BAB 13
METODA PENGUJIAN PERANGKAT LUNAK MENGGUNAKAN METRIK KOMPLEKSITAS SIKLOMATIK

Sebuah program pada dasamya tidak stabil karena terlalu kompleks untuk diuji dan terlalu kompleks untuk digunakan dalam produksi. Jika suatu program yang kompleks dapat diidentifikasi maka dapat pula diperbaiki untuk disederhanakan.

1. PENGANTAR

Dalam Bab ini diterangkan bagaimana menggunakan Metrik Kompleksitas Siklomatik dalam pengujian. Tujuan dari penggunaan metoda ini ialah mengidentifikasi dan mengkoreksi struktur utama kode yang kompleks untuk menguji dan untuk menunjukkan path yang melalui program yang diperlukan untuk pengujian. Teknik tersebut haruslah membantu seorang programmer dalam mendesign program.

Bagian ini membahas tentang metrik kualitas perangkat lunak, metrik kompleksitas siklomatik dan bagaimana menerapkannya pada pengujian software.

Metoda pengujian yang digambarkan, ditampilkan dalam 2 fase. Fase pertama adalah untuk kuantitas dan batas kompleksitas program guna memperbolehkan seluruh pengujian. Kuantifikasi ini disertai penggunaan ukuran kompleksitas yang menunjukkan jumlah minimum dari perbedaan path yang harus diuji. Fase kedua adalah pengujian aktual, dimana jumlah path yang diuji dipaksa untuk sama nilainya dengan ukuran kompleksitas.

Perhatian pertama ditujukan pada pembatasan kompleksitas program untuk meyakinkan pengujian. Akan tetapi sudut pandang ini merupakan penyederhanaan yang berlebihan dari masalah tersebut. Dalam mendisain program, tingkat disain menurut tipenya dihasilkan dalam bentuk bagan, seperti yang ditunjukkan gambar 74, yang ditampilkan secara nyata susunan sebuah fungsi program kedalam modul-modul yang berbeda. Jika bahasa pemrograman yang digunakan adalah FORTRAN, maka puncak modul menurut tipenya merupakan kode utama dan setiap modul-modul lain biasanya merupakan Subrutin, fungsi atau secara ekstemal disebut Program. Jika COBOL yang digunakan sebagai bahasa pemrograman, maka puncak kode utama dan modul- modul lainnya menurut tipenya berbentuk paragraf.

Disain modular penting bagi kualitas produk akhir, setiap modul-modul tersebut harus memiliki bentuk seperti :

- *Testability* - Usaha pengujian untuk validasi setiap modul harus teratur.
- *Comprehensibility* - Setiap modul harus dapat dibaca dan dimengerti.
- *Reusability* - Jika modui-modul didefinisikan dengan baik, boleh jadi reusable ada dalam sistim yang berbeda.
- *Maintainability* - Pekerjaan memodifikasi dan pengujian kembali setiap modul dalam fase operasional harus teratur.

Proses design modular ini diarahkan oleh 2 prinsip :

- Dekomposisi fungsional yang diwakili oleh hirarki design yang menghasilkan beberapa kebebasan. modul-modul kohesif yang menyediakail dekomposisi natural dari masalah tersebut
- Modularisasi juga diarahkan oleh ukuran atau pengujian setiap modul-modul Bahwa modularisasi harus menghindari modul-modul yang sifatnya sangat kompleks dan tidak stabil

Struktur prinsip pengujian dalam bahasan ini akan membatasi kompleksitas program melalui batasan dan kompleksitas di dalam modul-modul yang setiap poin-poinnya dapat diuji. Jika ditemukan modul-modul istimewa yang melebihi kompleksitas awal, untuk lebih jauhnya lagi akan diperlukan perbaikan disain. Sebagai contoh, jika modul 11 pada gambar 74 terlalu kompleks, kreasi dari beberapa pengujian di bawah modul 11 akan diperlukan. jadi, pendeknya, kemampuan pengujian program akan dijamin oleh batasan dari kompleksitas setiap modul dengan program.

2. UKURAN KOMPLEKSITAS

Ukuran kompleksitas akan dibatasi oleh jumlah dari pengembangan path dalam program pada saat disain dan kondisi pengkodean jadi pengujian akan teratur hingga kondisi selanjutnya. Satu dari sekian banyak aIasan untuk membatasi pengembangan path, sebagai pengganti batasan dasar pada panjang program, adalah mengikuti masalah: program yang relatif pendek dapat memenuhi keseluruhan jumlah path. Pendekatan di sini terlalu dibatasi oleh jumlah dasar (pengembangan) daripada path-path yang akan mewakili semua path ketika dikombinasikan.

Definisi 1
Bilangan sik/omatik v(G) dari graph G *dengan n simpul, e ruas dan* 1 *komponen penghubung ada/ah* :
v(G) = e - n + 1

Teorema 1
Jika G *graph terhubung kuat, maka bilangan siklomatik sesuai dengan jumlah maksimum dari path yang bebas linier.*

Sebagai contoh, grafik kontrol dalam gambar 75 mempunyai 7 simpul, (a) menuju (g), simpul masukkan dan keluaran (a) dan (g), dan 10 ruas.

Untuk mengaplikasikan teorema 1, grafik harus secara jelas dihubungkan, yang mana artinya diberikan 2 bentukkan (a) dan (b), ada beberapa path dari (a) ke (b) dan path dari (b) ke (a). Untuk lebih puasnya. kita gabungkan penambahan tepi dengan grafik, ya.ng mana cabang dari keluaran (g) ke masukkan (a), seperti ditunjukkan dalam gambar 76.

Teorema 1 sekarang telah diterapkan, dan kondisi jumlah maksimal dari path yang bebas di G' adalah 11 – 7 + 1. (G hanya mempunyai satu komponen penghubuug,jadi kita anggap p = 1). Umumnya kasus di mana p > 1 digunakan untuk design kompleksitas (lihat bagian 7). implikasi, dasar dari pengembangan path yang jika dikombinasikan akan mewakili semua path. Misalnya, 5 path ditunjukkan melaiui dasar bentuk :

b 1 :abcg

b2:a(bc)*g
b3:abefg
b4:adefg
b5:adfg

Catatan: Notasi (bc)*2 berarti iterasi looping (bc) sebanyak dua kali.

Jika ada path yang tidak beraturan dipilih harus sesuai dengan kombinasi linear dari path dasar b 1 – b5. Sebagai contoh, path abcbefg sesuai dengan b2 + b3 - bl. dan path a(bc)*3g sesuai dengan 2 * b2 – b1. Untuk mengetahui ini, penting untuk menjumlahkan tepi-tepi di G (gambar 77) dan menunjukkan dasar seperti vektor-vektor ruasnya (gambar 78).

Path abcbefg dijelaskan seperti pada vektor mas yang ditunjukkan oleh gambar 78, dan ini sesuai dengan b2 + b3 - b 1 di mana penambahan dan pengurangan adalah salah satu komponen pengerjaan Dalam cara yang serupa, path a(bc)*3g yang ditunjukkan dalam gambar 78 sama dengan 2 . b2 - b 1

Titik berat pembicaraaan ini adalah bahwa proses penambahan kelebihan tepi di G dibentuk hanya untuk membuat graph yang secara jelas dihubungkan, jadi Teorema 1 telah diterapkan. Pada saat penghitungan kompleksitas program atau pengujiaan program, mas yang lebih tidak dikeluarkan, tapi bisa jadi ditunjukkan oleh penambahan 1 pada jumlaah ruas. Jadi Kompleksitas v didefinisikan sebagai berikut :

$$\mathbf{v = (e + 1) - n + 1 = e - n + 2}$$

3. CONTOH

Graph dalam gambar 79 sampai 90 dijelaskan dalam susunan tambahan kompleksitas bagi hubungan antara jumlah kompleksitas dan intuisi pengertian kita tentang kompleksitas graph.

Suatu bahan penting dalam metodologi pengujian dibatasi oleh logika program hingga pengembangan suatu susunan, pertama, program harus dapat dimengerti, dan kedua, banyaknya pengujian yang dikehendaki untuk memeriksa logika yang tidak berlebihan.

Dalam praktek, besarnya program sering memiliki kompleksitas rendah dan program kecil mempunyai kompleksitas tinggi. Karena ini, praktek umum pada percobaan untuk membatasi kompleksitas dengan hanya mengontrol bagaimana routin akan ditempati adalah tidak sebanding dengan masukkannya. Ukuran kompleksitas telah digunakan dalam produksi sekelilingnya dengan membatasi kompleksitas setiap modul sampai 10. Programmer menghendaki penghitungan kompleksitas seperti pengembangan *routin,* dan jika memiliki lebih dari 10, kesemua yang dikehendaki telah tercapai dan subfungsi modular atau perancangan kembali pera.'lgkat lunak. Hanya situasi di mana batas 10 tampak seperti tidak beralasan dan adanya pengecualian adalah dalam besamya statemen CASE di mana jumlah pengembangan suatu kelompok mengikuti seleksi fungsi. (lihat gambar 86 sebagai contoh grafik dari statemen CASE)

4. PENYEDERHANAAN

Penyederhanaan pertama memperbolehkaan penghitungan v dengan menghitung *splitting nodes* pada graph. Kal ini mempunyai lebih daripada satu hasil dan digabungkan dengan sebuah kondisi dalam program sumber (gambar 91).

Dalam FORTRAN, bentuk pembagian akan digabungkan dengan IF, kondisi GOTO, penghitungan GOTO, atau statemen DO. jika S adalah jumlah dari *splitting nodes* dalam graph, maka v = S + 1. Sebagai contoh, dalam gambar 92, *splitting nodes* adalah (a), (b), (c), dan (cl), jadi v = 4 + 1

Dari setiap *splitting nodes* dalam graph akan digabungkan dengan sebuah predikat atau kondisi dalam program, ekspresi v = S + 1 dapat dihitung dengan kondisi penghitungan sederhana di program sumber Kenyataannya, jumlah dari kondisi adalah indikator kompleksitas yang lebih baik daripada jumlah predikat, dari suatu susunan predikat bisa mempunyai lebih dari satu kondisi, misal:

IF cl OR c2 THEN bl ELSE b2

Dengan adanya dua jalan terakhir maka predikat bisa menjadi benar, statemennya digambarkan dalam gambar 93.

Perhatikanlah bahwa kompleksitas dan suatu graph dan statemen adalah 3. Juga perhatikan bahwa statemen yang mengikuti adalah ekuivalen dan mempunyai kompleksitas 3.

```
IF cl THEN bl
ELSE IF c2 THEN b1
          ELSE b2
```

Jika suatu program mengandung predikatnya sebanyak n-buah, seperti pada statemen CASE dengan n CASE, n-buah predikat dikontribusikan n - 1 pada penghitungan S. Sebagai contoh, gambar 94 tentang CASE predikat (a) memiliki 3 hasil, lalu dua dikontribusikan ke S. lni berarti v = 2 + 1. Perhatikan bahwa statemen CASE dengan n case dapat disimulasikan dengan n - 1 di dalam statemen IF- THEN-ELSE, yang mana menghasilkan kompleksitas yang sama.

Penyederhanaan kedua memperbolehkan penghitungan e - n + 2 dengan menghitung daerah di dalam graph kontrol dengan menggunakan prinsip Euler :

Jika G *adalah graph terhubung planar dengan n simpul, e ruas dan r region, maka* $n - e + r = 2$.

Dengan mengubah susunan, kita dapatkan r = e - n + 2. Jadi jika graph G planar, maka penghitungan kompleksitasnya melalui penghitungan region (gambar 95).

5. KRITERIA STRUKTUR PENGUJIAN

Kriteria yang harus bisa terpenuhi untuk dapat melengkapi teknik struktur pengujian suatu program dengan kompleksitas v adalah :

1. Setiap hasil dari setiap keputusan harus dieksekusi pada bagian akhir.
2. Pada akhir path v harus dieksekusi.

Kekuatan dari teorema di bagian metodologi pengujian ini adalah menempatkan jumlah kompleksitas dari pengujian path v yang mempunyai 2 kedudukan penting :

1. Susunan pengujian dari path v dapat direalisasikan (jika hat ini dilanggar, bagian 8 akan mendemonstrasikan bagaimana suatu program dengan kompleksitas lebih kecil akan memenulu keperluan yang sama).
2. Pengujian yang berkisar tentang pengembangan path v adalah kombinasi linear secara berlebihan dari path basis.

Secara operasional, mengikuti pengalaman dengan teknik seperti ini telah diobservasi. Jika kompleksitas program-program kecil (range 1 - 5), maka teknik pengujian konvesional biasanya bisa memenuhi struktur kriteria pengujian.

Metodologi yang sering digunakan [MILL] menginginkan :
1. Semua statemen dieksekusi.
2. Setiap keputusan harus dieksekusi.

Secara tipikal/tipe susunan data pengujian mengisi kriteria :

tl : # (# notasi karakter selain A)
t2 : ABC^ (" notasi karakter selain X)
t3 : ABCX

Program SEARCH mengeksekusi setiap pengujian t 1 sarnpai t3, jadi data pengujian gagal untuk mendeteksi kesalahan.

Pengujian b4 menghasilkan BOOL = "true" dan COUNT = 0, jadi ini menunjukkan program tidak terdapat spesifikasi.

6. IDENTIFIKASI PENGUJIAN PATH: METODA DASAR

.

Teknik yang digambarkan di sini memberikan metodologi spesifik untuk mengidentitas susunan path kontrol dan data pengujian memenuhi kriteria struktur pengujian. Teknik, jika diaplikasikan, menghasilkan susunan data pengujian dan path kontrol sesuai dengan jumlah kompleksitas siklomatik sebuah program. Teknik ini secara lebih jelasnya disebut metoda dasar; ini berdasarkan kepada metoda struktur pengujian karena memberikan teknik yang lebih spesifik untuk menidentitas data uji yang aktual dan path pengujian.

Tahap pertama adalah path "dasar" fungsional yang melalui program yang mana merepresentasikan fungsi yang berhak dan tidak hanya *error* yang keluar. Pemilihan path dasar yang pertama tidak terikat apapun. Untuk merepresentasikan fungsi yang

dibuktikan dalam program, seperti menentang path yang salah yang menghasilkan pesan kesalahan atau memperbaiki kembali prosedur. Untuk pengujian dasar, semua kebutuhan fungsional untuk diimplementasikan pada metoda dasar. Juga lihat data yang akan menghasilkan kesalahan. lni seharusnya direalisasikan bahwa fungsional dasar dari path yang direpresentasikan sama dengan keputusan yang diambil dengan jalan khusus.

Tahap kedua adalah menidentifikasi path kedua dengan melokasir keputusan pertama dalam path dasar dan menyimpan hasilnya" sementara secara simultan jumlah maksimum dari keputusan dasar sama seperti path dasar. Ini sepertinya menghasilkan path kedua yang mana terdapat perbedaan secara minimum dari path dasar.

Tahap ketiga adalah kembali kepada keputusan pertama dari nilai path dasar dan mengidenifikasi serta menyimpan keputusan kedua pada path dasar. Hal ini akan menghasilkan path ketiga dengan perbedaan yang minimum terhadap path dasar. Kemudian uji lagi path ketiga ini.

Prosedur ini berlanjut hingga salah satu path melalui setiap keputusan dan mengambilnya dari nilai dasar, sementara keputusan lain menuju nilai dasar.

Penyeleksian path dasar tidak terikat apapun, tidaklah terlalu penting untuk menyusun data pengujian bagi sebuah program. Sehingga ada beberapa susunan data pengujian yang memenuhi kriteria. sruktur pengujian. Penerapan dari metoda dasar, bagaimanapun menurunkan susunan data uji dengan kedudukan sebagai berikut:
- path v yang bebas akan diturunkan
- setiap ruas dalam graph suatu program akan di dilintasi

Contoh kasus untuk gambar 98 (lihat halaman 435)

7. INTEGRASI PENGUJIAN

Pengertian dari kompleksitas diperoleh dari jumlah siklomatik suatu grafik. Dalam pembicaraan ini. bagaimanapun, dibatasi oleh grafik yang hanya dengan satu komponen Di bagian ini, kita akan mengadakan pendekatan umum untuk graph tertutup yang mempunyai beberapa komponen. Penerapannya akan menjadi ukuran kompleksitas disain; secara lebih spesifik, kita akan mengkuantitaskan usaha yang diminta untuk menampilkan integrasi pengujian dari beberapa modul dengan struktur disain.

Kita fokuskan untuk suatu modul, dan aplikasi pengujian pada level unit. Modul secara tipikal direpresentasikan dalam FORTRAN, PLl atau program PASCAL sebagai prosedur, fungsi, atau kode baris utama Dalam COBOL, modul secara tipikal diekspresikan sebagai paragraf yang mana referensinya berasal dari beberapa tempat didalam program.

Gambar 100 adalah representasi standar dari design di mana M adalah puncak dan disebut modul A dan Modul B. Design pada gambar 100 diimplikasikan dengan mengikuti M, A dan B adalah modul distrik. Kesemuanya itu memiliki spesifikasi internal dan memiliki data pengujian yang unik Modul A dan B dipanggil dari M.

Kesemuanya, mungkin juga dipanggil dalam konteks yang berbeda oleh modul yang lainnya dan bisa juga pada librari program. Bentuk ini berbeda dari situasi di mana kode A dan kode B akan dihubungkan dengan M.

Gambar 101 adalah graph yang menunjukkan algoritma yaang kita temukan dalam modul M, A dan B. Graph pada gambar 101 memiliki 3 komponen, M, A dan B; setiap graph .yang kita bicarakan hanya mempunyai 1 komponen.

Kita harus menambahkan ruas ekstra pada setiap komponen dalam graph untuk memenuhi kondisi konektifitas dari teorema 1. Karena itu, untuk lebih umumnya diekspresikan v = e – n + 2p; ini adalah ekspresi kompleksitas sistem dari disain dengan beberapa komponen graph, yang bertentangan dengan spesifik v = e - n + 2, yang diterapkan untuk 1 komponen.

Saat jumlah komponen sama dengan 3 (P=3), penghitungan kompleksitas yang diberikan v = 13 - 13 + 2*3. Design kompleksitas dari 6 representasi usaha pengujian yang diminta untuk menampilkan integrasi puncak bawah pada 3 modul M, A dan B. Sebagai contoh, gunakan strategi integrasi *Top-Down*, dengan mengikuti :

- *One test*, diperlukan untuk memeriksa kode dalam M. Sisa suatu simulasi aksi dari A dan B dipanggil untuk membiarkan pengujian M.
- *Two tests*, diperlukan untuk memeriksa logika A. Setiap panggilan di A dijalankan
- melalui M dalam susunan untuk merninta A. selama pengujian A, sisa sesuatu untuk B masih di tempat.
- *Three tests*, diperlukan untuk memeriksa logika B, seperti di alas, setiap 3 panggilan di B akan dijalankan *top-down* melalui M.

Indikasi Kuantifikasi Kompleksitas, ini memang kasus yang ditunjukkan di atas bahwa 6 pengujian diperlukan untuk mengintegrasi 3 modul.

Bentuk design kompleksitas pada graph dengan beberapa komponen sama dengan penjumlahan kompleksitas unit-level. Dengan contoh di alas, kompleksitas dapat dihitung dengan v = v(M) + v(A) + v(B) = 1 + 2 + 3. untuk pembuktiannya lihat [MCCA].

Penerapan dari kompleksitas design-level adalah berbeda daripada penerapan kompleksitas unit-level. Kompleksitas design-level tidak dibatasi kontrol kompleksitas unit-level: Aplikasi utama dari kompleksitas design adalah kuantitas usaha integrasi dari pengumpulan modul.

Terdapat kesempatan, pada saat kompleksitas disain dapat digunakan untuk membuat perbandingan kompleksitas relatif pada subsistem dalam keseluruhan design. Kuantitasi ini menunjukkan kompleksitas subsistem yang akan memberikan kestabilan pada beberapa atribut proyek daripada secara biasa digunakan kode garis (LOC). Sebagai contoh, jika kompleksitas disain pada subsistem adalah 2000, dan subsistem kedua kompleksitasnya 30, akan terdapat beberapa implikasi(misalnya, pengujian subsistem dan integrasi secara tertutup dihubungkan dengan kuantitas kompleksitas disain daripada ukuran fisik subsistem dalam kode garis).

8. TEKNIK REDUKSI

Pada saat metodologi ini diaplikasikan pada proyek berjalan atau ketika latihan pengujian dianalisa, biasanya hasil dari jumlah path yang diujikan lebih kecil daripada kompleksitas siklometrik. Konsep metodologi di bawah ini adalah kuantitas dan batas kompleksitas pada program yang lalu diperlukan untuk pengujian pada bagian akhir seperti yang dilalui kuantifikasi.

Ide pada bagian ini adalah bahwa jika pengujian aktual tidak bisa menemukan kompleksitas siklomatik, lalu pengujian yang lainnya dapat diimprovisasikan atau logika program dapat disederhanakan.

Kita asumsikan program yang telah ditulis. kompleksitas v telah dihitung. Jumlah path yang dijalakankan selama fase pengujian adalah kompleksitas aktual (ka). jika ka lebih kecil dari v, kondisinya seperti :

1. Program yang mengandung penambahan path dapat diuji. .
2. Kompleksitas v dapat dihasilkan oleh v - ka (v - ka keputusan dapat diubah dari program)
3. Porsi program yang dapat dihasilkan pada kode in-line (looping dengan panjang tetap telah digunakan dalam susunan ruang konservasi)

Pengujian path yang aktual dalam program dideterminasikan oleh data flow dan kondisi data pada keputusaan khusus. Karena data flow, jumlah path bisa direalisasikan di dalam program yang diberikan. Poin dari bagian ini adalah pada saat data flow dan kondisi data dipertimbangkan, harus menjadi path v yang terakhir atau program yang lainnya memang bisa dihasilkan.

Pertimbangkan Program berikut :

```
j = 1;
IF I <= 3 THEN I=2
          ELSE j = 14;
IF (I + J) <= 6 THEN OUTPUT (I,J);
```

Kompleksitasnya tiga dan graph kontrol ditunjukkan pada gambar 102.

Jelas bahwa ka = 2, hanya path yang direalisasikan pada grafik TT, FF--- satu path di mana 1 <= 3 adalah benar dan (I + J) <= 6 juga benar, dan path kedua di mana kedua kondisinya tidak ada penambahan path untuk diuji, dan tidak ada loop dengan panjang tetap, program bisa dihasilkan dengan kompleksitas 2 :

```
J = 1
IF I <= 3 THEN
          BEGIN
            I=2
           OUTPUT(I,J)
          END
            ELSE
               J = 14,
```

Kesimpulannya, kompleksitas siklomatik v pada program dapat dipikirkaan sebagai spesifikasi untuk menguji path. Jika program yang diberikan tidak memiliki path v akhir yang diujikan, lalu pengujian lainnya tidak lengkap atau terdapat logika eksesif yang bisa diubah. Jika ada logika yang tidak dapat diuji maka logika tersebut harus diubah kembali.

Untuk sistem yang lebih besar dan penerapan yang pasti, ini membentuk suatu objek yang mungkin sangat sulit dan tidak bisa dicapai Sebuah kasus yang mana hasilnya tidak mungkin ada adalah :

- Program defensif
- Tidak mempercayai hardware
- Kesalahan pemrograman

Setiap usaha seharusnya dibuat untuk menghindari hal di atas.

9. KOMPLEKSITAS PENTING (DASAR)

.

Pertanyaan menarik yang digabungkan dengan kompleksitas program adalah kuantitas dari struktur suatu program. Bagaimana kita menentukan suatu derajat kuantitas yang mana program yang telah ditulis hanya menggunakan struktur kontrol standar flow yang dibentuk seperti pada gambar 105.

Ini sangat renting untuk dipertimbangkan, suatu dasar pengurangan kompleksitas suatu program di mana v melebihi 10 adalah perbaikan subfungsi dari flow pengontrol utama sehingga menjadi subrutin atau fungsi. Akan berputar keluar jika struktur program (hanya menggunakan konstruksi SEQUENCE, UNTIL, WHILE, IF, CASE), secara kompleks dapat dikurangkan dalam cara straighforward. sebagai contoh, grafik pada gambar 106 dan 107, struktur programnya dapat dikurangkan pada suatu program dengan kompleksitas 1 dengan membuat satu-masukkan-satu-keluaran subgrafik di dalam fungsi.

Proses pengurangan adalah proses penempatan kembali subgraph yang cocok dengan bentuk *single-entry, single-exit.* Kompleksitas renting atau dasar adalah menentukan arah ke bawah dalam susunan struktur program tersebut.

Graph G' adalah graph reduksi yang dihasilkan dari perubahan semua bentuk yang cocok. *single-entry, single-exit* dari subgraph. Juga, ruas atas dan ruas yang melompat seharunya diubah jadi bentuk pertama adalah bentuk keputusan dan bentuk terakhir adalah bentuk pengumpulan. Kompleksitas dasar dari graph G ditentukan ev = v(G')

Meskipun G1 melalui G3 merniliki v = 5 (gambar 107 sampai 109), setiap subgraph G1 dapat diubah, dimana G3 tidak dapat dikurangkan pada semuanya. Jika terdapat graph yang kompleks, ini akan bisa ditentukan, G1 bisa dikurangkan di dalam subrutin, setiap kompleksitas yang kurang dari 10, tetapi G3 tidak dapat dikurangkan. Karena itu. modularisasi lebih jauh akan meminta program G3 dilempar keluar dan I program baru dirancang kembali.

Contoh pengurangan kompleksitas dengan menganut kepada struktur control standar seperti flow yang dibentuk pada gambar 110. Ini ditulis kembali dari gambar 96. Kompleksitas program adalah 6, sementara kompleksitas yang ditulis kembali adalah 4. Kompleksitas dasaar dari program adalah 3 , sementara kompleksitas dasar yang ditulis kembali adalah 2.

10. MODlFIKASI PROGRAM

Beberapa pendapat merniliki indikasi bahwa perawatan perangkat lunak dan modifikasi membutuhkan sebanyak 70 persen dari harga total daur hidup pembangunan perangkat lunak. Banyak dan aktifitas utama ini memperlihatkan modifikasai dan pengujian kembali keberadaan program, yang mana keberadaan metodologi sangatlah kecil. Pada bagian ini akan diperkenalkan prosedur untuk menampilkan modifikasi dan pengujian dalam cara yang lebih tersusun.

Jika bidang tambalan atau pemodifikasian program tidak dapat diubah dengan menggunakan struktur flow kontrol, perubahan secara tipikal dibatasi bentuk fungsional (bentuk yang tidak lebih dari hasil suatu tepi). Dalam pemrograman, tipe ini memperlihatkan modifikasi perubahan statemen fungsional seperti input, output dan statemen yang menampilkan penghitungan. Bertolak dari ini, flow kontrol perubah memperlihatkan moditikasi at au penyelipan statemen seperti GOTO's, IF's dan DO-LOOPS yang mana perubahan program menjadi terkontrol.

Metoda pemeriksaan statemen fungsional :

- Mengidentitas sernua strul'tur pengujian pada path yang mengandung bentuk yang harus diganti
- mengeksekusi kembali semua path yang memiliki bentuk yang harus diganti.

Contoh pada gambar 111 yang mengilustrasikan prosedur

Asumsikan bahwa program paada gambar 112 bisa dimodifikasi. Asumsikan juga bahwa bentuk X adalah yang diinginkan programmer dengan kode (b) dan (c) untuk dieksekusi.

Dua bagian selanjutnya akan memberikanlah langkah-langkah dalam menampilkan pengujian utama Langkah pertama adalah merubah identitas yang tidak dapat diuji dengan sesungguhnya. Tahap kedua adalah mengkuantitaskan jumlah pengujian yang diberikan struktur pengganti.

1. Mengevaluasi kompleksitas dasar.
2. Menguji kembali quantifikasi

Gambar 113 menggambarkan situasi di mana Y adalah nama kode cabang dalam dan X adalah kode dari cabang. tiga kasus khusus dalam latihan akan ditunjukkan pada gambar 114.

Dalam kasus di mana Y adalah subgrafik dengan data masukkan unik dan bentuk keluaran, kita dapat menghitung siklomatik kompleksitas v(Y). Seperti kasus, jumlah

dari path yang sesungguhnya adalah 2.v(Y)+!. Kasus pertama dan kedua terpenuhi, kode Y merupakan cabang dalam yang mempunyai single entry dan keluaran.

Sebagai contoh, dalam kasus 2, v(Y) = 3, tujuh pengujian diperlukan untuk menguji path. Tiga path seharusnya diuji melalui masukkan normal dari Y untuk mendemonstrasikan bahwa cabang balik di dalam X tidak di ambil. Tiga path lagi yang membawa cabaang baru di dalam Y menjalankan Y di tiga jalan yang berbeda yang lalu kembali ke X. Dan akhirnya, pengujian terakhir dibuat dari path yang tidak membawa path baru tetapi pengganti X.

11. KESIMPULAN

Dalam bagian ini, tahap operasional dari struktur pengujian dikonsilidasikan dan didaftar ke bawah.

Disain:
Jika algoritma ditulis dalam bahasa pemrograman tingkat tinggi, batas kompleksitas hingga 7. Pengalaman menunjukkan pada saat pengkodean, kompleksitas akan mendekati 10.

Jlka spesifikasi intemal dari modul software tennasuk jumlah kondisinya harus diuji, batas kondisinya sampai 6

Jika informasi tidak sesuai dengan fase disain, modul penghentian yang anda intuisikan akan mendekati kompleksitas 10 dalam submodul dengan kompleksitas kurang dari atau sama dengan 10

Fase Pengkodean :
Membuat grafik flow eksplisit untuk proses pemrograman.
Penghitungan kompleksitas siklomatik v dengan tiga metoda yang digambarkan di bagian 4.

Saat kompleksitas mendekati 10, keljakan kembali fase design dan perbaiki logika di dalam modul, dengan setiaap kompleksitas 10 atau kurang(Pengecualian pada statemen CASE atau proyek yang lebih spesifik).

Fase Pengujian Unit:

Mcnggunakan metoda dasar untuk mengidentitas pengujian path dan data hingga jumlah path terpenuhi dengan kriteria
- v adalah pengembangan path yang telah dijelaskan
- setiap tepi pada grafik flow dikemas pada akhir waktu.

Jika kriteria di atas tidak terpenuhi, maka kriteria yang lainnya :
- Banyak pengujian path eksis yang dapat dikerjakan.
- Program yang mengandung logika yang berlebihan bisa diperbaharui.

Simpanlah dokumentasi pada path yang telah diuji untuk fase perawatan, secara tipe dalam unit pengembangan.

Fase perawatan :
Dalam kasus perubahan statemen fungsional, semua path pengujian di dalam unit pengembangan memotong dengan diubahnya statemen fungsional.

Dalam kasus perubahan statemen kontrol :

- Jika kompleksitas dasar akan dibagi, jangan membuat perubahan; program akan menjadi tidak terawat. ambillah pendekatan perbedaan pada modifikasi.
- Di mana kompleksitas dasar tidak akan dibagi, quantitas dan pengujian kembali dengan 2*v(T) + 1

Gambar 115 mengilustrasikan langkah utama dalarn teknik struktur pengujian.

Referensi : Perry, W.E; A Structured Approach to Systems Testing; QED Information Science; 1983