BSM462
Yazılım Testi

Hafta – 14 – İkinci Kısım

Yazılımda Kusur Tespiti

Dr. Öğr. Üyesi M. Fatih ADAK

fatihadak@sakarya.edu.tr

İçerik

- Yazılım kusur tespiti
- Yazılım kusurlarına yaklaşım
- Yazılım kusur tahmini genel işleyiş
- Yazılım kusur tahmini tarihsel süreç
- Hacim ve karmaşıklık
- Kod satır sayısı (LOC)
- LOC ile KLOC arasındaki fark
- ABC metriği
- Halstead'ın karmaşıklık ölçümü
- Hacim karmaşıklık ilişkisi
- Kontrol akış metrikleri
- McCabe'nin cyclomatic karmaşıklığı
- Myer'in karmaşıklık hesabı
- Veri akış metrikleri
- İşlev noktası (Function Point)

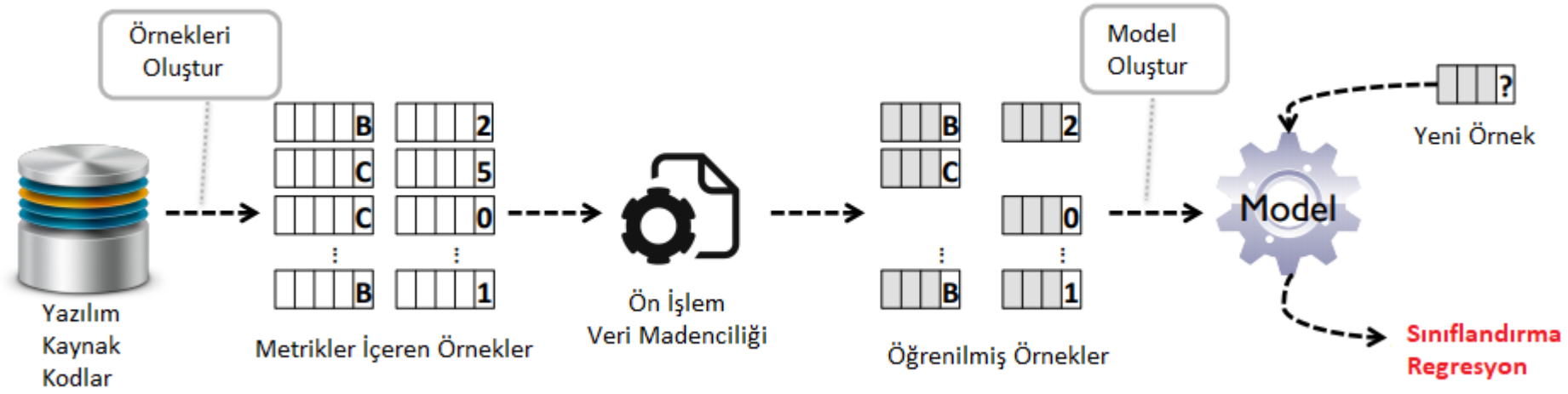
Yazılım Kusur Tespiti

- Yazılımda kusurlar genellikle beklentiler ve özelliklerden sapmalar olarak tanımlanır.
- Üç Problem Yaklaşımı
  - Yazılımdaki kusur sayısını tahmin et.
  - Yazılımın güvenirliğini zamana bağlı başarısızlık sayısı olarak tahmin et.
  - Tasarımın ve test sürecinin yazılım kusur sayısı üzerindeki etkisini anla ve analiz et.

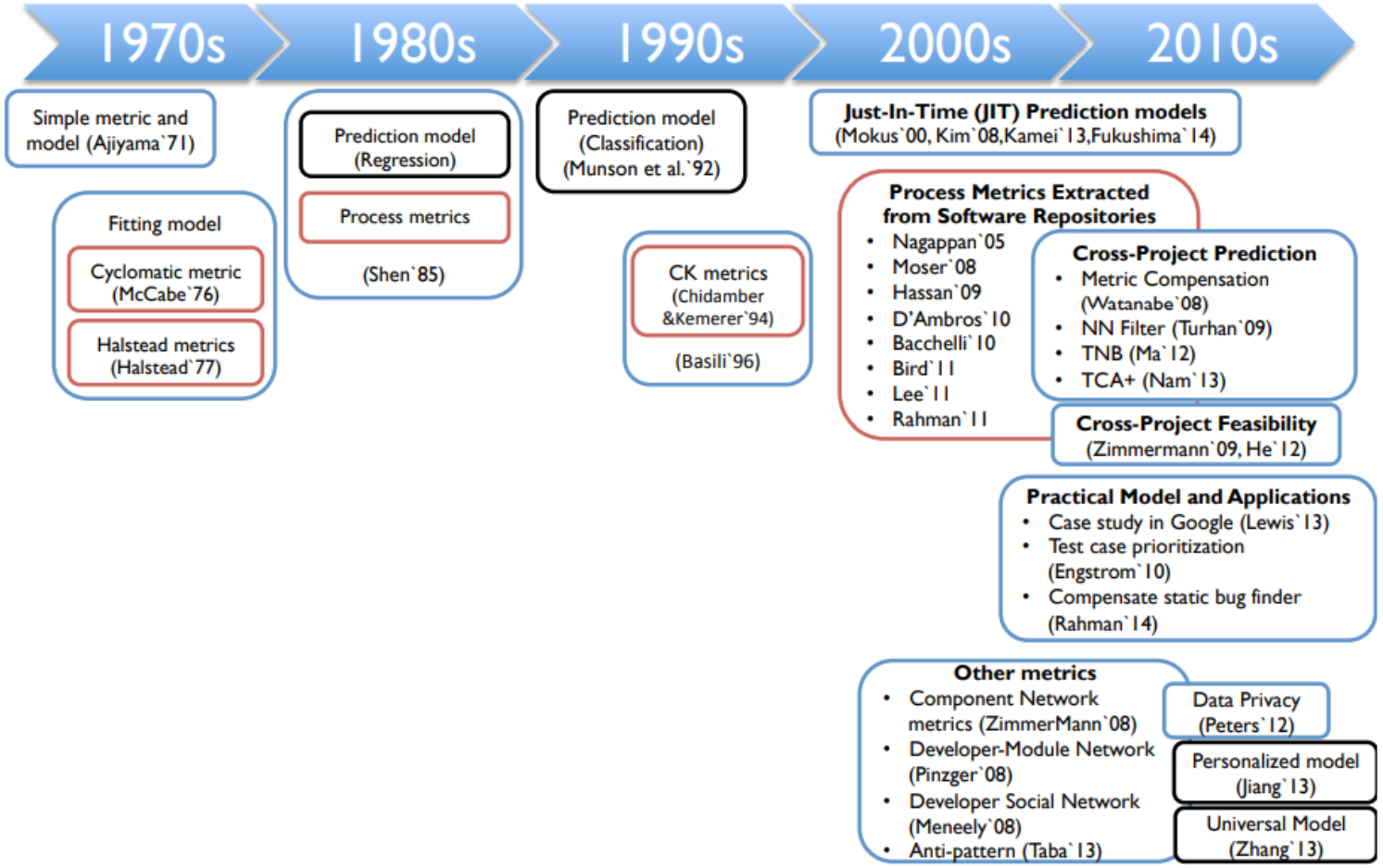
Yazılım Kusurlarına Yaklaşım

- Kusur Önleme (Defect prevention)
- Kusur Tespiti (Defect detection)
- Kusur Düzeltme (Defect correction)

Yazılım Kusur Tahmini Genel İşleyiş

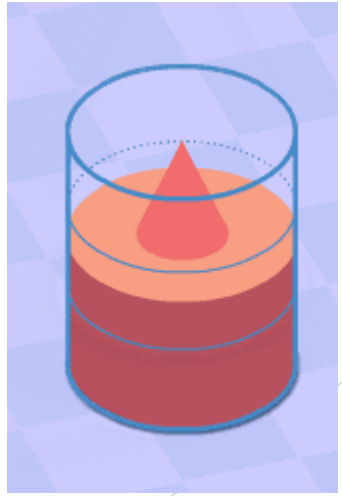


Yazılım Kusur Tahmini Çalışmaları Tarihsel Süreç



Yazılım Hacminin Ölçülmesi

- Yazılım hacim ölçme ilk çalışma 1975 yılında Maurice Halstead tarafından Purdue Üniversitesinde yapılmıştır.
- İlk akla gelebilen parametre kod satır sayıdır.
- Bunun operatörler, operandlar, fonksiyon çağrımları izlemiştir.



Hacim ve Karmaşıklık

- Kod satır sayısı (LOC)
- ABC Metriği
- McCabe'nin Cyclomatic Karmaşıklığı
- Myer'in Karmaşıklık Hesabı
- İşlev Noktası (Function Point)

Kod Satır Sayısı (LOC)

- Avantajları
  - Kolayca sayılabilir
  - Birçok yazılım kusur tahmininde kullanılan bir modeldir.
- Dezavantajları
  - Programlama diline bağımlı özelliktir.
  - Küçük ve işlevsel programlar cezalandırılabilir.

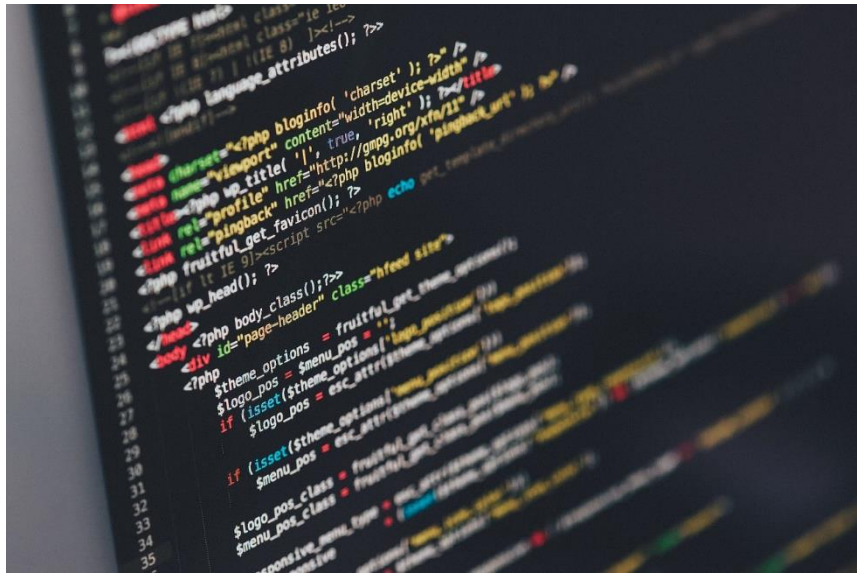
LOC ile KLOC arasındaki fark

- LOC
  - Yazılımın kaynak kodundaki boş ve yorum satırları dahil satır sayısıdır.
- KLOC (Kilo Lines of Code)
  - LOC değerinin 1000'e bölümü ile elde edilir.

- LOC için tavsiyeler
  - Bir kaynak dosyası 4 ile 400 satır arasında olmalıdır.
  - Bir fonksiyonun büyüklüğü 4 ile 40 satır arasında olmalıdır.
  - Bir kaynak dosyasında yorum satır sayısı en az %30 en fazla %75 olmalıdır.

Kod Satır Sayısı (LOC)

- LOC kullanılarak elde edilebilecek nitelikler
  - Üretkenlik = LOC / Bir Ay Kişi Sayısı
  - Kalite = Kusur / LOC
  - Maliyet = Para / LOC



Kod Satır Sayısı Yaklaşımının Zayıflığı

```
bool SearchList(char *cmd, char **list, unsigned n) {

    // search entire list for command string
    for(unsigned i=0; i < n; i++)
        if (stricmp(cmd, list[i]) == 0) return true;

    return false;
}
```

LOC= 8

```
bool SearchList(
    char *cmd,          // command string
    char **list,        // array of strings
    unsigned n)         // max. no. of elements
{
    unsigned i;

    // search entire list for command string

    for(unsigned i=0; i < n; i++)          // for each element
    {
        if (stricmp(cmd, list[i]) == 0)       // matched?
        {
            return true;                  // found match
        }
    }

    return false;                         // no match
}
```

Yorum ve boşluklar sayılır

LOC=19

Aynı Kod Farklı Satır Sayısı

ABC Metriği

- 1997 yılında Jerry Fitzpatrick tarafından tanıtılmıştır.
- A: Atama Sayısı
- B: Çağrım Sayısı (fonksiyon çağrım)
- C: Kontrol Sayısı (if/else gibi)
- Hesaplama sonucunda bir değer elde edilir. Bu yazılımın hacmini belirler.

$$|\text{ABC}| = \sqrt{(AxA) + (BxB) + (CxC)}$$

ABC Metriği

$$|\text{ABC}| = \sqrt{(AxA) + (BxB) + (CxC)}$$

- Bu değer <= 10 ise hacim en iyi düzeydedir.
- Bu değer <= 20 ise hacim kabul edilebilir düzeydedir.
- Bu değer 21 <= ABC <= 60 ise yeniden düzenlenmeye ihtiyacı var
- Bu değer 61 <= ise **kabul edilemez**.

ABC Metriği

- Avantajları
  - Hesaplanması kolaydır
  - Bazı nitelikleri içermesi sayesinde LOC'tan daha avantajlıdır.
  - Programcının kod yazım tarzından bağımsızdır.
  - Bir paket, sınıf, alt modül, dosya olması farketmez
- Dezavantajları
  - ABC kodun gerçek hacmi yerine çalışma boyutunu yansıtır.
  - Birkaç satır kodun olduğu bazı durumlarda sıfır değerini üretebilir.
  - Kod hacim ölçüsü için çalışmalarda ABC metriği **benimsenmemiştir.**

Halstead’in Karmaşıklık Ölçümü

- 1977 yılında Halstead tarafından bir metrik paketi tanıtıldı.
- Bunlar Halstead metrikleri olarak bilinir.
- Yazılım metrikleri alanında bilimsel bir denklem yazan ilk kişidir.
- Amacı LOC’a bir alternatif oluşturabilmekti.
- Bir programı token dizisi olarak sınıflandırır.
  - n1: Bir programda kullanılan operatörlerin tekil sayısı
  - n2: Bir programda kullanılan operandların tekil sayısı
  - N1: Bir programda kullanılan toplam operatör sayısı
  - N2: Bir programda kullanılan toplam operand sayısı

Halstead'in Karmaşıklık Ölçümü

- Fonksiyon tanımlamaları dikkate alınmaz.
- Program Sözlüğü: n=n1+n2
- Program Uzunluğu: N=N1+N2
- Program Hacmi

$$V = N * log_2(n)$$

$$V = 80 * log_2(25) \approx 371$$

Zorluk Düzeyi: Aynı operandlar birkaç kez kullanılmış ise hatalara yatkınlık artar.

$$D=(n1/2)*(N2/n2)$$

Program Seviyesi: Zorluğun tersidir. Zorluk düzeyi azaldıkça yüksek seviyeli program olur.

$$L=1/D$$

```
void sort ( int *a, int n ) {
int i, j, t;

   if ( n < 2 ) return;
   for ( i=0 ; i < n-1; i++ )  {
         for ( j=i+1 ; j < n ; j++ ) {
               if ( a[i] > a[j] ) {
                     t = a[i];
                     a[i] = a[j];
                     a[j] = t;
               }
         }
   }
}
```

3	<	3	{
5	=	3	}
1	>	1	+
1	-	2	++
2	,	2	for
9	;	2	if
4	(	1	int
4	)	1	return
6	[]		

1	0
2	1
1	2
6	a
8	i
7	j
3	n
3	t

	Toplam	Tekil
Operators	N1 = 50	n1 = 17
Operands	N2 = 30	n2 = 8

Halstead’in Karmaşıklık Ölçümü

- Gereken Programlama Çabası: Mevcut bir algoritmayı gerçek bir uygulamaya dönüştürmek için gerekli zihinsel aktivite

$$E = V*D$$

- Programlama Zamanı

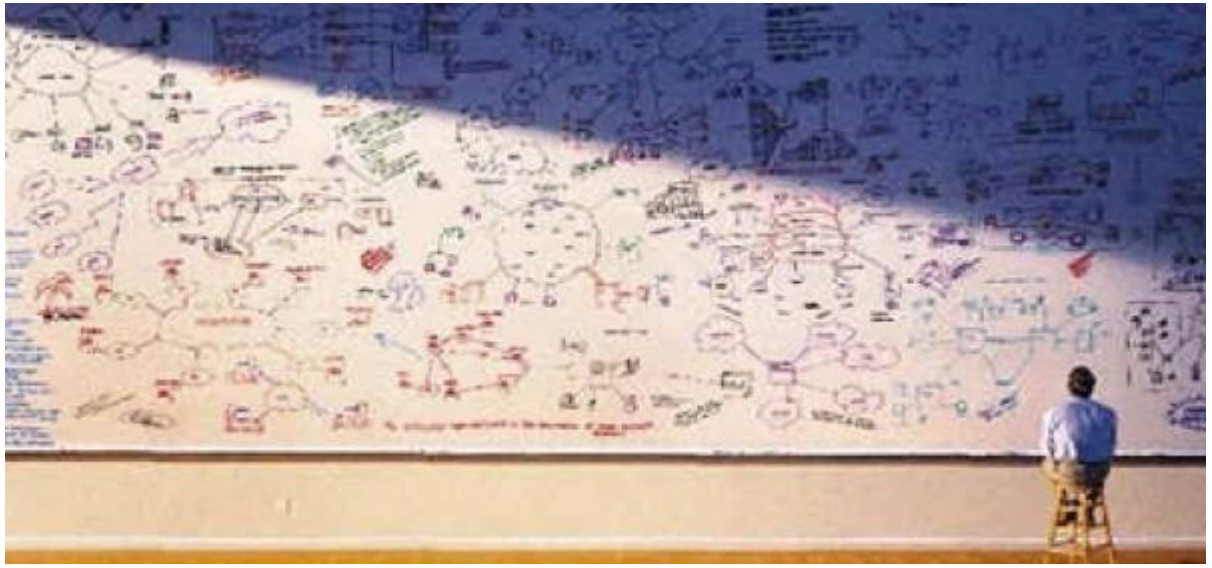
$$T = E / 18$$

Halstead’in Karmaşıklık Ölçümü

- Avantajları
  - Hesaplanması basittir. Program yapısının derinlemesine analizini gerektirmez.
  - Programların genel kalitesini ölçebilir.
  - Literatürdeki birçok çalışma, programlama gayretini, hata oranını ve onarım oranını tahmin etmek için bu ölçümleri kullanmıştır.
- Dezavantajları
  - Operand ve operatörleri ayırt etme zorluğu önemli bir problemdir.
  - Program seviyesinin hangi değeri program karmaşıklığını oluşturduğunu belirtmeden kod karmaşıklığının hacmi arttırdığını söyler.
  - Bu yöntem bir kodun yapısını, kalıtımı ve modüller arası etkileşimi **ölçememektedir**.

Hacim Karmaşıklık İlişkisi

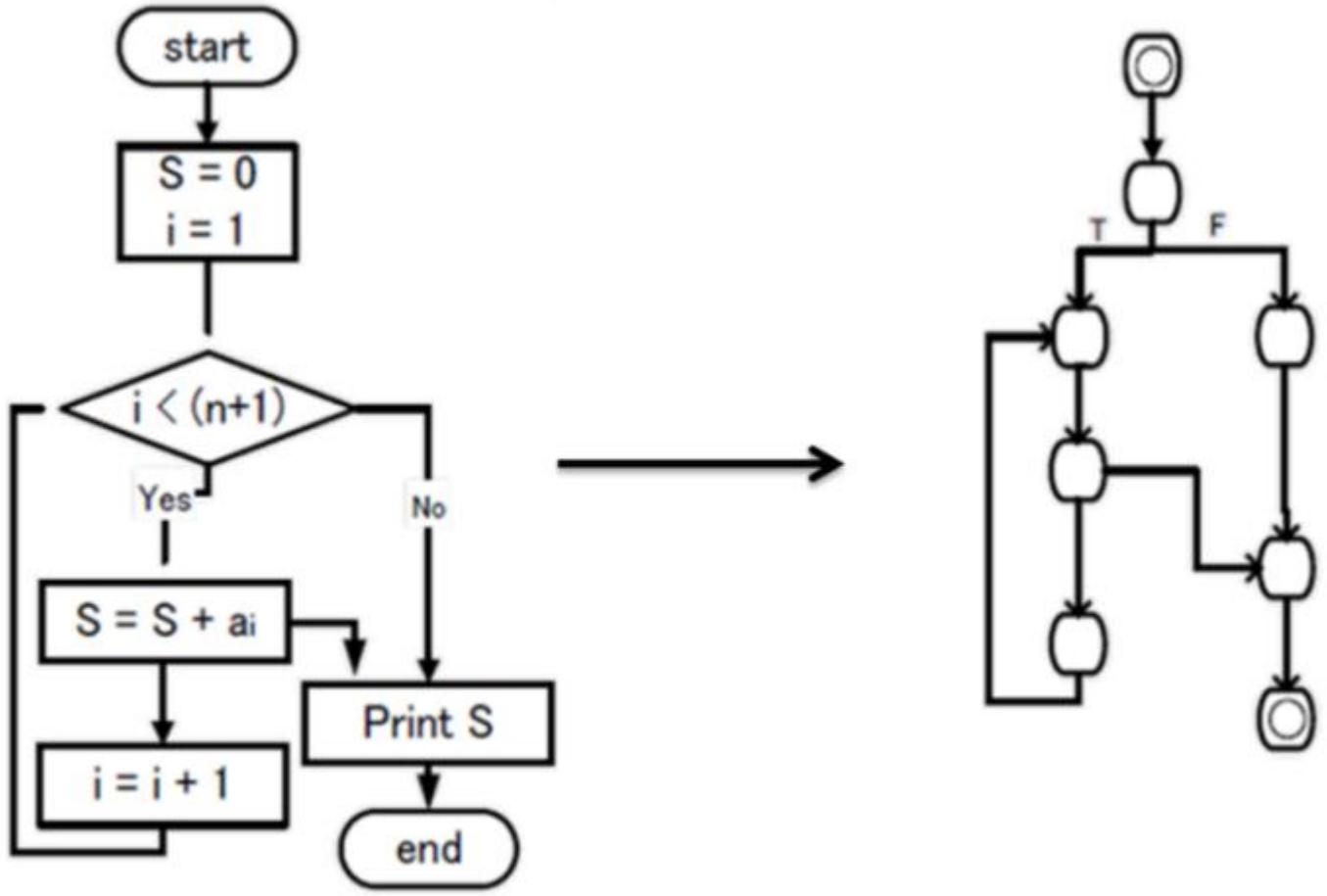
- Yazılım karmaşıklığı ve hacim arasındaki bağlantı göründüğü kadar kolay kurulamamaktadır.
- Karmaşıklık metrikleri iki ana kategoriye ayrılabilir
  - Kontrol Akış Metrikleri
  - Veri Akış Metrikleri



Kontrol Akış Metrikleri

- Bir programda yürütülen komutların dizilimiyle ilgilenir.
- Bu yaklaşım programın döngü ve iteratif akışını dikkate alır.
- Programın kontrol grafiğinin analizine dayanır.
- Bir modülün çağrım ve geri dönüş mekanizmasını graf üzerinde temsil eder.
  - Graftaki düğümler, hesaplama durumlarını ve ifadeleri temsil eder.
  - Graftaki kenarlar, kontrol akışını temsil eder.

Kontrol Akış Metrikleri



McCabe’nin Cyclomatic Karmaşıklığı

- 1976 yılında Thomas McCabe tarafından tanıtılmıştır.
- Kontrol akışını baz alan en popüler yaklaşımdır.
- Programın kontrol akışına dayanır.
- Bir modülün mantıksal karmaşıklığını ölçer
- Cyclomatic karmaşıklık, programdaki kontrol yapısını veya karar mantığını ölçer.
- Yapılan çalışmalar bu karmaşıklık ile kusurun ortaya çıkması arasında büyük bir korelasyonun olduğunu göstermiştir.
- Yazılım güvenirliği alanında popüler olmuş bir yaklaşımdır.

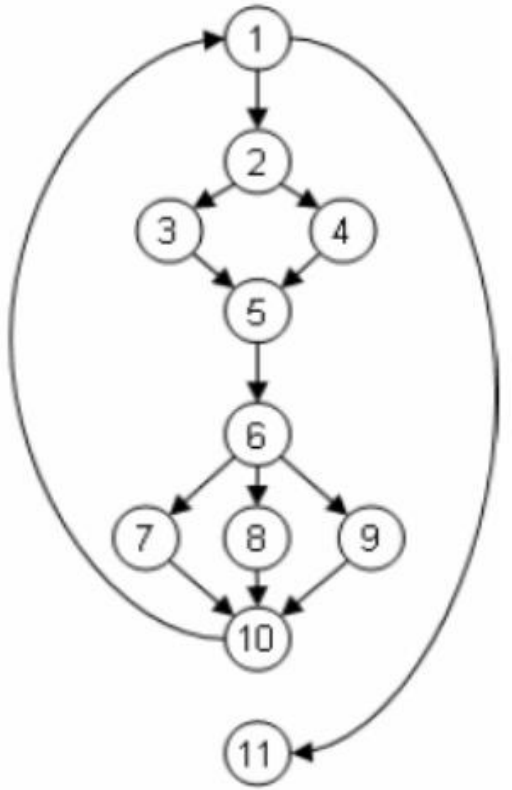
McCabe'nin Cyclomatic Karmaşıklığı

V = e – n + 2p

- e : Kenar Sayısı
- n : Düğüm Sayısı
- p : Bilişen Sayısı

V = 14 – 11 + 2x1 = 5

Node	Statement
(1)	`while(x<100){`
(2)	`  if (a[x] % 2 == 0) {`
(3)	`      parity =  0;`
	`   }`
	`   else {`
(4)	`      parity =  1;`
(5)	`   }`
(6)	`   switch(parity){`
	`      case 0:`
(7)	`        println( "a[" + i + "] is even");`
	`      case 1:`
(8)	`        println( "a[" + i + "] is odd");`
	`      default:`
(9)	`        println( "Unexpected error");`
	`    }`
(10)	`    x++;`
	`}`
(11)	`p = true;`



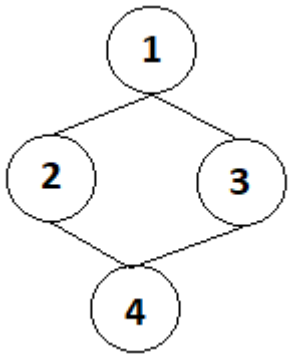
Myer'in Karmaşıklık Hesabı

- Tek koşul içeren basit akışların, McCabe'nin karmaşıklık hesabının ayırt edemeyeceğini belirtmiştir.
- Bu tip programlar aynı akış grafı ile temsil edilirken aslında farklı karmaşıklığa sahip olabilmektedirler.
- Örneğin aşağıdaki kod bloklarının her ikisin de McCabe'nin hesabına göre aynı karmaşıklığa sahip olduğu görülür.

$$V = 4 - 4 + 2 \times 1 = 2$$

```
IF x=O THEN sl
ELSE s2
```

```
IF x=O and y= I THEN sl
ELSE s2
```



Myer'in Karmaşıklık Hesabı

- Myer karmaşıklık hesaplarken bir değer yerine bir aralık olarak ölçmeyi önerir.

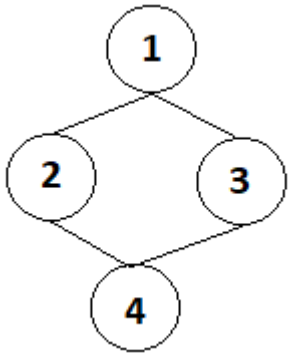
  [CC, CC+h]

- h: Tekil olarak koşul sayısı
  - Tek parametreli ise sıfır
  - n parametreli ise n-1 değerini alır.

- Burumda soldaki [2, 2], sağdaki ise [2, 3]

```
IF x=O THEN sl
ELSE s2
```

```
IF x=O and y= I THEN sl
ELSE s2
```



Veri Akış Metrikleri

- Modüller arasındaki bağlantıyı baz alır.
- Yüksek seviyeli tasarımlarda bu modül kullanılabilir.
  - Grafiksel Olarak İfade Etme
    - Modüller arasındaki veri akışını ifade eder.
    - Bu yaklaşımda tasarımın detaylarının bilinmesine gerek yoktur.
    - Birçok graf çeşidi kullanılabilir.
    - Call Graph (Yönlü çoklu graftır)
    - Düğümler: Modülleri, fonksiyonları, alt blokları temsil eder.
    - Kenarlar: Modüller arasındaki ilişkileri temsil eder.

İşlev Noktası (Function Point)

- FP'ler LOC benzeri metrikleri kullanmayıp, daha güvenilir bir kalite metriği oluşturma amacıyla geliştirilmişlerdir.
- Programlama dili veya kaynak kodu bağımsız değerlendirebilirler.
- Yazılım sürecinin ilk aşamalarında da hesaplanabilir.

UFP = Girdi Sayısı*w1 + Çıktı Sayısı*w2 + kullanıcı sorgu sayısı*w3 +

dosya sayısı*w4 + dış referans sayısı*w5

İşlev Noktası Ağırlıklar

	Basit	Ortalama	Kompleks
w1	3	4	6
w2	3	5	7
w3	3	4	6
w4	7	10	15
w5	5	7	10

DI: Etki Derecesi 0-5
TDI: Bütün etki dereceleri toplanır
VAF = TDIx0.01+0.65
FP= UFPxVAF

Örnek

Parametre	Adet
Girdi	24 (Orta)
Çıktı	16 (Orta)
Sorgu	22 (Kompleks)
Dosya	4 (Orta)
Referans	2 (Basit)

UFP = 24x4+16x5+22x6+4x10+2x5=358
TDI=52 olsun
VAF=1.17
FP=358x1.17
FP=418.86

Referanslar

- Madi, A., Zein, O. K., & Kadry, S. (2013). On the improvement of cyclomatic complexity metric. *International Journal of Software Engineering and Its Applications*, *7*(2), 67-82.
- Rawat, M. S., & Dubey, S. K. (2012). Software defect prediction models for quality improvement: a literature study. *International Journal of Computer Science Issues (IJCSI)*, *9*(5), 288.
- Nam, J. (2014). Survey on software defect prediction. *Department of Compter Science and Engineerning, The Hong Kong University of Science and Technology, Tech. Rep*.
- A. Abran and P. N. Robillard, Function points: a study of their measurement processes and scale transformations, Journal of Systems and Software, Vol. 25, No. 2, 1994, pp. 171-184
- Full Function Points: Counting Practices Manual, Edited by Software Engineering Laboratory Management Research Laboratory and …, Sep. 1997
- T. Fetcke, A Generalized Structure for Function Point Analysis, In International Workshop on Software Measurement, Lac Supérieur, Québec, Canada, Sep. 1999