

Yazılım Metrikleri (Ölçüler) (Software Metrics)

- **Yazılım metrikleri, yazılımların ölçülebilen alt düzeydeki niteliklerini ifade ederler.**
Örneğin; bir metottaki satır sayısı, bir sınıfın türetim zincirindeki derinliği, bir programın çalışması sırasında çıkan hata sayısı, bir modül üzerinde kaç kişinin çalıştığı gibi.
- Yıllar içinde "metrik" sözcüğünün anlamı genişlemiştir.
 - Örneğin; ölçülen nitelik, bu niteliğin ölçülme yöntemi ve ölçüm sonucu atanın sayı gibi. Yayınlarında bu kavramların hepsine metrik adı verilebilmektedir.
 - Bu durum anlam karmaşasına neden olduğundan ISO/IEC 25020 standardında metrik sözcüğü kullanılmamaktadır.
 - Bunun yerine; **ölçülen nitelik** (*property to quantify*), **ölçme yöntemi** (*measurement method*), **kalite ölçüm elemanı** (*quality measure element - QME*), **yazılım kalitesi ölçüsü** (*software quality measure - QM*) terimleri kullanılmaktadır (Bkz. 2.37).
 - Ancak makale ve bildirilerde metrik sözcüğü özellikle nitelik ve QME yerine hala kullanılmaktadır.
- Bu derste yazılım ürünlerinden **tasarım (kod) kalitesini** ölçmek ve değerlendirmek için kullanılan metrikler ele alınacaktır.

Metrik Türleri

Yazılımların kalitesini etkilediği düşünülen veriler (metrikler), bir yazılım projesinin farklı bileşenlerinden (kaynaklardan) elde edilebilirler.

- **Tasarım (kod) metrikleri:**
 - Doğrudan programın kodundan veya kodlama öncesindeki tasarımından elde edilirler.
 - İç niteliklerle veya dış niteliklerle ilgili olabilirler.
Örneğin; bir sınıfındaki metod sayısı, bir metottan çağırılan diğer metodların sayısı (derleme zamanı veya çalışma zamanı ölçülebilir).
- **Tarihsel (historical) metrikler:**
 - Yazılım projesinin geçmişi ile ilgili bilgi verirler.
 - Süreç metrikleri (*process metrics*) veya evrimsel metrikler (*evolution metrics*) olarak da adlandırılırlar.
 - Örneğin; şimdije kadar bir sınıfta belirlenen hata sayısı, bir sınıfın projeye eklenmesinden bu yana geçen süre (sınıfın yaşı), projede kaç sürüm oluşturulduğu, şimdije kadar bir sınıfa kaç satır eklendiği (veya silindiği) gibi.
- **Proje ekibi ile ilgili metrikleri (Developer metrics):**
 - Proje ekibinde çalışanların kişisel nitelikleri ile ilgili verilerdir.
 - Örneğin; bir programcının yaşı, meslekteki kaçinci yılı, mezun olduğu okul gibi.

Metrik Türleri (devamı)

- **Etkileşim (micro interaction) metrikleri:**
 - Proje ekibinin yazılım ile etkileşiminden elde edilirler.
 - Örneğin; bir sınıfın kodlanması için ortalama ne kadar süre ayrılıyor, iki kodlama arasında ne kadar ara veriliyor, bir sınıf üzerinde kaç programcı çalışıyor, bir programcının o sınıf üzerindeki çalışması çeşitli nedenlerle kaç defa kesiliyor gibi.
 - Bazı araştırmacılar proje ekibi metrikleri ile etkileşim metriklerini aynı grupta toplamaktadır.
 - Etkileşim metrikleri ile ilgili kaynaklar:
 - Taek Lee, et al. "Micro interaction metrics for defect prediction", ESEC/FSE 2011.
<https://doi.org/10.1145/2025113.2025156>
 - Taek Lee, et al. "Developer Micro Interaction Metrics for Software Defect Prediction", IEEE Transactions on Software Engineering, 2016.
<https://doi.org/10.1109/TSE.2016.2550458>
- Bu dersin ana konusu **nesneye dayalı tasarım kalitesi** olduğundan öncelikle yazılımların iç özelliklerine ilişkin tasarım (kod) metrikleri ele alınacaktır.

Tasarım (kod) Metriklerinin Türleri

- Yazılım tasarım metrikleri elde edildikleri ortama göre ikiye ayrılır:
 1. **Statik metrikler:** Program çalıştırılmadan, kaynak koddan ya da tasarım modelinden elde edilirler.
Örneğin; sınıfıktaki metod sayısı, sınıfın bağımlılığı, kodun satır sayısı gibi.
Bu metrikler yazılımın iç nitelikleri ile ilgilidir.
 2. **Dinamik metrikler:** Program çalışırken elde edilirler.
Örneğin; çalışma sırasında çağrılan metod sayısı, test sırasında bir sınıfta çıkan hata sayısı, bir metodun çalışma süresi gibi.
Bu metrikler yazılım ürünlerinin (sınıf, metod, modül) dış nitelikleri ile ilgilidir.
- Metrik değerleri iki şekilde elde edilebilir:
 1. **Doğrudan:** Yazılım (ya da tasarım modeli) üzerinde doğrudan ölçme (veya sayma) yapılarak
Örneğin, bir yazılım sınıfındaki satır sayısı
 2. **Dolaylı:** Yazılımlardan elde edilen veriler üzerinde hesaplar yapılarak
Örnekler: Bir programdaki hata yoğunluğu (hata / kod satırı oranı), Uyum (sonraki yansılarda)
- Metrikler farklı düzeydeki yazılım ürünlerinden toplanabilir:
 - Proje, mikroservis, sınıf, metod.
 - Ölçülmesi istenen birime göre metrikler uygun ürününden toplanmalı veya uygun düzeye uyarlanmalı.

Chidamber - Kemerer (CK) Metrik Kümesi

S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, No. 6, pp. 476-493, 1994.

- **Nesneye dayalı** yazılımlar için geliştirilen ilk temel metrik kümesi olduğundan önce CK kümesi ele alınacaktır.
- Bundan önce yayımlanan metrikler işlevsel programlar içindir.
Örneğin: kaynak kodu satır sayısı, açıklama yüzdesi, McCabe çevrimisel karmaşıklığı (*Cyclomatic Complexity*)
- Nesneye dayalı yazılımlarda tasarım sorunlarının (özellikle karmaşıklık) belirlenmesi hedeflenmektedir.
- Bu nedenle de **sınıfların karmaşıklığını** ölçebilecek metrikler hedeflenmiştir.
- Bu metrikler yazılımin **statik** yapısı (tasarımı) ile ilgilidir.
Metrikler programı çalıştırmadan elde edilebilirler.
- Sistemin dinamik davranışları ile ilgili verdikleri bilgiler kısıtlıdır.
- Belli bir programlama diline (gerçekleme) bağlı değildirler (C++, Java, C#, ...).
- İki yazılım firmasından toplanan veriler üzerinde denenmiştir.
A firması: C++ ile GUI kütüphaneleri yazıyor. 634 sınıfından veri toplanmıştır.
B firması: Yürületken eleman üreticisi. Smalltalk ile üretim ve otomasyon yazılımı yapıyor. 1459 sınıfından veri toplanmıştır.

Chidamber - Kemerer Nesneye Dayalı Tasarım Metrikleri

1. Weighted methods per class (WMC)

- Sınıf başına düşen ağırlıklı metod (sayısı)
- C sınıfının n adet metodu (M_1, M_2, \dots, M_n) varsa ve bu metodların karmaşıklıkları c_1, c_2, \dots, c_n ise WMC aşağıdaki gibi hesaplanır:

$$WMC = \sum_{i=1}^n c_i \quad \text{Metotların karmaşıklıkları } (c_i) \text{ farklı şekillerde hesaplanabilir.}$$

Örneğin McCabe yöntemi (*Cyclomatic Complexity*) kullanılabilir.

McCabe çevrimisel karmaşıklığı (*Cyclomatic Complexity*) :

- Bir programda (fonksiyonda) başlangıç ile bitiş arasında kaç farklı yol olduğunu gösterir.
 - Örneğin: içinde hiç dalandırma, döngü olmayan bir fonksiyonun McCabe karmaşıklığı 1'dir.
 - Tek bir IF deyimi varsa karmaşıklık 2 olur.
- T. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308-320, 1976.

- Tüm metodların karmaşıklıkları 1 olarak kabul edilirse WMC = n olur.

WMC Anlamı:

- WMC değeri o sınıfın geliştirilmesi ve bakımı için harcanması gereken zaman ve eforu gösterir.
- Metot sayısının fazlalığı alt sınıfların da karmaşıklığını arttırmır.
- Metot sayısı fazla olan sınıflar büyük olasılıkla daha özel işler yaparlar ve tekrar kullanılmaları zordur.
- Bu değerin küçük olması tercih edilir.

Deneysel Sonuçlar:

A : C++ ile yazılan GUI kütüphaneleri, 634 sınıf.

B : Smalltalk ile yazılan devre üretim programı kütüphaneleri, 1459 sınıf.

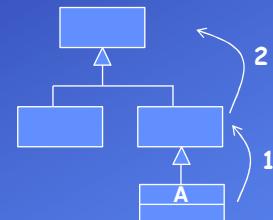
Tüm $c_i = 1$ kabul edilmiştir.

Yazılım	Median	Max	Min
A	5	106	0
B	10	346	0

- Çoğunlukla değerler düşük.
- 106'lık sınıfın hiç alt sınıfı yok.
- 87 metotlu bir sınıfın doğrudan 14 alt sınıfı ve onlardan ulaşılan 43 dolaylı alt sınıfı var. Tehlikeli olabilir.

2. Depth of inheritance tree (DIT)

- Kalıtım (türetim) ağacının derinliği.
- İncelenen sınıfın kalıtım (türetim) ağacında köke olan uzaklığıdır.
- Çoklu kalıtım (multiple inheritance) olan programlama dillerinde köke olan en uzun yol hesaplanır.

**Anlamı:**

- Bir sınıf ağaçta ne kadar derindeyse o kadar çok niteliği (veri ve metot) kalıtım yoluyla aldığı için davranışını kısıtlemek zorlaşır.
- Derinliği fazla olan ağaçlar tasarım karmaşıklığına işaret edebilir.
- Bir sınıfın ağaçta derinde olması birçok niteliği tekrar kullandığı anlamına da gelir.

2. Depth of inheritance tree (DIT) (devamı)

Deneysel Sonuçlar:

Yazılım	Median	Max	Min
A	1	8	0
B	3	10	0

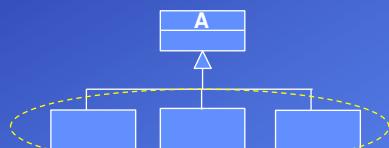
- Değerler düşük. Sınıfların çoğu kalıtım ağacının köküne yakın.
- Proje B'de değerler biraz daha yüksek çünkü Smalltalk dilinde her veri nesne olarak tasarlanmaktadır; tüm sınıflar "object" sınıfından türemektedir.
- Tekrar kullanılabilirlik dikkate alınmamış olabilir (olumsuz).
- Kalıtım (is-a) yerine sahip olma (has-a) ilişkisi veya sadelik tercih edilmiş olabilir (olumlu).
- Metrikleri birlikte kullanmak gereklidir.
 - Örneğin Proje A'da DIT değeri 8 olan biri sınıfı sadece 4 metod ve yerel veriler bulunuyor (WMC değeri düşük).
 - Ancak bu sınıfın nesneleri kalıtım yoluyla 132 metoda sahip oluyorlar.
 - Bu sınıfı gerçekleştirmek kolay olmuştur, ancak test edilmesi ve bakımının yapılması zor olabilir.

3. Number of children (NOC)

- Bir sınıfından doğrudan türetilen sınıfların sayısı

Anlamı:

- Çok sayıda sınıf türetilmesi birçok niteliğin tekrar kullandığı anlamına gelebilir.
- Çok sayıda sınıf türetilmesi türetimin yanlış kullandığı anlamına da gelebilir.
- NOC, bir sınıfın tasarım üzerindeki etkisini gösterir.



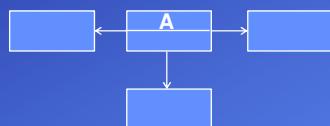
Deneysel Sonuçlar:

Yazılım	Median	Max	Min
A	0	42	0
B	0	50	0

- Değerler düşük. A'da sınıfların %73'ünün, B'de ise %68'nin alt sınıfı yok.
- Tasarım yöntemi olarak kalıtım (is-a) yerine sahip olma (has-a) tercih edilmiş olabilir (olumlu).
- Takım elemanları arasında iletişim eksikliği olduğu için tekrar kullanım düşük olabilir (olumsuz).

4. Coupling between objects (CBO)

- Bir sınıfın bağımlı olduğu diğer sınıfların sayısı.
- Burada bağımlılık, bir sınıfın diğerinin metodunu çağırması veya niteliklerine erişmesi olarak tanımlanmıştır.
- Kalitim ile oluşan bağımlılık dikkate alınmamıştır.



Anlamı:

- Sınıflar arası bağımlılığın yüksek olması modüler tasarıma aykırıdır ve sınıfın tekrar kullanabilirliğini azaltır.
- Bağımlılığı yüksek olan sınıf değişimlerden çok etkilenir ve bakımı zordur.
- Bağımlılığı yüksek olan sınıfın test işlemleri de karmaşıklaştırır.

Deneysel Sonuçlar:

Yazılım	Median	Max	Min
A	0	84	0
B	9	234	0

- B yazılımı Smalltalk ile yazılmıştır. Tüm veri tipleri sınıf olduğu için burada bağımlılık daha yüksektir.
- Bağımlılığı yüksek olan sınıfları incelemek gereklidir.
- Aracı (facade, controller) sınıfların da CBO değeri yüksek olabilir.
- Çok sayıda sınıfın bağımlılığının sıfır olması da dikkat çekici bir durumdur.

Sınıflar arası işbirliği sağlanmamış olabilir.

5. Response for a class (RFC)

- Bir sınıfın yanıt kümesi RS, o sınıfa bir mesaj geldiğinde etkinleşmesi mümkün olan tüm metodların oluşturduğu kümedir.

$$RS = \{M\} \cup_{\text{tüm } i} \{R_i\}$$

M, o sınıftaki tüm metodlar; R_i ise M_i metodunda çağırılan diğer metodlardır.

RS o sınıftaki tüm metodlar ve o metodların içinde doğrudan çağrılan diğer metodların kümesidir. Her metod kümede bir kez yer alır.

$$RFC = |RS|; \quad RFC \text{ çağrılan olası metodların sayısıdır.}$$

Anlamı:

- RFC'nin yüksek olması sınıfın karmaşık olduğunu gösterir.
- RFC değeri yüksek olan sınıfları sınamak ve hataları ayıklamak daha güçtür.
- Olası en yüksek RFC değeri o yazılımin sınaması için gerekli olan süre hakkında fikir verir.

Deneysel Sonuçlar:

Yazılım	Median	Max	Min
A	6	120	0
B	29	422	3

Sınıfların çoğunun RFC değeri düşüktür.

6. Lack of cohesion in methods (LCOM)

- Sınıfın uyumsuzluğu hakkında bilgi verir.
- Hesaplanması:
 - C sınıfının n adet metodu (M_1, M_2, \dots, M_n) bulunmaktadır.
 - $\{I_i\}$, M_i metodunun eriştiği sınıf nitelikleri (attribute) Kümesidir. $\{I_1\}, \{I_2\}, \dots, \{I_n\}$
 - P, kesişimleri boş küme olan niteliklerin Kümesidir. $P = \{ (I_i, I_j) | \{I_i\} \cap \{I_j\} = \emptyset \}$
 - $|P|$ ortak niteliklere erişmeyen (uyumlu olmayan?) metod çiftlerinin sayısını ifade eder.
 - Q, kesişimleri boş küme olmayan niteliklerin Kümesidir. $Q = \{ (I_i, I_j) | \{I_i\} \cap \{I_j\} \neq \emptyset \}$
 $|Q|$ ortak nitelikleri kullanan (ilgili/uyumlu?) metod çiftlerinin sayısını ifade eder.

$LCOM = |P| - |Q|$ eğer $|P| > |Q|$ ise; aksi durumda $LCOM=0$.

Anlamı:

- Metotların uyumlu (birbirileriyle ilgili) olmaları tercih edilir.
- Uyumsuzluk değerinin yüksek olması o sınıfın bölünmesi gerektiğini gösterebilir.
- Yüksek LCOM değeri yazılımın tasarımında sorun olduğunu gösterebilir.

Deneysel Sonuçlar:

Yazılım	Median	Max	Min
A	0	200	0
B	2	17	0

Sınıfların çoğunun LCOM değeri düşüktür.

C-K Metriklerinin ilgili olduğu nesneye dayalı tasarım aşamaları

Metric	Identification	Semantics	Relationships
WMC	X	X	
DIT	X		
NOC	X		
CBO			X
RFC		X	X
LCOM		X	

Metriklerin geçerliliğinin değerlendirmesi

- Önerilen metriklerin geçerli ve kullanılabılır olduğu (yazılım sistemlerinin kalitesi hakkında geçerli bilgi verdikleri) nasıl anlaşılır?
- Metriklerin verdiği sayısal değerlerin gerçek dünyayı temsil edip etmediği araştırılır.
- Gerçek yazılımlar üzerinde yapılan çalışmalar insan etkileşimi gerektirir, tüm olası senaryoları oluşturmak ve denemek zordur. Bu yöntem zahmetlidir, zaman alır.
- Bu makalede, formel bir kriter kümesi araştırılarak E. Weyuker tarafından geliştirilen kriterler ile yeni önerilen metrikler değerlendirilmiştir.

E. Weyuker, "Evaluating software complexity measures," *IEEE Trans. Software Eng.*, vol. 14, pp. 1357-1365, 1988.

Weyuker Kriterleri:

- Yazılım karmaşıklık metriklerinin uyması gereken 9 özellik belirtilmiştir.
- Fonksiyonlar için düşünülmüşlerdir. Nesneye dayalı yazılımlar (sınıflar) hedeflenmemiştir.
- Başka uzamanlar tarafından aşağıdaki iddialarla eleştirilmiştir:
 - Ölçeklenebilir değildir (karmaşıklık dışındaki metriklere uygulanamaz).
 - Sadece gerekli koşulları ortaya koymaktadır, yeterli koşulları değil.
 - Ölçme teorisine uygun değildir. Karmaşıklık tek metrikler ölçülebilecek basit bir kavram değildir.
- Weyuker ve ekibi bu eleştirilere kendi yayınlarında cevap vermiştir.

Weyuker metrik değerlendirme kriterleri (sınıflar için uyarlanmış):

μ bir yazılım metriği ise aşağıdaki 9 koşulu sağlamalıdır:

- Tüm sınıflar aynı metrik değerini vermemeli.
 P ve Q iki farklı sınıf olmak üzere $\exists P, \exists Q \quad \mu(P) \neq \mu(Q)$.
- c negatif olmayan bir sayı olmak üzere sadece sonlu sayıda sınıfın (P) karmaşıklık değeri $\mu(P) = c$ olabilir.
 Bu koşul zaten sağlandığı için C-K tarafından kullanılmamıştır.
- Bazı sınıflar aynı sonucu verebilir.
 P ve Q iki farklı sınıf olmak üzere $\exists P, \exists Q \quad \mu(P) = \mu(Q)$ olabilir.
- Tasarımın ayrıntıları (gerçekleme) önemlidir.
 P ve Q aynı işi yapan fakat tasarımları farklı iki sınıf ise $\mu(P) \neq \mu(Q)$ olabilir.
- Bir program parçasına (sınıfa) ekleme yapmak onun karmaşıklığını artırır.
 P ve Q iki farklı sınıf olmak üzere $\forall P, \forall Q \quad \mu(P) \leq \mu(P;Q)$ ve $\mu(Q) \leq \mu(P;Q)$.
 $P;Q$ iki sınıfın birleştirilmesi anlamına gelmektedir.

Weyuker metrik değerlendirme kriterleri devamı:

6. P ile R sınıfları arasındaki etkileşim Q ile R arasındakinden farklı olabilir.
 - a. $\exists P, \exists Q, \exists R \mu(P) = \mu(Q) \wedge \mu(P;R) \neq \mu(Q;R)$
 - b. $\exists P, \exists Q, \exists R \mu(P) = \mu(Q) \wedge \mu(R;P) \neq \mu(R;Q)$
7. Permütasyon önemlidir: Bir yazılımda satırların yerini değiştirirseniz karmaşıklığı da (metrik değeri) değişebilir.
Sınıflar için uygulanabilir olmadığı için C-K tarafından kullanılmamıştır.
8. Renaming Principle: Bir sınıfın sadece adının değiştirilmesi onun metrik değerini değiştirmez.
Bu koşul zaten sağlandığı için C-K tarafından kullanılmıştır.
9. Etkileşim karmaşıklılığı artırır: $\exists P, \exists Q \mu(P) + \mu(Q) < \mu(P;Q)$.

C-K Metriklerinin Weyuker kriterlerini sağlaması durumları:

Metrik	W1	W3	W4	W5	W6	W9
WMC	+	+	+	+	+	-
DIT	+	+	+	-	+	-
NOC	+	+	+	+	+	-
CBO	+	+	+	+	+	-
RFC	+	+	+	+	+	-
LCOM	+	+	+	-	+	-

- W2 "sonlu sayıda aynı değer" ve W8-"isim değişikliği" hepsi tarafından sağlanıyor.
- W7 "permutation" uygulanabilir değil.
- W9 "Etkileşim karmaşıklığı artırır" hiçbir tarafından sağlanmıyor.

Weyuker kriterleri ile ilgili eleştiriler:

- H. Zuse, "Support of experimentation by measurement theory," *Lecture notes in computer science*, vol. 706, Springer-Verlag, 1993, pp. 137-140.
https://link.springer.com/content/pdf/10.1007/3-540-57092-6_114
- B. Kitchenham, S. L. Pfleeger and N. Fenton, "Towards a framework for software measurement validation," *IEEE Transactions on Software Engineering*, vol. 21, no. 12, pp. 929-944, Dec. 1995.
doi: 10.1109/32.489070
- Zuse'nin eleştirisi:
 - Kriterler farklı ölçekleri gerektiriyor. Aralarında tutarsızlık var.
- Fenton'un temel eleştirisi: "Karmaşıklık çok bileşenli bir kavramdır, tek bir metrik (sayı) ile ifade edilemez".
 - Yazılımın karmaşıklığının birçok kalite niteliğini etkilediği düşünülmektedir (bilinmektedir, sezilmektedir): anlaşılırlık, esneklik, bakım kolaylığı, güvenirlilik, test kolaylığı, gerçekleme kolaylığı.
Çok sayıda ve bazen çatışan niteliği tek bir sayı ile ifade etmek olası değildir.
 - Kriterler (5, 6, 9) belli bir ölçüye zorunlu hale getirmemeli.
Bir nitelik, gereksinimlere bağlı olarak farklı ölçeklerde sonuç üreten farklı ölçme yöntemleri ile ölçülebilir.

Weyuker kriterleri ile ilgili eleştiriler (devamı):**Fenton'un eleştirileri (devamı):**

- *Kriter 5: Bir program parçasına (sınıfa) ekleme yapmak onun karmaşıklığını arttırır.*
 - Bu kriter boyut ile ilgilidir; yazılımın boyutu artarsa karmaşıklığının da artacağını ifade etmektedir.
 - Bu kriter kodun yapısal karmaşıklığını dikkate alır, ancak anlaşılırlığı (psikolojik karmaşıklığı) göz ardı etmektedir.
 - Bazen bir koda ekleme yapmak onu daha anlaşılır hale getirebilir.
- *Kriter 7: Bir program parçasında satırların yerini değiştirmek karmaşıklığı değiştirebilir.*
 - Bu kriter kodun psikolojik karmaşıklığını (anlaşılırlık) dikkate alır, ancak yapısal karmaşıklığını göz ardı etmektedir.
- *Kriter 9: Karmaşıklık değerleri toplanabiliyor.*
 - Bu kriter ölçmenin en azından aralık ölçüğünde olmasını gerektirir.
 - Ölçek ilgili niteliğin sağladığı ilişkilere bağlıdır. Ölçme yöntemi belli bir ölçeğin kullanılmasını zorunlu kılamaz.

Weyuker ve ekibi bu eleştirilere cevap vermiştir:

- S. Morasca et al., "Comments on 'Towards a framework for software measurement validation,'" *IEEE Transactions on Software Engineering*, vol. 23, no. 3, pp. 187-189, Mar. 1997.
doi: 10.1109/32.585506.

Nesneye dayalı tasarım metriklerine diğer örnekler:

- Yazılım sınıfları ile ilgili başka arabaşka araştırmacılar da çeşitli metrikler oluşturmuşlardır.
- Dersin ilerleyen bölümlerinde bu metriklerin kullanımı da ele incelenecektir.
- Bazı örnek metrikler aşağıda verilmiştir.
- **TCC** (*Tight Class Cohesion*): Bieman and B.K. Kang
 - Bir sınıfındaki doğrudan (sıkı) bağlı metod çiftlerinin sayısı, tüm olası metod çiftlerinin sayısına bölünür.
 $TCC = NDC(C) / NP(C)$
 İki metod ait oldukları sınıfın en az bir tane niteliğini ortak olarak kullanıylarsa doğrudan (sıkı) bağlıdır.
 - C sınıfında N adet metod varsa $NP(C) = N(N-1)/2$ (Olası tüm çiftler)
 - Bu değerin büyük olması o sınıfın metod uyumunun yüksek olduğunu gösterir.
- **WOC** (*Weight Of Class*): Lanza and Marinescu
 - Bir sınıfta sadece erişim amaçlı olmayan (*non-accessor*) metodların sınıfındaki tüm açık (*public*) metodlara oranı.
 - Bu sayının 1'e yakın olması o sınıfın işlevsel bir hizmet verdiği anlamına gelir.
 - Değerin küçük olması sınıfın daha çok veri tutma hizmeti verdiği anlamına gelir.
- **CAMC** (*Cohesion Among Methods in a Class*): Bansya, Etkorn, Davis, and Li
 - Bir sınıfta, ortak tipte parametreye sahip metodların oranı hesaplanır.

Yazılım kalitesinin metrikler ile ölçülmesindeki problemler ve hedefler:

- Çok sayıda (yüzlerce) metrik tanımlanmıştır ancak hepsi yararlı (kullanılabilir) bilgi sağlamamaktadır. Metriklerin gerçek projelerde sınanarak kalite nitelikleri veya tasarım kusurları ile ilişkileri ortaya koymalıdır.
 - Metrikler alt düzeydeki, küçük taneli (*fine grained*) niteliklere karşı düşen sayılardır. Asıl hedef daha yüksek düzeyeli nitelikleri değerlendirmek olduğundan metrikler ancak bir model ile birlikte değerlendirilebilirler (Bkz. ISO 25020).
 - Metriklerin değer aralıkları çok farklı olduğundan ($[1, \infty]$, $[0, 1]$) modelin içinde bu değerlerin birlikte nasıl kullanılacağına dikkat edilmelidir.
 - Metriklerin eşik değerlerini (iyi/kötü değerler) önceden belirlemek zordur. Bu zorluğu aşmak için kural tabanlı yerine öğrenme tabanlı yöntemler kullanılabilir.
 - Metrikleri yazılım geliştirmenin erken evrelerinde (örneğin tasarım) elde edilebilirlerse projenin ilerlemesinde kullanılabilirler.
- Nesneye dayalı tasarım bu konuda avantaj sağlar. Çünkü bu yönteme göre yazılım geliştirirken kodlamadan önce yazılımın yapısının tasarlanması gereklidir.

Dersin ilerleyen bölümlerinde bu maddelere degeinilecektir.