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* yang dilakukan secara– manual dan menggunakan *toolset*. Penerapan konsep tersebut dilakukan untuk menunjukkan perbedaan proses dari keduanya. Praktik yang mencakup penerapan *continuous integration* secara manual yaitu praktik penyimpanan versi secara manual, praktik pengujian kode program secara manual, praktik eksekusi *build* secara manual, dan praktik pengintegrasian modul secara manual. Sedangkan praktik yang mencakup penerapan *continuous integration* dengan menggunakan *toolset* yaitu praktik penyimpanan versi dengan *version control system tool*, praktik pengujian kode program dengan *automated testing tool*, praktik eksekusi *build* dengan *automated build tool* dan praktik pengintegrasian modul dengan *automated continuous integration tool*.

Studi kasus yang digunakan untuk menerapkan konsep pembangunan perangkat lunak dengan *continuous integration* 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 *developer*, yaitu Fachrul, Hernawati dan Yuanita.

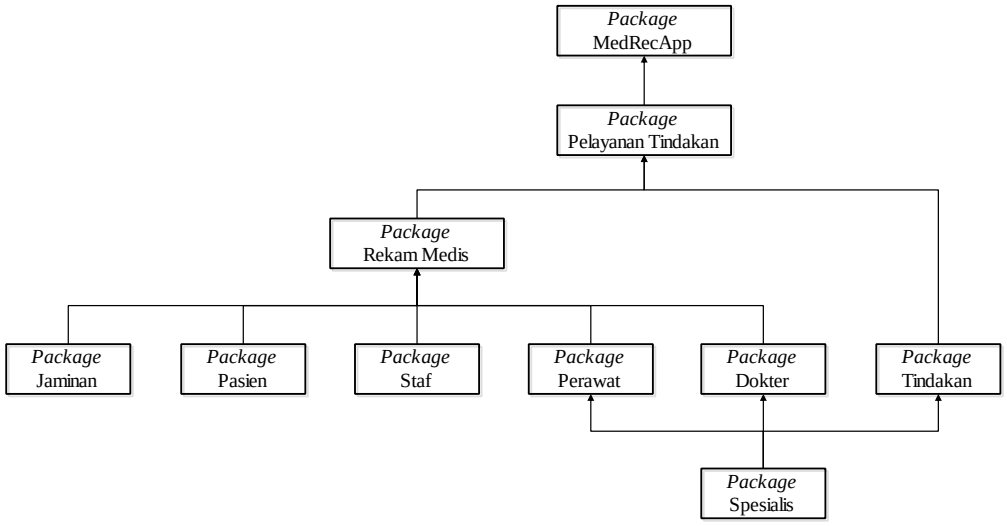
4.1. Modul aplikasi `medrecapp`

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

Tabel 4-1. Pembagian *package* aplikasi medrecapp

No.	Daftar *packages*	*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 *package* aplikasi medrecapp, terdapat kelas DAO (*Data Access Object*), kelas *entity*, kelas *GUI*, kelas *interface*, kelas *services*, dan kelas tabel model. Setiap kelas pada *package* memiliki beberapa *method*. *Package* aplikasi medrecapp terdiri dari sembilan *package* yaitu *package* spesialis, *package* jaminan, *package* pasien, *package* staf, *package* perawat, *package* dokter, *package* tindakan, *package* rekam medis, dan *package* pelayanan tindakan. Dependensi antar *package* aplikasi medrecapp dapat dilihat pada **gambar 4-1**.



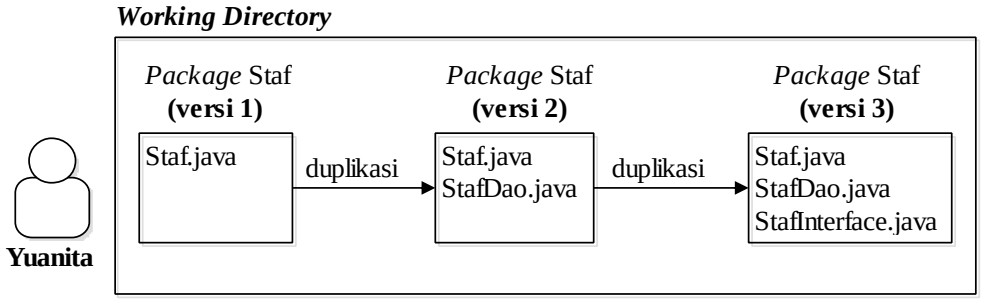
Gambar 4-1. *Package* pada aplikasi medrecapp

4.2. Praktik pembangunan aplikasi medrecapp secara manual

Pada sub bab ini akan dijelaskan tentang praktik pembangunan aplikasi medrecapp dengan *continuous integration* 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 pengintegrasian *package* secara manual.

4.2.1. Praktik penyimpanan versi secara manual

Secara umum, penyimpanan versi dilakukan oleh tim untuk menyimpan *history* dari setiap perubahan *package*. Tim yang tidak menggunakan bantuan *tool* dari *version control system* umumnya akan melakukan duplikasi *package* sebelum mengubah *package* tersebut. Hasil duplikasi *package* akan digunakan oleh tim sebagai *backup* untuk dapat kembali ke *package* yang belum diubah. Untuk membedakan hasil dari setiap *package* yang telah diduplikasi, tim perlu memberikan penamaan versi dan menambahkan informasi tentang detil perubahan yang terjadi pada *package* tersebut.

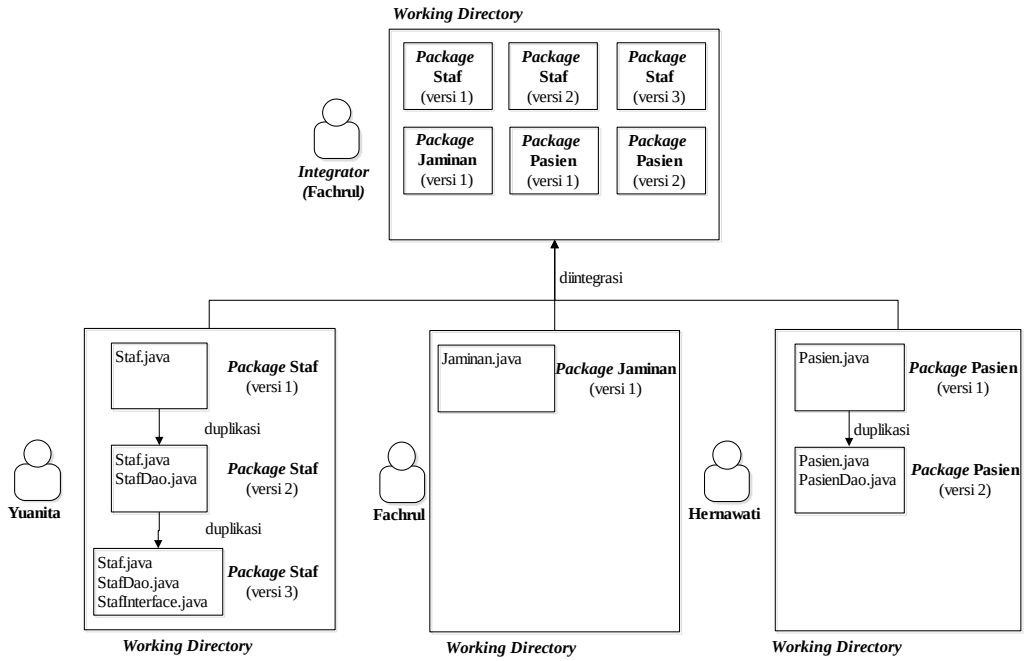


Gambar 4-2. Penyimpanan versi dengan cara manual

Pada **Gambar 4-2** dijelaskan bahwa Yuanita akan membuat *package* staf tanpa menggunakan bantuan *tool* dari *version control system.* Pembuatan tersebut dilakukan dengan cara melakukan duplikasi *package* staf terlebih dahulu sbelum mengubahnya. Hasil dari duplikasi *package* staf akan digunakan sebagai *backup* agar dapat kembali ke versi *package* sebelumnya. Yuanita akan melakukan penamaan versi *package* staf untuk dapat membedakan setiap hasil duplikasi yang dilakukan dan mencatat setiap perubahan yang terjadi pada setiap *package*.

Umumnya, setiap versi *package* yang dibuat para anggota tim 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 *tool* dari *version control system*, umumnya membutuhkan seorang *integrator* untuk mengelola semua versi *package* di tempat penyimpanan versi terpusat. *Integrator* tersebut akan memilih versi dari setiap *package* yang akan dijadikan paket aplikasi yang berisi *file* siap pakai.



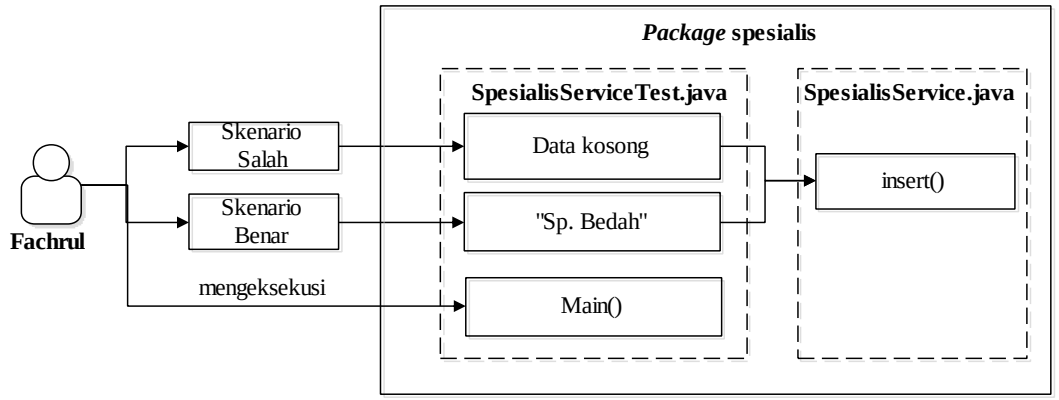
Gambar 4-3. Penggabungan versi modul secara manual

Pada **Gambar 4-3** dijelaskan bahwa Fachrul berperan sebagai *integrator* yang akan mengelola versi *package* di tempat penyimpanan versi terpusat. Setiap versi *package* yang dibuat Yuanita, Fachrul dan Herna akan disimpan di tempat penyimpanan versi terpusat, sehingga tidak salah dalam memahami versi *package* yang telah dibuat. Penyimpanan versi *package* dibuat untuk memudahkan *developer* ketika membutuhkan *package* dari hasil pekerjaaan *developer* lain. Semua versi *package* 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 *package* yang telah dibuat.

4.2.2. Praktik pengujian kode program secara manual

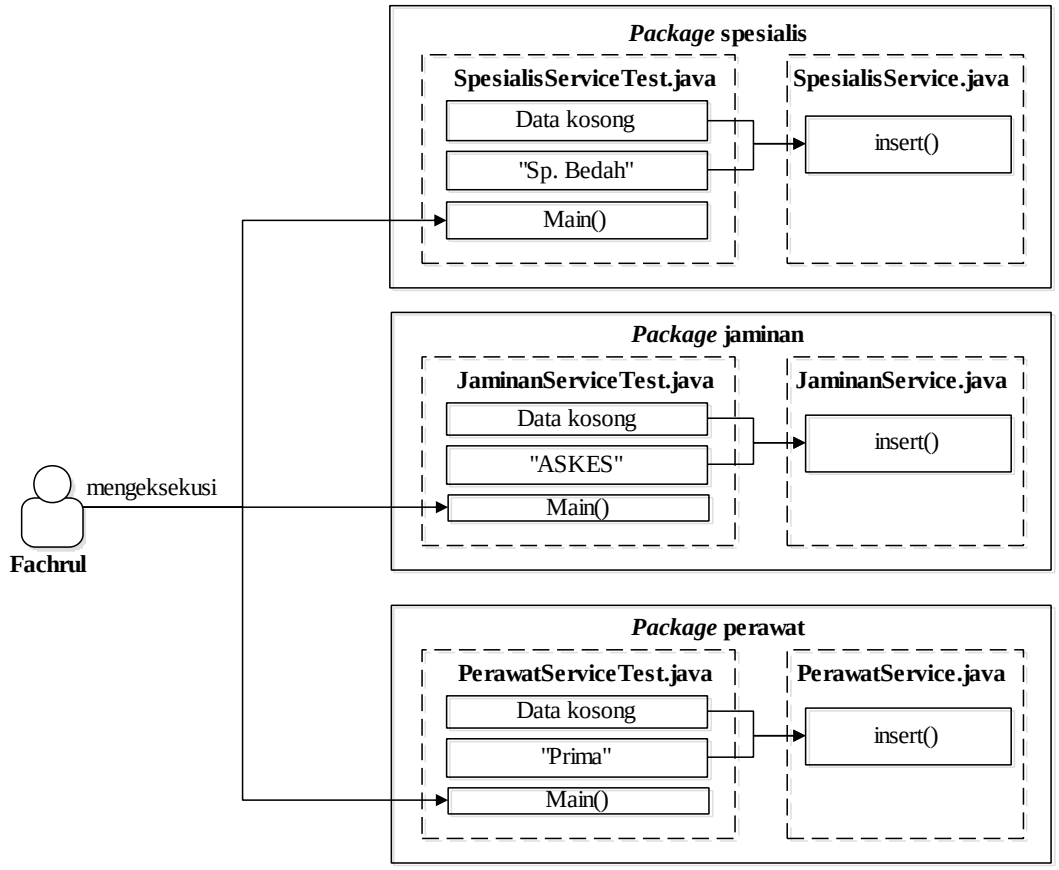
Package yang dikerjakan setiap *developer* akan ditambahi kelas-kelas. Pada setiap kelas, *developer* akan menambahi satu atau lebih *method*. Setiap *method* yang ditambahi ke dalam kelas harus diuji. Pengujian *method* (unit) dilakukan para *developer* untuk memastikan bahwa *functional requirement* dari *package* yang dibuat dapat dieksekusi serta minim dari kesalahan. Praktik pengujian kode program secara manual yang dicakup hanya pada tingkat pengujian unit dan integrasi

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



Gambar 4-4. Pengujian unit pada satu kelas *service*

Fachrul menguji unit pada kelas *service* spesialis. Untuk menguji unit pada kelas tersebut, Fachrul perlu membuat kelas pengujian. Kelas pengujian tersebut berisi *test case* salah dan *test case* benar terhadap satu unit kode program. Pengujian unit dilakukan Fachrul dengan cara mengeksekusi kelas pengujian tersebut. Ketika Fachrul membuat tiga kelas *service*, maka Fachrul perlu menambahkan *main method* pada tiga kelas pengujian dan mengeksekusi kelas pengujian tersebut satu per satu.



Gambar 4-5. Pengujian tiga kelas *service* secara manual

Unit pada kelas *GUI* perlu diuji. Untuk menguji unit pada *GUI*, tidak cukup hanya dengan menguji kode program saja. Diperlukan simulasi interaktif terhadap komponen *GUI*. *Developer* yang tidak menggunakan bantuan *functional testing tool*, perlu melakukan simulasi dengan skenario salah dan skenario benar secara manual dan berulang kali.

Fachrul menguji unit *GUI* pada kelas *GUI* spesialis. Untuk menguji unit *GUI* pada kelas tersebut, Fachrul perlu melakukan simulasi interaktif terhadap antarmuka spesialis. Pengujian unit GUI dilakukan Fachrul dengan cara mengeksekusi GUI spesialis 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.

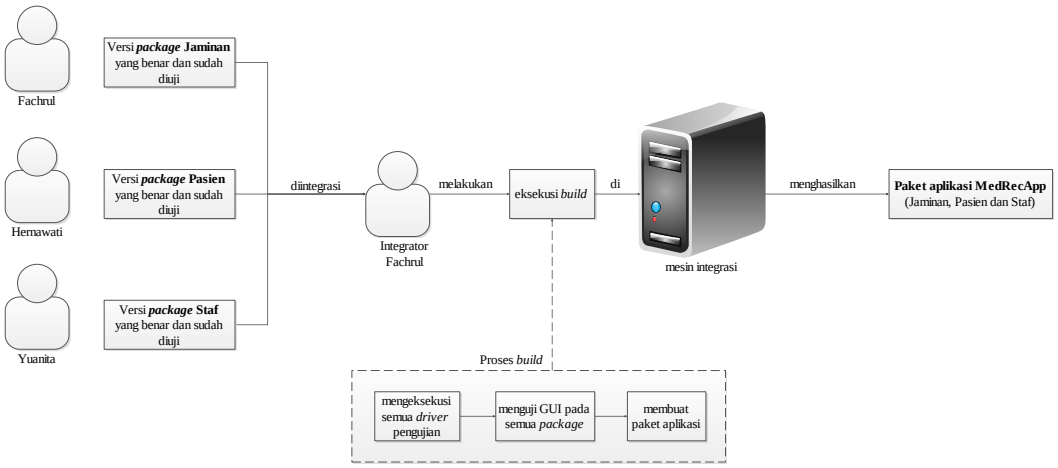
Package tingkat atas yang berdependensi dengan package tingkat bawah perlu dilakukan pengujian integrasi. pengujian tersebut dilakukan untuk menguji

kombinasi antar package dan menampilkan kesalahan pada interaksi antar method yang terintegrasi.

Yuanita membuat package rekam medis yang memiliki dependensi dengan lima package yaitu package jaminan, package pasien, package staf, package perawat dan package dokter. Sebelum Yuanita menguji rekam medis, maka perlu menguji package tingkat bawah. Pengujian tersebut dilakukan dengan mengeksekusi semua driver pengujian ditingkat bawah dan mensimulasikan semua skenario salah dan benar terhadap package di tingkat bawah.

4.2.3. Praktik eksekusi *build* secara manual

Package hasil pekerjaan dari para anggota tim akan digabungkan oleh *integrator*. Developer yang berperan sebagai *integrator* adalah Fachrul. Pembagian kerja pada anggota tim berdasarkan *package* aplikasi `medrecapp`, yaitu Fachrul membuat *package* Jaminan, Hernawati membuat *package* Pasien dan Yuanita membuat *package* Staf.



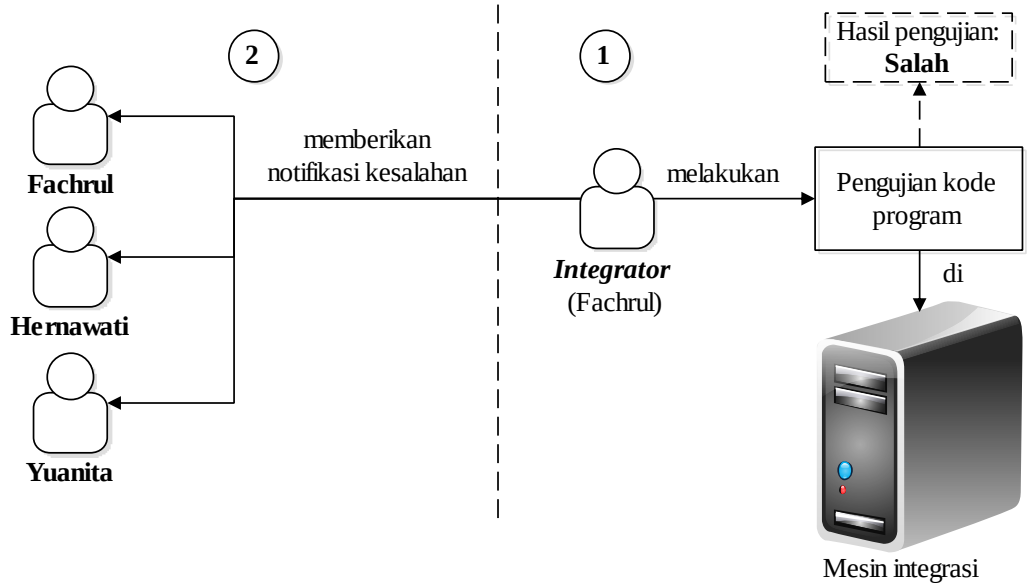
Gambar 4-6. Eksekusi *build* dengan cara manual

Pada **Gambar 4-6**, dijelaskan bahwa Fachrul akan menguji keseluruhan *package* dari hasil pekerjaan anggota tim yaitu *package* gabungan, yang terdiri dari *package* Jaminan, Pasien dan Staf. *Package* yang telah dibuat oleh setiap *developer* akan diuji berdasarkan pengujian unit dan integrasi. Setelah pengujian unit dan integrasi dilakukan, maka Fachrul akan melakukan eksekusi proses *build* pada *package* gabungan tersebut. Eksekusi proses *build* dilakukan mulai dari

eksekusi semua *driver* pengujian, menguji GUI pada semua *package* dan membuat paket aplikasi. Hasil akhir dari proses *build* perangkat lunak ini adalah paket aplikasi `medrecapp` yang terdiri dari *package* Jaminan, Pasien dan Staf.

4.2.4. Praktik pengintegrasian *package* secara manual

Pada sub bab ini, dijelaskan bahwa pengujian yang dilakukan oleh *integrator* hanya mencakup pengujian sebelum paket aplikasi dibuat, yaitu eksekusi semua *driver* pengujian dan menguji antarmuka pada semua *package*. Pengintegrasian *package* secara manual dilakukan oleh seorang *integrator* di mesin integrasi. Pada proses eksekusi *build*, *integrator* akan mengeksekusi semua *driver* pengujian dan menguji GUI pada semua *package* yang telah dibuat oleh anggota tim. Pengujian tersebut bertujuan 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 para anggota untuk dapat segera diperbaiki.

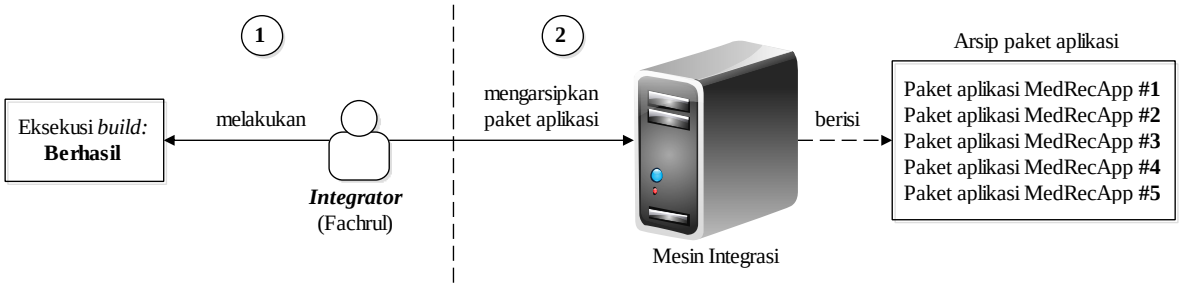


Gambar 4-7. Pemberian notifikasi kesalahan secara manual oleh *integrator*

Pada **Gambar 4-7** dijelaskan bahwa pengintegrasian *package* dilakukan oleh *integrator* secara manual. *Developer* yang berperan sebagai *integrator* adalah Fachrul. Fachrul akan mengeksekusi *build* yang ada di mesin integrasi. Pada **Gambar 4-7** bagian 1, dijelaskan bahwa integrator menguji kode program semua versi *package* yang benar dari para anggota tim. *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 para *developer* untuk dapat segera diperbaiki.

Integrasi *package* yang lulus uji akan dijadikan paket 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.

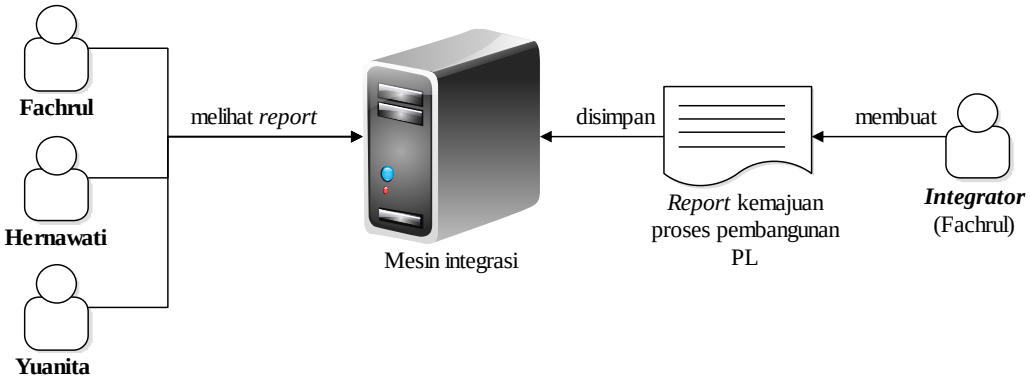


Gambar 4-8. Pengarsipan paket aplikasi MedRecApp secara manual oleh *integrator*

Pada **Gambar 4-8**, dijelaskan bahwa tim mengintegrasikan *package* tanpa menggunakan *automated continuous integration*. Fachrul yang berperan sebagai *integrator* akan mengeksekusi *build* pada mesin integrasi.

Pada **Gambar 4-8** bagian 1, jelaskan bahwa Fachrul berhasil mengeksekusi *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 melakukan pengarsipan 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, tim akan mengintegrasikan *package* secara manual. Pengintegrasian tersebut dilakukan oleh seorang *integrator* untuk membuat *report* kemajuan proses perangkat lunak di mesin integrasi.



Gambar 4-9. Pembuatan *report* kemajuan proses pembangunan perangkat lunak oleh *integrator*

Pada **Gambar 4-9** dijelaskan bahwa tim mengintegrasikan *package* tanpa menggunakan bantuan *automated continuous integration tool*. Fachrul berperan sebagai *integrator* yang akan membuat *report* kemajuan proses pembangunan perangkat lunak di mesin integrasi. *Report* kemajuan dibuat setiap proses eksekusi *build* selesai dan disimpan oleh *integrator* di mesin integrasi. Tujuan penyimpanan *report* kemajuan 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*

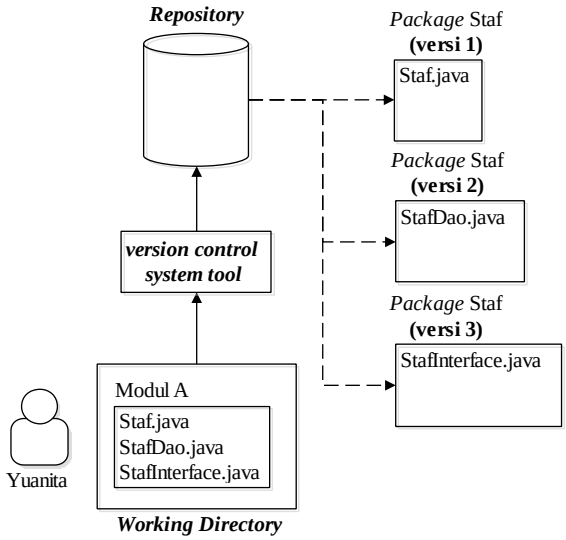
Pada sub bab ini akan dijelaskan tentang praktik pembangunan aplikasi medrecapp dengan *continuous integration* dan menggunakan bantuan *toolset*. Praktik dengan penggunaan *toolset* tersebut mencakup praktik penyimpanan versi dengan *version control system tool*, praktik pengujian kode program dengan *automated testing tool*, praktik eksekusi *build* dengan *automated build tool*, dan praktik pengintegrasian *package* dengan *automated continuous integration tool*.

4.3.1. Praktik penyimpanan versi dengan *version control system tool*

Pada sub bab ini akan menjelaskan tentang pengimplementasian penyimpanan versi yaitu penyimpanan versi *package* ke *repository*, pengunaan *centralized workflow* dan penyimpanan versi *package* dari *repository* lokal ke *repository* pusat.

Penyimpanan versi pada praktik *continuous integration* dengan bantuan *tool* dari *version control system* akan menyimpan semua versi *package* ke dalam

repository. *Tool* yang digunakan untuk penyimpanan versi adalah Git. Penyimpanan versi *package* tersebut dapat mempermudah *developer* untuk dapat melakukan *rollback* terhadap versi *package* tanpa perlu melakukan duplikasi *package*. Para anggota tim tidak perlu lagi melakukan secara manual dalam memberikan informasi tentang perubahan yang dilakukan terhadap *package*, karena Git akan mencatat waktu dan isi perubahan secara otomatis ketika menyimpan versi secara manual ke *repository*.

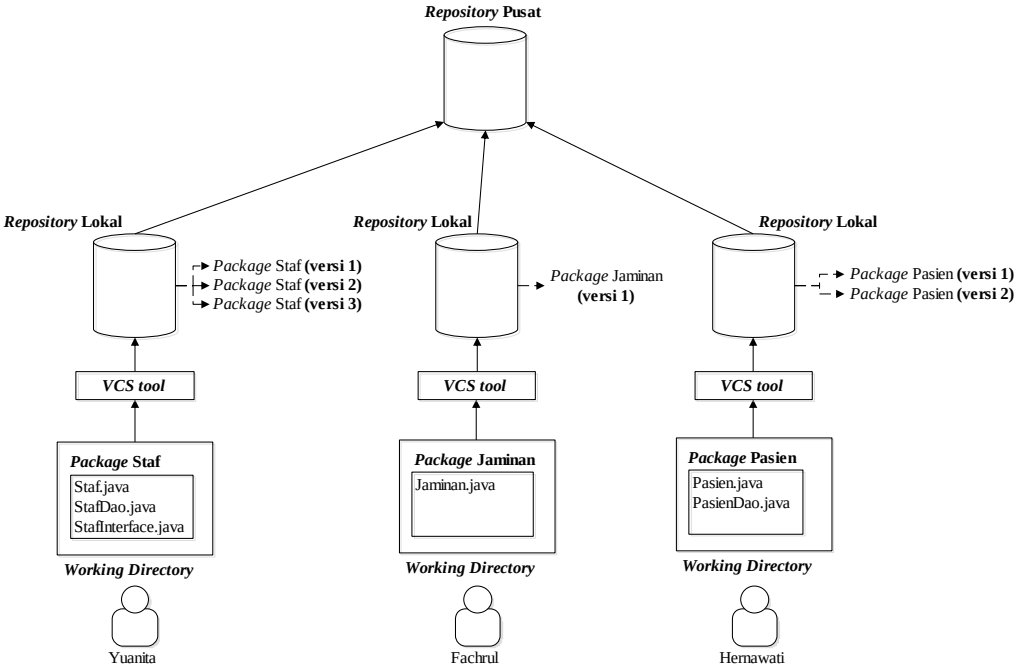


Gambar 4-10. Penyimpanan versi modul ke dalam *repository*

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

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. *Repository* dari setiap anggota tim tersebut, akan dihubungkan dengan sebuah *repository* pusat.

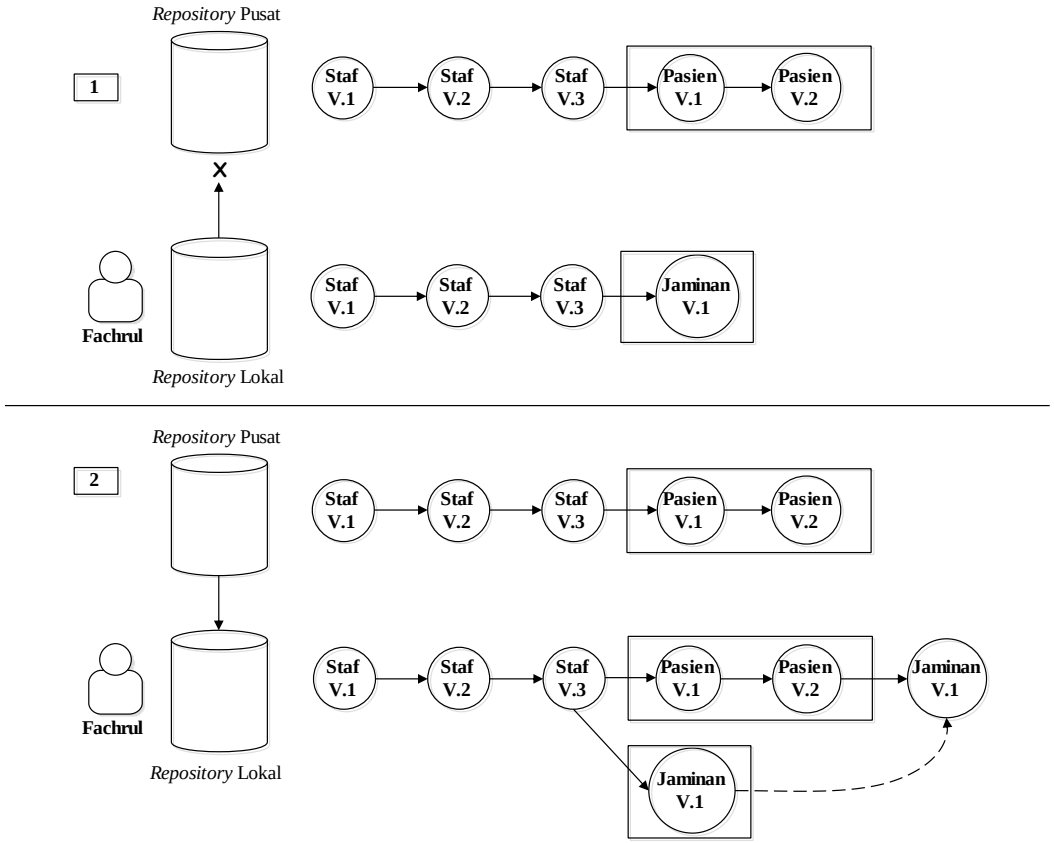
Penggunaan *repository* dengan cara *distributed* dan dihubungkan pada sebuah *repository* pusat disebut *centralized workflow*.



Gambar 4-11. *Centralized workflow*

Pada **Gambar 4-11** dijelaskan bahwa Yuanita mengerjakan pembangunan *package* staf, Fachrul mengerjakan pembangunan *package* jaminan dan Hernawati mengerjakan pembanguna *package* pasien. Setiap *developer* akan menyimpan versi dari *package* 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 *package* yang telah disimpan.

Anggota tim yang *repository* lokalnya belum terdapat versi terbaru dari *package* di *repository* pusat, tidak akan dapat menyimpan versi *package* ke *repository* pusat. Untuk mengatasi masalah tersebut, anggota tim perlu mengambil versi terbaru dari *package* di *repository* pusat terlebih dahulu. Semua versi terbaru dari *package* yang diambil dari *repository* pusat, akan digabungkan dengan versi *package* yang ada di *repository* lokal secara otomatis. Dengan menggunakan *tool* `Git`, setiap anggota tim dapat selalu memperbarui semua versi *package* 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 *package* yang terdapat di *repository* lokal ke *repository* pusat, karena *repository* lokal Fachrul belum terdapat versi terbaru dari *package di repository* pusat. Pada **Gambar 4-12** bagian 2 dijelaskan bahwa Fachrul telah mengambil versi terbaru dari *package* di *repository* pusat dan akan digabungkan dengan versi *package* yang terdapatdi *repository* local secara otomatis.

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

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

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

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

Fachrul menguji GUI spesialis dengan menggunakan bantuan FEST. Untuk menguji GUI tersebut Fachrul perlu membuat kelas pengujian yang akan menguji GUI spesialis. Kelas pengujian GUI spaeialis tersebut akan digunakan oleh FEST untuk mensimulasikan GUI secara otomatis.

Dengan menggunakan JUnit, pengujian integrasi yang dilakukan *developer* dapat minim dari *effort*. Developer dapat menguji semua *package* tingkat bawah hanya dengan satu kali eksekusi pengujian. Untuk mengotomasi semua pengujian tersebut, *developer* hanya perlu mengurutkan pengujian pada *test suite*.

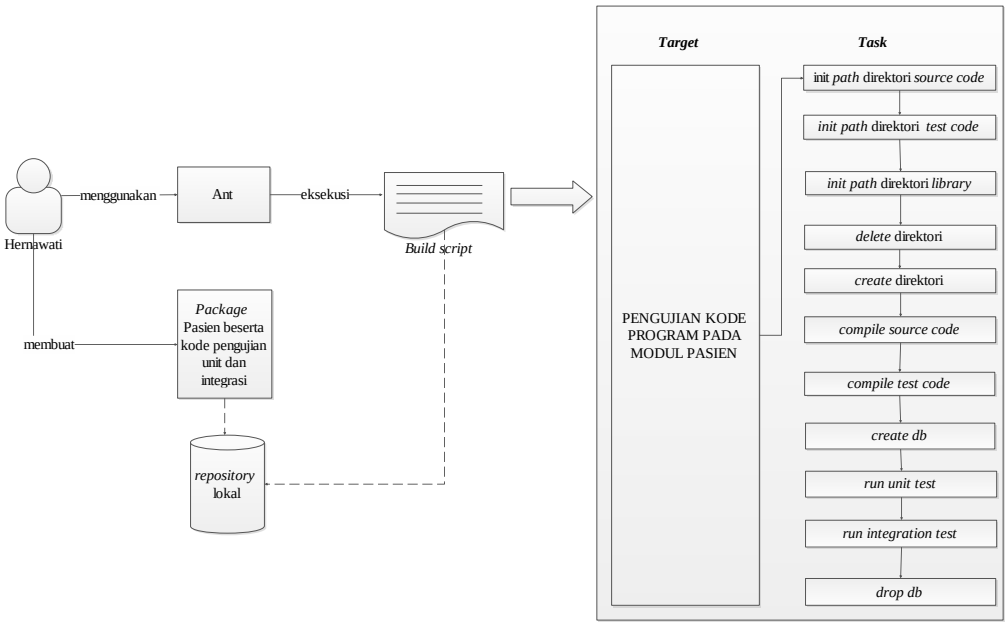
Yuanita menguji integrasi *package* rekam medis dengan menggunakan bantuan JUnit. Yuanita dapat mengeksekusi semua driver pengujian tingkat bawah dengan satu kali eksekusi pengujian. Yuanita perlu mengurutkan package tingkat bawah yaitu *package* jaminan, *package* pasien, *package* staf, *package* perawat dan *package* dokter pada *test suite.*

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

Pada sub bagian ini, tingkatan *automated build* yang dilakukan adalah *private* dan *integration build*. Proses *build* dilakukan dengan menggunakan *automated build tool* yaitu Ant. Kegiatan pengujian kode program dan

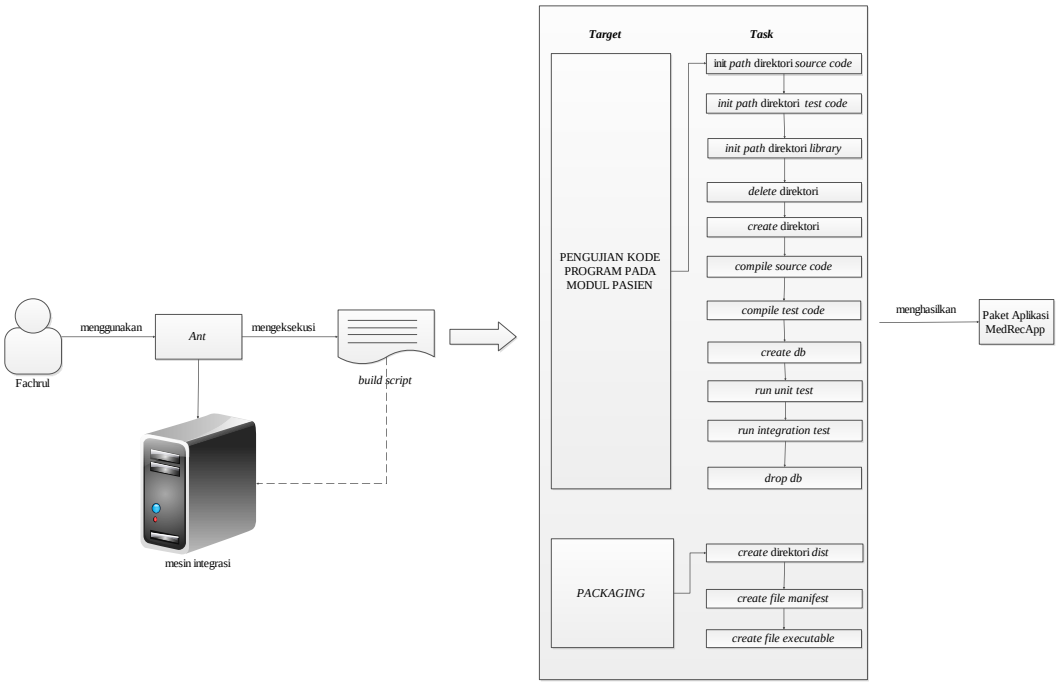
penyimpanan versi *package* yang sudah diubah ke *repository* lokal dapat diotomasi dengan bantuan *build script*. *Build script* tersebut berisi beberapa *target* dan *task* yang akan dieksekusi oleh Ant. Tim membuat *build script* untuk menyamakan proses alur kerja dari setiap anggota tim di mesin 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 anggota tim, tim perlu menentukan *target* dan *task* yang akan dilakukan oleh *automated build tool*. 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 *package* yang sudah diubah ke dalam *repository* lokal.



Gambar 4-13 . Eksekusi *private build*

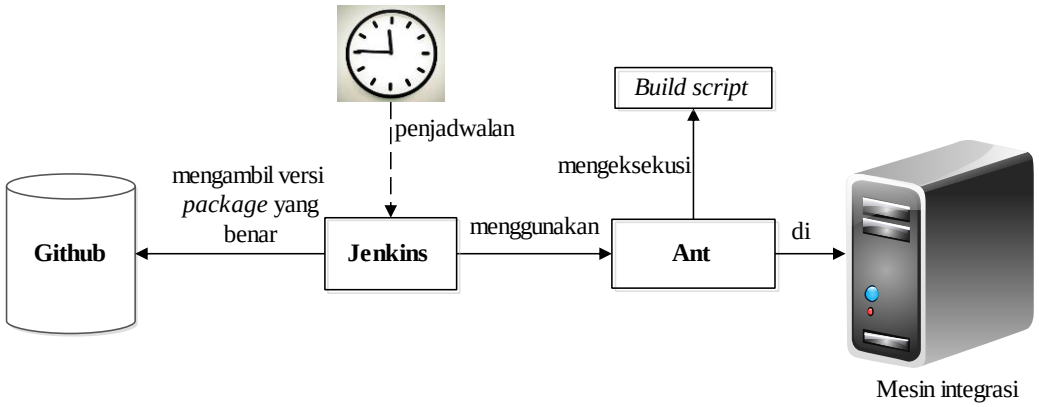
Pada **Gambar 4-13**, dijelaskan bahwa Hernawati menggunakan automated build tool yaitu Ant untuk mengeksekusi *build script* yang berisi *target* pengujian kode program yang terdiri dari 11 *tasks*. Task tersebut terdiri dari *init path* direktori *source code*, *init path* direktori *test code*, *init path* direktori *library*, *delete direktori*, *create direktori*, *compile source code*, *compile test code*, *create* db, *run unit test*, *run integration test* dan *drop* db.



Gambar 4-14. Eksekusi *integration build*

Pada **Gambar 4-14**, dijelaskan bahwa *integration build* dieksekusi oleh Fachrul dengan menggunakan *tool* 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 pengintegrasian modul dengan *automated continuous integration tool*

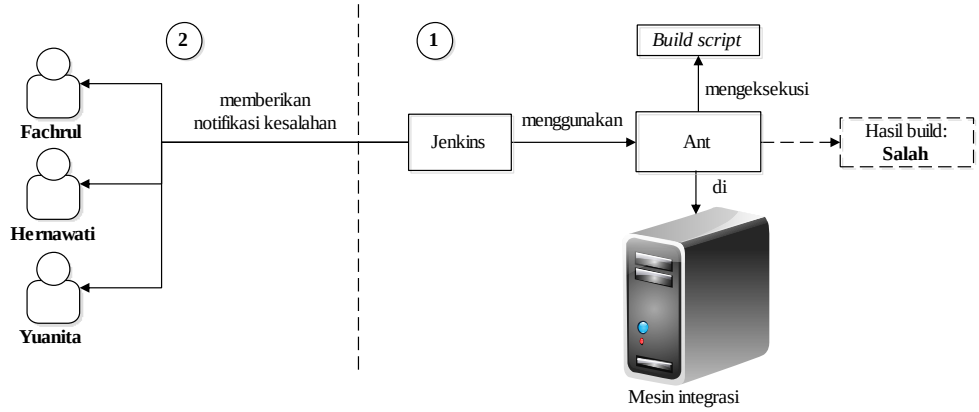


Gambar 4-15. Penjadwalan eksekusi *build script* pada mesin integrasi

Dengan menerapkan *automated continuous integration tool* dengan *tool* Jenkins, maka *integrator* tidak dibutuhkan lagi untuk melakukan proses eksekusi

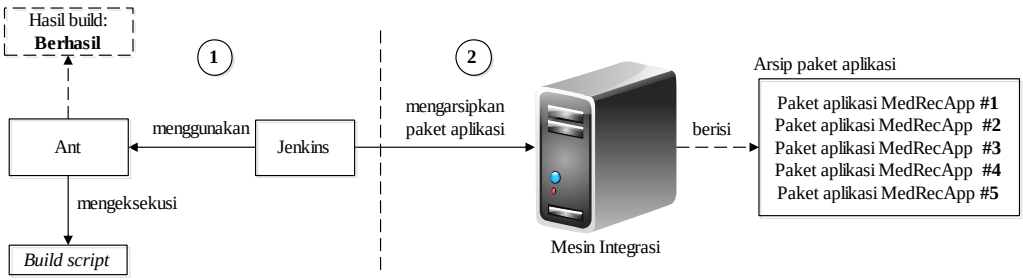
build script. Jenkins akan mengeksekusi *build script* dengan menggunakan Ant pada mesin integrasi. Tim hanya perlu menjadwalkan eksekusi *build*, kemudian Jenkins akan mengeksekusi *build script* sesuai dengan penjadwalan yang diatur oleh anggota tim sebelumnya.

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.



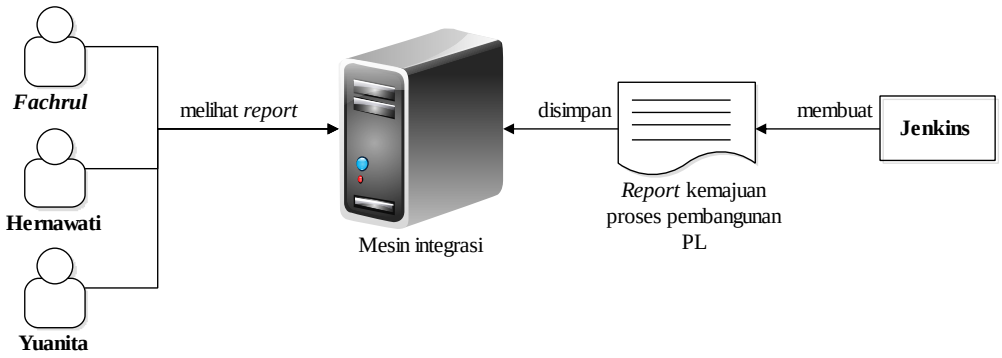
Gambar 4-16. Notifikasi kesalahan secara otomatis dari mesin integrasi

Pada **Gambar 4-16**, dijelaskan bahwa Jenkins mengeksekusi *build script* dengan menggunakan *tool* 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 continuous integration tool*, maka tim tidak lagi membutuhkan *integrator* untuk mengarsipkan paket aplikasi secara otomatis.



Gambar 4-17. Pengarsipan paket aplikasi oleh mesin integrasi secara otomatis.

Pada **Gambar 4-17**, 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 4-18. *Report* kemajuan proses pembangunan perangkat lunak secara otomatis.

Pada **Gambar 4-18**, dijelaskan bahwa Jenkins akan membuat *report* kemajuan proses pembangunan perangkat lunak yang disimpan di mesin integrasi. Kemudian anggota tim dapat melihat *report* tersebut pada mesin integrasi.