BAB IV
IMPLEMENTASI DAN PENGUJIAN

Pada Bab ini akan dijelaskan tentang strategi dan *tool* pada pembangunan perangkat lunak. Strategi dan *tool* tersebut digunakan sebagai solusi terhadap masalah yang terdapat pada metode pembangunan perangkat lunak secara manual. Strategi dan *tool* yang akan digunakan oleh anggota tim mencakup VCS (*Version Control System*), *testing*, dan *build*. Keseluruhan strategi dan *tool* tersebut akan diimplementasikan berdasarkan praktik pembangunan perangkat lunak dengan *Continuous Integration*.

Sub bab berikutnya akan dijelaskan mengenai validasi terhadap manfaat praktik *Continuous Integration*. Parameter yang digunakan untuk melakukan validasi dari manfaat *Continuous Integration* diantaranya pengurangan resiko pembangunan perangkat lunak, pengurangan proses yang berulang, visibilitas perangkat lunak yang lebih baik, dan peningkatan kepercayaan terhadap perangkat lunak yang dibangun.

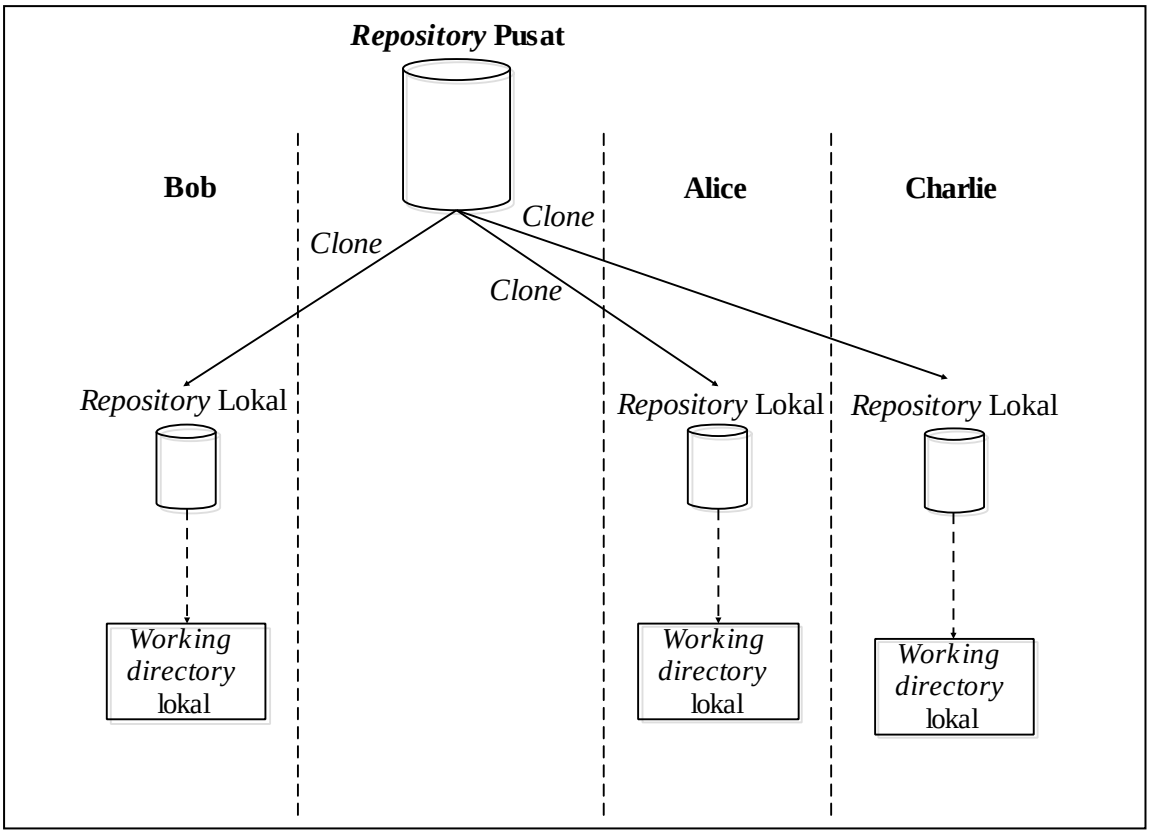
4.1 Strategi dan *tool* VCS

Penyimpanan versi dari setiap anggota tim dilakukan secara terpisah dan masih menggunakan cara yang manual, sehingga tim tersebut memerlukan strategi dan *tool* untuk melakukan penyimpanan versi. Strategi yang akan digunakan untuk melakukan penyimpanan versi pada tim tersebut adalah *distributed*, karena setiap perubahan yang terjadi pada hasil pekerjaan setiap anggota tim akan disimpan pada penyimpanan lokal terlebih dahulu.

Untuk menghindari penyimpanan versi perubahan kode program ke dalam *folder* baru, maka diperlukan sebuah *tool VCS* yang terdistribusi. Salah satu *tool VCS* yang terdistribusi adalah Git. Dengan Git setiap perubahan yang dilakukan anggota tim dapat disimpan berdasarkan waktu perubahannya secara otomatis.

Setiap versi kode program dari anggota tim akan dikumpulkan pada sebuah tempat penyimpanan versi (*repository*) terpusat. Penyimpanan versi terpusat dilakukan agar setiap anggota tim dimudahkan untuk mengelola versi kode programnya. Versi kode program dari setiap anggota tim harus dimulai dari *state* pada *working directory* yang sama, agar anggota tim dapat terhindar dari perbedaan *environment* yang mengakibatkan kesalahan pada pembangunan perangkat lunak. Untuk menyamakan *state* pada *working directory*, setiap anggota tim harus melakukan *clone* terhadap *repository* pusat. Dengan

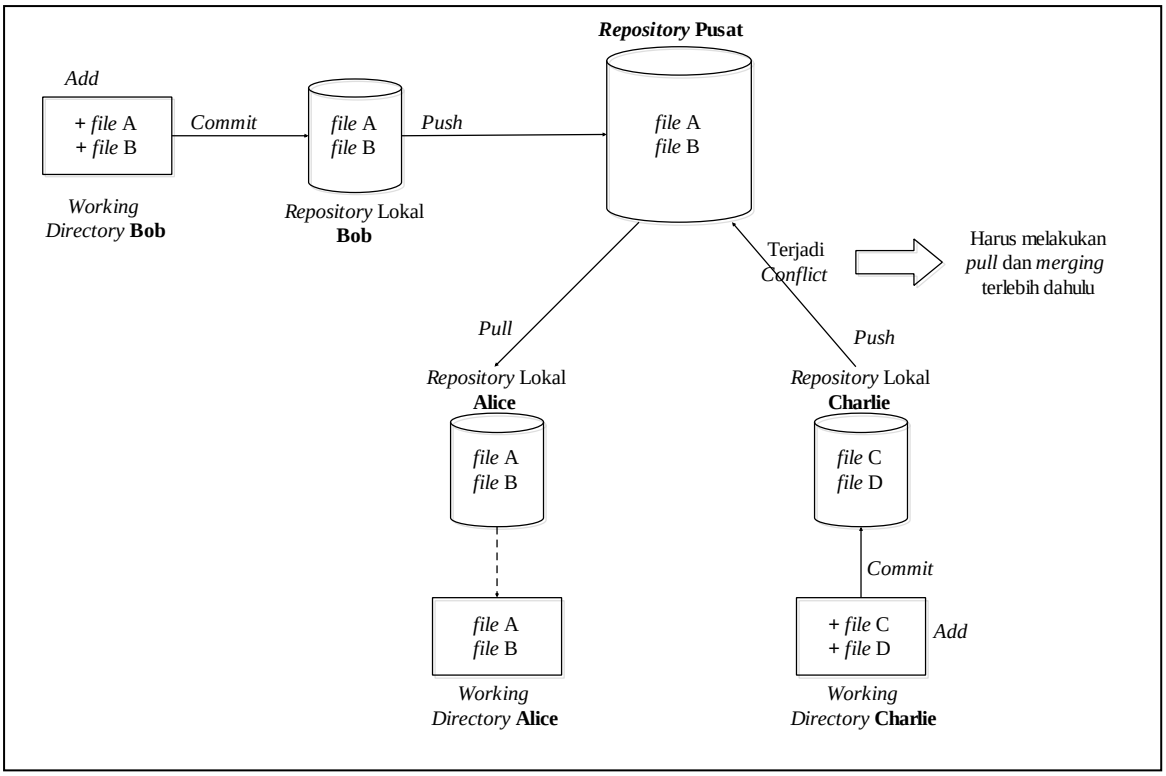
menjalankan perintah *clone*, *repository* dari setiap anggota tim dapat tersinkronisasi dengan *repository* pusat.



Gambar 4.1 *Clone repository* pusat

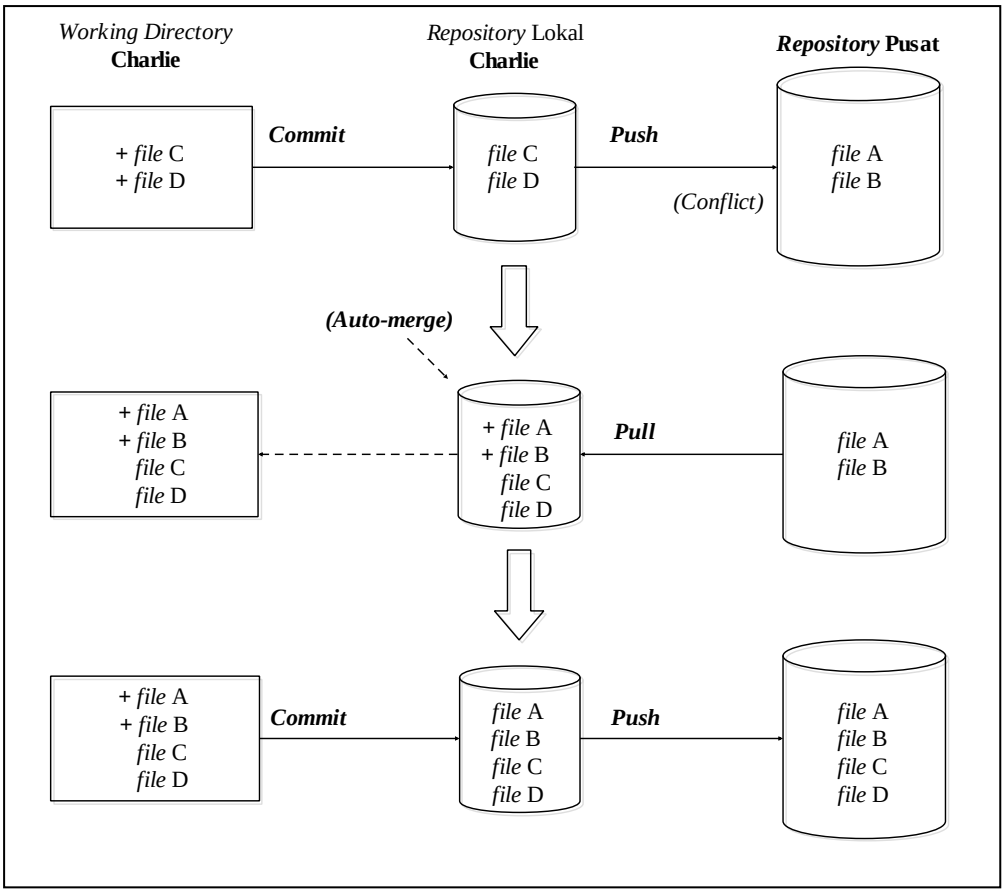
Anggota tim yang sudah melakukan *cloning* terhadap *repository* pusat akan memiliki sebuah *working directory* lokal. Pada *working directory* lokal, setiap anggota tim akan menambahkan beberapa *file* kode program baru. *File* kode program yang ditambahkan ke *working directory*, tidak akan dikenali git secara otomatis. Oleh karena itu, anggota tim perlu menjalankan perintah *add* setelah melakukan penambahan *file*, agar *file* tersebut dapat dikenali git. Perubahan kode progam yang dilakukan pada *file* yang sudah dikenali git, dapat disimpan dengan menjalankan perintah *commit*, sehingga pembuatan *folder* baru untuk menyimpan versi perubahan *file* dapat dihindari.

Pada penyimpanan versi kode program terpusat, setiap versi kode program dari anggota tim akan dikumpulkan pada sebuah *repository* pusat. Versi kode program yang telah sesuai untuk diintegrasikan akan dikirim oleh anggota tim ke *repository* pusat dengan perintah *push*. Hasil gabungan kode program yang ada di *repository* pusat dapat diambil oleh anggota tim dengan menggunakan perintah *pull*.



Gambar 4.2 Penyimpanan versi terpusat

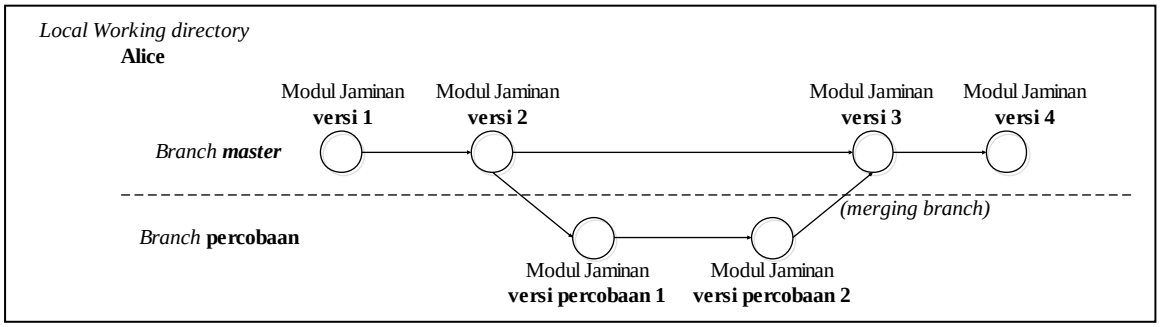
Jika anggota lain ingin melakukan *push* ke *repository* pusat, maka akan terjadi konflik. Konflik tersebut terjadi karena anggota tim tersebut tidak mempunyai perubahan terakhir dari *repository* pusat. Penyelesaian konflik dapat diselesaikan oleh anggota tim dengan melakukan *pull* dan *merging*. Git akan melakukan *merging* secara otomatis (*automerge*) apabila anggota tim mengubah *file* yang berbeda. Ketika anggota tim melakukan perubahan *file* pada baris yang sama, salah satu anggota tim harus melakukan *merging* secara manual dengan perintah *mergetool*.



Gambar 4.3 Penanganan *conflict* ketika melakukan *push*

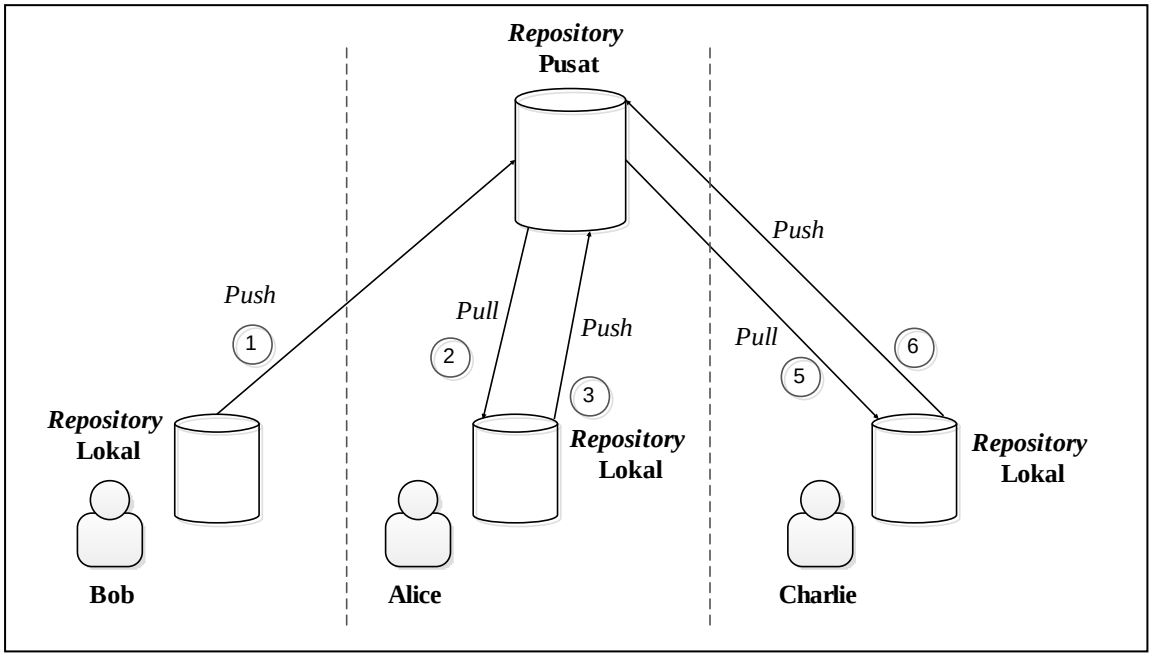
Pada proses pembangunan perangkat lunak, anggota tim umumnya tidak ingin merusak kode program yang ada di *mainline*, sehingga mereka membutuhkan cabang baru untuk melakukan proses percobaan. Proses percobaan yang dilakukan oleh anggota tim dapat menjadi versi alternatif dalam pembangunan perangkat lunak tersebut.

Menggabungkan hasil percabangan adalah pekerjaan yang sulit. Pada saat melakukan percabangan kemungkinan anggota tim mengubah banyak kode program pada beberapa *file* pada cabang tersebut. Jika terjadi konflik pada saat melakukan penggabungan cabang, maka anggota tim membutuhkan ketelitian yang tinggi untuk menyelesaikan konflik tersebut.



Gambar 4.4 Penggabungan cabang

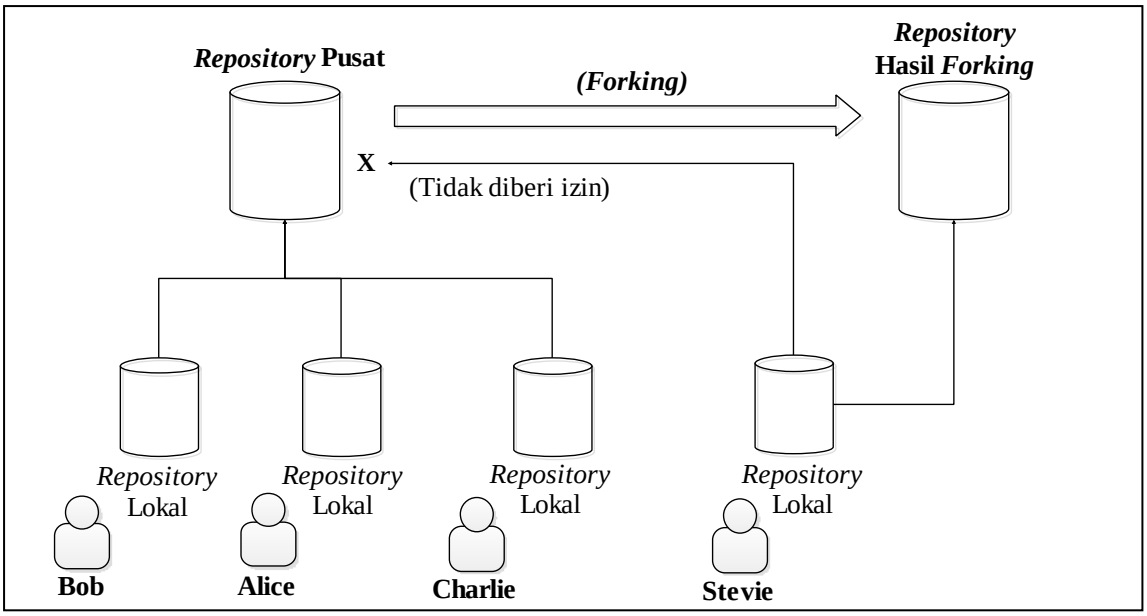
Untuk menghindari resiko yang terjadi saat penggabungan cabang, maka strategi yang akan digunakan anggota tim pada proses pembangunan perangkat lunak adalah *centralized workflow*. *Centralized workflow* adalah alur kerja pada git yang hanya bekerja pada satu *repository* pusat dengan satu cabang utama yaitu *master*.



Gambar 4.5 Strategi *Centralized Workflow*

Jika anggota tim ingin mengembangkan perangkat lunak tertentu namun tidak memiliki hak akses untuk menyimpan perubahan *source code*, maka anggota tim dapat memanfaatkan fitur *fork* yang ada pada Github. Anggota tim akan leluasa untuk mengembangkan perangkat lunak ke arah yang diinginkan di *repository* salinan dari repo yang di-*fork*.

Pada saat anggota tim memiliki perbedaan prinsip pada arah pengembangan perangkat lunak, maka anggota tim dapat melakukan *forking* pada github. Dengan *forking*, anggota tim dapat memiliki salinan repo dari *repository* pusat. Ketika anggota tim tersebut ingin memberikan kode programnya ke *repository* pusat, akan terjadi kegagalan, karena *repository* pusat tidak mengizinkan anggota tim tersebut untuk menyimpan perubahan kode program ke repo pusat, sehingga anggota tim harus melakukan *pull request* ke repo pusat terlebih dahulu. Ketika anggota tim repo pusat menerima *pull request* dari anggota tim tersebut, maka anggota tim repo pusat akan melakukan *merging*.



Gambar 4.6 *Forking* pada *repository* pusat

4.2 Strategi dan *tool Testing*

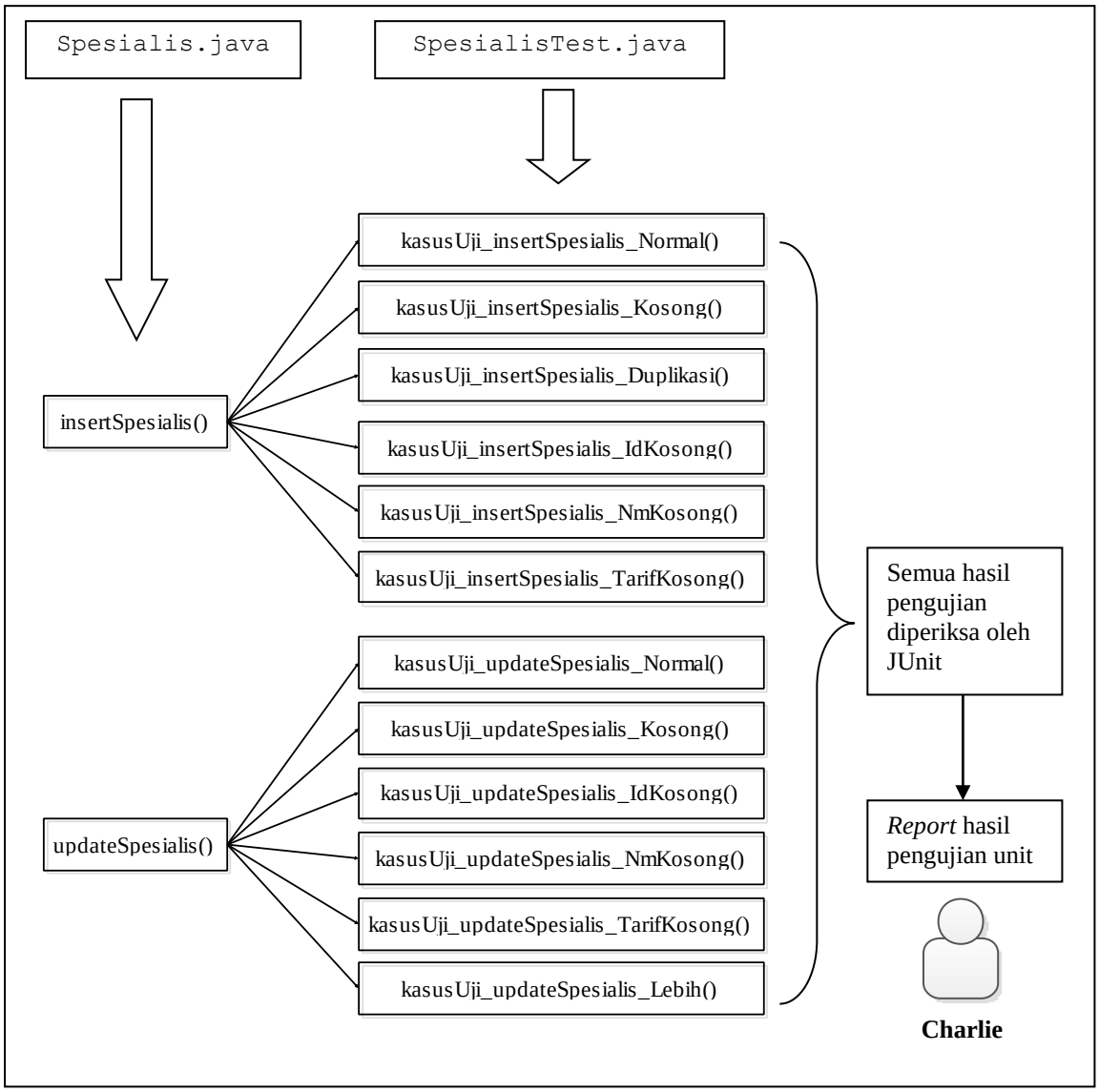
Kode program yang disimpan di *repository* pusat akan cenderung berubah mengikuti spesifikasi kebutuhan perangkat lunak yang diinginkan *customer*. Perubahan pada kode program tersebut dapat berpotensi dalam membuat kesalahan pada perangkat lunak. Untuk memastikan bahwa perangkat lunak yang sedang dibangun minim dari kesalahan, maka anggota tim perlu melakukan pengujian perangkat lunak. Dengan melakukan pengujian tersebut, kode program yang ditulis anggota tim dapat dipastikan memberikan *output* yang benar, sesuai dengan *input* yang diberikan.

Pada proses pembangunan perangkat lunak, anggota tim dituntut untuk beroperasi dengan konsep *feedback* yang cepat terhadap kesalahan. Dengan semakin cepatnya *feedback* terhadap kesalahan yang dilakukan anggota tim, maka pembangunan perangkat lunak tersebut akan semakin efisien. Agar anggota tim dapat menerapkan konsep *feedback* yang cepat terhadap kesalahan pada proses pembangunan perangkat lunak, maka dibutuhkan strategi dalam melakukan pengujian.

Strategi yang akan digunakan untuk melakukan pengujian kode program pada level unit adalah dengan membuat kasus uji setelah anggota tim selesai membuat unit kode program utamanya. Dengan strategi tersebut, anggota tim dapat mengetahui kesalahan kode program pada level unit dengan cepat, sehingga perbaikan dapat dilakukan sesegera mungkin.

Untuk menghindari pengujian kode program pada level unit secara manual dan berulang kali, maka diperlukan *tool unit testing*. Salah satu *tool* yang dapat digunakan

untuk melakukan pengujian unit adalah JUnit. Dengan JUnit pengujian unit yang dilakukan anggota tim dapat lebih efisien, karena yang akan memeriksa semua hasil kasus uji adalah JUnit. Selain itu, JUnit akan memberikan *report* dari hasil setiap kasus uji secara otomatis kepada pada anggota tim.

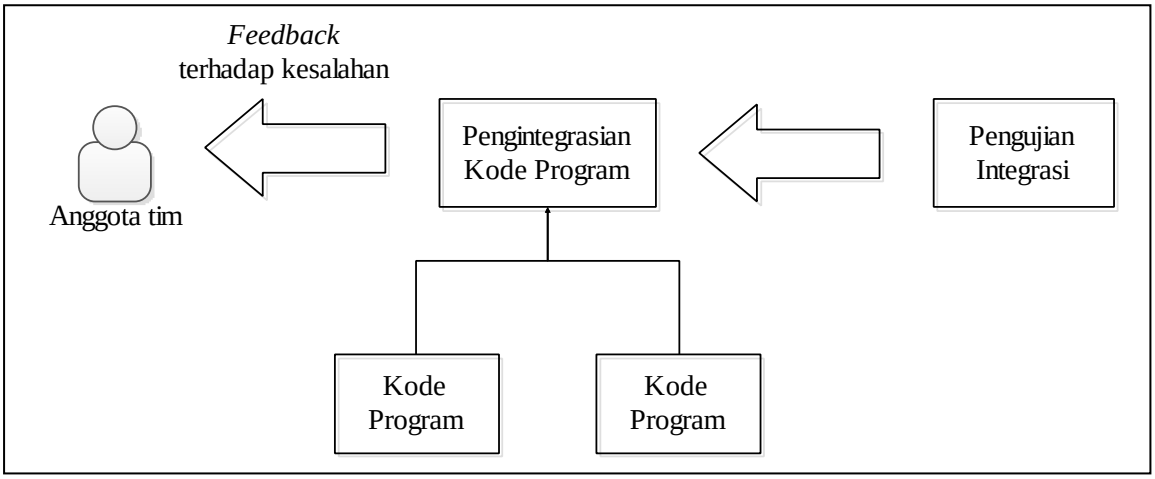


Gambar 4.7 Pengujian unit dengan menggunakan JUnit

Kode program yang telah selesai dilakukan pengujian unit, akan diintegrasikan menjadi modul-modul yang kecil. Modul-modul kecil tersebut akan diintegrasikan lagi menjadi modul yang lebih besar, hingga menjadi satu kesatuan aplikasi. Kode program yang sudah diuji pada level unit belum tentu terlepas dari kesalahan ketika diintegrasi, karena setiap unit kode program mungkin saja memiliki dependensi terhadap unit kode program yang lain. Kemungkinan kesalahan yang terjadi pada pengintegrasian kode

program akan semakin tinggi, seiring dengan rutinnya pengintegrasian kode program yang dilakukan anggota tim.

Untuk meminimalisasi kesalahan pada saat pengintegrasian kode program, anggota tim memerlukan strategi pengujian integrasi. Strategi yang akan digunakan anggota tim untuk melakukan pengujian kode program pada level integrasi adalah *incremental*. Pada strategi *incremental*, pengujian integrasi akan dilakukan anggota tim secara rutin, sehingga *feedback* terhadap kesalahan dari pengintegrasian kode program akan semakin cepat.

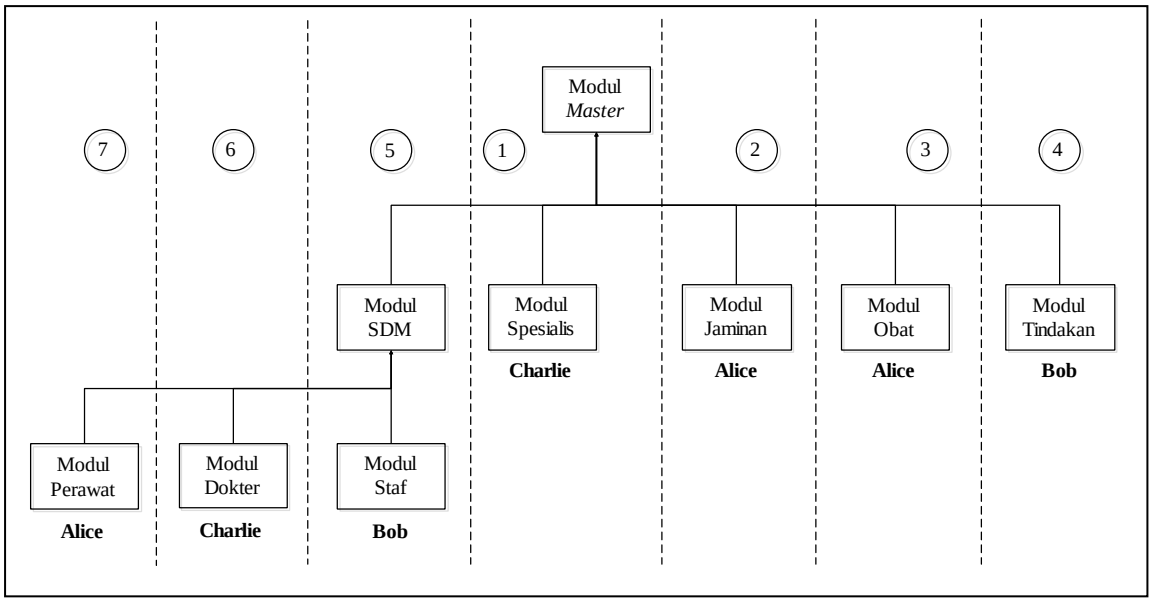


Gambar 4.8 Strategi pengujian integrasi secara *incremental*

Modul besar pada perangkat lunak yang dibangun oleh anggota tim, memiliki banyak dependensi terhadap modul-modul yang kecil. Jika anggota tim akan mengintegrasikan modul dari yang paling besar terlebih dahulu, maka anggota tim perlu membuat *stubs* pada pengujian integrasi. *Stub* digunakan anggota tim untuk dijadikan modul pengganti yang berperan sebagai modul yang akan dipanggil oleh modul yang sedang diuji. Anggota tim harus membuat semua kemungkinan *stub* yang akan dipanggil oleh modul tersebut, agar tingkat kemungkinan kesalahan pada pengintegrasian kode program dapat diminimalisasi. *Stub* yang sudah dibentuk anggota tim tidak akan digunakan lagi ketika modul-modul kecil penyusun modul besar tersebut telah selesai dibuat.

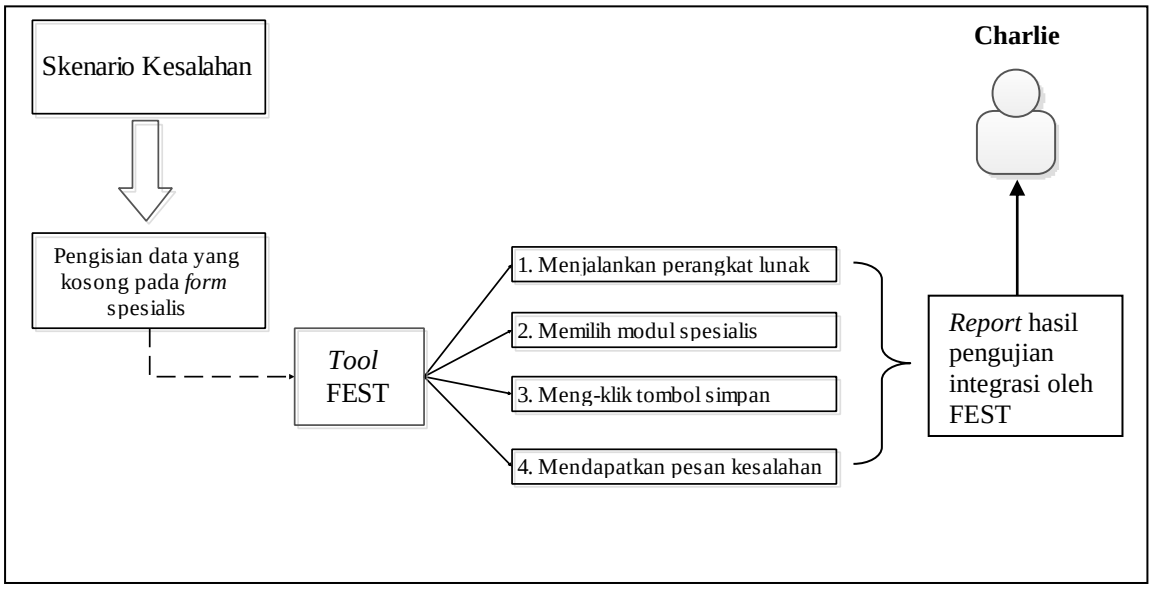
Anggota tim akan memerlukan *effort* yang besar untuk membuat *stubs* ketika melakukan pengujian integerasi. Oleh karena itu, strategi *incremental* yang akan digunakan untuk mengintegrasikan kode program adalah *bottom-up*. Dengan strategi *incremental* menggunakan *bottom-up*, anggota tim tidak perlu lagi membuat *stub*, karena

pengujian integrasi akan dimulai dari modul-modul terkecil yang sering dipanggil oleh modul yang lain.



Gambar 4.9 Strategi pengujian integrasi secara *incremental* dengan *bottom-up*

Untuk menghindari pengujian kode program pada level integrasi secara manual dan berulang kali, maka diperlukan *tool integration testing*. Salah satu *tool* yang dapat digunakan untuk melakukan pengujian integrasi adalah FEST. Dengan FEST, pengujian integrasi yang dilakukan anggota tim dapat lebih efisien, karena yang akan melakukan semua rangkaian kasus uji berdasarkan skenario kesalahan adalah FEST. Selain itu, FEST akan memberikan *report* pengujian integrasi dari semua skenario kesalahan secara otomatis kepada anggota tim.



Gambar 4.10 Pengujian integrasi dengan menggunakan FEST