BAB II
STUDI LITERATUR

Pada bab ini akan dijelaskan tentang gambaran umum praktik *continuous integration* yang diotomasi dengan menggunakan bantuan *toolset*. Gambaran umum tersebut akan digunakan sebagai acuan dalam membuat kerangka kerja pembangunan perangkat lunak dengan *automated continuous integration*. Praktik *automated continuous integration* mencakup tiga praktik lain yaitu *version control system*, *automated testing*, dan *automated build*. Pada setiap praktik akan dijelaskan tentang perbandingan dari beberapa *tools* yang dapat mendukung praktik tersebut.

2.1. *Automated continuous integration*
2.1.1. Tujuan *automated continuous integration*
2.1.1.1. Mengurangi risiko pembangunan perangkat lunak
2.1.1.2. Mengurangi proses manual yang berulang
2.1.1.3. Membuat visibilitas proyek menjadi lebih baik
2.1.1.4. Meningkatkan rasa percaya diri tim terhadap perangkat lunak
2.1.2. Prasyarat *automated continuous integration*
2.1.3. *Tools* pendukung *automated continuous integration*

2.2. *Version control system*
2.2.1. Tujuan *version control system*
2.2.2. Metode *version control system*
2.2.2.1. *Local version control system*
2.2.2.2. *Centralized version control system*
2.2.2.3. *Distributed version control system*
2.2.3. *Tools* pendukung *version control system*

2.3. *Automated testing*

Keberhasilan pembangunan *software* sangat ditentukan oleh hasil dari pengujian. Jika proses pengujian dilakukan dengan benar, maka *software* yang telah melewati pengujian tersebut dapat memiliki kualitas yang baik dan dapat dipertanggungjawabkan. Menurut Glenford J. Myers, *software testing* adalah suatu proses atau serangkaian proses pengujian yang dirancang oleh *developer* untuk memastikan bahwa kode program berfungsi sesuai dengan apa yang dirancang **[Myers]**. Inti dari *software testing* adalah verifikasi dan validasi *software*. Menurut Roger S. Pressman, verifikasi mengacu pada serangkaian kegiatan yang memastikan bahwa *software* telah mengimplementasi sebuah fungsi tertentu dengan cara yang benar. Sedangkan validasi mengacu pada satu set aktifitas yang memastikan bahwa *software* yang dibangun telah sesuai dengan kebutuhan *customer* **[Pressman]**.

Pengujian yang dilakukan secara manual membutuhkan prosedur baku dan ketelitian dari orang yang berperan sebagai penguji. Pada pembangunan perangkat lunak dengan *continuous integration*, proses pengujian akan dilakukan secara berulang kali, sehingga pengujian manual rawan terhadap kesalahan. *Automated testing* adalah proses pengujian *software* yang menggunakan bantuan *tool* pengujian. Proses pengujian dirancang agar dapat dilakukan secara otomatis oleh *tool* tersebut. *Tool* pengujian sangat diperlukan untuk membantu proses pengujian yang sifatnya berulang dan banyak.

2.3.1. Tujuan *automated testing*

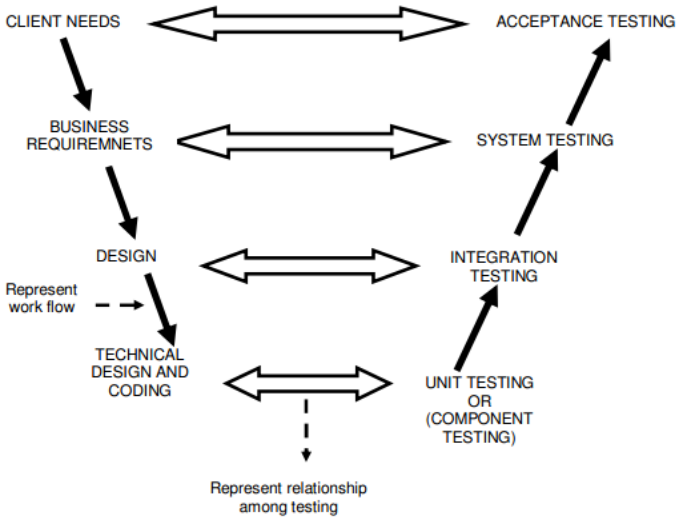
Tujuan penerapan praktik *automated testing* antara lain untuk mengotomasi proses eksekusi pengujian, proses analisis hasil pengujian, proses simulasi interaktif, dan proses pembuatan kerangka pengujian **[Peter]**.

1. Otomasi proses eksekusi pengujian. Dengan menerapkan praktik *automated testing*, semua pengujian dapat dieksekusi secara otomatis oleh *tool* pengujian. Untuk mengotomasi proses tersebut *developer* perlu membuat cakupan rangkaian pengujian terlebih dahulu. Dengan otomasi eksekusi pengujian, *developer* tidak lagi mengeksekusi pengujian satu per satu.

2. Otomasi proses analisis hasil pengujian. *Developer* dapat dimudahkan dalam menganalisis hasil pengujian perangkat lunak. Dengan menerapkan praktik *automated testing*, semua informasi hasil pengujian akan ditampilkan oleh *tool* pengujian kepada *developer* secara otomatis.
3. Otomasi proses simulasi interaktif. Produk perangkat lunak yang memerlukan interaksi dengan pengguna, tidak dapat diuji hanya dengan perintah baris kode saja. Untuk menguji antarmuka perangkat lunak tersebut, *tool* pengujian dapat digunakan untuk berinteraksi dengan antarmuka perangkat lunak secara otomatis.
4. Otomasi pembuatan kerangka pengujian. Untuk menguji perangkat lunak umumnya *developer* perlu membuat kerangka pengujian terlebih dahulu. Kerangka pengujian tersebut digunakan *developer* sebagai acuan dalam menguji perangkat lunak. Dengan *tool* pengujian, kerangka pengujian tersebut dapat dihasilkan secara otomatis, sehingga *developer* tidak lagi membuat kerangka pengujian secara manual.

2.3.2. Tingkatan *testing*

Menurut Patrick Oladimeji, untuk meningkatkan kualitas pengujian perangkat lunak dan menghasilkan metodologi pengujian yang sesuai di beberapa proyek, proses pengujian dapat diklasifikasikan ke tingkat yang berbeda **[Oladimeji]**. Tingkatan pengujian memiliki struktur hirarki yang tersusun dari bawah ke atas (lihat **gambar N-NN**). Setiap tingkatan pengujian ditandai dengan jenis *environment* yang berbeda misalnya *user*, *hardware*, data, dan *environtment variable* yang bervariasi dari setiap proyek. Setiap tingkatan pengujian yang telah dilakukan dapat merepresentasikan *milestone* pada suatu perencanaan proyek **[Ehmer]**.



Gambar N-NN. Tingkatan *software testing*

2.3.3.1. *Unit testing*

Unit testing juga dikenal sebagai pengujian komponen atau bagian terkecil dari perangkat lunak. Pengujian unit berada di tingkat pertama atau pengujian tingkat terendah. Pada tingkat pengujian unit, masing-masing unit *software* akan diuji. Pengujian unit umumnya dilakukan oleh seorang *programmer* yang membuat unit atau modul tertentu. *Unit testing* membantu menampilkan *bug* yang mungkin muncul dari suatu kode program. *Unit testing* berfokus pada implementasi dan pemahaman yang detil tentang sistem spesifikasi fungsional.

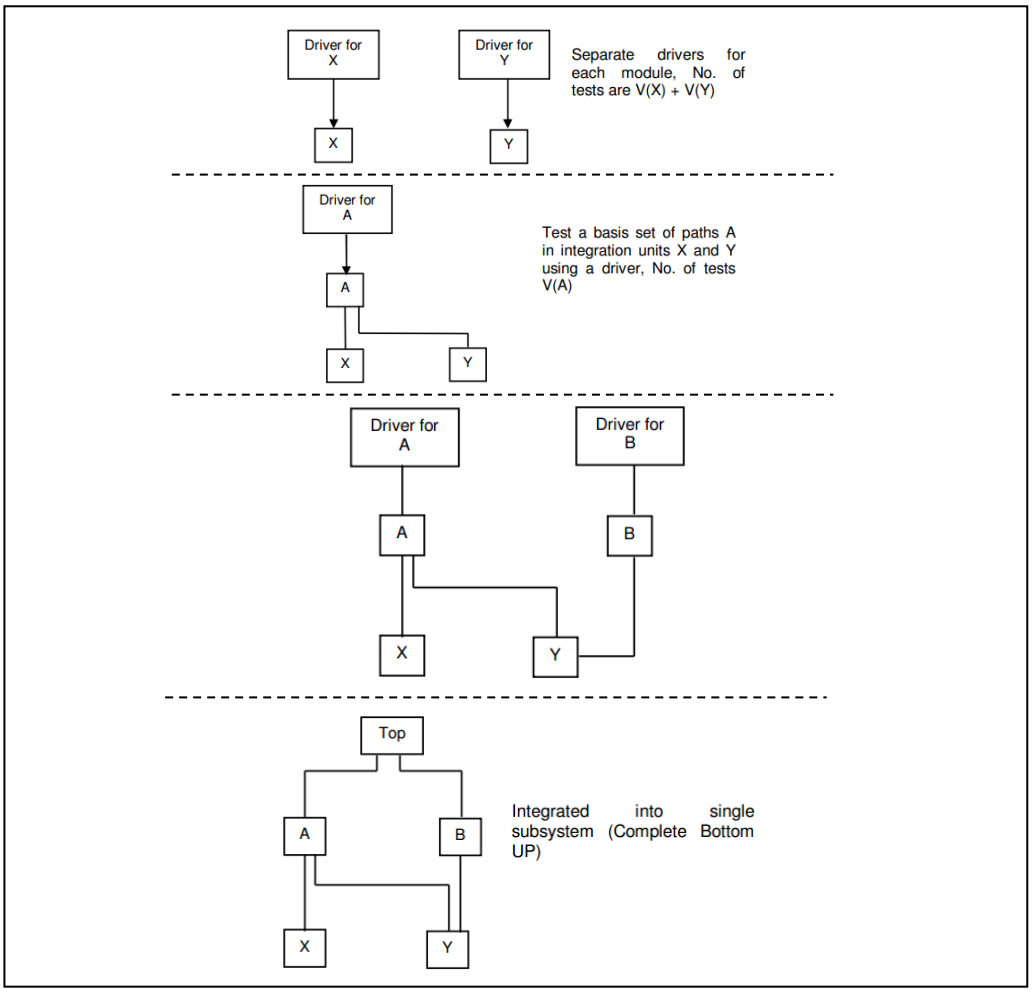
2.3.3.2. *Integration testing*

Integration testing adalah pengujian yang melibatkan penggabungan unit dari suatu program. Tujuan dari pengujian integrasi adalah untuk memverifikasi fungsional pogram serta kinerja dan kehandalan persyaratan yang ditempatkan pada *item* desain utama. Sekitar 40% dari kesalahan perangkat lunak dapat ditemukan selama pengujian integrasi, sehingga kebutuhan *integration testing* tidak dapat diabaikan **[Ehmer]**. Tujuan utama pengujian integrasi adalah untuk meningkatkan struktur integrasi secara keseluruhan sehingga memungkinkan pengujian yang detil pada setiap tahap dan meminimalkan kegiatan yang sama.

Pengujian integrasi secara *incremental* dapat diklasifikasikan menjadi dua yaitu *bottom-up* dan *top-down*.

1. *Bottom-up integration*

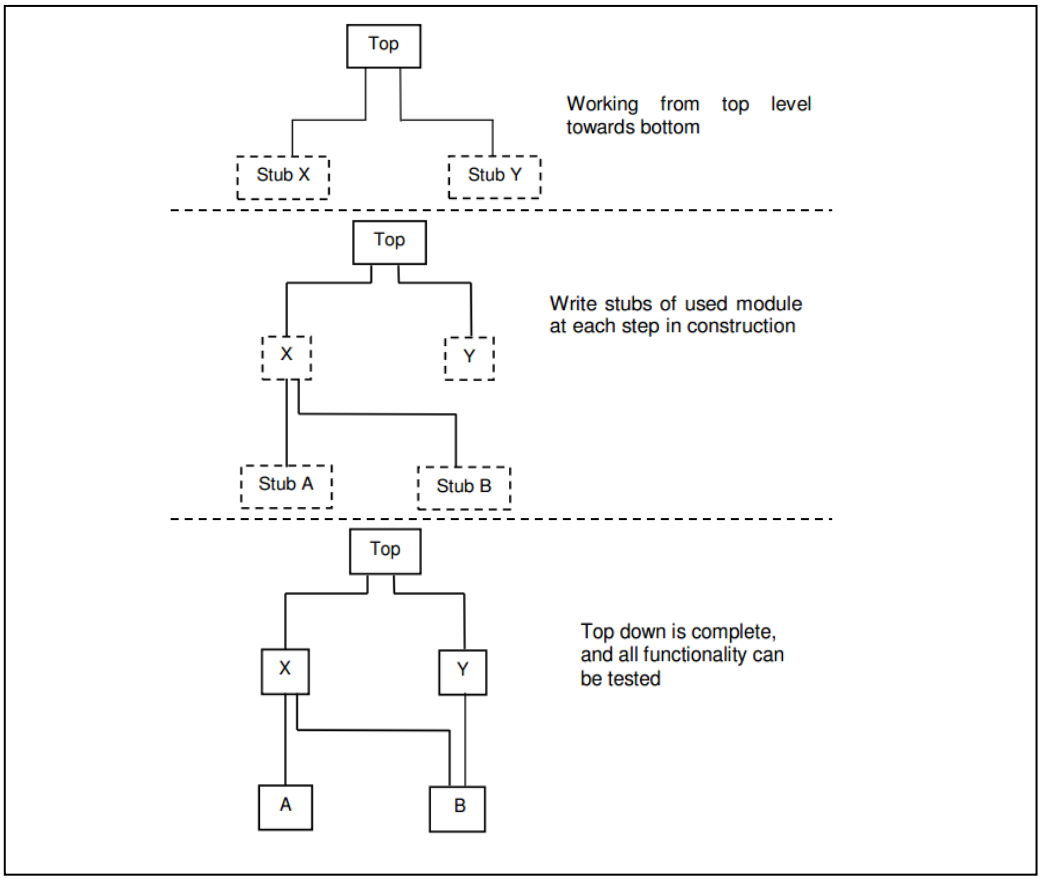
   Pada pendekatan *bottom-up integration*, pengujian dimulai dari bagian modul yang lebih rendah (lihat **gambar N-NN**). *Bottom-up integration* menggunakan *test driver* untuk mengeksekusi pengujian dan memberikan data yang sesuai untuk modul tingkat yang lebih rendah. Pada setiap tahap *bottom-up integration*, unit di tingkat yang lebih tinggi diganti dengan *driver* (*driver* membuang potongan-potongan kode yang digunakan untuk mensimulasikan prosedur panggilan untuk modul *child*) **[Ehmer]**.



Gambar N-NN. Pengujian integrasi dengan strategi *bottom-up*

2. *Top-down integration*

Pengujian *top-down integration* dimulai dari *parent* modul dan kemudian ke modul *child*. Setiap tingkat modul yang lebih rendah, dapat dihubungkan dengan *stub* atau pengganti modul tingkat bawah yang belum ada (lihat **gambar N-NN**). *Stub* yang ditambahkan pada tingkat yang lebih rendah akan diganti dengan komponen yang sebenarnya. Pengujian tersebut dapat dilakukan mulai dari luasnya terlebih dahulu ataupun kedalamannya. Penguji dapat memutuskan jumlah *stub* yang harus diganti sebelum tes berikutnya dilakukan. Sebagai *prototipe*, sistem dapat dikembangkan pada awal proses proyek. *Top-down integration* dapat mempermudah pekerjaan dan desain *defect* dapat ditemukan serta diperbaiki lebih awal. Tetapi, satu kelemahan dengan pendekatan *top-down* adalah *developer* perlu bekerja ekstra untuk menghasilkan sejumlah besar *stub* **[Ehmer]**.



Gambar N-NN. Pengujian integrasi dengan strategi *top-down*

2.3.3.3. *System testing*

Tingkatan utama pengujian atau inti dari pengujian adalah pada tingkat *system testing* **[Ehmer]**. Fase ini menuntut keterampilan tambahan dari seorang *tester* karena berbagai teknik struktural dan fungsional dilakukan pada fase ini. Pengujian sistem dilakukan ketika sistem telah di-*deploy* ke lingkungan standar dan semua komponen yang diperlukan telah dirilis secara *internal*. Selain uji fungsional, pengujian sistem dapat mencakup konfigurasi pengujian, keamanan, pemanfaatan optimal sumber daya dan kinerja sistem. *System testing* diperlukan untuk mengurangi biaya dari perbaikan, meningkatkan produktifitas dan mengurangi risiko komersial. Tujuan utama dari pengujian sistem adalah untuk mengevaluasi sistem secara keseluruhan dan bukan per bagian.

2.3.3.4. *Acceptance testing*

A*cceptance testing* adalah tingkat pengujian perangkat lunak yang menguji sistem untuk menilai bahwa fungsi-fungsi yang ada pada sistem tersebut telah berjalan dengan benar dan sesuai dengan kebutuhan pengguna. Umumnya, pada tingkat *acceptance testing* diperlukan keterlibatan dari satu atau lebih pengguna untuk menentukan hasil pengujian. *Acceptance testing* dilakukan sebelum membuat sistem yang tersedia untuk penggunaan aktual. *Acceptance testing* juga dapat melibatkan pengujian kompatibilitas apabila sistem dikembangkan untuk menggantikan sistem yang lama. Pada tingkat *acceptance testing*, pengujian harus mencakup pemeriksaan kualitas secara keseluruhan, operasi yang benar, skalabilitas, kelengkapan, kegunaan, portabilitas dan ketahanan komponen fungsional yang disediakan oleh sistem perangkat lunak.

2.3.3. *Tools* pendukung *automated testing*

Berikut adalah beberapa *tools* pendukung praktik *automated testing* berdasarkan bahasa pemrograman yang dapat digunakan (lihat **tabel N-NN**) dan kelebihan fitur dari setiap *tools* (lihat **tabel N-NN**). Daftar *tools* tersebut dapat digunakan sebagai referensi dalam menentukan *tool* pada praktik *automated testing*.

Tabel N-NN. *Tools* pendukung praktik *automated testing* berdasarkan kode pemrograman

No	*Testing tools*	Bahasa pemrograman		
		Java	PHP	.NET
1	JUnit	√		
2	FEST	√		
3	TestComplete	√	√	√
4	PHPUnit		√	
5	Selenium IDE	√	√	√
6	NUnit			√
7	JMeter	√	√	√

Tabel N-NN. *Tools* pendukung praktik *automated testing* berdasarkan fitur

No	*Testing tools*	Fitur							
		Desktop base	*Web base*	*GUI test*	*Open source*	Tingkatan pengujian			
						Unit	Integrasi	Sistem	*Acceptance*
1	JUnit	√	√		√	√	√		
2	FEST	√		√	√	√	√		
3	TestComplete	√	√	√		√	√	√	√
4	PHPUnit	√	√		√	√	√		
5	Selenium IDE		√	√	√	√	√	√	√
6	NUnit	√	√		√	√	√		
7	JMeter		√		√			√	

Referensi testing:

[Peter] Peter A. Vogel, An Integrated General Purpose Automated Test Environment

2.4. *Automated build*

2.4.1. Tingkatan *automated build*

2.4.1.1. *Private build*

2.4.1.2. *Integration build*

2.4.1.3. *Release build*

2.4.2. *Tools* pendukung *automated build*