

BAB III

KONSEP UMUM PEMBANGUNAN PERANGKAT LUNAK SECARA MANUAL DAN MENGGUNAKAN *TOOLSET*

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

3.1. Konsep umum pembangunan perangkat lunak secara manual

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

3.1.1. Konsep penyimpanan versi secara manual

Pada sub bab ini akan dijelaskan tentang detail penyimpanan versi yang umum dilakukan tim tanpa menggunakan bantuan *tool version*

control. Penyimpanan versi dilakukan tim untuk menyimpan *history* dari setiap perubahan modul. Tim yang tidak menggunakan bantuan *tool version control* umumnya akan menduplikasi modul sebelum mengubah modul tersebut. Hasil duplikasi modul digunakan tim sebagai *backup* untuk melakukan *rollback* terhadap modul. Untuk membedakan hasil dari setiap duplikasi modul, umumnya tim akan melakukan penamaan versi.

[GAMBAR]

Gambar 3-1. Penyimpanan versi dengan cara 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 ditambahkan ke dalam modul harus diuji. Pengujian unit tersebut 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 automated testing* akan membuat *driver* pengujian pada setiap kode pengujian. *Driver* pengujian digunakan setiap anggota tim untuk mengeksekusi kode pengujian tersebut. Ketika terjadi kesalahan pada hasil dari satu atau lebih kode pengujian tersebut, anggota tim akan memperbaikinya dan mengeksekusi kembali semua *driver* pengujian dari awal.

[GAMBAR]

Gambar 3-2. Pengujian unit dengan cara manual

Modul-modul hasil pekerjaan setiap anggota tim yang telah dilakukan pengujian unit, umumnya akan diintegrasikan oleh salah

satu anggota tim yang bertugas sebagai *integrator*. Modul dari hasil pengintegrasian modul setiap anggota tim tersebut harus diuji. Pengujian integrasi akan dilakukan *integrator* untuk memastikan bahwa *functional requirement* dari modul hasil integrasi dapat dieksekusi serta minim dari kesalahan.

Sebelum *integrator* melakukan pengintegrasian modul, umumnya tim akan menentukan strategi pengintegrasian modul terlebih dahulu. Strategi pengintegrasian modul yang dilakukan secara rutin, diklasifikasikan menjadi tiga cara, yaitu *top-down*, *bottom-up*, dan *sandwich*. Pada strategi *top-down*, *integrator* akan mengintegrasikan modul perangkat lunak dari tingkat atas ke tingkat bawah. Strategi pengintegrasian *top-down* umumnya digunakan ketika modul pada tingkat atas tidak memiliki banyak dependensi terhadap modul tingkat bawah. Tim yang menggunakan strategi *top-down* perlu membuat *stubs* sebagai pengganti modul-modul tingkat bawah yang belum dibuat. *Stubs* tersebut akan digunakan *integrator* untuk menguji hasil pengintegrasian modul-modul pada tingkat atas. Ketika tim telah selesai membuat modul-modul pada tingkat bawah, *stubs* tersebut tidak akan digunakan kembali.

[GAMBAR]

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

Pada strategi *bottom-up*, *integrator* akan mengintegrasikan modul perangkat lunak dari tingkat bawah ke tingkat atas. Strategi *bottom-up* umumnya digunakan ketika modul pada tingkat atas memiliki banyak dependensi terhadap modul pada tingkat bawah. Tim yang menggunakan strategi *bottom-up* tidak lagi memerlukan *stubs*, karena modul-modul pada tingkat bawah telah dibuat sejak awal. Untuk menguji hasil pengintegrasian modul-modul pada tingkat bawah, tim memerlukan *driver* sebagai pengganti modul tingkat atas yang belum

dibuat. *Driver* tersebut akan digunakan *integrator* untuk memanggil modul hasil pengintegrasian modul pada tingkat bawah.

[GAMBAR]

Gambar 3-4. Pengujian integrasi dengan strategi *bottom-up*

Pada strategi *sandwich*, *integrator* akan mengintegrasikan modul dengan dua cara, yaitu *top-down* dan *bottom-up*. Anggota tim yang bekerja dari modul tingkat atas akan membuat *stubs* untuk menggantikan modul-modul tingkat bawah yang belum selesai dikerjakan. Sedangkan anggota tim yang bekerja dari modul tingkat bawah akan membuat *driver* untuk menggantikan modul-modul tingkat atas yang belum selesai dikerjakan.

[GAMBAR]

Gambar 3-5. Pengujian integrasi dengan strategi *sandwich*

3.1.3. Konsep eksekusi *build* secara manual

Setelah *integrator* melakukan pengujian integrasi dari hasil penggabungan modul setiap anggota tim, *integrator* akan mengeksekusi *build* untuk mendapatkan paket aplikasi. Paket aplikasi tersebut berisi *file executable* atau *file* yang siap dipakai oleh *user*. Umumnya, *integrator* yang tidak menggunakan bantuan *tool automated build* akan melakukan proses *build* secara manual. Proses *build* tersebut diantaranya inisialisasi *path* kode program, penghapusan *file* hasil kompilasi, kompilasi kode program, dan pembuatan paket aplikasi yang siap pakai. Rangkaian proses tersebut dilakukan *integrator* secara berulang kali setiap menggabungkan modul setiap anggota tim.

[GAMBAR]

Gambar 3-6. Eksekusi *build* dengan cara manual

3.1.4. Konsep pengintegrasian modul secara manual

Pengintegrasian kode program yang dilakukan secara manual

3.2. Konsep umum pembangunan perangkat lunak menggunakan *toolset*

Konsep pake toolset itu kaya gimana?

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

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

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

3.2.4. Konsep pengintegrasian kode program dengan *tool continuous integration*