BAB 2
LANDASAN TEORI

Bab ini menjelaskan teori-teori yang digunakan sebagai landasan dalam penelitian Tugas Akhir. Di dalamnya juga akan dijelaskan istilah-istilah pengujian yang digunakan dalam laporan ini.

2.1 Uji Perangkat Lunak (Software Testing)

Penjelasan mengenai sub bab ini diambil dari buku Danniel Galin: Software Quality Assurance from Theory to Implementation Chapter 9.

Uji perangkat lunak atau yang lebih umum disebut *software testing* (penjelasan selanjutnya akan menggunakan istilah *software testing*) memiliki beberapa definisi oleh berbagai sumber. Definisi dari *software testing* menurut Myers' (1979, Chapter 10) adalah:

"*Testing is the process of executing a program with intention of finding errors.*"

Berdasarkan definisi tersebut, aktivitas yang terjadi dalam *software testing* terdiri dari pengujian kode program hingga kegiatan percobaan terhadap perangkat lunak (*software*) yang sudah berfungsi.

Definisi lain yang lebih formal dan terkontrol adalah dua definisi untuk *testing* yang dikeluarkan oleh IEEE Std 610.12 (IEEE, 1990), yaitu:

"(1) *The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component. (2) The process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software item.*"

Dengan melihat definisi *software testing* di atas, maka dapat disimpulkan bahwa menjalankan (*running*) program sebagai bagian dari proses pengujian tidak diperlukan.

Definisi yang lebih formal dan sesuai karakteristik dari *software testing* adalah sebagai berikut:

"*Software testing is a **formal** process carried out by a **specialized testing team** in which a software unit, several integrated software units or an entire software package are examined by **running the programs** on a computer. All the associated tests are performed according to **approved test procedures** on approved test cases.*"

Beberapa kata dan kalimat yang terdapat dalam definisi di atas dapat dikaji sehingga dapat dibedakan antara *software testing* dengan metode-metode penjaminan mutu perangkat lunak yang lain. Penjelasannya sebagai berikut:

1. *Formal*: Rencana pengujian perangkat lunak merupakan bagian dari pengembangan proyek dan perencanaan kualitas perangkat lunak, dijadwalkan dan menjadi inti dari perjanjian antara tim pengembang perangkat lunak dengan konsumen. Dengan kata lain, adanya pemeriksaan tambahan dari pimpinan tim pengembang tidak dianggap sebagai *software testing*.
2. *Specialized testing team:* Sebuah tim independent atau konsultan eksternal yang bergerak khusus dalam bidang pengujian ditugaskan untuk melaksanakan tugas-tugas dalam *testing software* agar dapat menghilangkan adanya penyimpangan-penyimpangan dan untuk menjamin dihasilkannya pengujian yang efektif karena dilakukan oleh para profesional yang sudah ahli.
3. *Running the programs*: Berbagai bentuk kegiatan penjaminan kualitas perangkat lunak yang tidak memasukkan kegiatan menjalankan aplikasi (*running the software*), seperti misalnya inspeksi kode program, tidak dapat disebut sebagai sebuah pengujian (*test*).

4. *Approved test procedures:* Proses pengujian yang dilakukan berdasarkan rencana pengujian (*test plan)* dan prosedur pengujian (*testing procedures*) yang telah disetujui dipakai oleh organisasi yang mengembangkan perangkat lunak.
5. *Approved test cases: Test case* yang akan diuji didefinisikan secara penuh oleh *test plan.* Diharapkan tidak ada penambahan maupun pengurangan yang terjadi selama pengujian.

Demikian telah dijelaskan bagaimana definisi dari *software testing.* Sedangkan tujuan dari *software testing* adalah sebagai berikut:

1. Untuk mengidentifikasi dan menyatakan sebanyak mungkin *error* yang dimiliki oleh perangkat lunak yang diuji
2. Untuk membawa perangkat lunak yang diuji ke tingkat kualitas yang dapat diterima, setelah perangkat lunak tersebut mengalami pembetulan atau koreksi atas *error* yang ditemukan.
3. Untuk melaksanakan uji-uji yang dibutuhkan secara efisien dan efektif, dalam keterbatasan *budget* dan waktu penjadwalan.

2.2 Uji Terotomatisasi (Automated Testing)

Penjabaran dalam sub bab ini diambil dari buku Danniel Galin: Software Quality Assurance from Theory to Implementation Chapter 10.

Uji terotomatisasi merupakan uji perangkat lunak yang dibantu dengan alat pengujian yang juga berbentuk perangkat lunak. Faktor-faktor yang mendukung berkembangnya perangkat lunak pengujian ini antara lain: penghematan biaya pengembangan, durasi pengujian yang dipersingkat, peningkatan kecermatan dalam pelaksanaan pengujian, peningkatan akurasi pengujian, dan peningkatan laporan hasil pengujian seperti pada proses statistik.

2.2.1. Proses dalam Uji Terotomatisasi

Uji terotomatisasi membutuhkan perencanaan pengujian (*test planning*), perancangan pengujian (*test design*), *test case preparation,* penyelenggaraan pengujian, *test log* dan laporan pengujian (*test report*), *re-testing* setelah

dilakukan koreksi terhadap *error* yang ditemukan (disebut dengan *regression test*), dan *final test log* beserta laporannya termasuk laporan perbandingan (*comparative reports*). *Test log* dan laporan pengujian mungkin dilakukan berulang-ulang.

Agar lebih jelas, berikut ini tabel yang menunjukkan perbandingan antara uji terotomatisasi dengan uji manual:

Tabel 2.1 Perbandingan Uji Terotomatisasi dan Uji Manual

Fase Proses Pengujian	Uji Terotomatisasi		Uji Manual	
	Dilakukan Secara Manual (M) / Otomatis (O)	Kegiatan Utama	Dilakukan Secara Manual (M) / Otomatis (O)	Kegiatan Utama
Test Planning	M	Mempersiapkan *test plan*	M	Mempersiapkan *test plan*
Test Design	M	Mempersiapkan *test database*	M	Mempersiapkan prosedur pengujian
Test Case Preparation	M	Mempersiapkan *test case* ke dalam *test case database*	M	Mempersiapkan *test case*
Penyelenggaraan Pengujian	O	Menjalankan pengujian secara terkomputerisasi	M	Menjalankan pengujian oleh *tester*
Persiapan Test Log dan Test Reports	O	Menghasilkan output secara terkomputerisasi	M	Dipersiapkan oleh tester
Regression Test	O	Menjalankan pengujian secara terkomputerisasi	M	Menjalankan pengujian oleh *tester*
Persiapan Test Log dan Test Report termasuk Comparative Reports	O	Menghasilkan output secara terkomputerisasi	M	Dipersiapkan oleh tester

2.2.2. Jenis-jenis Uji Terotomatisasi

Terdapat beberapa jenis uji terotomatisasi yang memanfaatkan perangkat lunak pengujian, yaitu:

1. *Code auditing:* menguji kode program apakah sudah memenuhi standard dan prosedur pemrograman yang sudah dispesifikasikan.
2. *Coverage monitoring:* menghasilkan laporan seberapa besar persentase *line coverage* yang dipenuhi pada saat mengimplementasikan *test case.*
3. *Functional Tests:* menggantikan uji *correctness* secara manual. Uji ini akan dibahas lebih lanjut karena merupakan jenis uji perangkat lunak yang digunakan dalam penelitian Tugas Akhir ini.
4. *Load Tests:* melakukan pengukuran terhadap perangkat lunak dalam hal *reaction time, processing time,* dan parameter-parameter lainnya.
5. *Test management:* mendukung proses *monitoring* kinerja dari setiap *item* dalam *test case.*

2.2.3. *Functional Testing* dan *Regression Testing*

Funtional testing bukanlah pengujian yang fokus pada seberapa cepat atau seberapa lambat suatu aplikasi menjalankan fungsinya. Pengujian yang fokus kepada hal ini dinamakan *performance testing* yang tidak akan dibahas dalam laporan Tugas Akhir ini. *Functional testing* juga tidak secara langsung menilai seberapa sempurnakah suatu aplikasi. Jenis uji ini tidak mempedulikan apakah terdapat penggunaan memori yang berlebihan, kecuali jika hal tersebut menyebabkan pengguna aplikasi tidak dapat menggunakan aplikasi sesuai dengan fungsinya.

Fokus dari *functional testing* adalah untuk memverifikasi sebuah aplikasi apakah berjalan sesuai dengan yang diharapkan. Terdapat beberapa cara untuk mengukur seberapa baik suatu aplikasi saat digunakan, seperti misalnya dengan menilai seberapa baik aplikasi tersebut dalam memenuhi spesifikasi atau *requirement* yang sudah ditentukan. Dengan kata lain, apakah aplikasi melakukan apa yang diinginkan oleh pengguna.

Functional testing menghasilkan daftar *error* atau *bug* yang ditemukan. Setelah dilakukan koreksi terhadap *error* tersebut, dibutuhkan proses pengujian ulang

terhadap keseluruhan fungsi atau sebagian. Pengujian ulang inilah yang disebut dengan *regression testing. Regression testing* juga mungkin dilakukan setelah terjadi penambahan atau pengurangan fitur dalam program. Pengujian ini dilakukan secara otomatis untuk keseluruhan fungsi dalam program untuk memverifikasi apakah hasil koreksi terhadap *error* dan atau hasil penambahan atau pengurangan fitur pada program telah dilakukan dengan hasil yang memuaskan. Pengujian tersebut juga dilakukan untuk memverifikasi apakah hasil koreksi dan atau hasil penambahan atau pengurangan fitur tidak menimbulkan adanya *error* atau *bug* baru pada fitur-fitur lain.

Regression test dilaksanakan dengan menggunakan database *test case* yang sudah dibuat sebelumnya pada saat *functional testing*. Oleh karena itu, jenis uji ini bisa dilakukan dengan usaha yang minimal.

2.3 Rational Robot dan Rational TestManager

Penjelasan mengenai subbab ini didapatkan dari *Rational® TestManager User's Guide version 2003.06.00.*

Rational Robot dan Rational TestManager merupakan perangkat lunak untuk membantu proses *functional testing* dan *performance testing.* Rational Robot menghasilkan *test script* secara otomatis dan Rational TestManager menghasilkan *test log* yang merupakan hasil pengujian terhadap suatu fungsi pada aplikasi juga secara otomatis. Penggunaan Rational Robot adalah untuk tujuan-tujuan berikut:

1. Melakukan *functional testing* secara utuh. Membuat dan merekam *test script* yang menelusuri aplikasi dan memverifikasi status suatu objek dengan menggunakan *verification point.*
2. Melakukan *performance testing* secara utuh. *Test script* yang dihasilkan digunakan untuk menentukan apakah sistem yang diuji bekerja dalam waktu (*response time*) yang telah ditentukan berdasarkan pengujian dengan berbagai macam *workload.*
3. Menguji aplikasi yang dibuat dengan IDE (Integrated Development Environment) seperti Java, HTML, Visual Basic, Oracle Forms,

Delphi, dan PowerBuilder. Pengujian bisa dilakukan terhadap objek-objek pada aplikasi tersebut meskipun objek tidak terlihat di tampilannya.

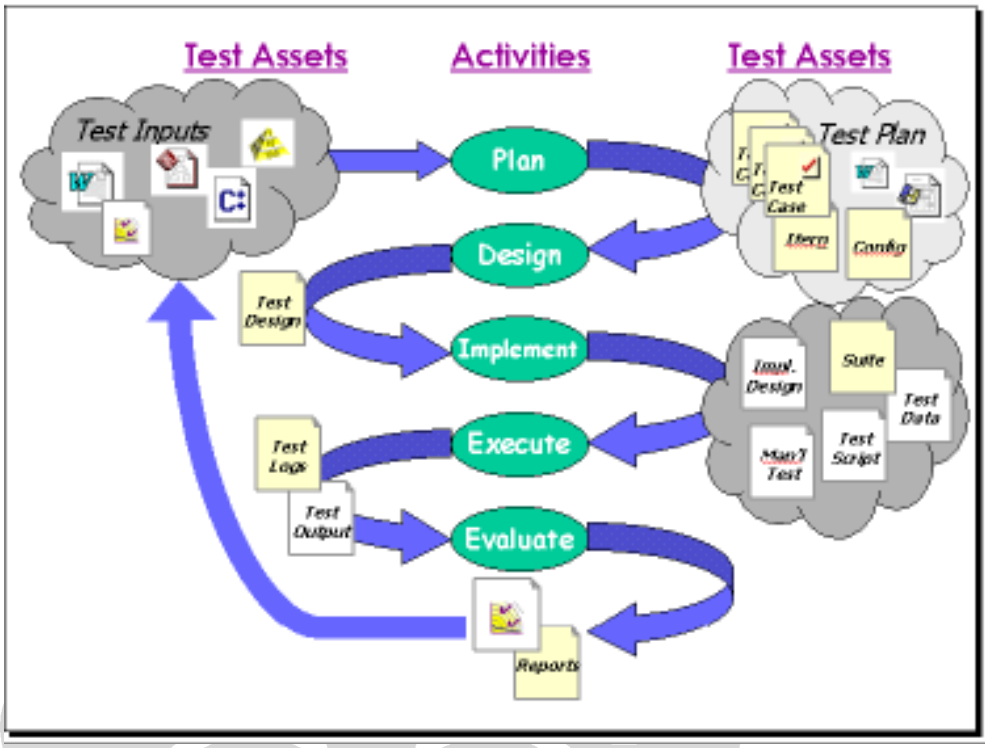
4. Mengumpulkan informasi mengenai aplikasi selama *test script* di-*run* (disebut dengan *playback)*. Informasi ini dapat dilihat di *test log* yang dihasilkan di Rational TestManager [RTM].

Rational Robot adalah kumpulan komponen untuk mengotomatisasi pengujian pada Microsoft Windows *client/server* dan aplikasi Internet yang berjalan pada Windows NT 4.0, Windows XP, Windows 2000, Windows 98, dan Windows Me. Komponen utama dari Robot adalah melakukan rekaman pengujian (*record*) dengan waktu yang singkat. Setelah rekaman selesai, Robot dapat mengulangi pengujian kembali (*playback*) dengan waktu yang lebih singkat dibandingkan dengan pengujian secara manual.

Rational Robot adalah sebuah alat untuk melakukan *regression functional testing* yang otomatis. Artinya, Rational Robot memiliki fungsi utama yaitu untuk melakukan *functional testing* yang juga mempermudah dan mempercepat proses *regression testing.*

Sedangkan Rational TestManager adalah *framework* yang terbuka yang menyatukan seluruh *tools*, *assets,* dan data yang dihasilkan maupun yang berhubungan dengan pengujian. Melalui *framework* yang tunggal ini, seluruh partisipan yang berhubungan dengan pengujian dapat mendefinisikan dan memperbaiki kualitas sistem yang akan dicapai. Rational TestManager adalah tempat di mana para anggota tim pengembang mendefinisikan rencana yang akan diimplementasikan.

Berikut ini merupakan gambar dan penjelasan *workflow* pengujian yang didukung oleh Rational TestManager:



Gambar 2.1 *Workflow* Proses Pengujian

Sumber: *Rational® TestManager User's Guide version 2003.06.00*

1. *Planning Test*

Kegiatan dalam *planning test* ini menjawab pertanyaan berikut: "Apa saja yang harus diuji sehingga memenuhi kualitas yang ditentukan?". Pada akhir kegiatan ini, akan dihasilkan *test plan* yang mendefinisikan apa yang akan diuji kemudian. Dalam TestManager, *test planning* terdiri dari kegiatan-kegiatan berikut:

a. Mengumpulkan dan mengidentifikasi *test input* yang dapat berupa *prototype*, spesifikasi fungsionalitas sistem, *requirement, source code,* dan lain sebagainya.

b. Membuat *test plan* yang dapat terdiri dari *test case folder* dan *test case*

c. Membuat *test case folder* yang digunakan untuk mengorganisasi *test case* secara hirarki.

d. Membuat *test case* yang merupakan *test asset* dalam TestManager. Pembuatan *test case* ini adalah untuk mendefinisikan unit-unit

sistem yang harus divalidasi untuk memverifikasi bahwa sistem yang diuji sudah bekerja sesuai dengan fungsinya

e. Mendefinisikan konfigurasi yang dibutuhkan dalam pengujian, termasuk konfigurasi *hardware* dan *software*. Misalkan pengujian dilakukan untuk menguji apakah sistem dapat berjalan di semua *operating system*, maka yang harus dibuat adalah konfigurasi untuk setiap *operating system*.

f. Mendefinisikan iterasi, yaitu kapan waktu yang tepat untuk menjalankan pengujian

2. Designing Test

Kegiatan dalam *designing test* ini menjawab pertanyaan: "Bagaimana caranya melakukan pengujian?". Kegiatan ini memberikan informasi mengenai aksi apa saja yang harus dilakukan pada sistem yang diuji dan perilaku serta karakteristik apa yang diharapkan untuk muncul jika sistem berjalan dengan baik.

3. Implementing Test

Aktivitas yang terjadi pada *implementing test* adalah perancangan dan pembuatan *test script* yang mengimplementasikan *test case* yang telah dibuat. Setelah selesai dibuat, *test script* dapat dihubungkan dengan *test case* yang sesuai.

4. Executing Test

Aktivitas dalam *executing test* adalah menjalankan (run) implementasi pengujian yang telah dibuat.

5. Evaluating Test

Aktivitas *evaluating test* termasuk:

a. Menentukan validitas dari uji yang dijalankan.

b. Menganalisis output yang dihasilkan untuk menentukan hasil pengujian.

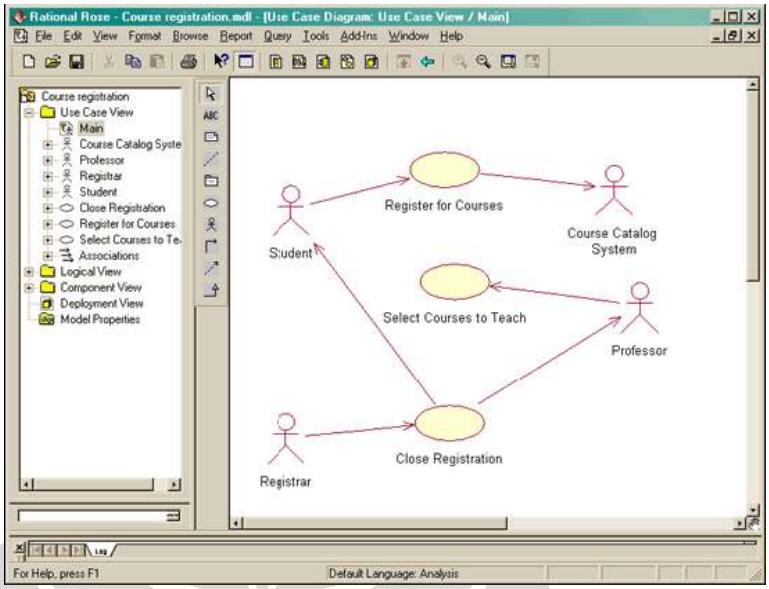
2.4 Pembuatan Test Case dengan Menggunakan Use Case

Berdasarkan *Generating Test Cases From Use Cases* oleh Jim Houmann, dalam sebuah proyek pengembangan perangkat lunak, *use case* mendefinisikan kebutuhan-kebutuhan sistem perangkat lunak itu sendiri. Pembuatan *use case* dilakukan di awal pengembangan, sehingga pernyataan fungsionalitas produk sudah tersedia pada iterasi awal. *Use case* memberikan informasi kepada konsumen mengenai apa yang diharapkan, kepada pengembang mengenai apa yang harus dibuat, kepada *technical writer* mengenai apa yang harus didokumentasikan, dan kepada *tester* mengenai apa yang harus diuji.

Bagi pengujian perangkat lunak, yang terdiri dari aktivitas-aktivitas yang saling berhubungan, masing-masing dengan *artifact* dan *deliverable*-nya, pembuatan *test case* merupakan langkah dasar yang pertama. Kemudian prosedur pengujian dirancang untuk *test case* tersebut, dan terakhir, *test script* dibuat untuk mengimplementasikan prosedur. *Test case* merupakan kunci terhadap proses ini karena *test case* mengidentifikasi dan mengkomunikasikan kondisi-kondisi yang akan diimplementasi dalam pengujian dan merupakan hal yang sangat diperlukan untuk memverifikasi implementasi kebutuhan produk. Semua hal ini digunakan untuk meyakinkan bahwa produk yang dihasilkan memenuhi kebutuhan-kebutuhan sistem.

2.4.1 Use Case

Use case didasarkan pada *Unified Modeling Language (UML)* dan dapat direpresentasikan secara visual dalam diagram *use-case.* Berikut adalah gambar contoh diagram *use-case* yang menggambarkan kebutuhan untuk sistem registrasi akademis di universitas.



Gambar 2.2 Contoh Diagram *Use-case*

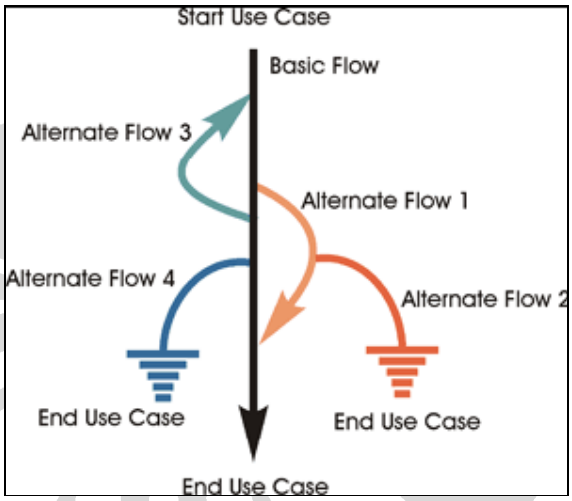
Sumber: *Generating Test Cases From Use Cases*

Lingkaran oval dari diagram *use-case* di atas merepresentasikan *use case*, sedangkan bentuk orang-orangan merepresentasikan aktor, yang dapat berupa manusia atau sistem lain. Garis bermata panah menggambarkan komunikasi yang terjadi antara seorang aktor dan sebuah *use case*. Diagram *use-case* ini telah cukup menggambarkan secara garis besar bagaimana kebutuhan sistem. Namun, masing-masing *use case* harus dijabarkan secara lebih mendetail bagaimana alur kerjanya dari keadaan awal hingga keadaan akhir yang diharapkan. Penjelasan *use case* ini disebut dengan *use case specification* seperti yang ditunjukkan pada tabel di bawah ini:

Tabel 2.2 *Use Case Specification*

Nama *use case*	
Aktor	
Kondisi Awal	
Kondisi Akhir	
Basic flow	
Alternate flow	
Prioritas	

Bagian terpenting dari sebuah *use case* untuk pembuatan *test case* adalah aliran *event*. Dua bagian utama dari aliran *event* adalah *basic flow* dan *alternate flow*. *Basic flow* mendefinisikan apa saja yang terjadi dalam keadaan normal pada saat *use case* dijalankan. Sedangkan *alternate flow* menjelaskan perilaku yang terjadi pada sistem dalam keadaan yang tidak normal atau variasi dari keadaan normal.



Gambar 2.3 *Basic Flow* dan *Alternate Flow* pada *Use Case*

Sumber: *Generating Test Cases From Use Cases*

Gambar *basic flow* dan *alternate flow* pada *use* case di atas merepresentasikan struktur yang biasa terdapat dalam aliran *event*. Garis lurus menggambarkan *basic flow* dari suatu *event,* dan garis yang melengkung merepresentasikan *alternate flow*. Dalam suatu aliran *event*, *alternate flow* dapat kembali menjadi *basic flow*, sedangkan *alternate flow* yang lain menuju akhir dari *use case*. Baik *basic flow* maupun *alternate flow* harus dijelaskan secara lebih mendetail dalam tahapan *event* atau *subflow.*

Berikut adalah contoh *basic flow* dan *alternate flow* dari *use case specification* untuk *use case Register for Courses*.

Tabel 2.3 Contoh *Basic Flow* dan *Alternate Flow*

Nama *use case*	*Register for Courses*
Basic flow	1. Login 2. Memilih "Create a Schedule" 3. Mendapatkan informasi mata kuliah 4. Memilih mata kuliah 5. Men-*submit* jadwal 6. Sistem menampilkan jadwal yang sudah lengkap
Alternate flow	1. Langkah pertama dari *basic flow,* login, gagal karena *password* dan atau *username* tidak valid. 2. Keluar dari sistem 3. Berdasarkan langkah 5 dari *basic flow*, jika persyaratan tidak lengkap, mata kuliah sudah penuh, atau jadwal bentrok, maka sistem tidak akan memasukkan mahasiswa ke dalam mata kuliah tersebut 4. Berdasarkan langkah 3 dari *basic flow*, jika sistem *down* maka informasi mata kuliah tidak dimunculkan, ada pesan yang ditampilkan, dan *use case* berakhir 5. Jika masa regristasi sudah berakhir, maka ada pesan yang ditampilkan dan *use case* berakhir

Sumber: *Generating Test Cases From Use Cases*

2.4.2 Skenario *Use Case*

Terdapat satu hal yang perlu diketahui sebelum penjelasan mengenai pembuatan *test case* dari *use case* dimulai, yaitu skenario *use case*. Sebuah skenario *use case* merupakan sebuah *path* yang utuh dari *use case*. Pengguna dari sistem yang utuh dapat melalui banyak *path* ketika mereka menjalankan fungsionalitas yang dijelaskan dalam *use case*. Satu *basic flow* merupakan satu skenario. *Basic flow* yang digabungkan dengan *alternate flow* 1 merupakan skenario yang lain. *Basic flow* yang digabungkan dengan *alternate flow* 2 adalah skenario yang lain lagi, dan seterusnya.

Tabel berikut ini menunjukkan semua kemungkinan skenario yang terbentuk dari *basic flow* dan *alternate flow.*

Tabel 2.4 Daftar Skenario Berdasarkan Kombinasi *Basic Flow* dan *Alternate Flow*

Skenario 1	*Basic flow*			
Skenario 2	*Basic flow*	*Alternate flow* 1		
Skenario 3	*Basic flow*	*Alternate flow* 1	*Alternate flow* 2	
Skenario 4	*Basic flow*	*Alternate flow* 3		
Skenario 5	*Basic flow*	*Alternate flow* 3	*Alternate flow* 1	
Skenario 6	*Basic flow*	*Alternate flow* 3	*Alternate flow* 1	*Alternate flow* 2
Skenario 7	*Basic flow*	*Alternate flow* 4		
Skenario 8	*Basic flow*	*Alternate flow* 3	*Alternate flow* 4	

Sumber: *Generating Test Cases From Use Cases*

Ketujuh skenario di atas akan digunakan sebagai dasar dalam pembuatan *test case*.

2.4.3 Pembuatan Test Case

Sebuah *test case* adalah kumpulan *test input*, *execution condition*, dan *expected result* yang dibentuk untuk tujuan tertentu, misalnya untuk menguji suatu *path* dalam program atau untuk memverifikasi dipenuhinya suatu *requirement*. *Test case* dibutuhkan untuk memverifikasi implementasi kebutuhan sistem apakah sudah berhasil dan memenuhi kebutuhan tersebut.

Berikut ini tiga tahapan dalam pembuatan *test case*:

1. Membentuk seluruh skenario *use case* yang mungkin untuk masing-masing *use case*
2. Membuat paling tidak satu *test case* untuk masing-masing skenario
3. Memasukkan nilai yang akan digunakan untuk pengujian

2.4.3.1 Pembuatan skenario

Dengan membaca *use case specification*, tahap pembentukan skenario dapat dilakukan dengan membuat semua kombinasi antara *basic flow* dan *alternate flow*. Tabel berikut memperlihatkan contoh daftar skenario berdasarkan *use case Register for Courses* yang *use case specification*-nya telah diberikan sebelumnya.

Tabel 2.5 Contoh Daftar Skenario

Nama skenario	*Flow* awal	*Alternate*
Skenario 1 – pendaftaran sukses	*Basic flow*	
Skenario 2 – login gagal	*Basic flow*	*Alternate flow* 1
Skenario 3 – keluar sistem	*Basic flow*	*Alternate flow 2*
Skenario 4 – sistem *down*	*Basic flow*	*Alternate flow 4*
Skenario 5 – masa registrasi habis	*Basic flow*	*Alternate flow 5*
Skenario 6 – tidak dapat mendaftarkan diri pada suatu mata kuliah karena persyaratan tidak lengkap, mata kuliah sudah penuh, atau jadwal bentrok	*Basic flow*	*Alternate flow 3*

Sumber: *Generating Test Cases From Use Cases*

2.4.3.2 Pembuatan *test case*

Dari skenario yang telah dibuat pada langkah sebelumnya, dapat dibentuk minimal satu *test case* untuk masing-masing skenario. Namun, diperlukan analisis dan *review* yang lebih mendalam pada masing-masing skenario ini karena dalam satu skenario bisa dibentuk lebih dari satu *test case*. Misalnya, jika suatu skenario dideskripsikan secara kurang mendetail, seperti deskripsi pada skenario 6 yang mengkombinasikan *basic flow* dengan *alternate flow* 3 berikut:
"Tidak dapat mendaftarkan diri pada suatu mata kuliah karena persyaratan tidak lengkap, mata kuliah sudah penuh, atau jadwal bentrok."

Oleh karena itu, diperlukan *test case* tambahan agar pengujian dapat dilakukan terhadap semua *event*. Langkah selanjutnya adalah membaca ulang *use case specification* dan mencari kondisi-kondisi atau elemen-elemen data yang dibutuhkan untuk menjalankan skenario yang beragam. Untuk *use case Register for Course*, kondisi yang ada misalnya *student ID, password,* mata kuliah yang dipilih, dan lain-lain.

Untuk dapat membuat *test case* dengan jelas dan mudah, dapat menggunakan matriks seperti yang akan diperlihatkan pada tabel berikut. Perhatikan baris pertama. Kolom pertama berisi *test case ID*, kolom kedua merupakan deskripsi singkat mengenai *test case* termasuk skenario yang berkaitan dengan *test case*, dan kolom-kolom yang lain kecuali kolom terakhir berisi elemen-elemen data yang akan digunakan dalam implementasi pengujian.

Tabel 2.6 Matriks *Test Case* untuk *Use Case Register for Course*

Test case ID	**Skenario/ Kondisi**	***Student ID***	***Password***	**Mata Kuliah yang Dipilih**	**Syarat lengkap**	**Mata kuliah dibuka**	**Jadwal tidak bermasalah**	**Hasil yang diharapkan**
RC 1	Skenario 1 – pendaftaran sukses	V	V	V	V	V	V	Jadwal dan nomor konfirmasi ditampilkan
RC 2	Skenario 2 – login gagal	I	N/A	N/A	N/A	N/A	N/A	Pesan *error*, kembali ke halaman login
RC 3	Skenario 3 – pengguna yang sudah berhasil login keluar dari aplikasi	V	V	N/A	N/A	N/A	N/A	Halaman login ditampilkan
RC 4	Skenario 4 – sistem *down*	V	V	N/A	N/A	N/A	N/A	Pesan *error* ditampilkan, kembali ke *basic flow* langkah 2
RC 5	Skenario 5 – masa registrasi habis	V	V	N/A	N/A	N/A	N/A	Pesan *error* ditampilkan, kembali ke *basic flow* langkah 2
RC 6	Skenario 6 – gagal registrasi karena mata kuliah sudah penuh	V	V	V	V	I	V	Pesan *error* ditampilkan, kembali ke *basic flow* langkah 3
RC 7	Skenario 6 – gagal registrasi karena syarat tidak lengkap	V	V	V	I	V	V	Pesan *error* ditampilkan, kembali ke *basic flow* langkah 4
RC 8	Skenario 6 – gagal registrasi karena jadwal bentrok	V	V	V	V	V	I	Pesan *error* ditampilkan, kembali ke *basic flow* langkah 4

Sumber: *Generating Test Cases From Use Cases*

Perhatikan bahwa dalam matrix tersebut tidak ada nilai data yang dimasukkan. Masing-masing *cell* berisi V, I, atau N/A. Artinya, V merepresentasikan *valid*, I untuk *invalid*, dan N/A berarti bahwa tidak perlu memasukkan nilai data dalam *cell* tersebut.

2.4.3.3 Memasukkan nilai data yang digunakan untuk pengujian

Setelah seluruh *test case* berhasil dibentuk, semuanya harus di-*review* dan divalidasi untuk memastikan kebenarannya dan untuk mengetahui adanya *test case* yang *redundant* atau yang kurang. Kemudian, setelah semuanya sudah lengkap dan *valid*, langkah terakhir adalah mengganti nilai I dan V dengan nilai data. Tanpa menggunakan data, *test case* (atau prosedur pengujian) tidak dapat diimplementasikan atau dieksekusi. *Test case* tersebut hanya merupakan deskripsi dari kondisi, skenario, dan *path*. Oleh karena itu, nilai-nilai data yang nyata perlu dimasukkan untuk implementasi pengujian. Tabel berikut ini menunjukkan matriks *test case* dengan nilai-nilai yang menggantikan V dan I pada matriks sebelumnya.

Tabel 2.7 Matrik *Test Case* dengan Nilai Data

Test case ID	**Skenario/ Kondisi**	***Student ID***	***Password***	**Mata Kuliah yang Dipilih**	**Syarat lengkap**	**Mata kuliah dibuka**	**Jadwal tidak bermasalah**	**Hasil yang diharapkan**
RC 1	Skenario 1 – pendaftaran sukses	jheuman	abc123	M101 E201 S101	Yes	Yes	Yes	Jadwal dan nomor konfirmasi ditampilkan
RC 2	Skenario 2 – login gagal	Jheuman1	N/A	N/A	N/A	N/A	N/A	Pesan *error*, kembali ke halaman login
RC 3	Skenario 3 – pengguna yang sudah berhasil login keluar dari aplikasi	jheuman	abc123	N/A	N/A	N/A	N/A	Halaman login ditampilkan
RC 4	Skenario 4 – sistem *down*	jheuman	abc123	N/A	N/A	N/A	N/A	Pesan *error* ditampilkan, kembali ke *basic flow* langkah 2
RC 5	Skenario 5 – masa registrasi habis	jheuman	abc123	N/A	N/A	N/A	N/A	Pesan *error* ditampilkan, kembali ke *basic flow* langkah 2
RC 6	Skenario 6 – gagal registrasi karena mata kuliah sudah penuh	jheuman	abc123	M101 E201 S101	Yes	M101 penuh	Yes	Pesan *error* ditampilkan, kembali ke *basic flow* langkah 3
RC 7	Skenario 6 – gagal registrasi karena syarat tidak lengkap	jheuman	abc123	M101 E201 S101	Tidak untuk E201	Yes	Yes	Pesan *error* ditampilkan, kembali ke *basic flow* langkah 4
RC 8	Skenario 6 – gagal registrasi karena jadwal bentrok	jheuman	abc123	M101 E201 S101	Yes	Yes	E201 dan S101 bentrok	Pesan *error* ditampilkan, kembali ke *basic flow* langkah 4

Sumber: *Generating Test Cases From Use Cases*