BAB IV
STUDI KASUS *CONTINUOUS INTEGRATION* SECARA MANUAL DAN MENGGUNAKAN *TOOLSET*

Pada bab ini akan dijelaskan tentang penerapan konsep pembangunan perangkat lunak dengan *continuous integration* (CI) yang dilakukan secara manual dan menggunakan *toolset*. Penerapan konsep tersebut dilakukan untuk menunjukkan perbedaan proses dari keduanya. Praktik yang mencakup penerapan CI secara manual yaitu praktik penyimpanan versi secara manual, praktik pengujian kode program secara manual, praktik eksekusi *build* secara manual, dan praktik integrasi modul secara manual. Sedangkan praktik yang mencakup penerapan CI dengan menggunakan *toolset* yaitu praktik penyimpanan versi dengan *version control system* (VCS) *tools*, praktik pengujian kode program dengan *automated testing tools*, praktik eksekusi *build* dengan *automated build tools* dan praktik integrasi modul dengan *automated CI tools*.

Studi kasus yang digunakan untuk menerapkan konsep pembangunan perangkat lunak dengan CI secara manual dan menggunakan *toolset* adalah aplikasi rekam medis berbasis *java desktop*, yang bernama `medrecapp`. Pembangunan aplikasi tersebut dilakukan oleh satu tim yang terdiri dari tiga orang *developers*, yaitu Fachrul, Hernawati dan Yuanita.

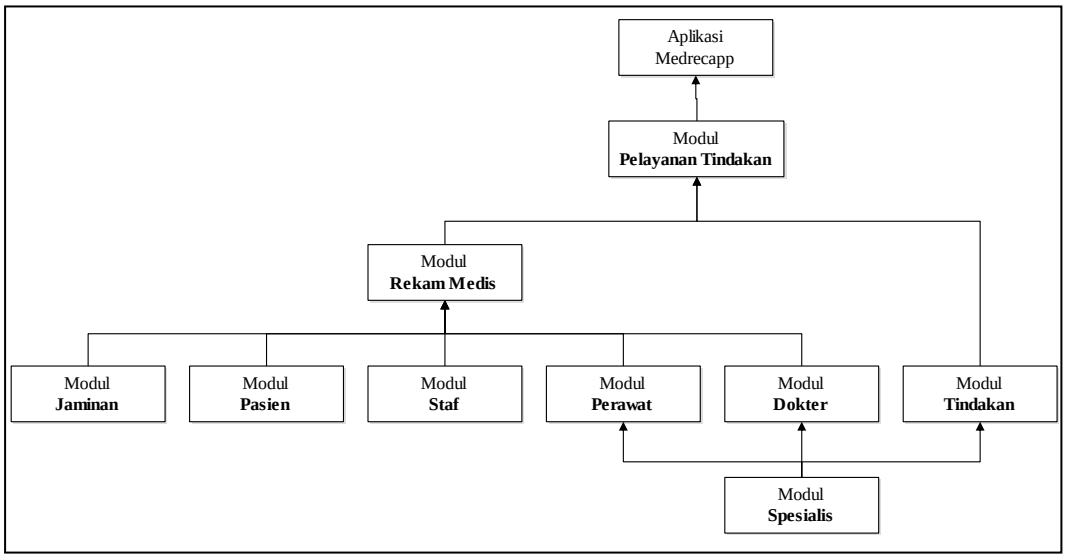
4.1. Modul aplikasi `medrecapp`

Pada sub bab ini akan dijelaskan tentang pembagian modul aplikasi `medrecapp` untuk mendukung penerapan konsep CI. Pembagian modul tersebut dilakukan tim untuk meningkatkan produktivitas pekerjaan setiap *developer* (lihat **tabel 4-1**). Setiap modul memiliki perbedaan area fungsional dengan modul yang lain. Para *developer* akan lebih mudah membuat kode program berdasarkan pada satu area fungsional.

Tabel 4-1. Pembagian modul aplikasi `medrecapp`

No.	Daftar modul	*Developers*		
		Fachrul	Hernawati	Yuanita
1	`Spesialis`	✓	-	-
2	`Jaminan`	✓	-	-
3	`Pasien`	-	✓	-
4	`Staf`	-	-	✓
5	`Perawat`	✓	-	-
6	`Dokter`	-	-	✓
7	`Tindakan`	-	✓	-
8	`Rekam medis`	-	-	✓
9	`Pelayanan tindakan`	-	✓	-

Pada setiap modul aplikasi `medrecapp`, terdapat kelas `DAO` (*Data Access Object*), kelas *`entity`*, kelas `GUI`, kelas *`interface`*, kelas *`service`*, dan kelas `tabel model`. Modul aplikasi `medrecapp` terdiri dari sembilan modul yaitu modul `spesialis`, modul `jaminan`, modul `pasien`, modul `staf`, modul `perawat`, modul `dokter`, modul `tindakan`, modul `rekam medis`, dan modul `pelayanan tindakan`. Dependensi antar modul aplikasi `medrecapp` dapat dilihat pada **gambar 4-1**.



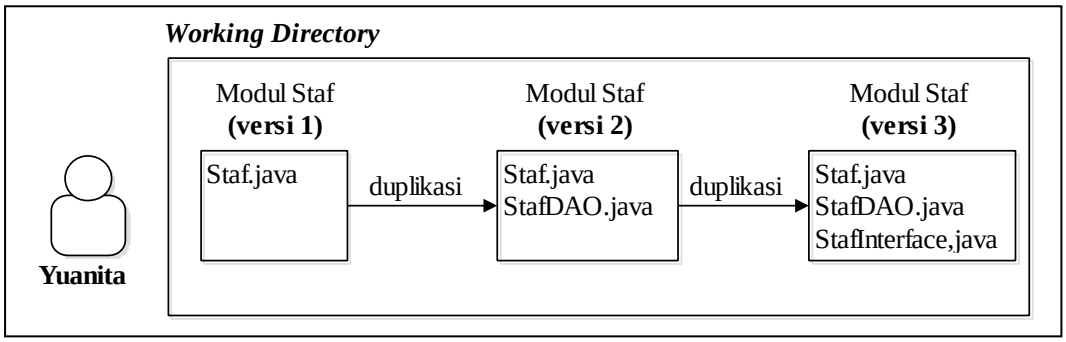
Gambar 4-1. Modul pada aplikasi `medrecapp`

4.2. Praktik pembangunan aplikasi `medrecapp` secara manual

Pada sub bab ini akan dijelaskan tentang praktik pembangunan aplikasi `medrecapp` dengan CI tanpa menggunakan bantuan *toolset*. Praktik manual tersebut mencakup praktik penyimpanan versi secara manual, praktik pengujian kode program secara manual, praktik eksekusi *build* secara manual dan praktik integrasi modul secara manual.

4.2.1. Praktik penyimpanan versi secara manual

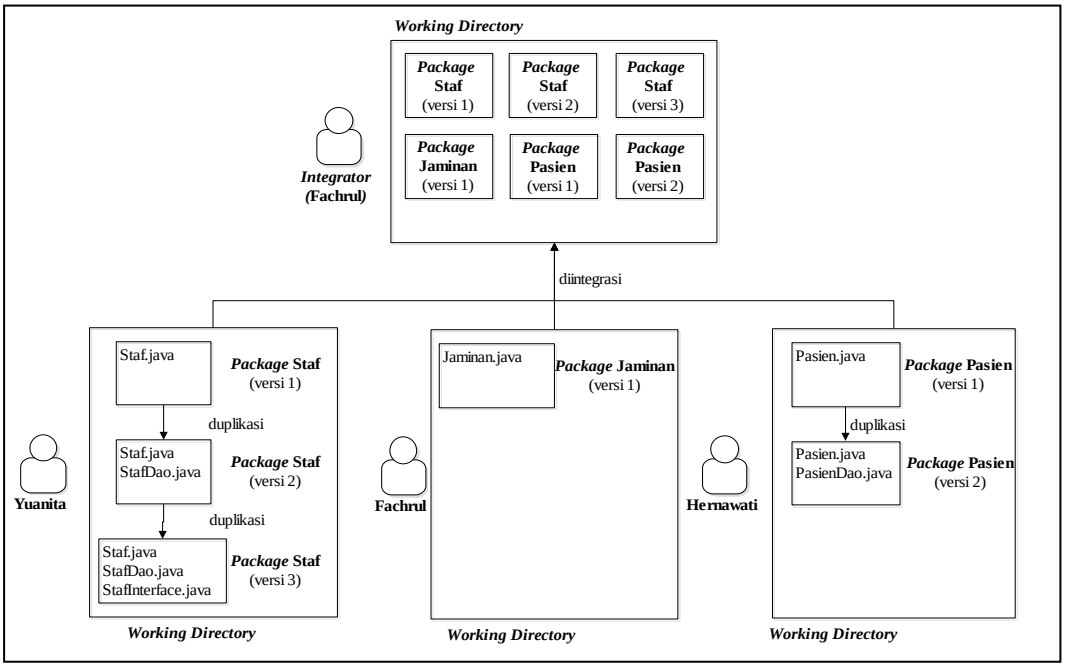
Penyimpanan versi dilakukan oleh para *developer* untuk menyimpan *history* dari setiap perubahan modul. *Developer* yang tidak menggunakan bantuan VCS *tools* umumnya akan menduplikasi modul sebelum mengubah modul tersebut. Hasil duplikasi modul akan digunakan oleh tim sebagai *backup* untuk dapat kembali ke modul yang belum diubah. Untuk membedakan hasil dari setiap modul yang telah diduplikasi, tim perlu melakukan penamaan versi dan menambahkan informasi tentang detil perubahan yang telah dilakukan pada modul tersebut.



Gambar 4-2. Praktik penyimpanan versi dengan cara manual

Pada **Gambar 4-2** dijelaskan bahwa Yuanita akan membuat modul `staf` tanpa menggunakan bantuan VCS *tools*. Penyimpanan versi modul `staf` dilakukan Yuanita dengan cara menduplikasi modul `staf` terlebih dahulu sebelum mengubahnya. Hasil dari duplikasi modul `staf` akan digunakan Yuanita sebagai *backup* untuk dapat kembali ke versi modul sebelumnya. Yuanita akan melakukan penamaan versi modul `staf` untuk dapat membedakan setiap hasil duplikasi yang dilakukan dan mencatat setiap perubahan yang terjadi pada setiap modul.

Umumnya, setiap versi modul yang dibuat para *developer* akan disimpan di tempat penyimpanan versi terpusat. Kegiatan tersebut dilakukan agar tim tidak salah dalam memahami versi modul yang telah dibuat. Tim yang tidak menggunakan VCS *tools*, umumnya 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.



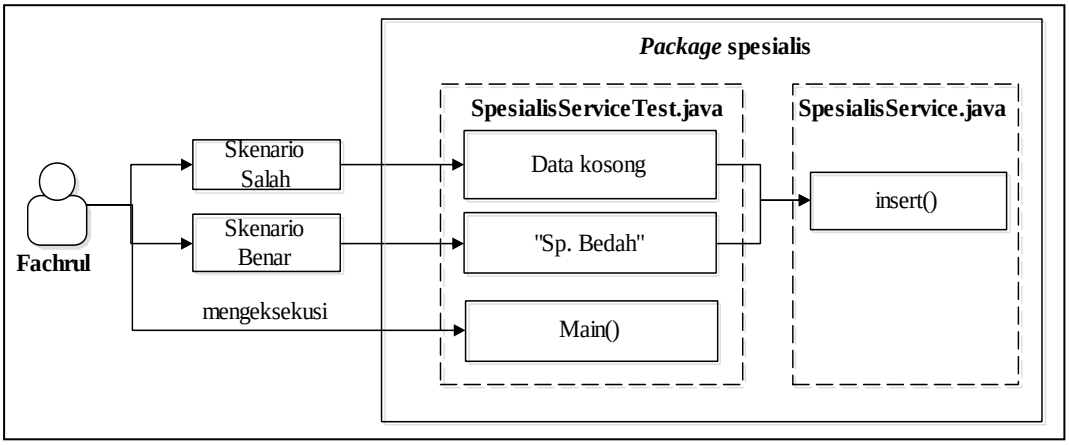
Gambar 4-3. Penggabungan versi modul secara manual

Pada **Gambar 4-3** dijelaskan bahwa Fachrul berperan sebagai *integrator* yang akan mengelola versi modul di tempat penyimpanan versi modul terpusat. Setiap versi modul yang dibuat Yuanita, Fachrul dan Herna akan disimpan di tempat penyimpanan versi terpusat agar mereka tidak salah dalam memahami versi modul yang telah dibuat. Selain itu, dengan penyimpanan versi modul terpusat *developer* dapat dimudahkan untuk mengambil versi modul yang benar dari *developer* yang lain. Semua versi modul di tempat penyimpanan versi terpusat akan dikelola oleh Fachrul sebelum dijadikan paket aplikasi yang berisi *file* siap pakai. Ukuran kapasitas tempat penyimpanan terpusat akan semakin membesar seiring dengan bertambahnya jumlah versi modul yang disimpan.

4.2.2. Praktik pengujian kode program secara manual

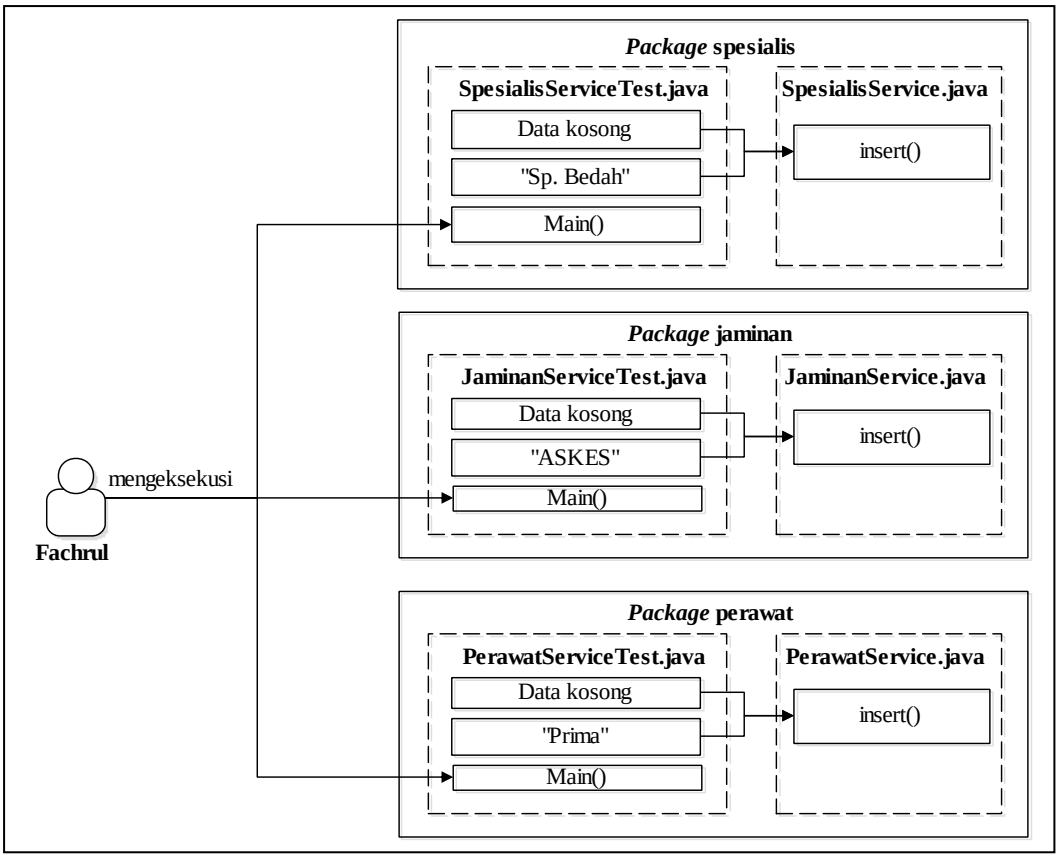
Modul yang dikerjakan setiap *developer* akan ditambahi kelas-kelas. Setiap kelas yang ditambahi ke dalam modul harus diuji. Pengujian unit dilakukan para *developer* untuk memastikan bahwa *functional requirement* dari modul yang dibuat dapat dieksekusi serta minim dari kesalahan.

Untuk menguji suatu kelas, tim memerlukan kelas pengujian. Pada setiap kelas pengujian, *developer* akan menambahi satu atau lebih *test case*. *Developer* yang tidak menggunakan bantuan *automated testing tools*, perlu menambahkan *main method* pada setiap kelas pengujian, agar kelas pengujian tersebut dapat dieksekusi.



Gambar 4-4. Pengujian unit secara manual oleh Fachrul

Fachrul menguji kelas `service` pada modul `spesialis`. Untuk menguji kelas tersebut, Fachrul perlu membuat kelas pengujian. Kelas pengujian tersebut berisi *test case* salah dan benar terhadap setiap *method* pada kelas yang akan diuji. Pengujian unit dilakukan Fachrul dengan cara melakukan *trigger* eksekusi kelas pengujian. Ketika Fachrul membuat tiga kelas pengujian, maka Fachrul perlu menambahkan *main method* pada tiga kelas pengujian tersebut dan melakukan tiga kali *trigger* eksekusi kelas pengujian satu per satu.



Gambar 4-5. Pengujian tiga kelas secara manual oleh Fachrul

Untuk menguji kelas `GUI`, diperlukan suatu simulasi interaktif terhadap komponen GUI. *Developer* yang tidak menggunakan bantuan *functional testing tools*, perlu melakukan simulasi skenario salah dan skenario benar secara manual dan berulang kali terhadap suatu antarmuka modul.

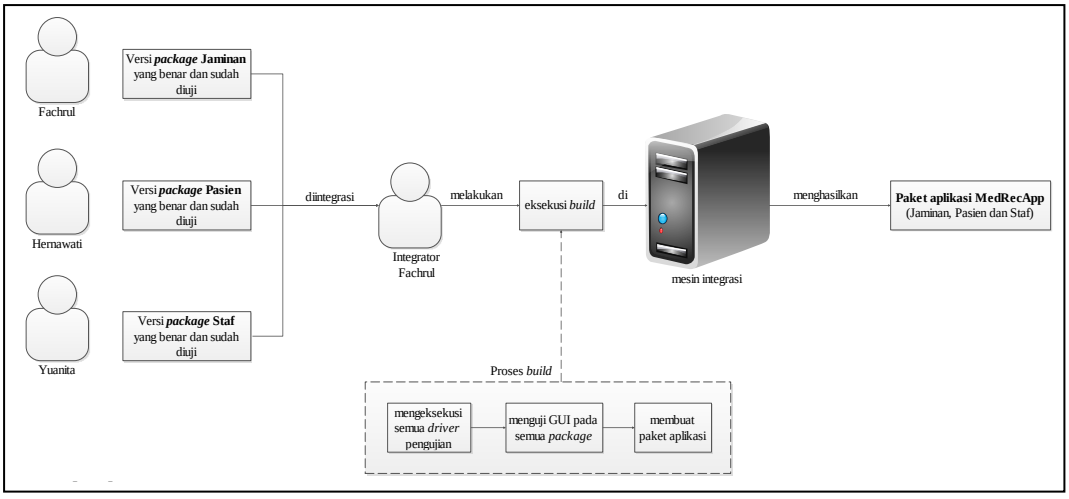
Fachrul menguji kelas `GUI` pada modul `spesialis`. Untuk menguji kelas `GUI` tersebut, Fachrul perlu melakukan simulasi interaktif terhadap antarmuka modul `spesialis`. Pengujian kelas `GUI` dilakukan Fachrul dengan cara melakukan *trigger* eksekusi kelas `GUI` dan mensimulasikan skenario salah dan skenario benar. Ketika terdapat kesalahan pada satu atau lebih hasil pengujian, Fachrul harus segera memperbaiki kesalahan tersebut dan mengulangi semua skenario salah dan benar dari awal.

Modul tingkat atas yang tergantung kepada modul tingkat bawah perlu dilakukan pengujian integrasi. Pengujian tersebut dilakukan untuk menguji kombinasi antar modul dan menampilkan kesalahan pada interaksi antar kelas yang terintegrasi.

Yuanita membuat modul `rekam medis` yang tergantung kepada lima modul lain yaitu modul `jaminan`, modul `pasien`, modul `staf`, modul `perawat` dan modul `dokter`. Sebelum Yuanita menguji modul `rekam medis`, maka Yuanita perlu menguji modul tingkat bawah terlebih dahulu. Yuanita menguji integrasi modul `rekam medis` dengan melakukan *trigger* eksekusi semua *driver* pengujian ditingkat bawah satu per satu dan mensimulasikan semua skenario salah dan benar terhadap modul di tingkat bawah. Kegiatan tersebut dilakukan Yuanita untuk dapat mengetahui penyebab kesalahan yang mungkin terjadi pada pengujian modul `rekam medis`.

4.2.3. Praktik eksekusi *build* secara manual

Modul hasil pekerjaan dari para *developer* akan digabungkan oleh *integrator*. *Developer* yang berperan sebagai *integrator* adalah Fachrul. Berdasarkan pembagian pekerjaan pembangunan aplikasi `medrecapp`, Fachrul akan membuat modul `Jaminan`, Hernawati akan membuat modul `Pasien` dan Yuanita akan membuat modul `Staf`.



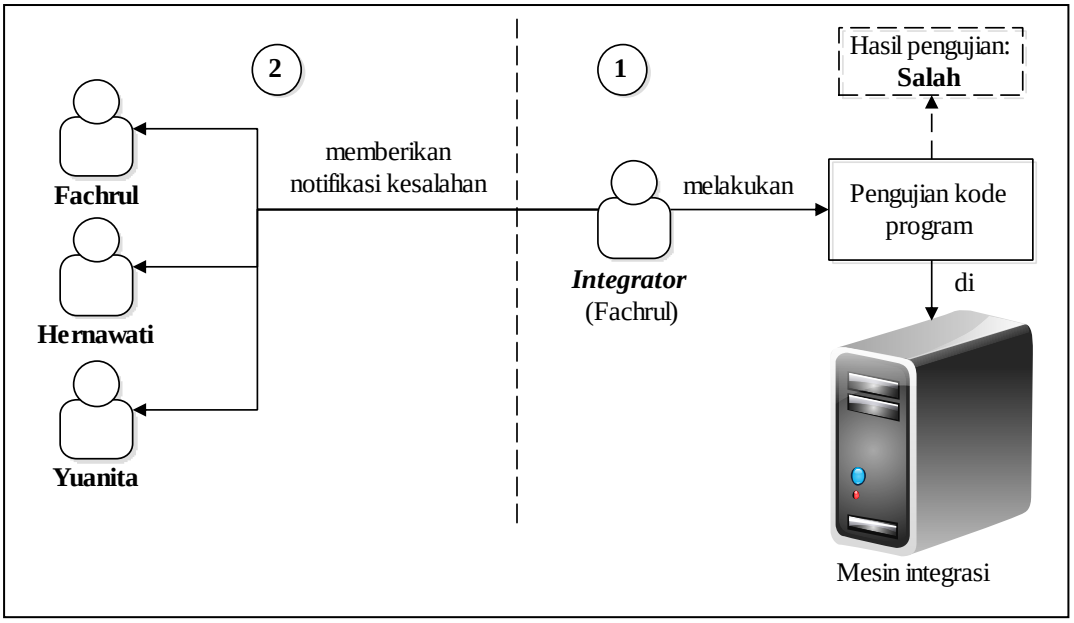
Gambar 4-6. Eksekusi *build* dengan cara manual

Pada **Gambar 4-6**, dijelaskan bahwa Fachrul akan menggabungkan ketiga modul tersebut dan melakukan serangkaian *trigger* eksekusi *build*. *Trigger* eksekusi *build* yang dilakukan Fachrul dimulai dari *trigger* eksekusi semua *driver* pengujian, *trigger* eksekusi kode program untuk menguji GUI semua modul dan *trigger* eksekusi pembuatan paket aplikasi. Hasil akhir dari proses *build* perangkat

lunak tersebut adalah paket aplikasi `medrecapp` yang terdiri dari modul `Jaminan`, `Pasien` dan `Staf`.

4.2.4. Praktik integrasi modul secara manual

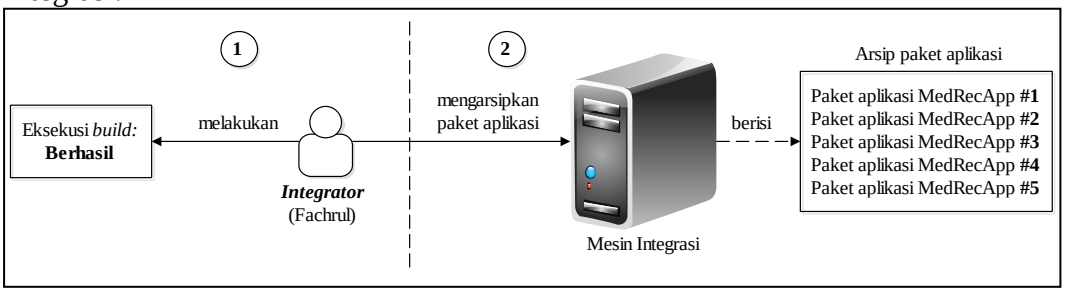
Integrasi modul akan dilakukan oleh seorang *integrator* di mesin integrasi. Pada proses eksekusi *build*, *integrator* akan melakukan *trigger* eksekusi semua *driver* pengujian dan *trigger* eksekusi kode program untuk menguji GUI semua modul yang telah dibuat setiap *developer*. Pengujian tersebut dilakukan untuk memastikan bahwa paket aplikasi yang akan dibuat dapat minim dari kesalahan. Ketika terjadi kesalahan pada satu atau lebih hasil pengujian, *integrator* perlu menginformasikan kesalahan tersebut kepada setiap anggota tim untuk dapat segera diperbaiki.



Gambar 4-7. Pemberian notifikasi kesalahan secara manual oleh *integrator*

Pada **Gambar 4-7** dijelaskan bahwa integrasi modul dilakukan oleh *integrator* secara manual. *Developer* yang berperan sebagai *integrator* adalah Fachrul. Fachrul akan melakukan *trigger* eksekusi *build* di mesin integrasi. Pada **Gambar 4-7** bagian 1, dijelaskan bahwa *integrator* menguji kode program semua versi modul yang benar dari setiap *developer*. *Integrator* menemukan kesalahan pada hasil pengujian dan mencatat kesalahan tersebut secara manual. Pada **Gambar 4-7** bagian 2, dijelaskan bahwa Fachrul menginformasikan kesalahan tersebut secara manual kepada setiap *developer* untuk dapat segera diperbaiki.

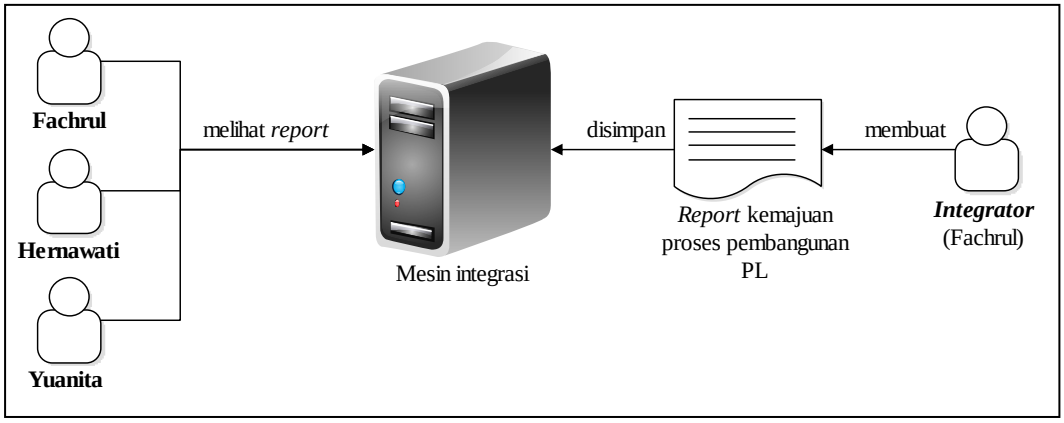
Integrasi modul yang lulus dari pegujian akan dijadikan paket aplikasi berisi *file* siap pakai di mesin integrasi. Untuk mendapatkan *history* dari semua paket aplikasi yang telah dibuat, maka paket aplikasi perlu diarsipkan. Pengarsipan paket aplikasi tersebut dilakukan secara manual oleh seorang *integrator* di mesin integrasi.



Gambar 4-8. Pengarsipan paket aplikasi `medrecapp` oleh *integrator*

Pada **Gambar 4-8**, dijelaskan bahwa tim mengintegrasikan modul tanpa menggunakan *automated CI tools*. Fachrul berperan sebagai seorang *integrator* yang akan melakukan *trigger* eksekusi *build* pada mesin integrasi. Pada **Gambar 4-8** bagian 1, dijelaskan bahwa Fachrul berhasil melakukan *trigger* eksekusi *build* pada mesin integrasi. Hasil dari eksekusi *build* tersebut adalah paket aplikasi yang berisi *file* siap pakai. Pada **Gambar 4-8** bagian 2, dijelaskan bahwa Fachrul mengarsipkan paket aplikasi di mesin integrasi secara manual. Pengarsipan paket aplikasi dilakukan untuk menyimpan *history* dari perubahan paket aplikasi.

Arsip dari aplikasi tersebut, akan dijadikan *milestone* dari kemajuan proses pembangunan perangkat lunak. Untuk mendapatkan informasi tentang kemajuan proses pembangunan perangkat lunak, *integrator* akan mengarsipkan paket aplikasi secara manual. Pengarsipan paket tersebut dilakukan oleh seorang *integrator* untuk membuat laporan kemajuan proses perangkat lunak di mesin integrasi.



Gambar 4-9. Pembuatan laporan kemajuan proses aplikasi `medrecapp` oleh *integrator*

Pada **Gambar 4-9** dijelaskan bahwa tim mengintegrasikan modul tanpa menggunakan bantuan *automated CI tools*. Fachrul berperan sebagai *integrator* yang akan membuat laporan kemajuan proses pembangunan aplikasi `medrecapp` di mesin integrasi. Laporan kemajuan tersebut akan dibuat Fachrul setiap selesai melakukan *trigger* eksekusi *build* dan disimpan di mesin integrasi. Tujuan penyimpanan laporan kemajuan tersebut di mesin integrasi adalah agar para *developer* dapat melihat hasil proses pembangunan perangkat lunak hanya dengan mengaksesnya di mesin integrasi.

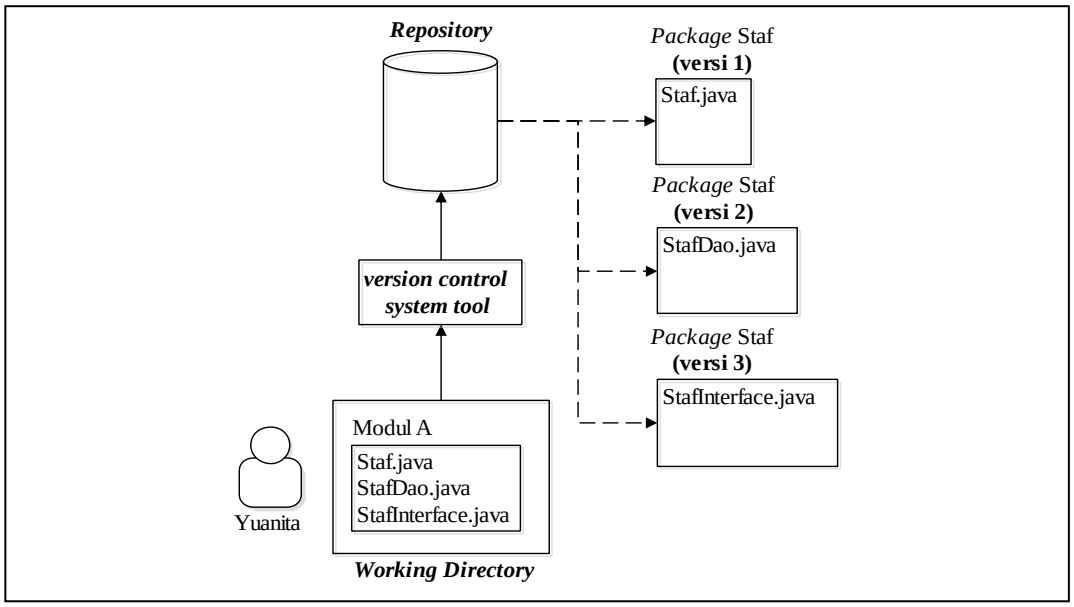
4.3. Praktik pembangunan aplikasi `medrecapp` menggunakan *toolset*

Pada sub bab ini akan dijelaskan tentang praktik pembangunan aplikasi `medrecapp` dengan CI dan menggunakan bantuan *toolset*. Praktik dengan penggunaan *toolset* tersebut mencakup praktik penyimpanan versi dengan *VCS tools* (Git), praktik pengujian kode program dengan *automated testing tools* (JUnit dan FEST), praktik eksekusi *build* dengan *automated build tools* (Ant), dan praktik integrasi modul dengan *automated CI tools* (Jenkins).

4.3.1. Praktik penyimpanan versi dengan *VCS tools*

Pada sub bab ini akan dijelaskan tentang praktik penyimpanan versi yang dilakukan tim pada pembangunan aplikasi `medrecapp` dengan menggunakan bantuan VCS *tools*. Penyimpanan versi pada praktik *automated* CI dengan bantuan *VCS tools* akan dilakukan *developer* dengan menyimpan semua versi modul ke dalam *repository*. *Tools* yang digunakan setiap *developer* untuk penyimpanan versi adalah Git. Penyimpanan versi modul tersebut dapat

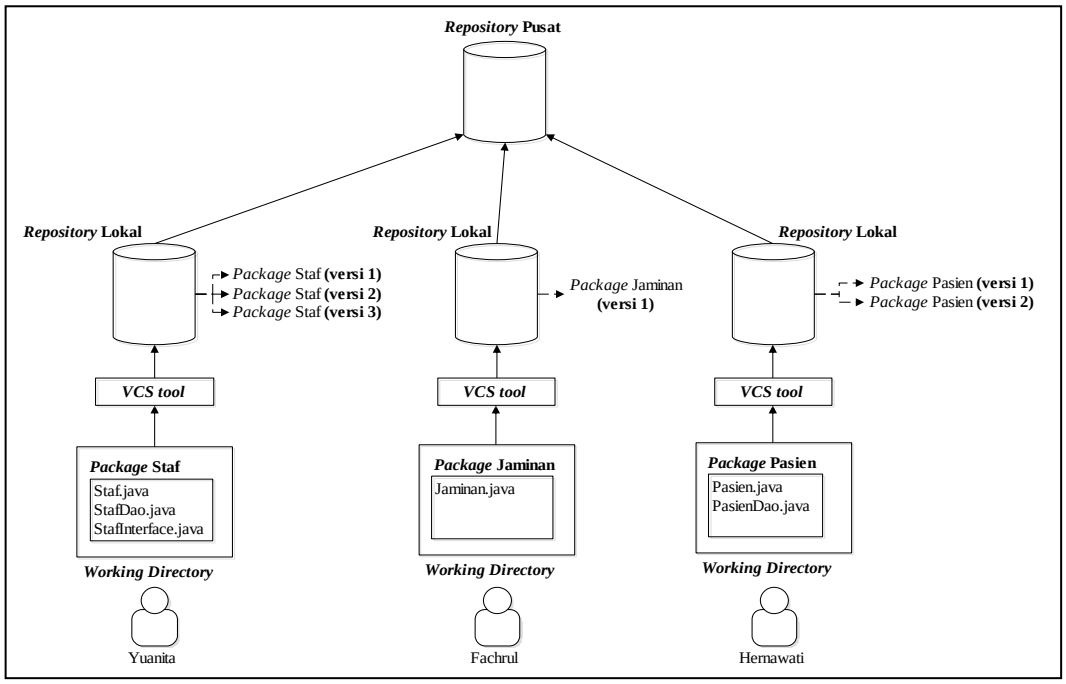
mempermudah *developer* untuk dapat melakukan *rollback* terhadap versi modul tanpa perlu melakukan duplikasi modul. Para *developer* tidak perlu lagi melakukan secara manual dalam membuat informasi tentang perubahan yang dilakukan terhadap modul, karena Git akan mencatat waktu dan isi perubahan secara otomatis ketika *developer* menyimpan versi secara manual ke *repository*.



Gambar 4-10. Penyimpanan versi modul ke dalam *repository*

Pada **Gambar 4-10** dijelaskan bahwa Yuanita melakukan penyimpanan versi dari modul `staf` yang telah diubah dengan menggunakan Git. Modul yang telah diubah akan disimpan ke dalam *repository*, sehingga tidak perlu melakukan duplikasi modul `staf`. Waktu dan isi perubahan modul `staf` akan dicatat secara otomatis oleh Git pada saat penyimpanan ke *repository*. Penyimpanan di *repository* hanya menyimpan perubahan yang terjadi pada modul `staf`, sehingga kapasitas penyimpanan yang digunakan dapat lebih kecil dibandingkan dengan cara manual.

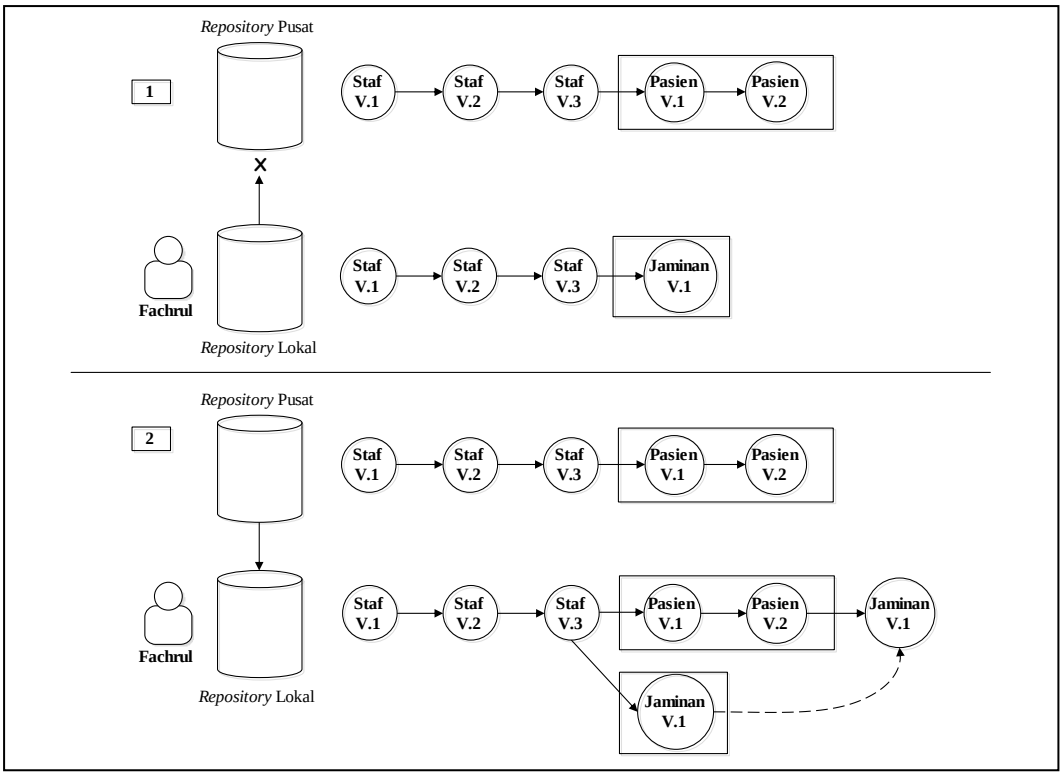
Umumnya, cara penggunaan *repository* untuk menerapkan praktik VCS adalah *distributed*. Dengan menggunakan cara *distributed*, setiap *developer* akan memiliki *repository* pada mesin lokal. *Repository* dari setiap *developer* tersebut, akan dihubungkan dengan sebuah *repository* pusat. Penggunaan *repository* dengan cara *distributed* dan dihubungkan pada sebuah *repository* pusat disebut *centralized workflow*.



Gambar 4-11. *Centralized workflow*

Pada **Gambar 4-11** dijelaskan bahwa Yuanita mengerjakan pembangunan modul `staf`, Fachrul mengerjakan pembangunan modul `jaminan` dan Hernawati mengerjakan pembanguna modul `pasien`. Setiap *developer* akan menyimpan versi dari modul yang telah diubah ke dalam *repository* lokal terlebih dahulu sebelum disimpan ke *repository* pusat. Pekerjaan tersebut dilakukan agar setiap *developer* tidak salah dalam memahami versi modul yang telah disimpan.

Anggota tim yang *repository* lokalnya belum terdapat versi terbaru dari modul di *repository* pusat, tidak akan dapat menyimpan versi modul ke *repository* pusat. Untuk mengatasi masalah tersebut, anggota tim 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 `Git`, setiap anggota tim dapat selalu memperbarui semua versi modul dari *developer* lain tanpa harus menyimpan duplikasi versi modul secara manual.

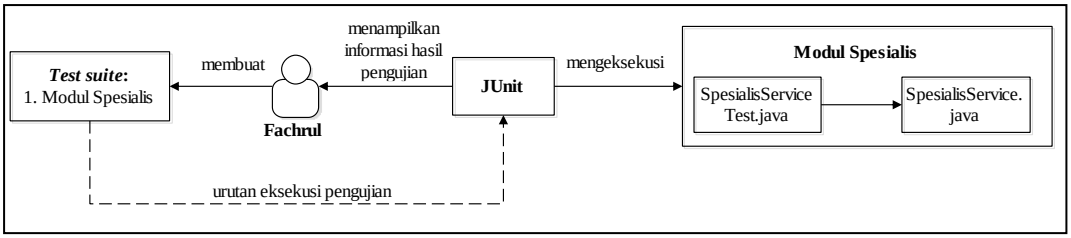


Gambar 4-12. Penggabungan versi modul

Pada **Gambar 4-12** bagian 1 dijelaskan bahwa Fachrul tidak dapat menyimpan versi modul yang terdapat di *repository* lokal ke *repository* pusat, karena *repository* lokal Fachrul belum terdapat versi terbaru dari modul *di repository* pusat. Pada **Gambar 4-12** bagian 2 dijelaskan bahwa Fachrul telah mengambil versi terbaru dari modul di *repository* pusat dan akan digabungkan dengan versi modul yang terdapat di *repository* local secara otomatis.

4.3.2. Praktik pengujian kode program dengan *automated testing tools*

Developer dapat mengurangi usaha pada pengujian unit dengan mengunakan bantuan *automated testing tools*. Berdasarkan **Tabel 2-4** *tools* yang mendukung pengujian unit dengan bahasa pemrograman java salah satunya adalah JUnit. Dengan JUnit, *developer* tidak perlu lagi membuat *driver* pengujian pada setiap kelas pengujian. Selain itu, JUnit menyediakan `test suite` untuk membuat rangkaian pengujian yang akan dieksekusi secara berurutan.



Gambar 4-13. Pengujian unit dengan menggunakan JUnit

Fachrul menguji kelas `service` pada modul `spesialis` dengan menggunakan `JUnit`. Setelah Fachrul membuat kelas pengujian `spesialis`, Fachrul dapat mengeksekusi kode pengujian tersebut tanpa perlu menambahkan `main method` di kelas pengujian. Kemudian ketika Fachrul membuat tiga kelas pengujian, Fachrul tidak perlu menambahkan `main method` pada ketiga kelas tersebut. Selain itu, Fachrul juga dapat mengeksekusi ketiga kelas pengujian hanya dengan satu kali *trigger* eksekusi `test suite` yang berisi urutan kelas pengujian.

Developer dapat mengurangi usaha dalam menguji kelas `GUI` dengan menggunakan bantuan *functional testing tools*. Berdasarkan **Tabel 2-4** *tools* yang mendukung pengujian GUI dengan menggunakan bahasa pemrograman java salah satunya adalah FEST. Dengan FEST *developer* dapat menguji kelas `GUI` secara berulang kali tanpa mengeluarkan usaha yang besar. Seain itu, penggunaan FEST dapat dieksekusi dengan menggunakan `test suite` dari JUnit.

Fachrul menguji kelas `GUI` modul `spesialis` dengan menggunakan bantuan FEST. Untuk menguji GUI tersebut Fachrul perlu membuat kelas pengujian yang akan menguji kelas `GUI` modul `spesialis`. Pengujian GUI `spesialis` tersebut akan disimulasikan oleh FEST secara otomatis.

Dengan menggunakan JUnit, usaha pada pengujian integrasi yang dilakukan *developer* dapat diminimalisasi. *Developer* dapat menguji semua modul tingkat bawah hanya dengan satu kali *trigger* eksekusi pengujian. Untuk mengotomasi semua pengujian tersebut, *developer* hanya perlu mengurutkan pengujian pada `test suite`.

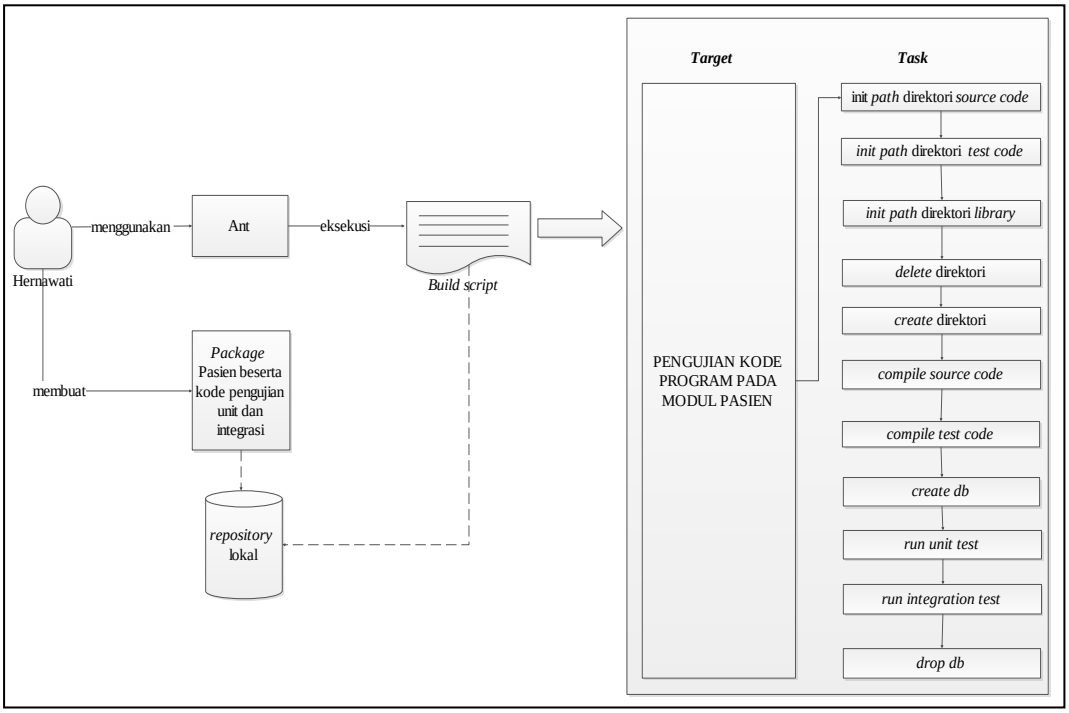
Yuanita menguji integrasi modul `rekam medis` dengan menggunakan bantuan JUnit. Yuanita dapat menguji semua kelas pada modul tingkat bawah dengan hanya satu kali *trigger* eksekusi `test suite`. Yuanita perlu

mengurutkan modul tingkat bawah yaitu modul `jaminan`, modul `pasien`, modul `staf`, modul `perawat` dan modul `dokter` pada `test suite`.

4.3.3. Praktik eksekusi *build* dengan *automated build tools*

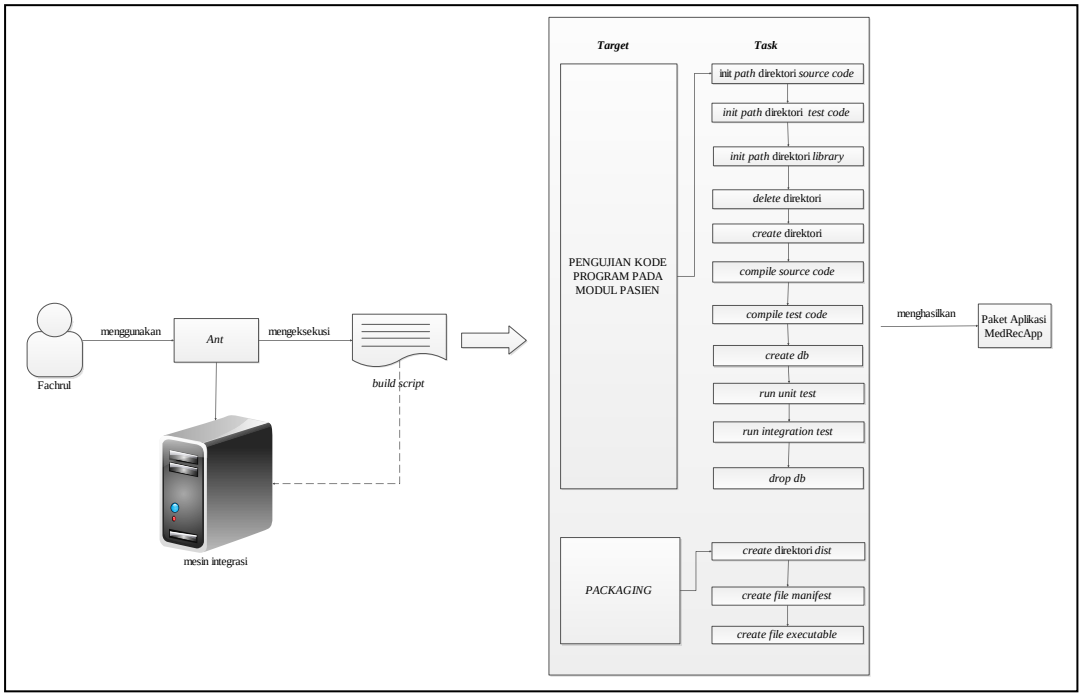
Proses *build* dilakukan dengan menggunakan *automated build tools* yaitu Ant. Kegiatan pengujian kode program dan penyimpanan versi modul yang sudah diubah ke *repository* lokal dapat diotomasi dengan bantuan *build script*. *Build script* tersebut berisi beberapa *target* dan *task* yang akan dieksekusi oleh Ant. Tim membuat *build script* untuk menyamakan proses alur kerja dari setiap anggota tim di mesin lokal serta mengotomasi proses *build* yang akan dilakukan oleh *integrator* di mesin integrasi.

Build script yang dieksekusi oleh Ant di mesin lokal setiap anggota tim disebut *private build*. Untuk menyamakan alur kerja setiap *developer*, tim perlu menentukan *target* dan *task* yang akan dilakukan oleh Ant. Setiap *target* dapat terdiri dari beberapa *task* dan bergantung pada *target* yang lain. Beberapa *target* yang ada pada *private build* mencakup eksekusi pengujian kode program dan penyimpanan versi modul yang sudah diubah ke dalam *repository* lokal.



Gambar 4-14. Eksekusi *private build*

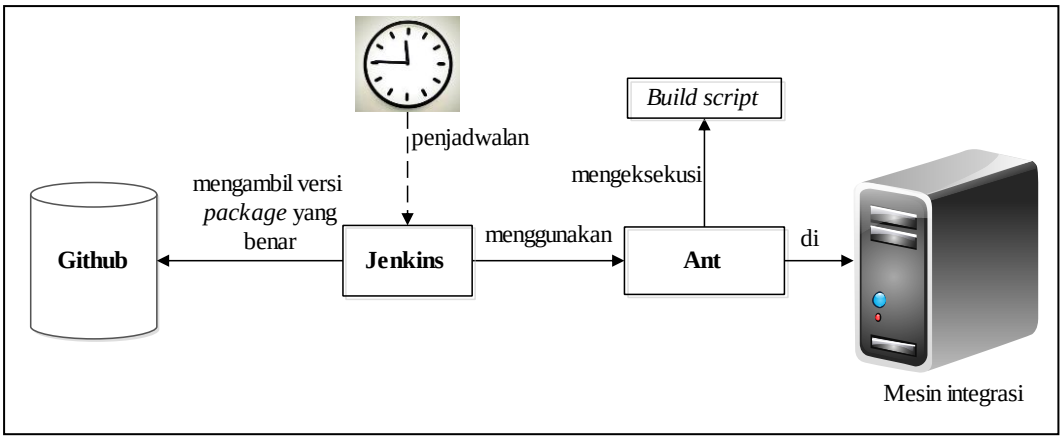
Pada **Gambar 4-14**, dijelaskan bahwa Hernawati menggunakan *automated build tools* yaitu Ant untuk mengeksekusi *build script* yang berisi *target* pengujian kode program yang terdiri dari 11 *tasks*. *Task* tersebut terdiri dari inisialisasi *path* direktori kode program, inisialisasi *path* direktori kode pengujian, inisialisasi *path* direktori *library*, menghapus direktori, membuat direktori baru, kompilasi kode program, kompilasi kode pengujian, membuat *database*, menjalankan *unit testing*, menjalankan *integration testing* dan menghapus *database*.



Gambar 4-15. Eksekusi *integration build*

Pada **Gambar 4-15**, dijelaskan bahwa *integration build* dieksekusi oleh Fachrul dengan menggunakan Ant. Ant akan mengeksekusi *build script* yang terdiri dari serangkaian *target* dan *task*. Ant akan mengotomasi JUnit untuk menguji program berdasarkan `test suite`.

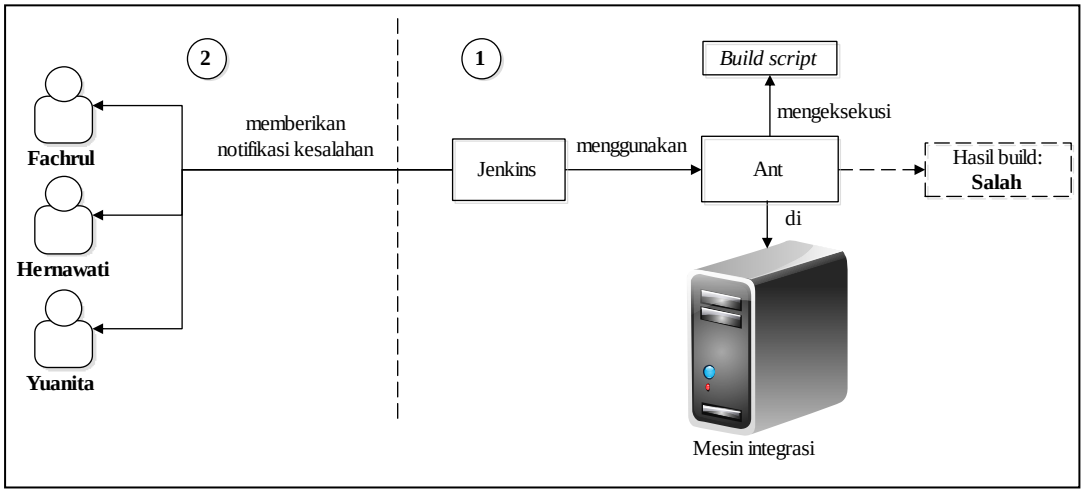
4.3.4. Praktik integrasi modul dengan *automated CI tools*



Gambar 4-16. Penjadwalan eksekusi *build script* pada mesin integrasi

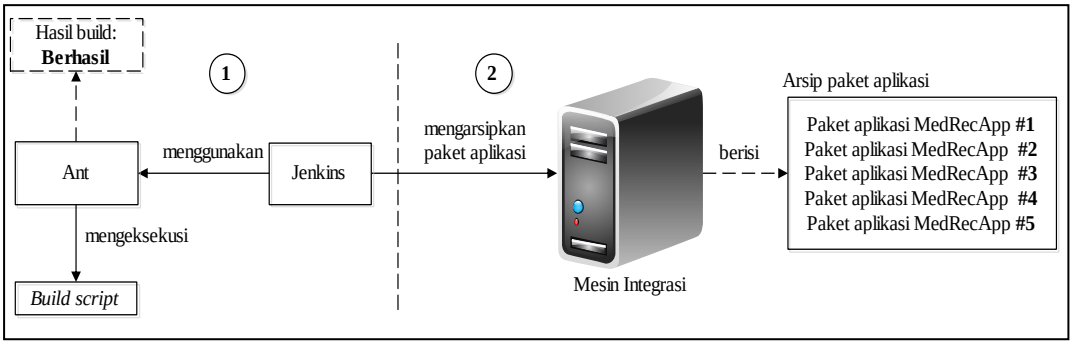
Dengan menerapkan *automated CI tools* dengan Jenkins, maka *integrator* tidak dibutuhkan lagi untuk melakukan proses eksekusi *build script*. Jenkins akan mengeksekusi *build script* dengan menggunakan Ant pada mesin integrasi. Tim hanya perlu menjadwalkan eksekusi *build*, kemudian Jenkins akan mengeksekusi *build script* sesuai dengan penjadwalan yang diatur oleh tim.

Pada setiap eksekusi *integration build*, mesin integrasi akan menguji kode program dan paket aplikasi secara otomatis. Dengan menggunakan Jenkins, maka tim tidak lagi membutuhkan seorang *integrator* pada mesin integrasi untuk menginformasikan kesalahan pada satu atau lebih hasil pengujian. Notifikasi kesalahan tersebut akan diinformasikan oleh Jenkins kepada setiap anggota tim secara otomatis.



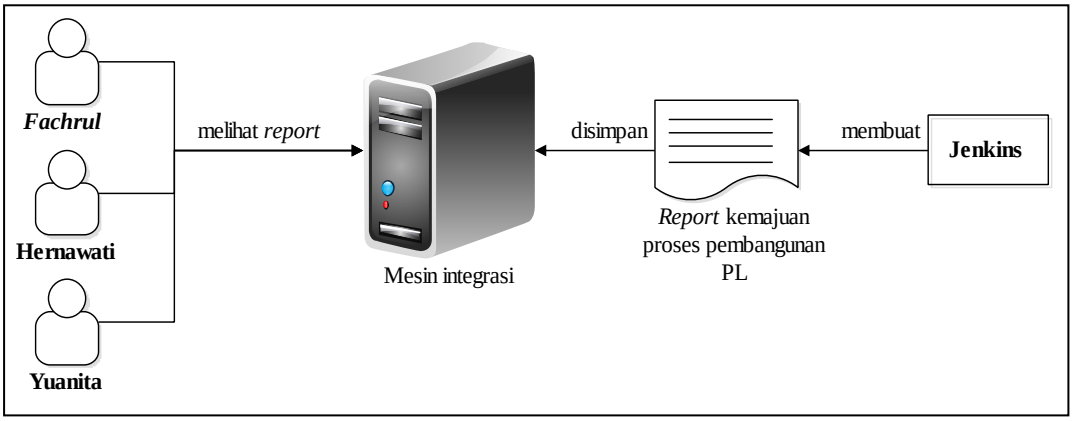
Gambar 4-17. Notifikasi kesalahan secara otomatis oleh Jenkins

Pada **Gambar 4-17**, dijelaskan bahwa Jenkins mengeksekusi *build script* dengan menggunakan Ant. Jika Jenkins menemukan kesalahan pada hasil *build* yang ada pada mesin integrasi, maka Jenkins akan memberikan notifikasi kesalahan apabila hasil *build* pada mesin integrasi gagal. Dengan *automated CI tools*, maka tim tidak lagi membutuhkan *integrator* untuk mengarsipkan paket aplikasi.



Gambar 4-18. Pengarsipan paket aplikasi oleh mesin integrasi secara otomatis.

Pada **Gambar 4-18**, dijelaskan bahwa Jenkins menggunakan Ant untuk mengeksekusi *build script*. Kemudian setiap hasil paket aplikasi yang berhasil di-*build* akan diarsipkan oleh Jenkins di mesin integrasi.



Gambar 4-19. Laporan kemajuan proses pembangunan perangkat lunak secara otomatis.

Pada **Gambar 4-19**, dijelaskan bahwa Jenkins akan membuat laporan kemajuan proses pembangunan perangkat lunak yang disimpan di mesin integrasi. Kemudian anggota tim dapat melihat laporan tersebut pada mesin integrasi.

4.4. Kesimpulan studi kasus

Lesson learn

Dengan menerapkan praktik automated CI, pekerjaan developer menjadi lebih efisien. Setiap developer hanya perlu membuat kode program serta membuat kode pengujian unit dan integrasi. Setelah kode pengujian selesai dibuat, developer hanya perlu menjalankan pengujian tersebut. Jika hasil pengujian tidak menampikan kesalahan, maka simpan versi modul tersebut ke repository dan kembali membuat kode program.

Integrasi modul harus dilakukan secara rutin, agar kesalahan pada kode program dapat segera diketahui sebelum kode program semakin banyak dan kompleks.

Sebelum menerapkan praktik automated CI, diperlukan perencanaan aplikasi yang benar, karena ketika terjadi perubahan pada rencana awal, misalnya perubahan struktur kode program, maka tim tersebut perlu merencanakan ulang praktik tersebut dari awal.

Pada praktik automated CI diperlukan komunikasi tim yang baik, agar setiap pekerjaan yang dilakukan anggota tim tidak redundan.