# Perancangan Perangkat Lunak

## Pengukuran Kualitas Perangkat Lunak (Software Metric)

Avinanta Tarigan


Universitas Gunadarma

# Outline

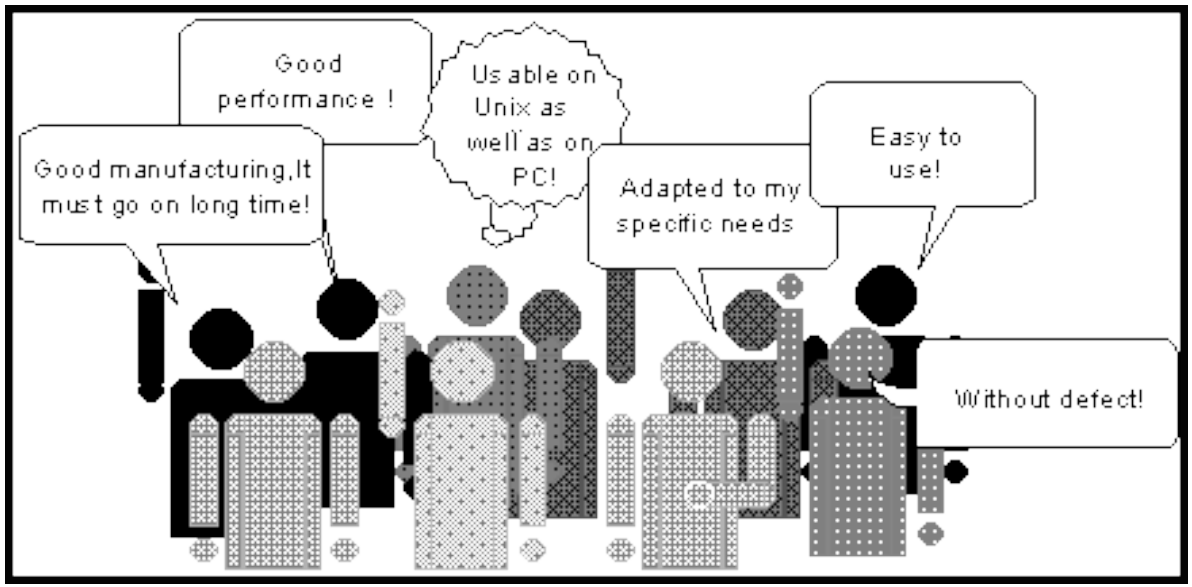


# Perbedaan Cara Pandang I

- " a function of how much it changes the world for the better" (DeMarco)
- McConnell's karakteristik kualitas internal and external:
  - external : bagian yang langsung berhadapan dg pengguna, internal : yang tidak langsung berhadapan dg pengguna.
- "fitness for purpose" (Juran)
- "zero defects" (Crosby)
- "the totality of features and characteristics of a product or service that bear on its ability to satisfy specified or implied needs" (ISO)
- "the degree to which the attributes of the software enable it to perform its intended end use" (DoD)
- Programmer: sulit untuk meyakinkan bahwa kode mereka tdk sesuai dg kebutuhan user

# Perbedaan Cara Pandang II

- Business Analyst: fungsionalitas & spesifikasi teknis adalah segalanya
- Project Manager: budget dan deadlines
- End-User: ??????



# Problem Pada Pengukuran

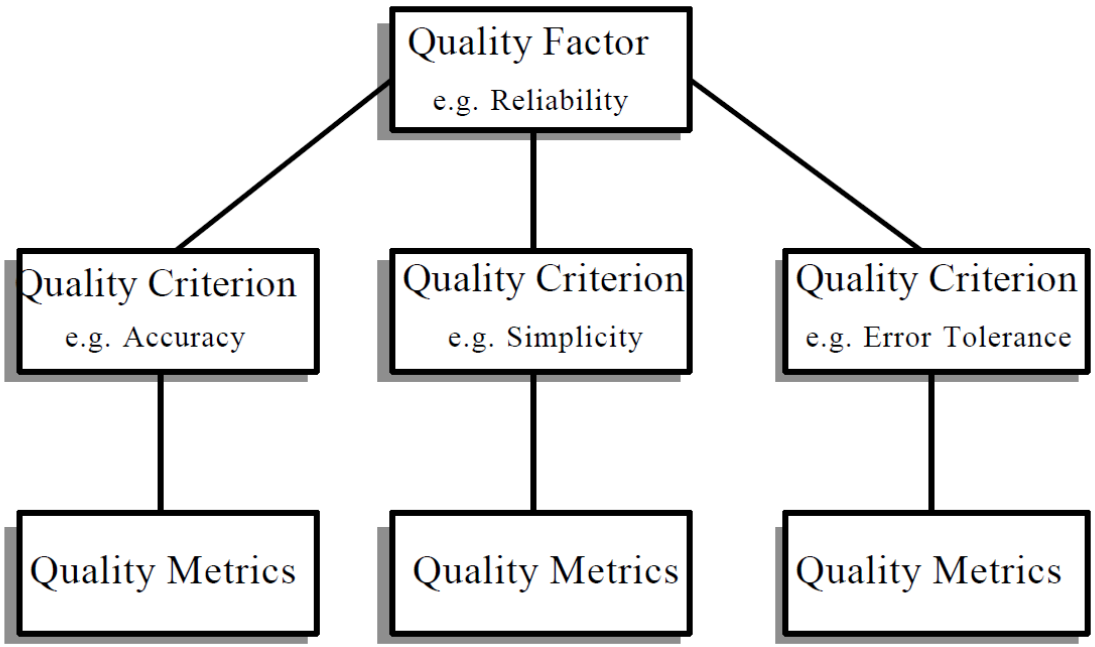
- Tidak didasari pada hukum kuantitas fisika
- Pengukuran tidak langsung: masih dalam perdebatan
- Beberapa berpendapat: kualitas PL tidak dapat diukur
- Multidimensional
- Beberapa standard menggunakan pengukuran kualitatif yang subyektif dan membutuhkan pengalaman penilai dalam pengembangan perangkat lunak

# Lha terus ?

- Q: Jika pengukuran tidak absolut mengapa harus dipelajari ?
- A: Gunanya:
  - berisi cara-cara sistematik dalam menilai kualitas PL untuk digunakan sebagai pegangan dalam pengembangan PL
  - kesadaran atas cara-cara tersebut akan mempengaruhi cara pengembang dalam mengembangkan PL

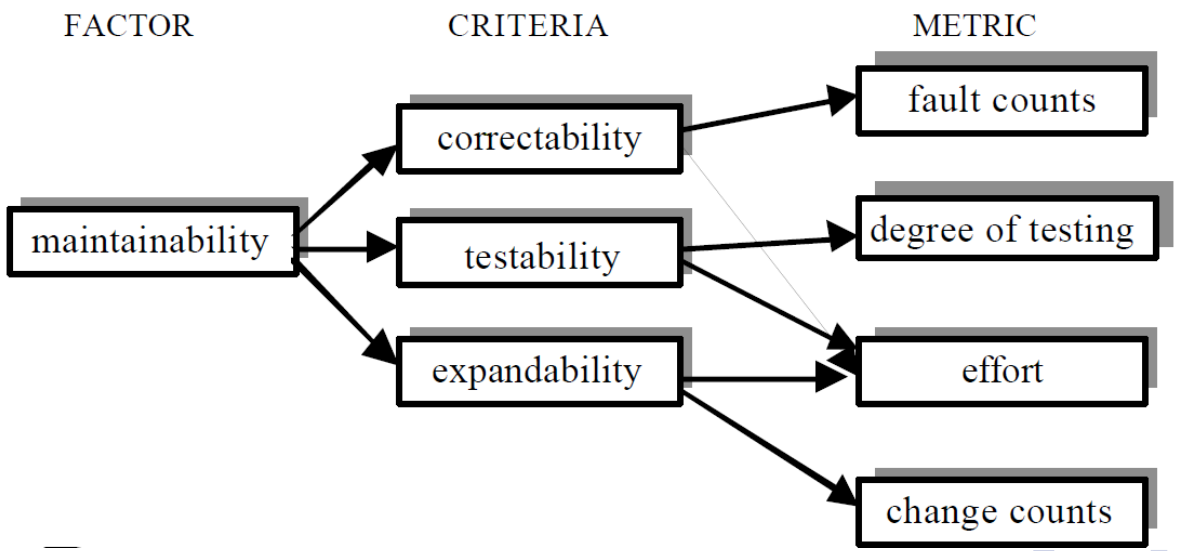
# Model Hirarki I

- Dikembangkan pada tahun 70-an dimana gaya tersentralisasi mainframe sdg jaya-jayanya
- Dua model hirarki : Boehm (1978) and McCall (1977)

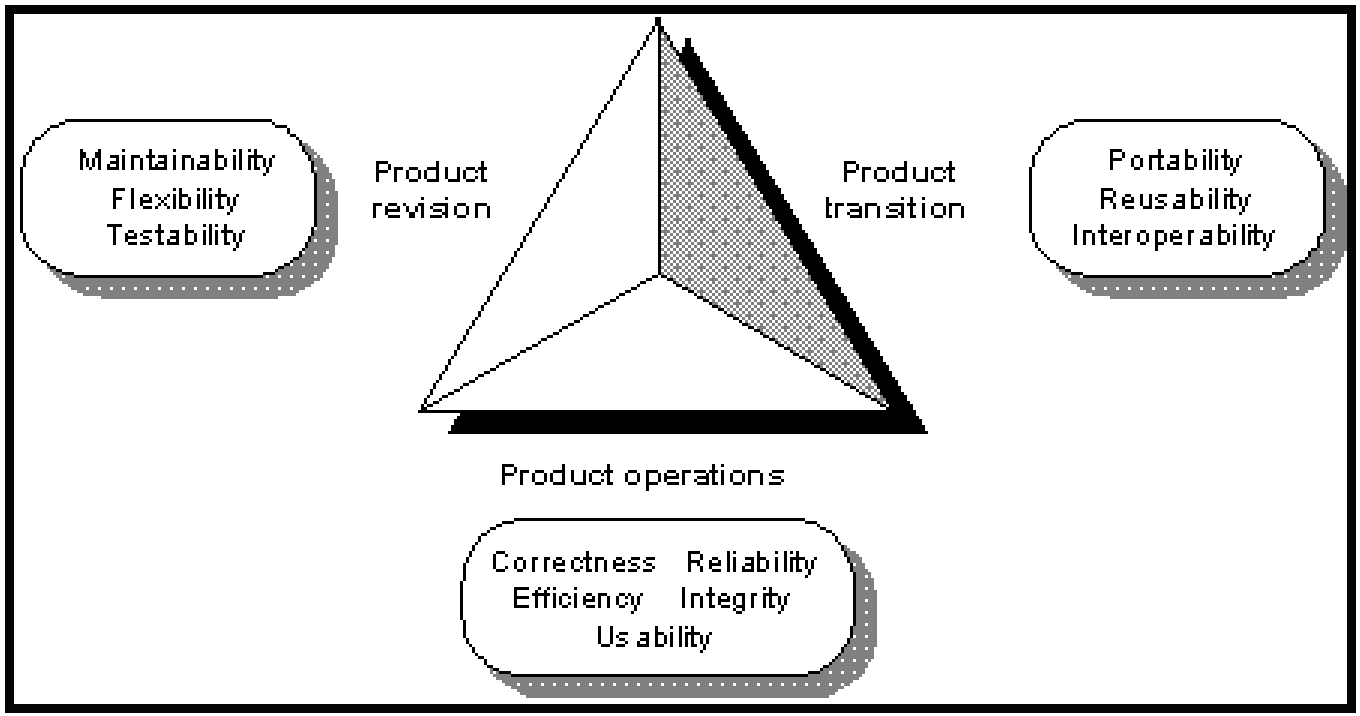


# Model Hirarki II

- Ide dasar model hirarki
  - Fokus penilaian ada pada produk PL final
  - Karakteristik pada abstraksi / level yang tinggi
  - Faktor kualitas terlalu tinggi / tidak langsung (indirect) sehingga harus dipecah menjadi beberapa kriteria yang lebih rendah dan semakin bisa diukur
  - Kriteria2 tersebut dipecah lagi menjadi beberapa penilaian dengan ruang lingkup yang lebih kecil (metrik)



# Faktor Kualitas McCall I




- McCall, Richards, Walters: Kategorisasi faktor kualitas dan karakteristik
- Problem: tidak mudah dalam penilaian tsb

# Faktor Kualitas McCall II

- Perbandingan McCall dengan yang lain :

| No. | Software quality factor | McCall 's classic model | Alternative factor models | |
|---|---|---|---|---|
| | | | Evans and Marciniak model | Deutsch and Willis model |
| 1 | Correctness | + | + | + |
| 2 | Reliability | + | + | + |
| 3 | Efficiency | + | + | + |
| 4 | Integrity | + | + | + |
| 5 | Usability | + | + | + |
| 6 | Maintainability | + | + | + |
| 7 | Flexibility | + | + | + |
| 8 | Testability | + | | |
| 9 | Portability | + | + | + |
| 10 | Reusability | + | + | + |
| 11 | Interoperability | + | + | + |
| 12 | Verifiability | | + | + |
| 13 | Expandability | | + | + |
| 14 | Safety | | | + |
| 15 | Manageability | | | + |
| 16 | Survivability | | | + |

- 41 metrics 23 criteria quality factors.

# Faktor Kualitas McCall III

- menggunakan checklist apakah kondisi yang dipertanyakan ada dalam requirement (R), design (D), atau implementasi (I).
- dijawab dengan ya (bernilai 1) atau tidak (bernilai 0).
- contoh dalam kriteria "completeness" faktor "correctness".

1. unambiguous references (input, function, output) [R,D,I];
2. all data references defined, computed, or obtained from external source [R,D,I];
3. all defined functions used [R,D,I];
4. all referenced functions defined [R,D,I];
5. all conditions and processing defined for each decision point [R,D,I];
6. all defined and referenced calling sequence parameters agreed [R,D,I];

# Faktor Kualitas McCall IV

7. all problem reports resolved [R,D,I];
8. design agrees with requirements [D];
9. code agrees with design [I].

- Menghitung metrik "completeness" :

$$\frac{1}{3} \times \left( \frac{\textit{Number Yes for R}}{7} + \frac{\textit{Number Yes for D}}{8} + \frac{\textit{Number Yes for I}}{8} \right)$$

# Faktor Kualitas ISO 9126 I

- Derivatif dari McCall
- Manifestasi dari "TOTALITAS" fitur PL thd kebutuhan pengguna (requirement)
- Functionality
  - suitability, accuracy, interoperability, compliance, security
- Reliability
  - maturity, fault-tolerance, recoverability
- Usability
  - understandability, learnibility, operability
- Efficiency
  - time behavior, resource behavior
- Maintainability

# Faktor Kualitas ISO 9126 II

- analyzability, changeability, stability, testability

- Portability
  - adaptability, instalability, conformance, replaceability

# Problem Pengukuran Kualitatif I

- Pengukuran Kualitas: diukur dg obyek pembanding yang ada dalam kesamaan kondisi dan konsep yang telah didefinisikan sebelumnya
- Subyektif, harus dilakukan oleh yang benar2 expert, punya pengalaman banyak, dan netral
- Pengukuran tidak langsung: manifestasi kualitas PL bukan kualitas PL yang sesungguhnya
- Konflik:
  - Integrity vs efficiency : control of access requires additional code and processing
  - Usability vs efficiency : improvements in hci may increase the amount of code and power required.
  - Maintainability and Testability vs efficiency : well structured code is easy to test but less efficient.

# Problem Pengukuran Kualitatif II

- Portability vs efficiency : optimised software leads to a decrease in portability.
- Flexibility, reusability and interoperability vs efficiency.
- Flexibility and reusability vs integrity : general and flexible data structures increase the data security and protection problem.
- Interoperability vs integrity : potential for accidental access and opportunity for deliberate access.
- Reusability vs reliability : reusable software tends to be general.

# Prinsip

- Ukuran (measure): indikasi kuantitatif thd atribut suatu obyek
- Metrik: derajat ukuran kuantitatif dari atribut suatu obyek
- Dalam Rekayasa Perangkat Lunak: Indikator adalah metrik untuk melihat
  - proses pengembangan PL
  - proyek PL
  - produk itu sendiri

# Problem

- Kompleksitas PL
- Perbedaan pandang kompleksitas PL
- Pengembangan berbagai macam ukuran akan menimbulkan konflik dalam ukuran itu sendiri
- Pendekatan sains: tidak mudah untuk melakukan experimen yang relevan

# Produk Metrik I

- Metrik untuk analisa model:
  - Fungsionalitas
  - Besarnya sistem yang dikembangkan
  - Kualitas spesifikasi
- Metrik untuk desain model:
  - Metrik arsitektur
  - Pengukuran pada level komponen
  - Desain antarmuka
  - Pengukuran untuk desain berorientasi obyek
- Metrik untuk kode sumber (source code):
  - Kompleksitas
  - Panjang kode sumber
- Metrik untuk pengujian (testing):
  - Kerusakan (bugs)

# Produk Metrik II

- Efektivitas pengujian
- In-process

# Metrik untuk Fungsionalitas I

- Function point Metric (FP) oleh Albrecht 1979
- Tujuan untuk estimasi :
  - besar upaya / biaya
  - banyak error
  - banyak komponen / line of code
- Metode Pengukuran :

| Domain | Banyak | | Simple | Average | Complex | Total |
|---|---|---|---|---|---|---|
| External Input | hasilEI | × | 3 | 4 | 6 | = |
| External Output | hasilEO | × | 4 | 5 | 7 | = |
| External Inquiries | hasilEQ | × | 3 | 4 | 6 | = |
| Internal Logical Files | hasilILF | × | 7 | 10 | 15 | = |
| External Interface Files | hasilEIF | × | 5 | 7 | 10 | = |
| | | | | | Total | = *total* |

# Metrik untuk Fungsionalitas II

- Function point :

$$FP = total \times \left(0.65 \times 0.01 \times \sum F_j\right)$$

- Dimana $F_j$, $j = (1...14)$ adalah value adjustment factors (VAF) didasarkan pada pertanyaan di bawah ini:
  1. Apakah sistem memerlukan backup dan recovery yang handal?
  2. Apakah komunikasi data khusus diperlukan?
  3. Adakah fungsi proses yang terdistribusi?
  4. Apakah performa sistem kritikal?
  5. Apakah sistem akan melayani permintaan yang besar (utilisasi)?
  6. Apakah sistem memerlukan data entry online?
  7. Apakah data entry online memerlukan input dari beberapa terminal yang bersamaan?
  8. Apakah ILF diupdate online?
  9. Apakah in-out-inquiries kompleks?

# Metrik untuk Fungsionalitas III

10. Apakah internal process kompleks?
11. Apakah kode didesain agar dapat digunakan kembali?
12. Apakah konversi dan instalasi dimasukkan dalam desain?
13. Apakah sistem akan diinstall lebih dari satu instance dalam organisasi yang berbeda?
14. Apakah sistem didisain untuk dapat digunakan dengan mudah oleh pengguna?

- Setiap pertanyaan dijawab dengan skala 0...5 (0=tidak penting, 5=penting sekali)
- Berdasarkan "pengalaman" / historical data, nilai FP dapat digunakan untuk mengestimasi line-of-code atau banyaknya resources yang digunakan dalam pengembangan

# Metrik untuk Fungsionalitas IV

- Contoh penggunaan:
  Misalnya sebuah PL mempunyai FP = 40. Berdasarkan pengalaman, 1FP = 100 line-of-code dan 20FP=3 programmer / month dengan gaji 1 programmer 4 juta per bulan. Berarti estimasi PL tersebut akan mempunyai 4000 line-of-code, dibutuhkan 3 programmer selama 2 bulan pengerjaan, dan biaya sebanyak 24 juta untuk membayar programmer tsb.

# Metrik untuk Kualitas Spesifikasi I

- Davis et.al.
- Kebutuhan (requirement) :

$$n_r = n_f + n_{nf}$$

- dimana $n_f$ adalah banyaknya kebutuhan fungsional dan $n_{nf}$ adalah banyaknya kebutuhan non-fungsional
- Dibutuhkan beberapa orang reviewer untuk menjawab pertanyaan / pengukuran
- Hal yang diukur:
  - Specificity : $Q_1 = \frac{n_{ui}}{n_r}$ dimana $n_{ui}$ adalah banyaknya kebutuhan yang diinterpretasikan sama oleh reviewer

# Metrik untuk Kualitas Spesifikasi II

- Completeness : $Q_2 = \frac{n_u}{n_i \times n_s}$ dimana $n_u$ adalah banyaknya kebutuhan fungsional yang unik, $n_i$banyaknya input (stimuli) dalam / disebabkan oleh spesifikasi yang bersangkutan, dan $n_s$ banyaknya state yang dispesifikasikan. Hal ini mengukur persentase fungsionalitas yang penting dan sudah dispesifikasikan.
- Disempurnakan dengan $Q_3 = \frac{n_c}{n_c + n_{nv}}$ dimana $n_c$ adalah kebutuhan yang sudah divalidasi sedangkan $n_{nv}$ yang belum

# Metrik untuk Desain Arsitektur I

- Metode kotak-hitam
- Card dan Glass (1990):
  - kompleksitas struktur
  - kompleksitas data (kompleksitas antarmuka internal)
  - kompleksitas sistem
- Arsitektur berhirarki (call-and-return):
  - kompleksitas struktur dari modul $i$ : $S(i) = f_{out}^2(i)$ dimana $f_{out}(i)$ adalah fan-out module $i$
    - fan-out: banyak modul yang menjadi sub-ordinat langsung dari modul tsb.
    - bagaimana dengan fan-in ?
  - kompleksitas data modul $i$ : $D(i) = \frac{v(i)}{f_{out}(i)+1}$
    - $v(i)$ adalah banyaknya variabel input dan output sebagai parameter dan output modul $i$

# Metrik untuk Desain Arsitektur II

- kompleksitas sistem: $C(i) = S(i) + D(i)$
- Interpretasi: $C(i)$ tinggi - potensial upaya akan besar dalam integrasi dan pengujian

- Fenton (1991) menggunakan morfologi yang simpel:
  - $size = n + a$
  - $n$ adalah banyaknya node, dan $a$ adalah banyaknya arc
  - *depth* (kedalaman) adalah path yang paling panjang ditempuh dari root ke akar
  - *width* (kelebaran) adalah maksimum banyaknya node yang ada dalam setiap level hirarki
  - Mengukur coupling dengan sangat simpel dalam konteks integrasi

# Metriks untuk Desain Berorientasi Obyek I

- Biasanya, penilaian subyektif & berdasarkan pengalaman
- Problem: ukuran dan kompleksitas
- Metrik OO berdasarkan Whitmire:
  1. Besar (size) :
     1. populasi: mengukur banyaknya statik entitas OO (class, operations, dll)
     2. volume: hampir sama dg populasi, tp berdasarkan perubahannya thd waktu
     3. panjang (length): mengukur panjangnya rantai interkoneksi antar klas
     4. fungsionalitas: mengukur "value" software thd penggunanya
  2. Kompleksitas: karakteristik struktural interkoneksi antar class
  3. Coupling (ikatan): koneksi, ikatan, kolaborasi antara elemen-elemen

# Metriks untuk Desain Berorientasi Obyek II

4. Sufficiency (kecukupan): apakah abstraksi (class) mempunyai fitur yang sesuai dg konteks software tsb dikembangkan
5. Completness (kelengkapan): hampir sama dengan sufficiency, tetapi lebih ke arah kelengkapan fitur
6. Cohesion: tingkat kerjasama antara elemen dalam mencapai tujuan dikembangkannya PL tsb
7. Primitiviness: tingkat koherensi dalam class2 yang didefinisikan
8. Similarity: apakah ada kesamaan dalam fungsi, struktur, perilaku, tujuan class-class yang didefinisikan
9. Volatility: tingkat apakah perubahan dimungkinkan dalam desain oleh karena perubahan requirement

# Metrik Berorientasikan Class I

- Harisson, Counsell, Nithi: indikator kuantitatif untuk desain OO
- **Method inheritance factor (MIF)**: mengukur tingkat inheritance class dalam method dan attribute
- diketahui:
  - $M_a(C_i)$ = banyaknya methods yang dapat dipanggil (invoke) dalam asosiasi dg class $C_i$
  - $M_d(C_i)$ = banyaknya methods yang dideklarasikan dalam class $C_i$
  - $M_i(C_i)$ = banyaknya methods yang diturunkan (dan tidak dioverride) dalam $C_i$
- Maka MIF dapat diukur dengan

$$MIF = \frac{\sum M_i(C_i)}{\sum M_a(C_i)}$$

# Metrik Berorientasikan Class II

- dimana $i = 1...T_c$, $T_c$ banyaknya class dalam arsitektur PL tsb, dan

$$M_a(C_i) = M_d(C_i) + M_i(C_i)$$

- **Coupling Factor (CF)** : mengukur faktor ikatan antara class-class

$$CF = \frac{\sum_i \sum_j is\_client\,(C_i, C_j)}{(T_c^2 - T_c)}$$

- dimana $i, j = 1...T_c$, $T_c$ banyaknya class dalam arsitektur PL tsb, dan

$$is\_client(C_c, C_s) \begin{cases} 1 & \text{ada relasi antara client class } C_c \text{ dengan server class } C_s, \text{ dimana } C_c \neq C_s \\ 0 & \text{tidak ada relasi} \end{cases}$$

# Metrik untuk Kode Sumber I

- Halstead mengusulkan hukum kuantitatif dalam pengembangan PL
- Pengukuran :
  - $n_1$= banyaknya operator (secara distinct) yang digunakan dalam program
  - $n_2$= banyaknya operands (secara distinct) yang ada dalam program
  - $N_1$= jumlah munculnya operator
  - $N_2$= jumlah munculnya operand
- Panjangnya program (lines of code):

$$N = n_1 \, log_2 \, n_1 + n_2 \, log_2 \, n_2$$

- Potensial minimum volume dari algoritma ( bits )

# Metrik untuk Kode Sumber II

$$V = N\,log_2\,(n_1 + n_2)$$

- Volume ratio $L$ : rasio volume dari program yang paling kecil (kompak) dengan volume program yang sebenarnya

$$L = \frac{2}{n_1} \times \frac{n_2}{N_2}$$

# Metriks untuk Pengujian I

- Memprediksi (estimasi) upaya (effort) dalam pengujian
- Cyclomatic Complexity sebagai dasar pengukuran kompleksitas
- Module dengan CC yang besar akan lebih berpotensial untuk error dibandingkan module dengan CC yang kecil
- Halstead Effort:
  - Input: $PL$ (program level), $V$ (volume program)

$$PL = \frac{1}{\left(\left(\frac{n_1}{2}\right) \times \left(\frac{N_2}{n_2}\right)\right)}$$

$$e = \frac{V}{PL} \tag{1}$$

- Presentase upaya untuk menguji modul $k$:

# Metriks untuk Pengujian II

$$p\_upaya\_pengujian(k) = \frac{e(k)}{\sum e(i)}$$

- Dimana $e(k)$ didapat dari perhitungan 1 dan $\sum e(i)$ adalah total dari Halstead effort seluruh modul yang ada dalam PL tsb.

# Metrik untuk Pemeliharaan (Maintenance) I

- IEEE standard 981.1-1988 : Software Maturity Index
- Indikasi stabilitas produk PL berdasarkan banyaknya perubahan yang terjadi setelah implementasi
- Ukuran:
  - $M_T$ = banyaknya modul dalam release saat ini
  - $F_c$ = banyaknya modul dalam release saat ini yang sudah dirubah
  - $F_a$= banyaknya modul dalam release saat ini yang ditambahkan
  - $F_d$= banyaknya modul dalam release sebelumnya yang dihapus
- Perhitungan Software Maturity Index:

$$SMI = \frac{(M_T - (F_a + F_c + F_d))}{M_T}$$

- Interpretasi: jika SMI mendekati 1.0 maka produk mendekati keadaan stabil dalam konteks perubahan akibat kesalahan dalam pengembangan sebelumnya

# Kesimpulan (sementara)

- Software Metrik menyediakan metode kuantitatif untuk mengukur kualitas dari atribut internal produk PL
- Pengembang mendapat estimasi terlebih dahulu sebelum waktu pengembangan
- Fokus: fungsionalitas, data, perilaku / behavior
- Tahap pengukuran:
  - Arsitektur (konvensional dan OO)
  - Component-Level: cohesion - coupling - complexity (belum dibahas dalam slide ini, silahkan cari dan bahas bersama)
  - User Interface (juga belum dibahas lho)
  - Source code
  - Pengujian
  - Pemeliharaan

# End

Terimakasih