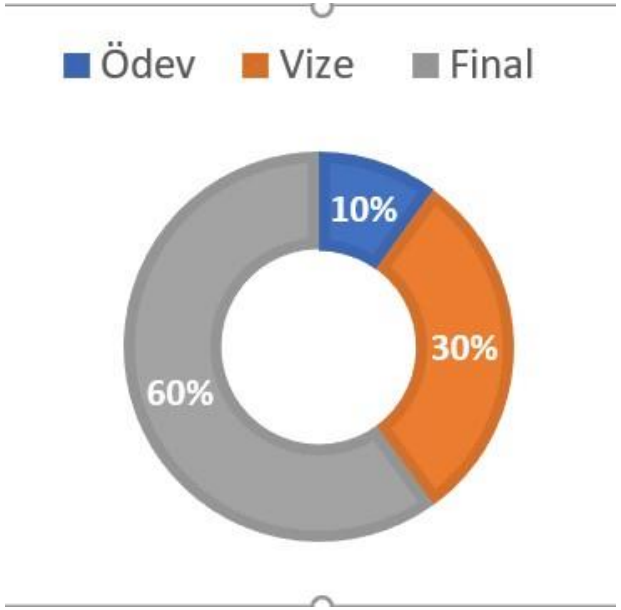
Yazılım Mühendisliği
1906003082015

Dr. Öğr. Üy. Önder EYECİOĞLU
Bilgisayar Mühendisliği

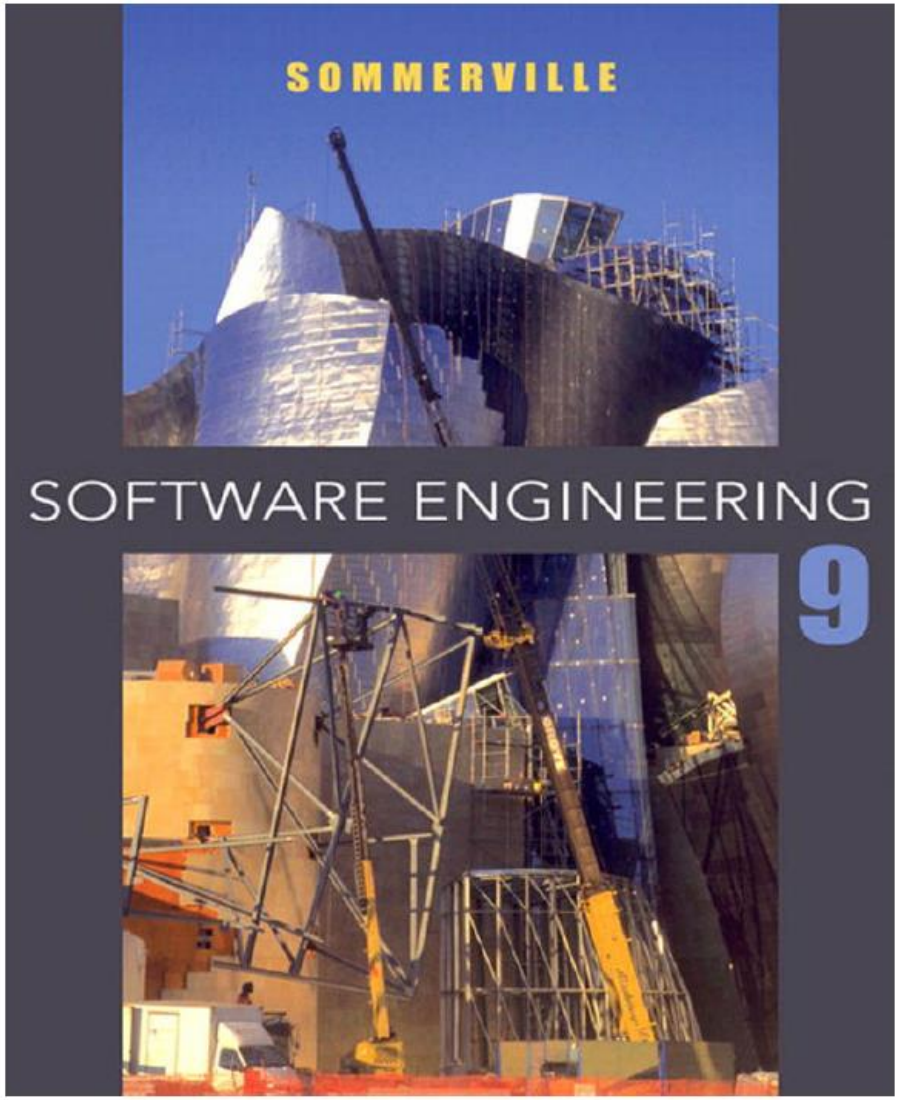
Giriş

Ders Günü ve Saati:
Salı: 09:15-13:00
Devam zorunluluğu %70



HAFTA	KONULAR
Hafta 1	Yazılım Mühendisliğine Giriş
Hafta 2	Yazılım Geliştirme Süreç Modelleri
Hafta 3	Yazılım Gereksinim Mühendisliği
Hafta 4	Yazılım Mimarisi
Hafta 5	Nesneye Yönelik Analiz ve Tasarım
Hafta 6	Laboratuar Çalışması: UML Modelleme Araçları
Hafta 7	Yazılım Test Teknikleri
Hafta 8	Ara Sınav
Hafta 9	Yazılım Kalite Yönetimi
Hafta 10	Yazılım Bakımı - Yeniden Kullanımı ve Konfigürasyon Yönetimi
Hafta 11	Yazılım Proje Yönetimi (Yazılım Ölçümü ve Yazılım Proje Maliyet Tahmin Yöntemleri)
Hafta 12	Yazılım Proje Yönetimi (Yazılım Risk Yönetimi)
Hafta 13	Çevik Yazılım Geliştirme Süreç Modelleri
Hafta 14	Yazılım Süreci İyileştirme, Yeterlilik Modeli (CMM)

Kaynaklar



9.
KALİTE ÖLÇÜM VE DEĞERLENDİRMELERİ

Giriş

BÖLÜM HEDEFLERİ

- Yazılım kalite ölçütlerinin incelenmesi
- Yazılım ölçütlerinin test adımlarında kullanımının anlaşılması
- Kalite Güvenirliğinin kestirimini ve modellerini tanima

9. HAFTA İÇERİĞİ

- Yazılım Ölçümü
- Yazılım Kalite Ölçütleri (Metrikleri)
- Yazılım Kalite ölçüm Yöntemleri

Yazılım Ölçümü

- «Konuştuğun şeyi ölçebiliyor ve rakamlarla ifade edebiliyorsan, o şey hakkında bir şeyler biliyorsun demektir. Aksi takdirde, bildiğin şeyler değersiz ve tatmin etmeyen bilgi kalabalığıdır». Lord Kelvin
- «Ölçmediğin şeyi kontrol edemezsin» Tom De Marco
- Ölçme ? (Yönetici, Tasrımcı, Mühendis)

Yazılım Ölçümü

Yöneticiler:

- Yazılım üretimi için maliyetin ne olacağına
- Farklı bölüm çalışanlarının ücretlerini belirlemek için personel üretkenliğini ölçmeye
- Farklı projelerde geliştirilen yazılım ürünlerini kıyaslayabilmek için yazılım kalitesini ölçmeye
- Projeler için ölçülebilir hedefler belirlenmesi örneğin, yazılım sisteminin nasıl güvenilir olması.
- Şirkette kullanımına başlamak üzere, yöntem veya yazılım geliştirme araçlarının etkinliğinin ölçülmesine

gereksinim duymaktadırlar.

Yazılım Ölçümü

Yazılım Mühendislerinin:

- Süreç ölçümleri ile sistemin kalitesini izlemeye örneğin, tasarım süresince yapılan değişiklikler veya farklı gözden geçirme ya da test adımlarında bulunan hatalar,
- Kesin ölçülebilir terimlerle kalite ve performans gereksinimlerini belirlemeye
- Sertifika için ürün ve süreç özelliklerini ölçmeye. (Örneğin modüllerin 100 kod satırdan fazla olmamasına)
- Var olan ürünler ve uygulanan süreçlerden faydalanarak gelecekteki ürün geliştirmeler için tahmin yapabilmeye

gereksinimleri vardır.

Yazılım Ölçümü

Ölçüm:

- Gelişmenin tek yolu!
- CMM(Capability Maturity Model) için gerekli
- Yazılım Sürecinin Gelişmesi için gerekli
- Yazılım Gelişimi Boyunca Ölçüm;
  - Verimliliği Belirler.
  - Kaliteyi Değerlendirir.
  - Tahmin Yaptırır.
  - Risk Analizi Yaptırır.

Yazılım Kalitesi

- Yazılım endüstrisinin karakteristik özelliği, hızla ve sürekli olarak gelişmesi ve değişmesinin yanı sıra, dünya ölçeğinde pek çok başarısız uyarlamaya sahne olmasıdır.
- Yazılımın beklenen kalite düzeyinde olmamasının ve dolayısıyla da müşteri beklentilerini kesin ve net bir şekilde karşılayamamasının neden olduğu maliyetler, ürüne ve ürünü üreten şirketin başarısına yönelik ciddi tehditleri de beraberinde getirmektedir.
- Bu durum daha kaliteli yazılım ürünlerine olan gereksinimi artırmaktadır.

Yazılım Kalitesi

- Amerikada yazılım projelerinin;
- %47'si kullanılmamakta
- %29'u müşteri tarafından kabul edilmemekte
- %19'u başladıktan sonra iptal edilmekte ya da büyük ölçüde değiştirilerek yeniden başlatılmakta
- %3'ü kimi değişikliklerden sonra kullanılmakta
- %2'si ancak teslim alındığı gibi kullanılmakta

olduğu görülmüştür.

- Aynı zamanda 2002 yılında NIST tarafından yapılan bir araştırma sonucunda yazılım hatalarının Amerikan ekonomisine yıllık maliyeti 59 Milyar $ olarak tahmin edilmiştir

Yazılım Kalite Metrikleri

Metrikler yazılımın ya da işlemlerin verilen özniteliklerin hangilerine sahip olduklarını gösteren nümerik ölçülerdir. Metriklere örnek olarak; “her bin satır koddaki hata sayısı”, “ortalama yazılım modülü boyutu”, verilebilir. Metrikler yazılım yaşam süreci boyunca toplanır ve analiz edilirler ve bize aşağıda verilen konularda yardımcı olurlar;

- Yazılım kalite seviyesinin belirlenmesi
- Proje zamanlama kestirimi(tahmin)
- Zaman ilerlemesinin izlenmesi
- Yazılım boyutu ve karmaşıklığının hesaplanması
- Proje maliyetinin belirlenmesi
- Süreçlerin iyileştirilmesi

Yazılım Kalite Metrikleri

Büyüklük (Size)	LOC (SLOC,CLOC,BLoC, C&SLOC), Volume
Karmaşıklık (Complexity)	Çevrimsel Karmaşıkık (Cyclomatic Complexity), Mantıksal Yoğunluk (Logical Density)
Yapısallık (Structure)	Esas Karmaşıklık (Essential Complexity)
Modülerlik (Modularity)	Entegrasyon Karmaşıklığı(Integration Complexity)
Nesneye Yönelik (Object Oriented)	Bağımlı Sınıf Sayısı (Coupling Between Objects), Sınıf İçindeki Metod Sayısı (Weighted Methods/Class), Türetilmiş Alt Sınıf Sayısı (Number of Children)...

Yazılım Kalite Metrikleri

Bakım Yapılabilirlik (Maintainability)	3MI, 4MI
Gereksiz Kod (Redundant Code)	Ölü Kod/Veri Sayısı (Dead code/Data counts)
İşlev Noktaları (Function Points)	İdeal olarak orjinal dökümantasyondan üretilir ya da son çare olarak geri gelme yöntemiyle kullanılabilir.
Gramer (Grammar)	GOTO, ALTER, I/O, IF, EVALUATE, max, IF/EVALUATE'de bulunan koşulları sayıları
Kod Tekrarları (Duplicate Code)	Kopyala/Yapıştır ile üretile kod satır sayıları

Yazılım Kalite Metrikleri

	Metrik Adı	Yılı
McCabe	Cyclomatic Complexity	1976
Myers	Fan-In & Fan-Out	1978
Albrecht	Function Points	1979
Halstead	Software Science	1977
Yourdon & Constantine	Fan-In & Fan-Out	1979
Basili & Perricone	LOC	1984
Withrow	LOC	1990
Capers-Jones	Function Points & LOC	1992
Lorentz	OO – 11 Metrics	1993
Chidamber & Kemerer	OO – 6 Metrics	1994
Robert Martin	OO – 5 Metrics	1998

Yazılım Kalite Metrikleri

Metric Type	Metric	Definiton
McCabe	v(G) ev(G) iv(G) LOC	Cyclomatic Complexity Essential Complexity Design Complexity Lines of Code
Halstead	N V L D I E B T	Length Volume Level Difficulty Intelligent Content Effort Error Estimate Programming Time
Line Count	LOCode LOComment LOBlank LOCodeAndComment	Lines of Code Lines of Comment Lines of Blank Lines of Code and Comment
Operator/Operand	UniqOp UniqOpnd TotalOp TotalOpnd	Unique Operators Unique Operands Total Operators Total Operands
Branch	BranchCount	Total Branch Count

SATIR SAYISI-LOC

- Toplam satır sayısı
- Boş olan / olmayan (blank / non-blank)
- SLOC (source LOC) : Comment ve boş olanların haricindeki
- LLOC (logical LOC) : Program ifadesi sayısı
- comment
- mixed (code + comment)

SATIR SAYISI-LOC

C	COBOL

`#include <stdio.h>`
 	
`int main(void) {`
 	
`printf("Hello World");`
`return 0;`
`}`
	`000100 IDENTIFICATION DIVISION.`
	`000200 PROGRAM-ID. HELLOWORLD.`
	`000300`
	`000400*`
	`000500 ENVIRONMENT DIVISION.`
	`000600 CONFIGURATION SECTION.`
	`000700 SOURCE-COMPUTER. RM-COBOL.`
	`000800 OBJECT-COMPUTER. RM-COBOL.`
	`000900`
	`001000 DATA DIVISION.`
	`001100 FILE SECTION.`
	`001200`
	`100000 PROCEDURE DIVISION.`
	`100100`
	`100200 MAIN-LOGIC SECTION.`
	`100300 BEGIN.`
	`100400 DISPLAY " " LINE 1 POSITION 1 ERASE EOS.`
	`100500 DISPLAY "Hello world!" LINE 15 POSITION`
	`10.`
	`100600 STOP RUN.`
	`100700 MAIN-LOGIC-EXIT.`
	`100800 EXIT.`

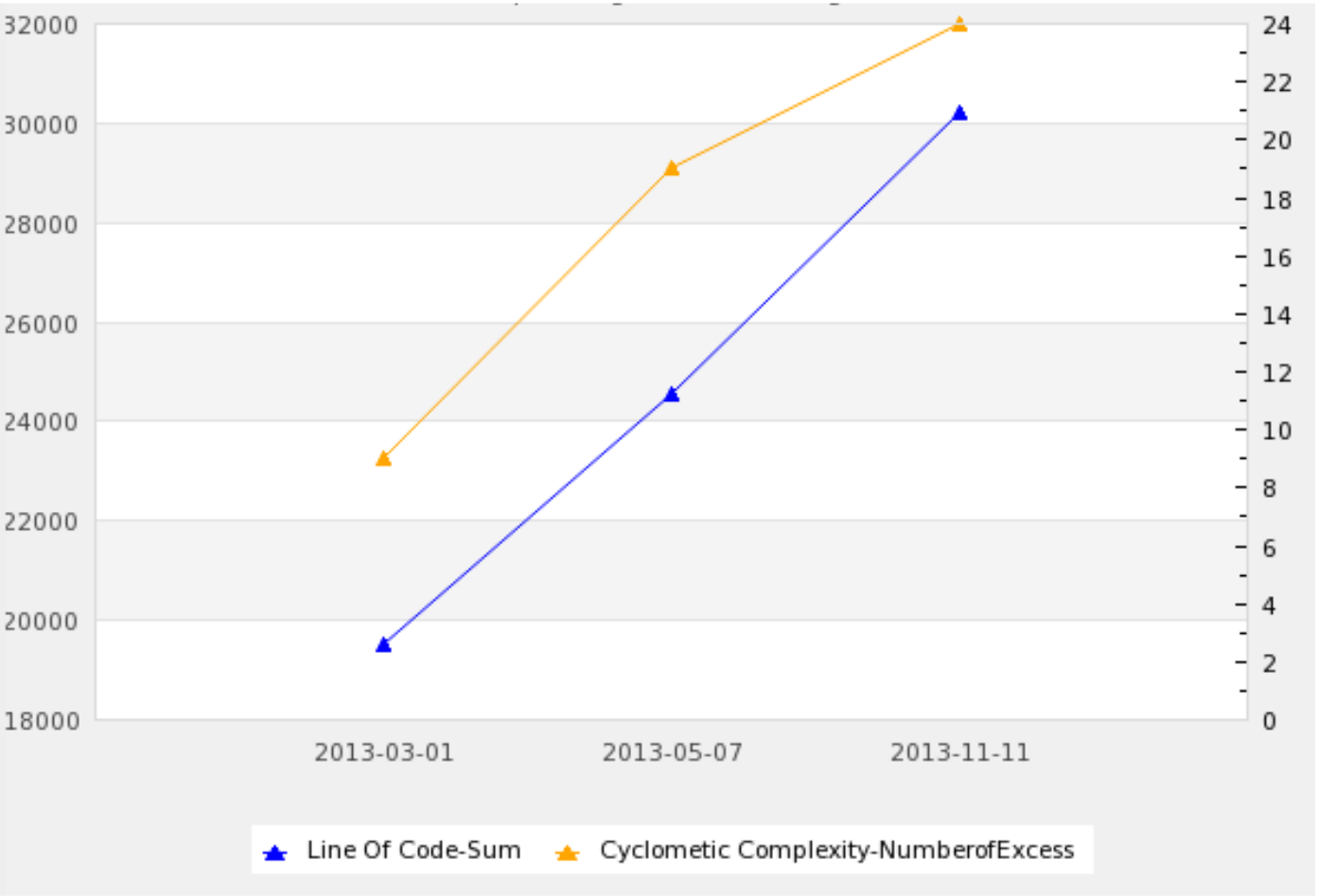
Lines of code: 5 (boşluklar hariç)	Lines of code: 17 (boşluklar hariç)

SATIR SAYISI-LOC-Avantaj

- Hesaplaması kolay
- LLOC'un hesaplanması için parser gerekli
- Tek satırda içiçe geçmiş ifadeler kullanılmışsa anlamsız
- Yazılım büyüklük ve geliştirme ihtiyacı kestirimini hesaplamada kullanılıyor. (COCOMO)
- Bakım yapılabilirlik metriğinin (MI) hesaplanmasında kullanılıyor.

SATIR SAYISI-LOC-Avantaj

- Hata sayısı ile orantılı
- Büyük modüller genellikle daha çok hata içerir.
- Hata sayısı kestirimi için daha iyi ama daha
- karmaşık yöntemler var.



SATIR SAYISI-LOC-Dezavantaj

- Programın yapısını ihmal eder.
- Program kodu metinden daha fazlasıdır!
- Farklı geliştiriciler tarafından farklı programlama dillerinde yazılmış modülleri karşılaştırmak zordur.
- Bazı diller gereğinden uzun kod yazmayı gerektirir.
- Built-in fonksiyonalitenin olması ya da olmaması (örneğin Arayüz geliştirme)
- Yapısal uzunluk (örneğin C'deki h dosyaları)

SATIR SAYISI-LOC-ÖNERİLER

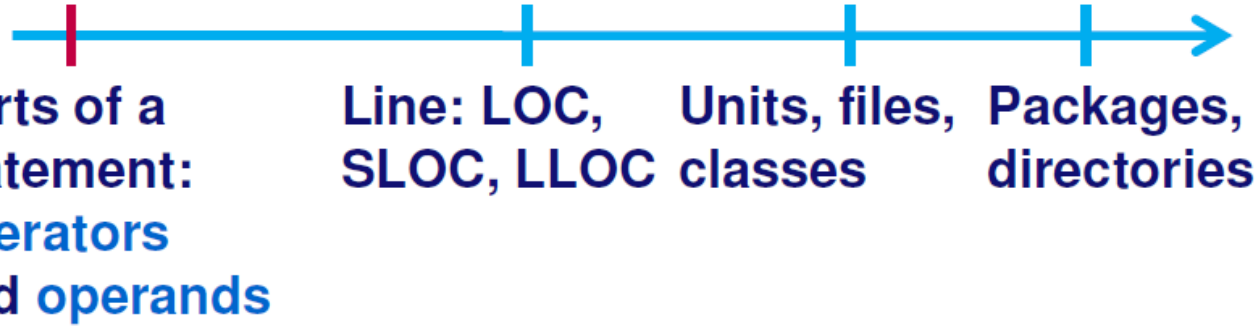
- Bir dosyanın en az %30'u, en fazla %75'i yorum satırı olmalıdır.
- Eğer dosyanın üçte birinden daha az yorum satırı varsa, yetersizdir ya da zayıf ifade edilmiştir.
- Eğer %75'ten fazla yorum satırı varsa o artık bir program değil, dokümandır.
- İstisna : Bazen bir iyi dokümante edilmiş bir header dosyasında yorum satırı oranı %75'i geçebilir.

Halstead Metrikleri

- Maurice Halstead tarafından 1977'de geliştirildi ve o zamandan beri etkin şekilde kullanıldı ve test edildi.
- Kod karmaşıklığı için sağlam bir gösterge
- Sıklıkla bakım yapılabilirlik metriği olarak kullanılır.
- OO yazılımlar için değil, prosedürel yazılımlar (COBOL, Fortran) için uygundur.

Halstead Metrikleri

- Kaynak kodu bir dizi işaretçi olarak ve her işaretçiyi de operatör ya da operand olarak sınıflandırarak yapılan bir yorumlamaya dayalı metriklerdir.
- Bir metod içersindeki operatör ve operand sayılarının 30'u geçmemesi beklenmektedir



- **Operators:**
  - **traditional (+,++, >), keywords (return, if, continue)**
- **Operands**
  - **identifiers, constants**

Halstead Metrikleri

- Halstead 4 temel metriği

	Toplam	Tekil
Operatörler	N1	n1
Operandlar	N2	n2

- benzersiz (farklı) operatörlerin sayısı (n1)
- benzersiz (farklı) işlenenlerin sayısı (n2)
- toplam operatör sayısı (N1)
- toplam işlenen sayısı (N2)

Halstead Metrikleri

- Halstead 4 temel metriği

	Toplam	Tekil
Operatörler	N1	n1
Operandlar	N2	n2

- Program uzunluğu: N=N1+N2
- Cümle uzunluğu: n=n1+n2
- Program Hacmi: programın bilgi içeriği

V=N*log2(n)

Halstead'in hacmi (V) bir algoritmanın uygulanmasının boyutunu açıklar

Halstead Metrikleri

- Bir fonksiyonun hacmi, 20 den büyük 1000 den küçük olmalıdır.
- Bir dosyanın hacminin en az 100 ve en fazla 8000 olması gerekir.

Difficulty level (D)

Programın zorluk seviyesi veya hata olasılığı (D), programdaki tekil operatörlerin sayısıyla orantılıdır. D ayrıca, toplam işlenen sayısı ile tekil işlenenlerin sayısı arasındaki orantı ile orantılıdır. (yani, eğer aynı işlenenler programda birçok kez kullanılıyorsa, hatalara daha yatkındır)

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

Halstead Metrikleri

Program level (L)

Program seviyesi (L), programın hata ihtimalinin tersidir. Yani Düşük seviye bir program, yüksek seviyeli bir programdan ziyade hatalara daha yatkındır.

$$L = 1 / D$$

Effort to implement (E)

Bir programı (E) gerçekleştirme veya bir programı anlama çabası, hacim ve programın zorluk seviyesi ile orantılıdır.

$$E = V * D$$

Halstead Metrikleri

Time to implement (T)

Bir programı (T) uygulama veya anlamanın zamanı çaba ile orantılıdır. Halstead, bu çabaları 18'e bölerek, saniyeler içinde zamanın bir yaklaşımı olduğunu buldu..

$$T = E / 18$$

Teslim edilen hataların sayısı (B)

Teslim edilen hataların sayısı (B), yazılımın genel karmaşıklığı ile ilişkilidir.

$$B=(E^2/3)/3000$$

Bir dosyada teslim edilen hatalar 2'den küçük olmalıdır. Deneyimler, C veya C ++ ile programlama yaparken, bir kaynak dosyada neredeyse her zaman B önerilerinden daha fazla hata içerdiğini göstermiştir. B, dinamik test için önemli bir metriktir: Teslim edilen hataların sayısı, bir modüldeki hataların sayısını tahmin eder. Hedef olarak en azından testte modülden çok sayıda hata bulunmalıdır.

Halstead Metrikleri

Metric	Meaning	Formula
n	Vocabulary	$n_1 + n_2$
N	Size	$N_1 + N_2$
V	Volume	$N * \log2\ n$
D	Difficulty	$n_1/2 * N_2/n_2$
E	Effort	$V * D$
B	Errors	$V / 3000$
T	Testing time	E / k

Halstead Örnek

```
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;
               }
         }
   }
}
```

- Ignore the function definition
- Count operators and operands

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

	Total	Unique
Operators	N1 = 50	n1 = 17
Operands	N2 = 30	n2 = 7

$V = 80 \log_2(24) \approx 392$

Inside the boundaries [20;1000]

Halstead Örnek

i = i+1	Toplam	Tekil
Operatörler	N1 = 2 **(=, +)**	n1 = 2 **(=, +)**
Operandlar	N2 = 3 **(i, i, 1)**	n2 = 2 **(i, 1)**
i++	**Toplam**	**Tekil**
Operatörler	N1 = 1 **(++)**	n1 = 1 **(++)**
Operandlar	N2 = 1 **(i)**	n2 = 1 **(i)**

Metrik	i=i+1	i++
Volume V (NLog2n)	V =12	V =2
Difficulty D (n1/2)*(N2/n2)	D = 1.5	D = 0.5
Effort E (V*D)	E = 18	E = 1

SONUÇ : “i=i+1” yerine “i++” kullanıldığında efor 18 kat azalıyor.

McCabe Cyclomatic SAYILAR-Çevrimsel Karmaşıklık

- 1976'da McCabe tarafından önerildi.
- Bir program modülünde doğrusal olarak bağımsız yolların(Kontrol akışı) sayısını ölçer.
- En yaygın ve kabul edilen yazılım metriklerinden birisidir, programlama dili ve yazım şeklinden bağımsızdır.
- Bir programın doğruluğu ve güvenirliği için en kapsamlı ölçütdür.
- Bir metod ya da sınıfta icra edilmesi gereken test sayısı hakkında fikir verir. (Test sayısı >= ÇK)

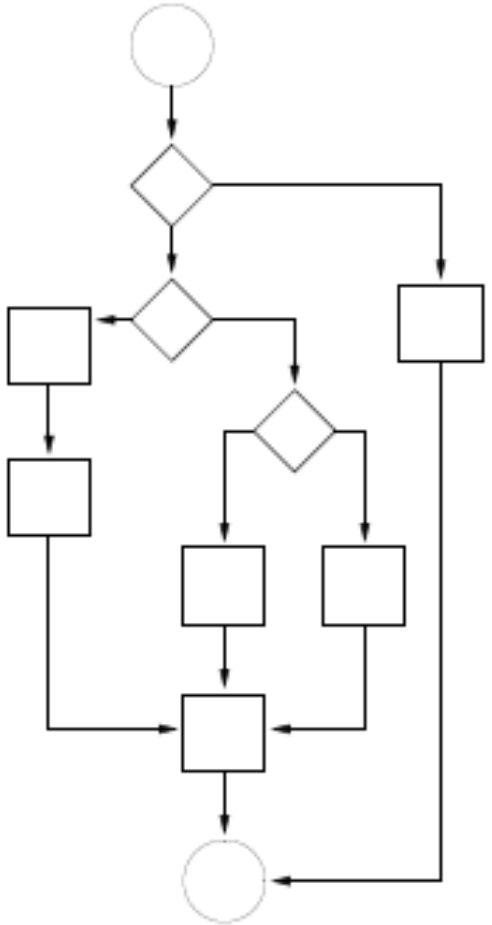
McCabe Cyclomatic SAYILAR-Çevrimsel Karmaşıklık

- v(G) = #edges - #vertices + 2

For control flow graphs

- v(G) = #binaryDecisions + 1, or
- v(G) = #IFs + #LOOPs + 1

Boundaries
- v(function) ≤ 15
- v(file) ≤ 100

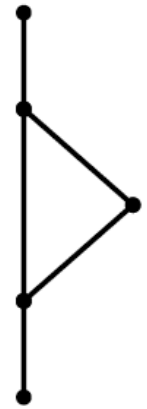


McCabe Cyclomatic SAYILAR-Çevrimsel Karmaşıklık

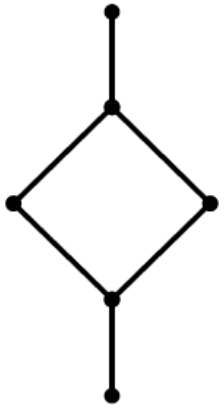
```
void sort ( int *a, int n ) {
int i, j, t;

   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;
               }
         }
   }
}
```

- Count IFs and LOOPs
- IF: 2, LOOP: 2
- v(G) = 5
- Structural complexity

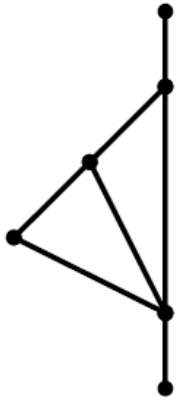
McCabe Cyclomatic SAYILAR-Çevrimsel Karmaşıklık



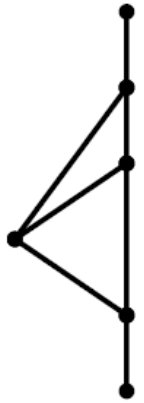
if (i) ;



if (i) ; else ;

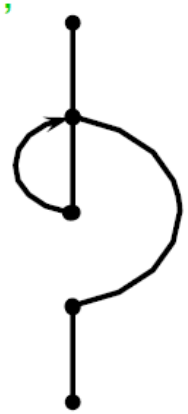


if (i && j) ;

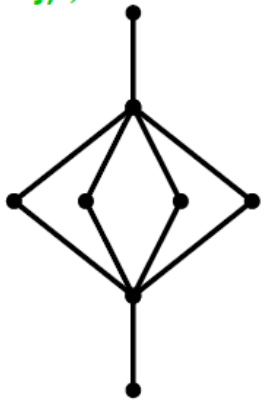


if (i || j) ;

do ; while (i);

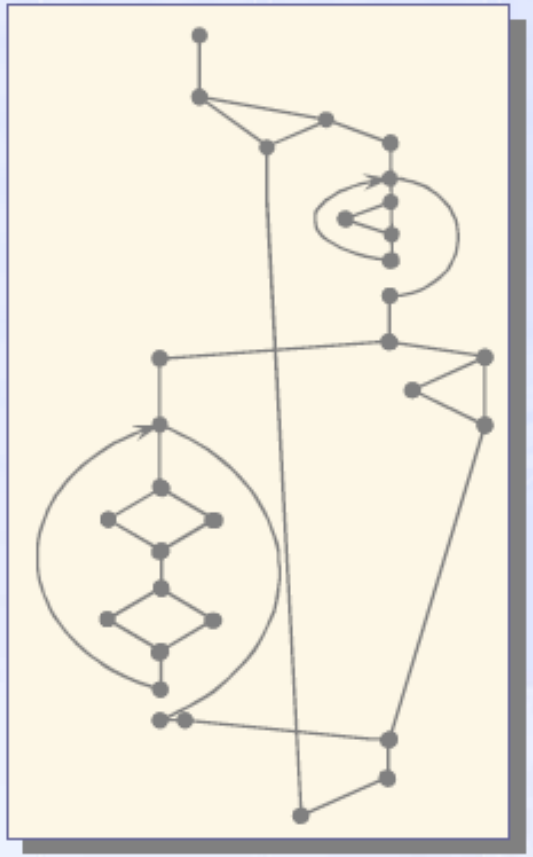


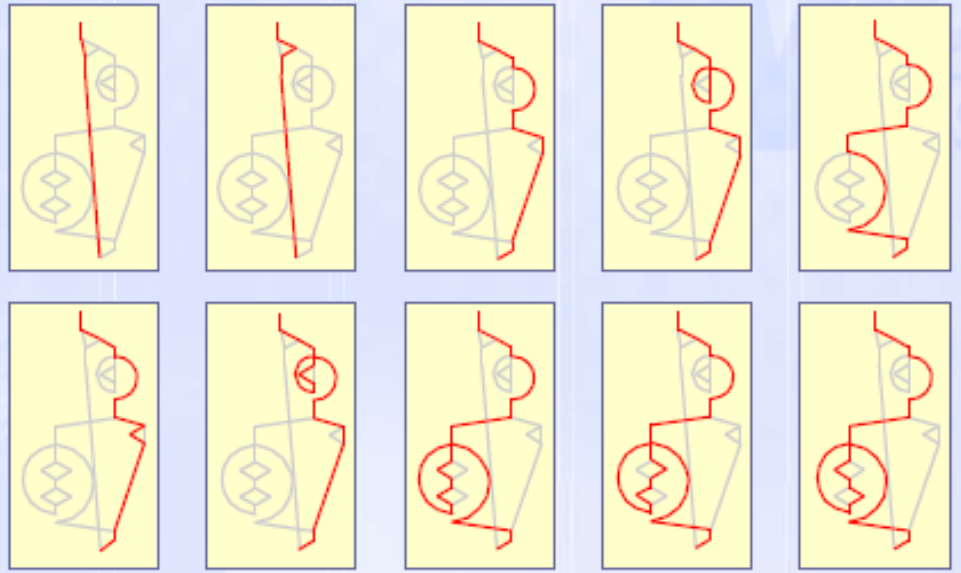
while (i) ;



switch(i) { case 0: break; ... }

McCabe Cyclomatic SAYILAR-Çevrimsel Karmaşıklık



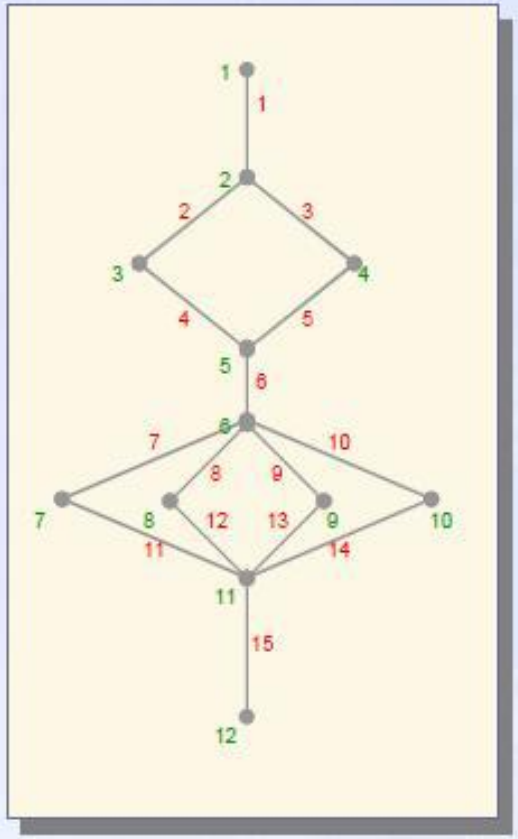


- Cyclomatic Complexity = 10

  Means that 10 Minimum Tests will:

  - Cover All the Code
  - Test Decision Logic

McCabe Cyclomatic SAYILAR-Çevrimsel Karmaşıklık



Count edges and nodes

Use the formula:

v(G) = e - n + 2

Example:

v(G) = 15 - 12 + 2

v(G) = 5

McCabe Cyclomatic SAYILAR-Çevrimsel Karmaşıklık

- v(G), koşul dallarının (conditional branches) sayısıdır.
- Eğer sadece sıralı ifadeler içeriyorsa o programın v(G) değeri 1'dir.
- Bir fonksiyon için, v(G) koşul dalları nokta sayısının bir eksiğidir.
- Çevrimsel karmaşıklığın artması fonksiyondaki işlem yollarının artmasına ve programın daha zor anlaşılmasına neden olur.

McCabe Cyclomatic SAYILAR-Çevrimsel Karmaşıklık

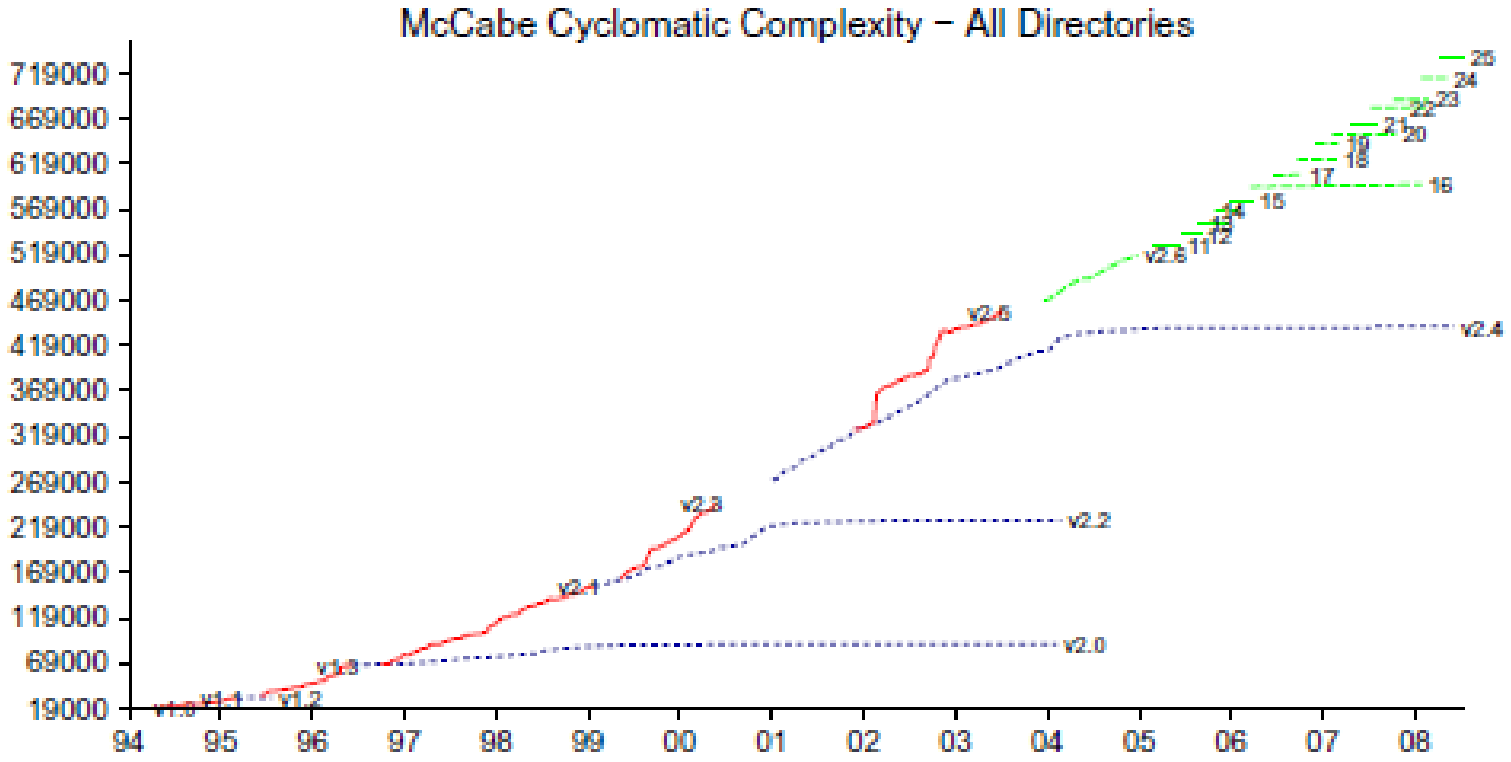
Çevrimsel Karmaşıklığı Artıran Faktörler

- İf ifadesi (programa yeni bir dal ekler)
- for ve while döngüleri gibi Çevrim yapıları
- Bir switch bloğundaki her case parçası
- Bir try bloğundaki her catch(...) parçası
- Exprl?expr2:expr3 yapısı
- Önişlemciler (#if, #ifdef, #ifndef, #elif)

❑ Çevrimsel Karmaşıklık ve Risk (Cyclomatic Complexity)

• 1 – 10	Basit işlem, düşük risk
• 11- 20	Daha karışık, orta risk
• 21 – 50	Karışık, yüksek
• >50	Test edilemez, ÇOK YÜKSEK RİSK

McCabe Cyclomatic SAYILAR-Çevrimsel Karmaşıklık



Linux kernel MCCabe karmaşıklığı

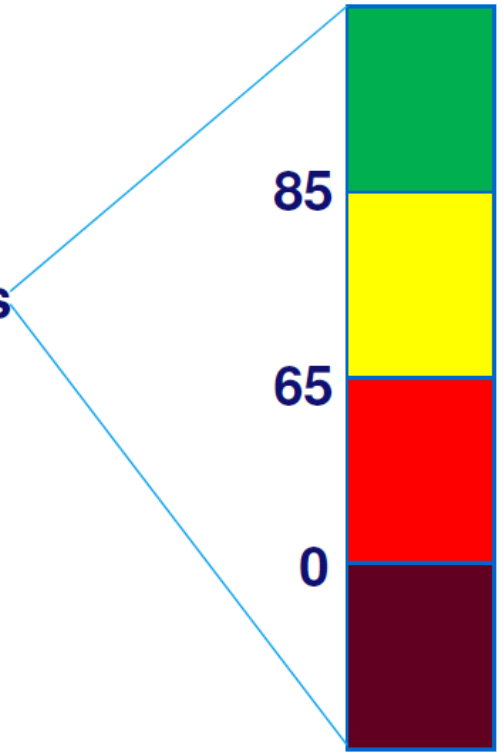
Bakım Yapılabilirlik İndeksi (MI)

$$MI_1 = 171 - 5.2\ln(V) - 0.23V(g) - 16.2\ln(LOC)$$

Halstead McCabe LOC

$$MI_2 = MI_1 + 50\sin\sqrt{2.46\,perCM}$$

% comments



- MI_2 can be used only if comments are meaningful
- If more than one module is considered – use average values for each one of the parameters
- Parameters were estimated by fitting to expert evaluation
  - BUT: few middle-sized systems!

Bakım Yapılabilirlik İndeksi (MI)

```
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;
               }
         }
   }
}
```

- Halstead's V ≈ 392
- McCabe's v(G) = 5
- LOC = 14
- $MI_1 \approx 96$
- Easy to maintain!

NESNEYE YÖNELİMLİ METRİKLER

- Chidamber & Kemerer OO Metrikleri
- McCabe Sınıf karmaşıklık metrikleri

NESNEYE YÖNELİMLİ METRİKLER

- Chidamber & Kemerer OO Metrikleri

Chidamber & Kemerer OO Metrikleri

Nesne yönelimli tasarım için kullanılan bu metrik grubu, bir sistemin bütün olarak değerlendirilmesinden ziyade sistemdeki sınıfları geniş bir şekilde değerlendirmeyi amaçlar. Chidamber & Kemerer, nesne yönelimli tasarım için altı adet metrik tanımlar.

1. Sınıfın ağırlıklı metot sayısı (Weighted Methods per Class - WMC)
Sınıfın ağırlıklı metot sayısı (WMC); bir sınıftaki metotların karmaşıklık derecesi veya sayısıdır. Bir sınıfın metotlarının karmaşıklığı ve sayısı, sınıfın geliştirilmesine ve bakımına harcanacak zaman-çaba hakkında fikir verir (9). Bu metrik, bir sistemdeki sınıfların ortalaması hesaplanarak ölçülür (10). WMC; nesne yönelimli bir yazılımın anlaşılırlığını, yeniden kullanılabilirliğini ve dayanıklılığını/bakılabilirliğini ölçmede kullanılır.

2. Kalıtım ağacının derinliği (Depth of Inheritance Tree - DIT)
Sınıfın kalıtım ağacının köküne uzaklığıdır (9). Bu değerin çok yüksek olması test edilebilirliğin çok düşük olduğunu, aksi halde nesne yönelim ilkelerinin fazla kullanılmadığını gösterir (12). Bu metrik; verimliliği, yeniden kullanımı, anlaşılırlığı ve test edilebilirliği ölçer.

Chidamber & Kemerer OO Metrikleri

3. Alt sınıf sayısı (Number of Children - NOC)
Bir sınıftan direk türetilmiş alt sınıfların sayısıdır. NOC, kalıtımlı ifadeler dikkate alınarak hesaplanır. Bir sınıf hiyerarşisinin genişliğini ölçer. Alt sınıf sayısının fazla olması; yeniden kullanımın yüksek olduğunu, daha çok hatanın oluşabileceğini (13), test esnasında harcanacak zamanı-çabayı ve kalıtımın yanlış kullanıldığını gösterir (9). NOC; verimlilik, yeniden kullanılabilirlik ve test edilebilirlik düzeyini ölçer.

4. Nesne sınıfları arasındaki bağımlılık (Coupling Between Object Classes - CBO)
Bir sınıf içindeki özellik (attribute) ya da metotların (method) diğer sınıfta kullanılması ve sınıflar arasında kalıtımın olmaması durumunda iki sınıf arasında bağımlılıktan bahsedilebilir (13). CBO; verimliliği ve yeniden kullanılabilirliği ölçmede kullanılır.

Chidamber & Kemerer OO Metrikleri

5. Sınıfın tetiklediği metot sayısı (Response for a class - RFC)
Bir sınıftan bir nesnenin metotları çağrılması durumunda, bu nesnenin tetikleyebileceği tüm metotların sayısı RFC değerini verir. Yani, bir sınıfta yazılan ve çağırılan toplam metot sayısıdır (9). Bu metrik; sınıf seviye tasarım metriklerinden olup (14); anlaşılabilirliği, dayanıklılığı, karmaşıklığı ve test edilebilirliği ölçmede kullanılır (11).

6. Metotlardaki uyum eksikliği (Lack of cohesion in methods - LCOM)
LCOM, n adet kümenin kesişiminden oluşan kümelerdeki uyumsuzlukların sayısıdır ve metotlardaki benzerlik derecesini ölçer. Metotlardaki uyum eksikliği; bir sınıfın, iki veya daha fazla alt sınıfa ayrıldığını gösterir ve karmaşıklığı arttırır. (9,15). Yapılan bir çalışmada, LCOM ölçütünün uyum özelliğini çok da iyi ayırt edemediği ispatlanmıştır (3). Literatürde LCOM2, LCOM3 (13) ve LCOM4 (9,16) adlarıyla yer alan farklı LCOM metrik tanımları da mevcuttur. LCOM metriği test ediciye; verimlilik ve yeniden kullanılabilirlik derecesi hakkında bilgi verir.