

Yazılım Metrikleri (Ölçüler) (Software Metrics)

Yazılım metrikleri, yazılımların ölçülebilen nitelikleri ve bu niteliklere sayıların karşı düşürülmesi yöntemleridir.

Örneğin; bir metottaki satır sayısı, bir sınıfın türetim zincirindeki derinliği, bir programın çalışması sırasında çıkan hata sayısı gibi.

Yıllar içinde "metrik" sözcüğüne farklı anlamlar yüklenmiştir.

Örneğin; ölçülen nitelik, bu niteliğin ölçülme yöntemi, ölçüm sonucu atanan sayı gibi.

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. 2.38).

Ancak makale ve bildirilerde metrik sözcüğü özellikle nitelik ve QME yerine hala kullanılmaktadır.

Metrik Türleri

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

- **Tasarım (kod) metrikleri:**

Doğrudan programın kodundan veya kodlama öncesindeki tasarımdan elde edilirler.

İç niteliklerle veya dış niteliklerle ilgili olabilirler.

Örneğin; bir sınıftaki metot sayısı, bir metottan çağırılan diğer metotların sayısı.

- **Tarihsel (historical) metrikler:**

Yazılım projesinin geçmişi ile ilgili bilgi verirler.

Süreç metrikleri (*process metrics*) veya evrimsel metrikler (*evolution metrics*) olarak da adlandırılırlar.

Örneğin; şimdiye kadar bir sınıfta belirlenen hata sayısı, bir sınıfın projeye eklenmesinden bu yana geçen süre (sınıfın yaşı), projede kaç sürüm oluşturulduğu, şimdiye 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 kaçınıcı 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.

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>

Bazı araştırmacılar proje ekibi metrikleri ile etkileşim metriklerini aynı grupta toplamaktadırlar.

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; sınıftaki metot 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 metot sayısı, test sırasında bir sınıfta hata çıkması, bir metodun çalışma süresi gibi.
Bu metrikler yazılım ürünlerinin (sınıf, metot, 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
Örneğin, bir programdaki hata yoğunluğu (hata / kod satırı oranı)

Chidamber - Kemerer 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 çevrimsel karmaşıklığı (*Cyclomatic Complexity*)

Yazılım ürünlerinden tasarım kalitesinin ölçülmesi hedeflenmektedir.

Önerilen metrikler Weyuker karmaşıklık ölçme prensipleri ile sınırlanmıştır.

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

İki yazılım firmasından toplanan veriler üzerinde denenmiştir.

A firması: C++ ile GUI kütüphaneleri yazıyor. 634 sınıftan veri toplanmıştır.

B firması: Yarıiletken eleman üreticisi. Smalltalk ile üretim ve otomasyon yazılımı yapıyor. 1459 sınıftan veri toplanmıştır.

Chidamber - Kemerer Metrik Kümesi

Teorik Çıkış Noktası:

Gredy Booch nesneye dayalı tasarımın dört temel aşaması olduğunu belirtmiştir:

1. *Identification*: Sınıfların (ve nesnelerin) varlıklarının bulunması (belirlenmesi)
2. *Semantics*: Sınıfların (ve nesnelerin) anlamlarının (sorumluluklarının) belirlenmesi
3. *Relationships*: Sınıflar (ve nesneler) arasındaki ilişkilerin belirlenmesi
4. *Implementation*: Sınıfların (ve nesnelerin) gerçekleştirilmesi

Buna göre nedenle nesneye dayalı tasarımın (OOD) temeli sınıflardır.

Bu nedenle de sınıfların karmaşıklığını ölçebilecek metrikler hedeflenmiştir.

- Bu metrikler yazılımın **statik** yapısı (tasarımı) ile ilgilidir.
- Programı çalıştırmadan elde edilebilirler.
- Sistemin dinamik davranışı ile ilgili verdikleri bilgiler kısıtlıdır.
- Belli bir programlama diline (gerçekleme) bağlı değildirler (C++, Java, C#, ...).

Chidamber - Kemerer Nesneye Dayalı Tasarım Metrikleri

1. Weighted methods per class (WMC)

Sınıf başına düşen ağırlıklı metot (sayısı)

C sınıfının n adet metodu ($M_1, M_2, \dots, M_n$) varsa ve bu metotların karmaşıklıkları $c_1, c_2, \dots, c_n$ ise WMC aşağıdaki gibi hesaplanır:

$$WMC = \sum_{i=1}^n c_i$$

Metotların karmaşıklıkları (c_i) farklı şekillerde hesaplanabilir. Örneğin McCabe yöntemi (*Cyclomatic Complexity*) kullanılabilir.

Tüm metotların karmaşıklıkları 1 olarak kabul edilirse $WMC = n$ olur.

McCabe çevrimsel karmaşıklığı (*Cyclomatic Complexity*) :

Bir programda (fonksiyonda) başlangıç ile bitiş arasında kaç farklı yol olduğunu gösterir.

Örneğin; içinde hiç dallanma, 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.

WMC Anlamı:

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

Deneyisel 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.

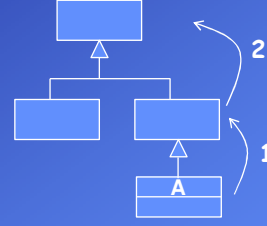
87 metotlu bir sınıfın doğrudan 14 alt sınıfı ve onlardan ulaşılan 43 dolaylı alt sınıfı var. Tehlikeli olabilir.

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 kestirmek 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.

Kalıtım yerine sahip olma ilişkisi veya sadelik tercih edilmiş olabilir.

Metrikleri birlikte kullanmak gerekir.

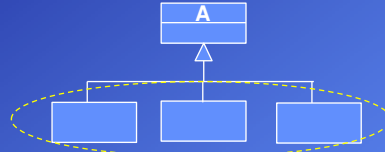
Örneğin Proje A'da DIT değeri 8 olan biri sınıfta sadece 4 metot ve yerel veriler bulunuyor (WMC değeri düşük).

Ancak bu sınıfın nesneleri kalıtım yoluyla 132 metoda sahip oluyorlar.

Bu sınıfı gerçeklemek kolay olmuştur, ancak test edilmesi ve bakımının yapılması zor olabilir.

3. Number of children (NOC)

Bir sınıftan doğrudan türetilen sınıfların sayısı



Anlamı:

- Çok sayıda sınıf türetilmesi birçok niteliğin tekrar kullandığı anlamına gelebilir.
- Çok sayıda sınıf türetilmesi türetimin yanlış kullandığı anlamına gelebilir.
- NOC, bir sınıfın tasarım üzerindeki etkisini gösterir. Çok sayıda alt sınıfı olan sınıf, tekrar kullanıldığı ve birçok sınıfı etkilediği için dikkatle test edilmelidir.

Deneyisel 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.

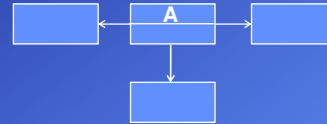
Tasarım yöntemi olarak kalıtım tercih edilmemiş olabilir. (Sahip olma?)

Takım elemanları arasında iletişim eksikliği olduğu için tekrar kullanım düşük olabilir.

4. Coupling between objects (CBO)

Bir sınıfın bağımlı olduğu diğer sınıfların sayısı.

Burada bağımlılık, bir sınıfın diğerinin metodunu çağırması veya niteliklerine erişmesi olarak tanımlanmıştır.



Anlamı:

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

Deneyisel 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 gerekir.

Aracı (facade, controller) sınıfların CBO değeri yüksek olabilir.

5. Response for a class (RFC)

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

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

M, o sınıftaki tüm metotlar; R_i ise M_i metodunda çağırılan diğer metotlardır.

RS o sınıftaki tüm metotlar ve o metotların içinde doğrudan çağırılan diğer metotların kümesidir. Her metot kümede bir kez yer alır.

$RFC = |RS|$; RFC çağırılan olası metotların 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 yazılımın 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.

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.

$$P = \{ (I_i, I_j) \mid \{I_i\} \cap \{I_j\} = \emptyset \}$$

$|P|$ ortak niteliklere erişmeyen metot çiftlerinin sayısını ifade eder.

Q, kesişimleri boş küme olmayan niteliklerin kümesidir.

$$Q = \{ (I_i, I_j) \mid \{I_i\} \cap \{I_j\} \neq \emptyset \}$$

$|Q|$ ortak nitelikleri kullanan (ilgili ?) metot çiftlerinin sayısını ifade eder.

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

Anlamı:

- Metotların uyumlu (birbirleriyle 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
<i>WMC</i>	X	X	
<i>DIT</i>	X		
<i>NOC</i>	X		
<i>CBO</i>			X
<i>RFC</i>		X	X
<i>LCOM</i>		X	

Metriklerin geçerliliğinin değerlendirilmesi

Önerilen metriklerin geçerli ve kullanılabilir 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.

Yazılım (karmaşıklık) metriklerinin uyması gereken 9 özellik belirtilmiştir.

Nesneye dayalı yazılımlar hedeflenmemiştir.

Başka uzmanlar tarafından eleştirilmiştir: Ölçeklenebilir değildir (karmaşıklık dışındaki metriklere uygulanamaz); sadece gerekli koşulları ortaya koymaktadır, yeterli koşulları değil.

Weyuker metrik değerlendirme kriterleri (sınıflar için uyarlanmış):

μ bir yazılım metriği ise aşağıdaki 9 koşulu sağlamalıdır:

1. Tüm sınıflar aynı metrik değerini vermemeli.
P ve Q iki farklı sınıf olmak üzere $\exists P, \exists Q \quad \mu(P) \neq \mu(Q)$.
2. 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.
3. Bazı sınıflar aynı sonucu verebilir.
P ve Q iki farklı sınıf olmak üzere $\exists P, \exists Q \quad \mu(P) = \mu(Q)$ olabilir.
4. Tasarımın ayrıntıları (gerçekleme) önemlidir.
P ve Q aynı işi yapan fakat tasarımları farklı iki sınıf ise $\mu(P) \neq \mu(Q)$ olabilir.
5. 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.
 $\exists P, \exists Q, \exists R \quad \mu(P) = \mu(Q) \wedge \mu(P+R) \neq \mu(Q+R)$
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 kullanılmamıştır.
9. Etkileşim karmaşıklılığı artırır: $\exists P, \exists Q \quad \mu(P) + \mu(Q) < \mu(P+Q)$.
Burada birinci "+" aritmetik toplama ikinci "+" ise iki sınıfın birleştirilmesi anlamına gelmektedir.

C-K Metriklerinin Weyuker kriterlerini sağlama 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" hiçbiri tarafından sağlanmıyor.

Fenton'un Weyuker kriterleri ile ilgili eleştirisi:

Fenton'un temel eleştirisi karmaşıklık gibi çok bileşenli bir kavramın tek bir metrik ile ifade edilemeyeceğidir.

Yazılımın karmaşıklığının birçok kalite niteliğini etkilediği düşünülmektedir (bilinmektedir, sezilmektedir): anlaşılabilirlik, esneklik, bakım kolaylığı, güvenilirlik, test kolaylığı, gerçekleştirme kolaylığı.

Çok sayıda ve bazen çatışan niteliği tek bir sayı ile ifade etmek olası değildir.

Weyuker (karmaşıklık) kriterleri ile ilgili eleştirisi:

5. Bir program parçasına (sınıfa) ekleme yapmak onun karmaşıklığını arttırır.

P ve Q iki farklı sınıf olmak üzere $\forall P, \forall Q \mu(P) \leq \mu(P+Q)$ ve $\mu(Q) \leq \mu(P+Q)$.

Bu kriter boyut ile ilgilidir; yazılımın boyutu artarsa karmaşıklığının da artacağını ifade etmektedir.

Ancak bu kriter anlaşılabilirliği göz ardı etmektedir.

Bazen bir koda ekleme yapmak onu daha anlaşılır hale getirebilir.

6. P ile R sınıfları arasındaki etkileşim Q ile R arasındakinden farklı olabilir.

$\exists P, \exists Q, \exists R \mu(P) = \mu(Q) \wedge \mu(P+R) \neq \mu(Q+R)$

Bu kriter ise boyutu değil anlaşılabilirliği dikkate almaktadır.

Boyutun aynı miktarda artması karmaşıklığı aynı miktarda arttırmıyor.

Bu iki kriter çelişmektedir.

Diğer tasarım metriklerine örnekler:

Chidamber ve Kemerer'in çalışmasından sonra başka araştırmacılar da nesneye dayalı yazılımları değerlendirmek için ortaya çeşitli metrikler koymuşlardır.

Bazı örnek metrikler aşağıda verilmiştir.

- **TCC** (*Tight Class Cohesion*): Bieman and B.K. Kang

Bir sınıftaki doğrudan (sıkı) bağlı metot çiftlerinin sayısı, tüm olası metot çiftlerinin sayısına bölünür. $TCC = NDC(C) / NP(C)$

İki metodun doğrudan (sıkı) bağlıdır eğer bu metotlar ait oldukları sınıfın en az bir tane niteliğini ortak olarak kullanıyorlarsa.

C sınıfında N adet metot varsa $NP(C) = N(N-1)/2$ (Olası tüm çiftler)

Bu değerin büyük olması o sınıfın metot uyumunun yüksek olduğunu gösterir.

- **WOC** (*Weight Of Class*): Lanza and Marinescu

Bir sınıfta sadece erişim amaçlı olmayan (*non-accessor*) metotların sınıftaki tüm açık (*public*) metotlara oranı.

Bu sayının 1'e yakın olması o sınıfın işlevsel bir hizmet verdiği anlamına gelir.

Değerin küçük olması sınıfın daha çok veri tutma hizmet verdiği anlamına gelir.

- **CAMC** (*Cohesion Among Methods in a Class*): Bansiya, Etzkorn, Davis, and Li

Bir sınıfta, ortak tipte parametreye sahip metotların 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 koyulmalıdır.

- Metrikler alt düzeydeki, küçük taneli (*fine grained*) niteliklere karşı düşen sayılardır.

Asıl hedef daha yüksek düzeyli 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 yöntemle göre yazılım geliştirirken kodlamadan önce yazılımın yapısının tasarlanması gerekir.

Dersin ilerleyen bölümlerinde bu maddelere değinilecektir.