksekusi kode pengujian dengan hanya satu kali eksekusi *test suite*.



Gambar 3-18. Pengujian unit dengan menggunakan *unit testing tool*

Para anggota tim yang telah menggunakan *functional testing tool* tidak lagi melakukan skenario salah dan skenario benar terhadap komponen *GUI* secara manual. Semua skenario tersebut dapat diotomasi oleh *functional testing tool*. Untuk mengotomasi pengujian unit pada komponen *GUI*, para anggota tim perlu membuat kode pengujian terlebih dahulu. Para anggota tim yang telah membuat kode pengujian unit pada komponen *GUI*, dapat melakukan pengujian antarmuka modul secara berulang kali tanpa mengeluarkan *effort* yang besar.



Gambar 3-19. Pengujian unit *GUI* dengan menggunakan *functional testing tool*

Dengan kedua *tool* tersebut, pengujian integrasi dapat lebih efisien. Para anggota tim tidak lagi membuat *driver* pengujian pada setiap modul. Selain itu para anggota tim tidak lagi mengeksekusi *driver* pengujian dan antarmuka modul satu per satu. Pengujian integrasi dapat dilakukan oleh para anggota tim dengan satu kali eksekusi pengujian. Untuk mengotomasi pengujian integrasi, anggota tim yang menguji modul pada tingkat atas hanya perlu mengatur urutan pengujian pada *test suite*. Kedua *tool* tersebut akan menguji integrasi modul berdasarkan urutan pada *test suite*.

[GAMBAR]

Gambar 3-20. Pengujian integrasi dengan menggunakan *unit* dan *functional testing tool*

Pengujian sistem pada paket aplikasi juga dapat diotomasi. Umumnya pengujian yang dicakup pengujian sistem adalah pengujian kebutuhan fungsional terhadap paket aplikasi. Dengan *functional testing tool*, pengujian kebutuhan fungsional dapat dilakukan secara otomatis. Untuk mengotomasi pengujian paket

aplikasi terhadap kebutuhan fungsional, anggota tim perlu merekam aktifitas atau skenario penggunaan aplikasi terhadap kebutuhan fungsional. Rekaman tersebut akan digunakan *functional testing tool* untuk mengotomasi pengujian fungsional secara berulang kali, sehingga *effort* yang dikeluarkan anggota tim untuk menguji paket aplikasi lebih sedikit.

[GAMBAR]

Gambar 3-21. Pengujian sistem dengan menggunakan *functional testing tool*

Proses pada pengujian penerimaan juga dapat diotomasi. Dengan *functional testing tool*, tim dapat memperagakan penggunaan paket aplikasi terhadap kebutuhan fungsional secara otomatis. Selain itu *functional testing tool* dapat memberikan sehingga tim dapat dimudahkan dalam membuat dokumen penerimaan paket aplikasi ke customer.

[GAMBAR]

Gambar 3-22. Pengujian penerimaan dengan menggunakan *functional testing tool*

Pada pengujian integrasi dengan strategi *top-down* maupun *bottom-up*, para anggota tim

Dengan kedua tool pengujian tersebut, para anggota tim yang menguji modul dengan dependensi terhadap modul yang lain dapat mengurutkan pengujian dari modul tingkat bawah ke modul tingkat atas.

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 N-NN. Pengujian integrasi dengan bantuan *tool integration testing*

3.2.3. Konsep eksekusi *build* dengan *automated build tool*

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

Build script yang dieksekusi oleh *automated build tool* di mesin lokal setiap anggota tim, disebut *private build*. Untuk menyamakan alur kerja setiap anggota tim, tim perlu menentukan *target* dan *task* yang akan dilakukan oleh *automated build tool* di 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 eksekusi pengujian kode program dan penyimpanan versi modul yang sudah diubah ke dalam *repository* lokal.

[GAMBAR]

Gambar 3-23. Eksekusi *private build*

Untuk mengotomasikan semua kegiatan yang akan dilakukan *integrator* di mesin integrasi, tim perlu menentukan *target* dan *task* pada *build script* yang akan dieksekusi oleh *automated build tool*. *Build script* yang dieksekusi oleh *automated build tool* di mesin integrasi untuk membuat paket aplikasi, disebut *integration build*. Umumnya, *target* pada *integration build* mencakup eksekusi pengujian kode program dan pembuatan paket aplikasi.

[GAMBAR]

Gambar 3-24. Eksekusi *integration build*

Paket aplikasi hasil *integration build* tersebut dapat diuji dan di-*deploy* ke *customer environment* secara otomatis. Untuk mengotomasikan kegiatan tersebut, tim perlu menentukan *target* dan *task* pada *build script* yang akan dieksekusi oleh *automated build tool*. *Build script* yang dieksekusi oleh *automated build tool* di mesin integrasi untuk *deploying* paket aplikasi, disebut *release build*. Umumnya target pada *release build* mencakup pengujian paket aplikasi dan *deploy* paket aplikasi ke *customer environment*.

[GAMBAR]

Gambar 3-25. Eksekusi *release build*

3.2.4. Konsep pengintegrasian modul dengan *automated continuous integration tool*

Pada umumnya, tim yang tidak menggunakan *tool* dari *automated continuous integration* akan membutuhkan seorang *integrator* untuk menggunakan *automated build tool* dalam mengeksekusi *integration build* dan *release build*. *Integrator* akan mengambil versi modul yang benar dari *repository* pusat, kemudian mengeksekusi *integration build* dan *release build*. Dengan menggunakan *tool* dari *automated continuous integration* pada mesin integrasi, tim tidak lagi memerlukan seorang *integrator* untuk menggunakan *automated build tool*, karena penggunaan *automated build tool* dapat diotomasi dan dijadwalkan. *Automated continuous integration* juga dapat mengambil versi modul yang benar dari *repository* pusat secara otomatis berdasarkan jadwal tersebut.

[GAMBAR]

Gambar 3-26. Penjadwalan eksekusi *build script* pada mesin integrasi

Pada setiap eksekusi *integration build* dan *release build*, mesin integrasi akan menguji kode program dan paket aplikasi 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*, 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 3-27. Notifikasi kesalahan secara otomatis dari mesin integrasi

Dengan menggunakan *automated continuous integration tool*, 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 3-28. 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 di mesin integrasi.

[GAMBAR]

Gambar 3-29. *Report* kemajuan proses pembangunan perangkat lunak secara otomatis