TEKNIK PENGUJIAN PERANGKAT LUNAK

Oleh : Rahmat Robi Waliyansyah, M.Kom.

LATAR BELAKANG

- Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan merupakan review puncak terhadap spesifikasi, desain dan pembuatan program.
- Pengujian perangkat lunak menghabiskan upaya 30-40% dari total pekerjaan proyek.
- Untuk proyek yang membahayakan nyawa manusia, biaya pengujian bisa 3-5 X proyek biasa.

Tujuan Pengujian

1. Menjalankan program untuk menemukan error.
2. Test case yang bagus adalah yang memiliki kemungkinan terbesar untuk menemukan error yang tersembunyi.
3. Pengujian yang sukses adalah yang berhasil menemukan error yang tersembunyi.

Prinsip Pengujian

- Harus bisa dilacak hingga sampai ke kebutuhan customer.
- Harus direncanakan sejak model dibuat.
- Prinsip Pareto: 80% error uncovered.
- Dari lingkup kecil menuju yang besar.
- Tidak bisa semua kemungkinan diuji.
- Dilakukan oleh pihak ketiga yang independen.

Testablility

- Kemudahan untuk diuji.
- Karakteristiknya:
  - *Operability: mudah digunakan.*
  - *Observability: mudah diamati.*
  - *Controlability: mudah dikendalikan.*
  - *Decomposability: mudah diuraikan.*
  - *Simplicity: lingkup kecil, semakin mudah diuji.*
  - *Stability: jarang berubah.*
  - *Understandability: mudah dipahami.*

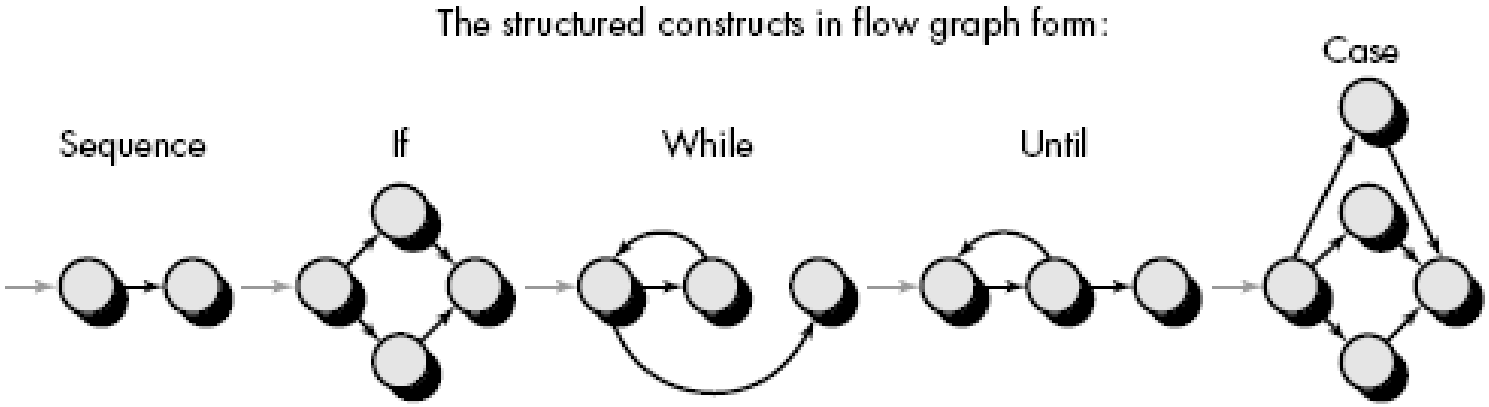
Desain Kasus Pengujian

- **Black Box Testing**
  - Memastikan fungsional perangkat lunak berjalan.
  - Kesesuaian input dengan output.
  - Tidak memperhatikan proses logic internal.
- **White Box Testing**
  - Pengamatan detail prosedur.
  - Mengamati sampai level percabangan kondisi dan perulangan.

White Box Testing

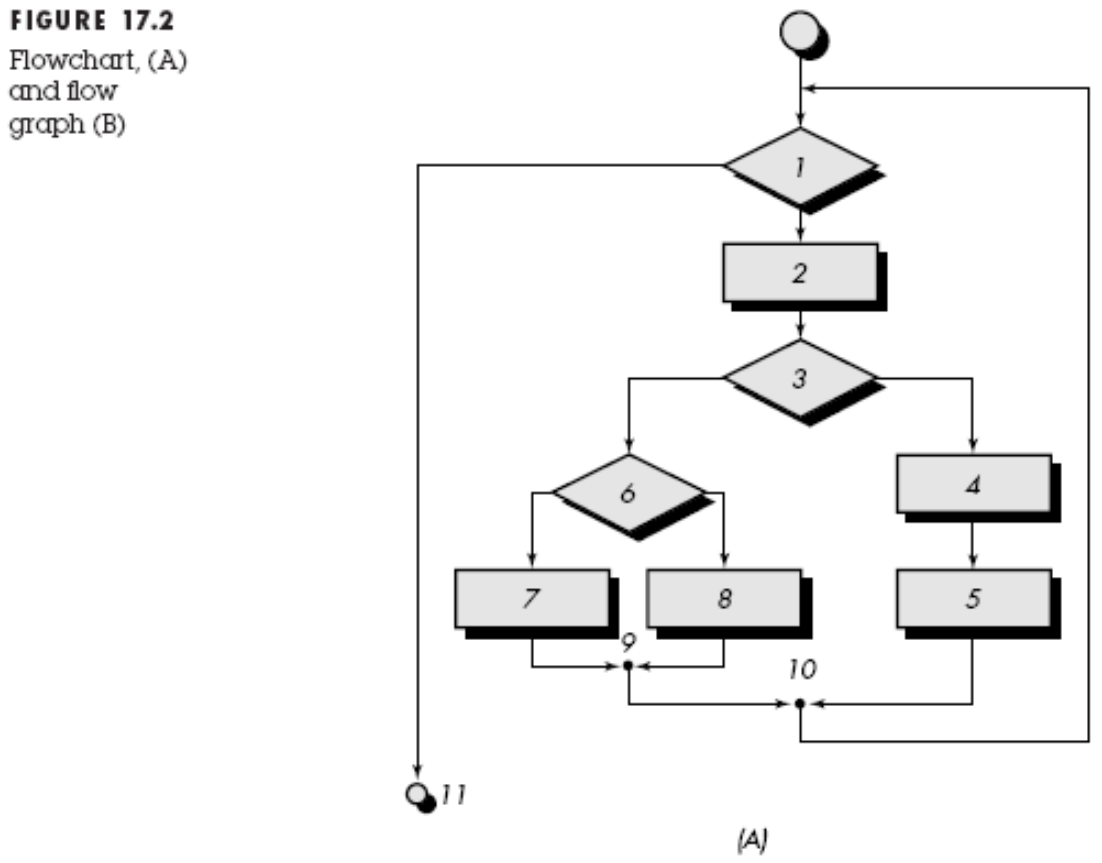
- Metode: basis path testing.
- Memakai notasi flow graph.

FIGURE 17.1
Flow graph notation



KOMPLEKSITAS CYCLOMATIC

- Menunjukkan jumlah skenario pengujian yang harus dilakukan untuk menjamin cakupan seluruh program.

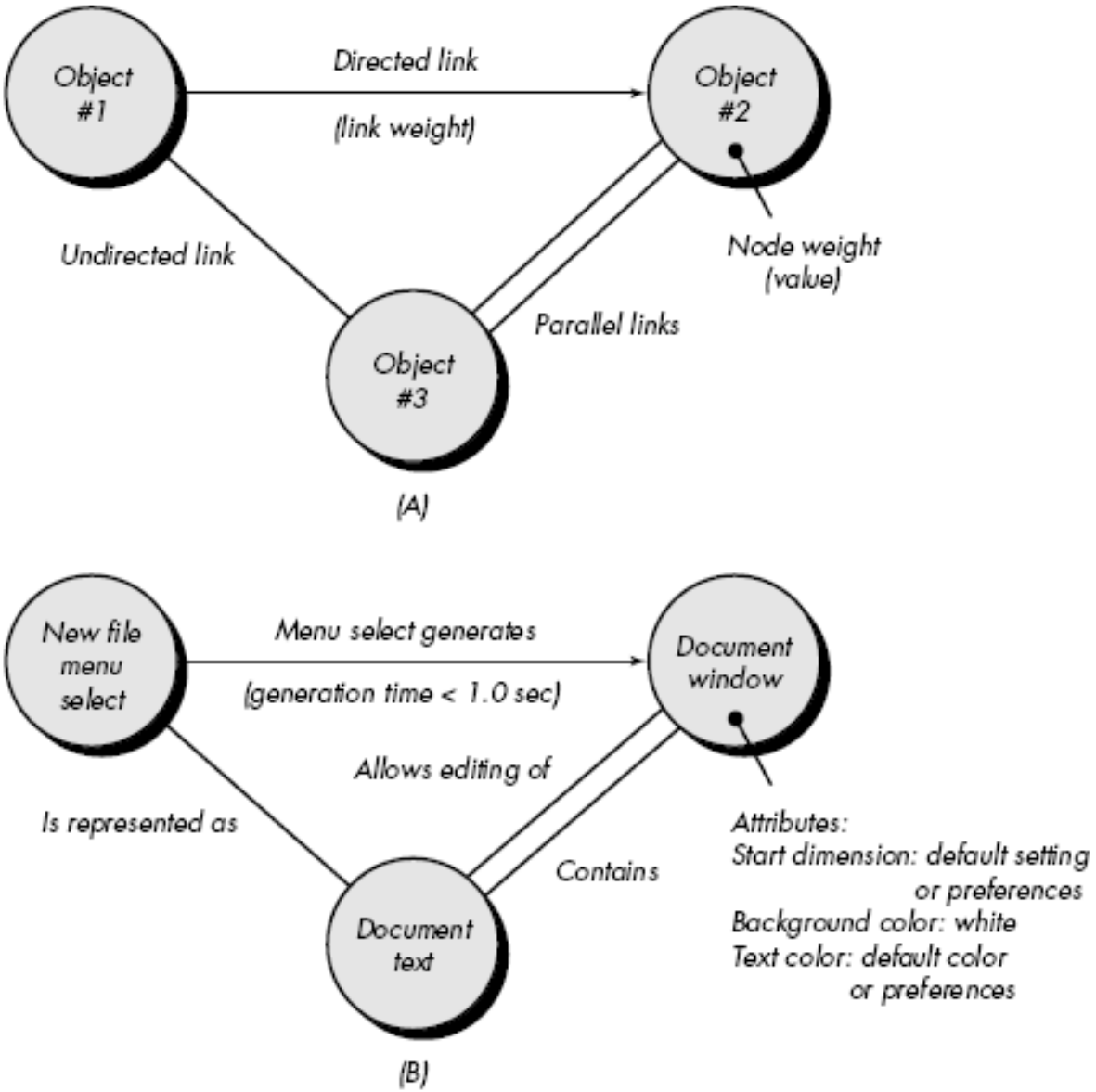


Contoh White Box Testing

```
PROCEDURE average;

    *  This procedure computes the average of 100 or fewer
       numbers that lie between bounding values; it also computes the
       sum and the total number valid.

    INTERFACE RETURNS average, total.input, total.valid;
    INTERFACE ACCEPTS value, minimum, maximum;

    TYPE value[1:100] IS SCALAR ARRAY;
    TYPE average, total.input, total.valid;
        minimum, maximum, sum IS SCALAR;
    TYPE i IS INTEGER;
1   i = 1;
    total.input = total.valid = 0;                    2
    sum = 0;
    DO WHILE value[i] <> -999 AND total.input < 100   3
4       increment total.input by 1;
        IF value[i] >= minimum AND value[i] <= maximum   6
5           THEN increment total.valid by 1;
7                sum = sum + value[i]
            ELSE skip
        ENDIF
8       increment i by 1;
9   ENDDO
    IF total.valid > 0   10
11      THEN average = sum / total.valid;
12      ELSE average = -999;
13  ENDIF
END average
```

Black Box Testing – Graph Based

FIGURE 17.9
(A) Graph notation
(B) Simple example



BLACK BOX TESTING – EQUIVALENCE PARTITIONING

1. If an input condition specifies a *range*, one valid and two invalid equivalence classes are defined.
2. If an input condition requires a specific *value*, one valid and two invalid equivalence classes are defined.
3. If an input condition specifies a member of a *set*, one valid and one invalid equivalence class are defined.
4. If an input condition is *Boolean*, one valid and one invalid class are defined.

- Contoh: Input NPM dalam SIAMIK
  - Jika dikosongi?
  - Jika diisi dengan format yang salah?
  - Jika diisi dengan NPM yang benar?

Black Box Testing – Analisa Nilai Batas

1. Menguji untuk input di sekitar batas atas maupun bawah sebuah range nilai yang valid.
2. Menguji nilai maksimal dan minimal.
3. Menerapkan (1 & 2) untuk output.
4. Menguji batas struktur data yang dipakai. Misal ukuran array.

Black Box Testing - Perbandingan

- Spesifikasi kebutuhan yang sama dimungkinkan menghasilkan aplikasi/perangkat lunak yang berbeda.
- Skenario pengujian pada aplikasi yang demikian bisa digunakan untuk skenario pengujian aplikasi serupa yang lain.

Skenario Pengujian Khusus

- Pengujian GUI.
- Pengujian arsitektur client/server.
- Pengujian dokumentasi dan fasilitas bantuan.
- Pengujian sistem waktu nyata.

STRATEGI PENGUJIAN PERANGKAT LUNAK

- Membahas langkah-langkah yang harus dikerjakan sebagai bagian dari pengujian.
- Kapan dilaksanakan? Berapa usaha, waktu dan sumber daya yang digunakan?
- Meliputi: perencanaan, desain test case, pelaksanaan, koleksi data dan evaluasi.

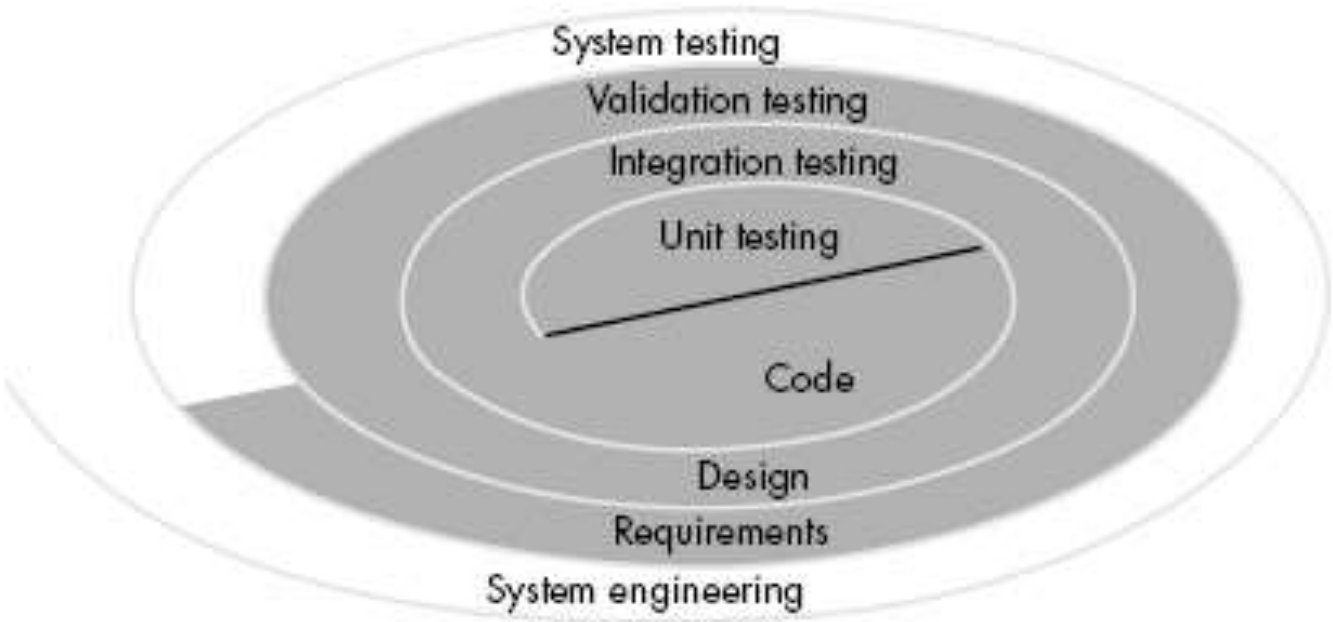
Kaidah Umum Pengujian

- Dimulai dari pengujian tingkat komponen menuju integrasi.
- Titik yang berbeda dimungkinkan memakai teknik pengujian yang berbeda.
- Pengujian dilakukan oleh developer dan (untuk proyek yang besar) tim independen.
- Testing dan debugging adalah berbeda. Namun debugging pasti berkaitan dengan strategi testing apapun.

STRATEGI PENGUJIAN

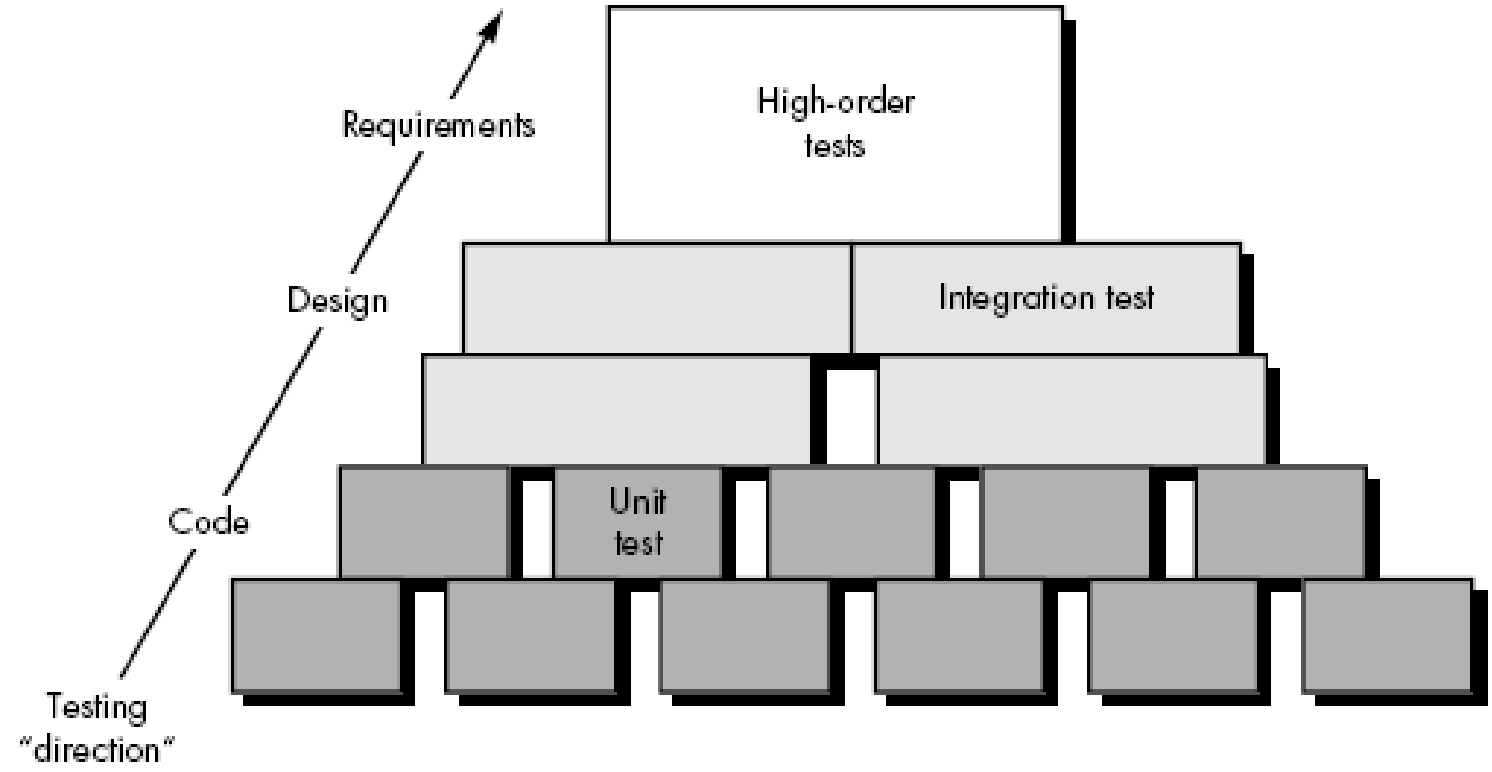
- Dimulai dari unit testing terhadap source code hingga system testing terhadap spesifikasi kebutuhan.

FIGURE 18.1
Testing strategy



LANGKAH PENGUJIAN

FIGURE 18.2
Software testing steps



UNIT TESTING

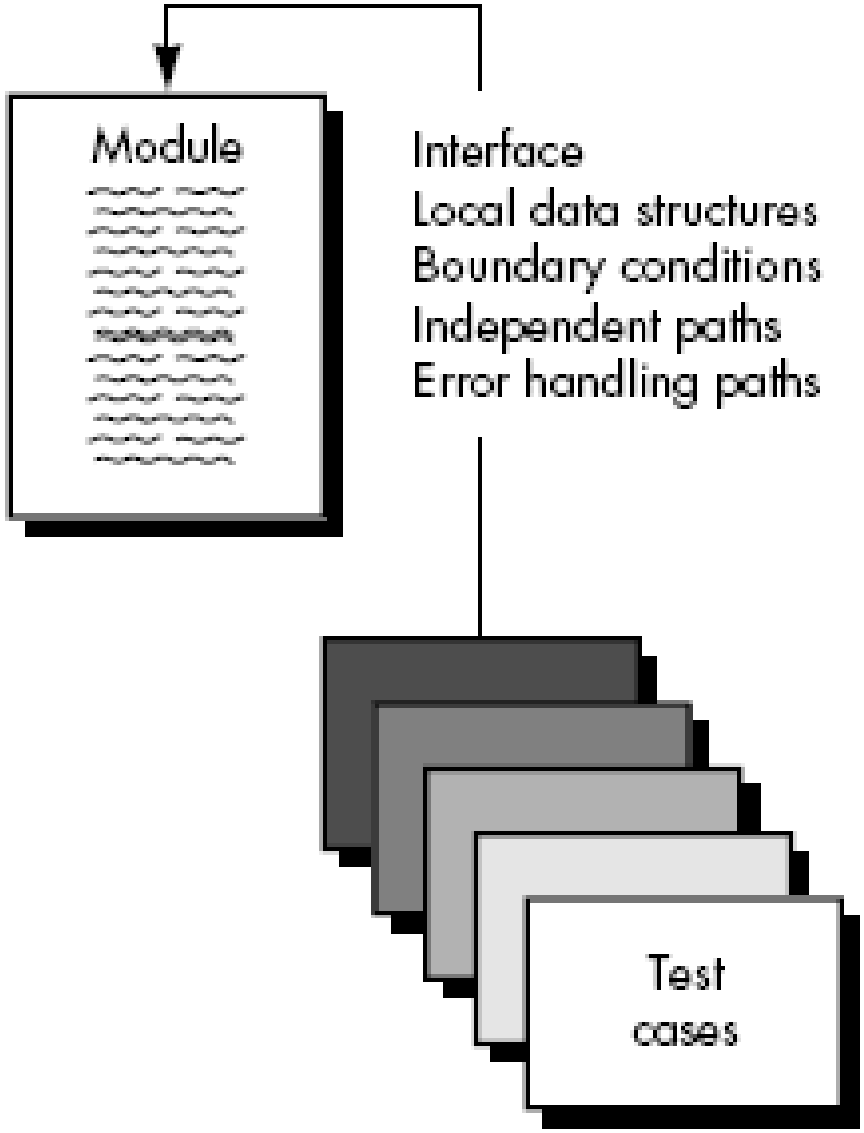
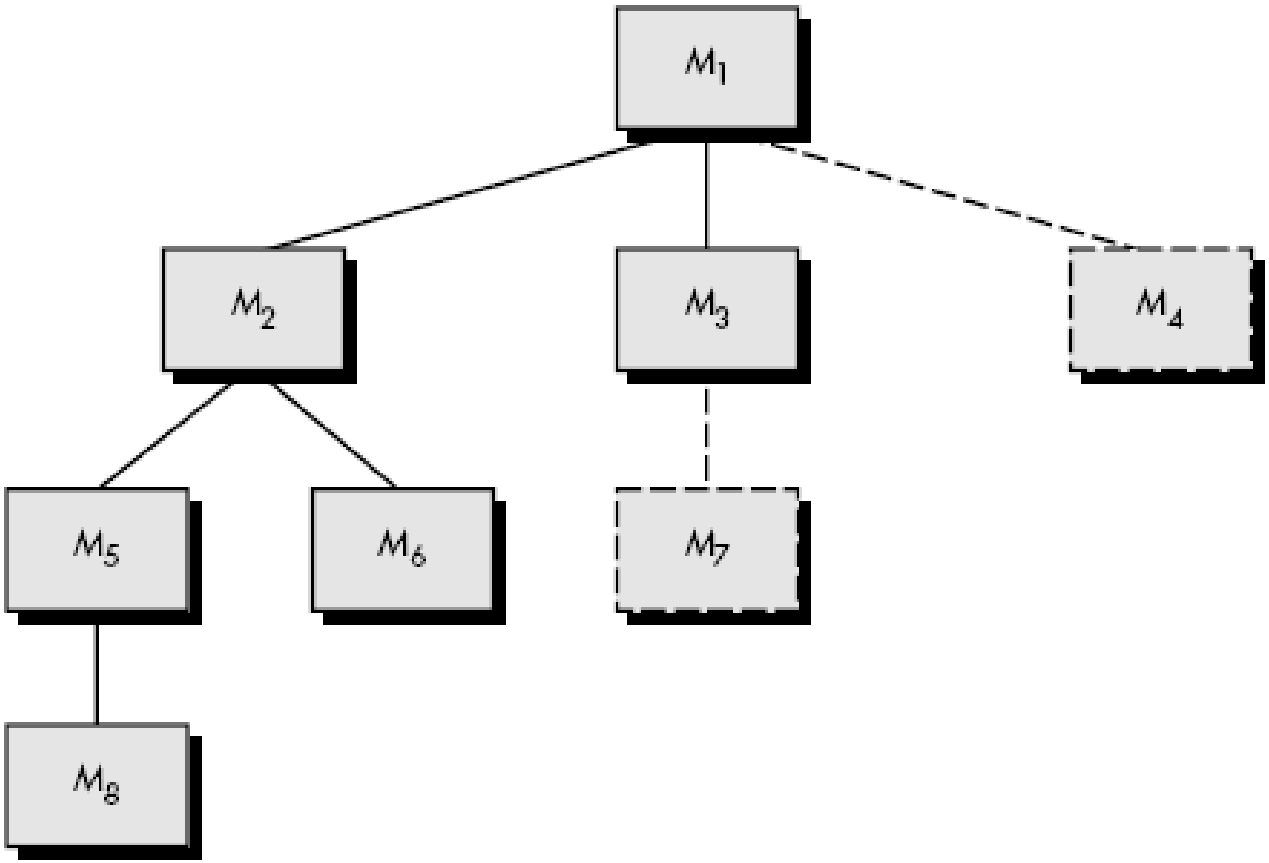


FIGURE 18.4
Unit test

INTEGRATION TESTING

- Top – down integration

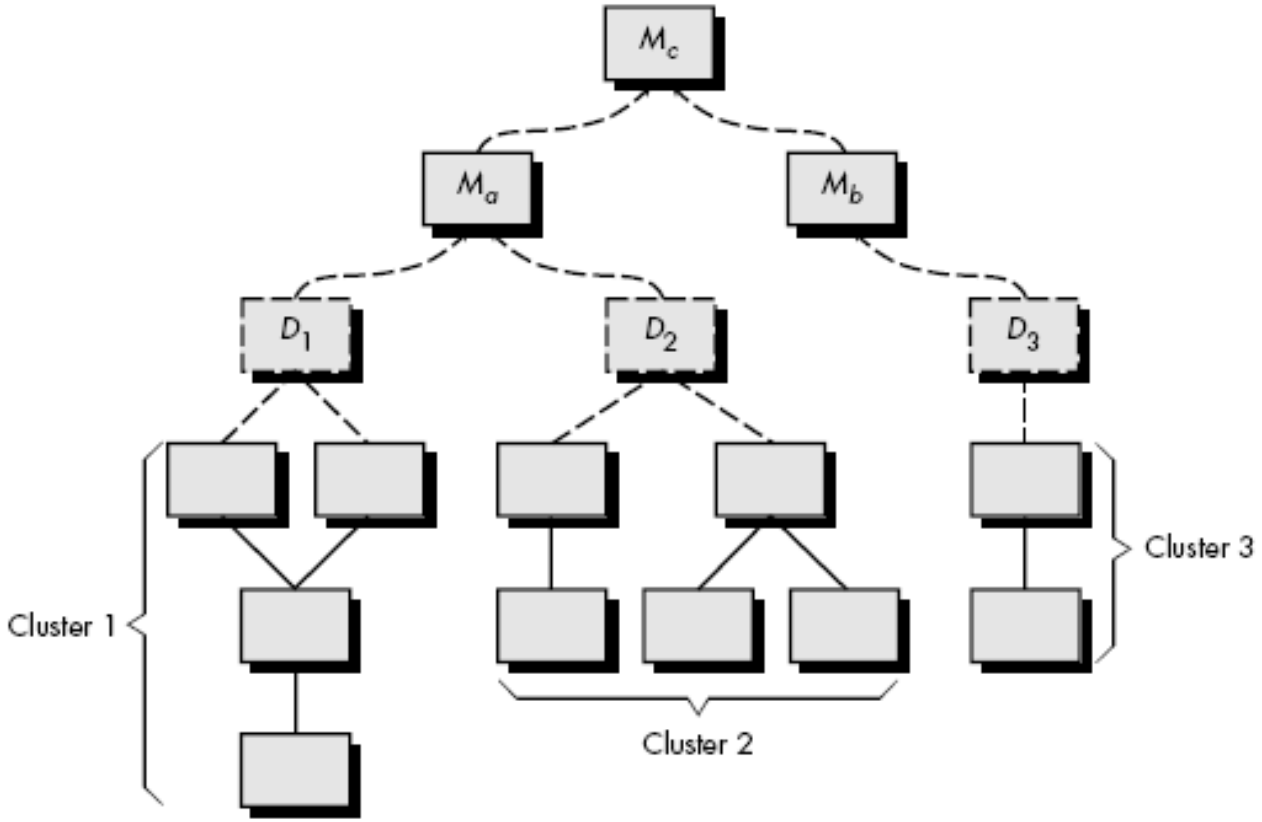
FIGURE 18.6
Top-down integration



INTEGRATION TESTING

- Bottom – up integration

FIGURE 18.7
Bottom-up integration



INTEGRATION TESTING

- Regression testing: dilakukan pengujian setiap kali ada modul baru yang diintegrasikan atau ada modul yang berubah.
- Smoke testing: test daily, untuk proyek jenis kritis-waktu.

Validation Testing

- Disebut sukses jika fungsi perangkat lunak dapat diterima oleh customer (berdasarkan dokumen SKPL).
- Alpha test: dilakukan di tempat developer oleh customer pada lingkungan yang terkendali.
- Beta test: dilakukan di tempat customer tanpa melibatkan developer pada lingkungan yang tak terkendali.

System Testing

- Menguji sistem berbasis komputer secara menyeluruh, termasuk juga hubungannya dengan sistem yang lain.
- Diantaranya:
  - **Recovery testing**, jika system failure.
  - **Security testing**, jika terjadi serangan.
  - **Stress testing**, terhadap jumlah, frekuensi dan volume pekerjaan.
  - **Performance testing**, untuk mengukur pemakaian sumber daya.

DEBUGGING

- Memperbaiki error yang ditemukan pada saat testing (yang sukses).
- Kaidah dasar sebelum debug:
  - Apakah penyebab bug dihasilkan kembali oleh bagian program yang lain?
  - Apakah bug selanjutnya yang mungkin muncul jika bug diperbaiki?
  - Apa yang bisa dilakukan untuk mencegah bug terjadi untuk pertama kalinya?