Tasarım Kusurlarının (*Design Flaws, Defects, Smells*) Belirlenmesi

- Kaliteli (doğru) bir tasarımın belli kalite niteliklerini (bkz. Kalite standartları) sağlaması beklenir.
- **Tasarım kusurları** (ya da kod kusurları) kaliteli bir tasarımın sahip olması gereken özelliklerden sapmalar olarak tanımlanırlar.
- Tasarım kusurları, kötü kokular (***bad smells***) olarak da adlandırılırlar.
- Tasarım kusuru içeren program modülleri derlemede, testlerde ya da programın çalışması sırasında hata vermeyebilirler.
- Ancak bu yapısal kusurlar -türlerine bağlı olarak- aşağıdaki sorunlara neden olurlar ve özellikle bakım maliyetini yükseltirler.
  - Yazılımın esnekliğini azaltırlar.
    - Yazılımı değiştirmek (değişen isteklere uyarlamak) zordur.
    - Yeni özellikler eklemek zordur.
  - Tekrar kullanılabilirliği azaltırlar.
  - Olası hatalara açıktırlar.
    - Sisteme yapılacak yeni eklemeler (ya da değişiklikler) kusurlu modüllerde sıklıkla mantık hatalarına neden olur.
  - Hataları gizlerler.
    - Hatanın kaynağını (hangi sınıf, hangi metot) bulmak kolay değildir.

Tasarım Kusurlarına Örnekler:

- Nesneye dayalı tasarımlarda sık karşılaşılan kusurlarla ilgili deneyimle oluşmuş bilgiler, görüşler ve isimlendirmeler bulunmaktadır.
- Yazılım dünyasında isimlendirilmiş tasarım kusurlarına ilişkin örnekler:
  - God Class: Karmaşık, uyumu kötü (fazla sorumluluk yüklenmiş) ve başka sınıfların verilerine yoğun olarak erişen sınıf
  - Schizophrenic Class: Birden fazla iş yüklenmiş, ara yüzünde farklı konularda hizmetler barındıran, uyumu kötü sınıf
  - Data Class: Anlamlı bir işi (sorumluluğu) olmayan, büyük ölçüde verilerden oluşan ve bu verileri doğrudan dışarıya açan sınıf
  - Brain method: Çok fazla iş yapan, uzun, fazla dallanma ve fazla iç içe blok içeren karmaşık metot. Sınıfın bütün sorumluluğu tek bir metoda yüklendiğinde ortaya çıkar.
  - Intensive Coupling: Bir metot çok sayıda başka metodu çağırıyor ve çağırılan bu metotlar başka sınıfta toplanmışlar.
  - Shotgun Surgery: Sınıfın bir metoduna farklı çok sayıda sınıftan çağrı yapılıyor.
  - Refused Parent Bequest: Alt sınıf, üst (taban) sınıflardan aldığı veri ve metotlardan yararlanmıyor.

Tasarım Kusurlarının Nedenleri:

1. Tasarımın iyi düşünülmeden kodlamaya geçilmiş olması. Tasarım kalıplarının (*design patterns*) dikkate alınmaması.
2. Zaman baskısı nedeniyle projenin gelecekteki durumu düşünülmeden anlık "çözümler" bulunması.

Teknik borç (*Technical debt*):

- Tasarıma yeterli zaman ayrılmadığında başta zaman kazanıldığı düşünülse de daha sonra (bakım aşamasında) ödenecek bir borç (iş gücü) oluşur.
- Bu borç tasarımdaki kusurlar nedeniyle oluşur.
- Çoğunlukla daha sonra tasarımı düzeltmek için harcanan süre (borç) başlangıçta tasarımın doğru bir biçimde oluşturulmak için harcanandan daha fazladır.
- Cunningham: "The debt incurred through the speeding up of software project development, which results in a number of deficiencies ending up in high maintenance overheads".
- McConnell: "A design or construction approach that's expedient in the short term but that creates a technical context in which the same work will cost more to do later than it would cost to do now (including increased cost over time)".

Tasarım kusurlarının metriklerle belirlenmesi (*Detection Strategies*)

- Kusurların belirlenmesinde modellere gereksinim vardır (tanı koyma).
  - Model, tasarım kusurunu açık biçimde tanımlamalı (tarif etmeli).
  - Alt düzeydeki doğrudan ölçülebilen özellikler ile üst düzeydeki kusurlar arasında ilişki kurmalı (Bkz. ytk02: ISO 2500mn SQuaRE standartları).

Yöntemlerin Türleri:

1. Kural Tabanlı Yöntemler:

- Belli bir kusuru bulmak için metriklerden (M1, M2, ...) ve belli referans değerlerinden oluşan kurallar uzamanlar tarafından belirlenir.

  Örneğin; M1 > 150 ve M2 < 0.3 → A tipi kusur vardır.
- Eğer bir yazılım biriminin (örneğin, mikroservis, sınıf, fonksiyon) metrik değerleri bu kurala uyuyorsa o birimde belli bir kusur var demektir.
- Kurallar daha karmaşık yapıda olabilir; bulanık mantık (*fuzzy logic*) kullanılabilir.

2. Makine Öğrenmesi Tabanlı Yöntemler:

- Kusurlu/kusursuz olduğu bilinen birimlerin (mikroservis, sınıf, metot) metrikleri kullanılarak eğitilen bir model bilgisayar yazılımı tarafından oluşturulur.
- Bu model daha sonra başka birimlerde kusur olup olmadığını belirlemek için kullanılır.

Tasarım kusurlarının kural tabanlı yöntemlerle belirlenmesi

Kural tabanlı modelin oluşturulması:

Kusurları bulurken kullanılacak model GQM yöntemi ile oluşturulabilir.

Hedef (G):

- Belli bir kusuru bulmak (kusurun tanımı yapılır)
- Kusurun tanımı yapılırken iyi (kabul edilebilir) tasarımın özellikleri dikkate alınır, bu özelliklerden yaygın olarak karşılaşılan sapmalar kusur olarak tarif edilir.

Sorular (Q):

- Kusurun tanımında yer alan sorular (belirtiler "*symptoms*") sorulur.
- Bu sorular, tasarımın veya programın ölçülebilir özellikleri ile ilgilidir.

Örneğin:

Sınıf çok büyük mü?

Metot karmaşık mı?

Sınıfın uyumu düşük mü?

Metot çok sayıda başka metoda bağımlı mı?

Kural tabanlı modelin oluşturulması (devamı):

Metrikler (M):

- Önceki bölümde sorulan yanıtlarını nicel olarak verebilecek metrikler belirlenir.

Örneğin; bir sınıftaki metot sayısı, bir metottaki iç içe blok sayısı, bir metottan çağırılan diğer metotların sayısı.

Değerlendirme ve Karar:

- Birden çok sorunun yanıtına ilişkin metriğin nasıl bir araya getirilip kusurun varlığı/yokluğu konusunda nasıl karar verileceğine ilişkin bir yöntem oluşturulur.
- Metrikler üzerinde aritmetik ve/veya mantıksal işlemler yapılabilir.
- Sonuçlar veya metriklerin değerleri önceden belirlenen bazı referans değerleri ile karşılaştırılır.

Referans (eşik) değerleri iyi/orta/kötü tasarımların özelliklerini temsil ederler.

Örnek: "God Class" belirlemek için örnek bir modelin oluşturulması

- Hedef (G) :

  "God Class" belirlemek. Karmaşık, uyumu kötü ve başka sınıfların verilerine yoğun olarak erişen sınıf

- Sorular (Q):

  Q1. Sınıfın karmaşıklığı yüksek mi?

  Q2. Sınıfın uyumu kötü mü?

  Q3. Sınıf başka sınıfların verilerine yoğun olarak erişiyor mu?

- Metrikler (M):

  M1. WMC (*Weighted methods per class*)

  M2. TCC (*Tight Class Cohesion*)

  M3. ATFD (*Access to Foreign Data*):

- Değerlendirme ve Karar: Metriklerden hedefe ulaşmak için gerekli olan kuralın belirlenmesi.

  Bu örnekte mantıksal (lojik) bir denklem oluşturulmuştur.

  God Class = (WMC > çok yüksek) VE (TCC < düşük) VE (ATFD > AZ)

  Buradaki sorunlardan biri de eşik (referans) değerlerinin belirlenmesidir.

  "Az", "çok", "çok yüksek" hangi değerlere karşı düşmektedir?

Metrikler ve Eşik (Referans) Değerleri (*Thresholds*):

- Metriklerin değerlerinin doğru bir biçimde yorumlanabilmesi için anlamlı eşik değerlerine (referans noktalarına) gerek vardır.

  Normal, az, çok, yüksek, düşük vs.

  Örneğin,

  - Bir insan hangi durumda uzun boyludur? 1.85m ya da 2m'den mi uzunsa?
  - Bir yazılım metodunda "normalde" kaç satır olur?

- Yazılım dünyasında kesin ("tam doğru") referans değerleri belirlenemese de pratikte kullanılabilecek değerlerin belirlenebilmesi amaçlanır.
- Metrikler için eşik değerleri belirlerken iki kaynaktan yararlanılır:

1. **İstatistiksel Yöntem (Otomatik):**

   Var olan yazılım projeleri incelenir ve belli metrik değerlerinin istatistiksel dağılımı (ortalama, ortanca değer (*median*), standart sapma) belirlenir.

2. **Anlamsal Eşik Değerleri (Uzman (insan) kararı):**

   Deneyimler (gözlemler) sonucu oluşmuş, genelde kabul gören eşik değerleri.

   Örneğin iç içe 3 veya daha az döngü "normal", daha çoğu "fazla" olarak nitelendirilebilir.

Metrikler ve Eşik Değerleri (*Thresholds*) (devamı)

1. **İstatistiksel Yöntem:**

Var olan yazılım projeleri incelenir ve belli metrik değerlerinin istatistiksel dağılımı (ortalama "AVG", ortanca değer (*median*), standart sapma "STDEV") belirlenir.

a) Ortalama ve standart sapma kullanılması:

- Eğer metrik en azından aralık (*interval*) ölçeğinde ise ortalama ve standart sapma kullanılabilir.
- Buna göre aşağıdaki sınıflandırma yapılabilir:

  Düşük: AVG - STDEV

  Yüksek: AVG + STDEV

  Çok Yüksek: (AVG + STDEV) · 1.5

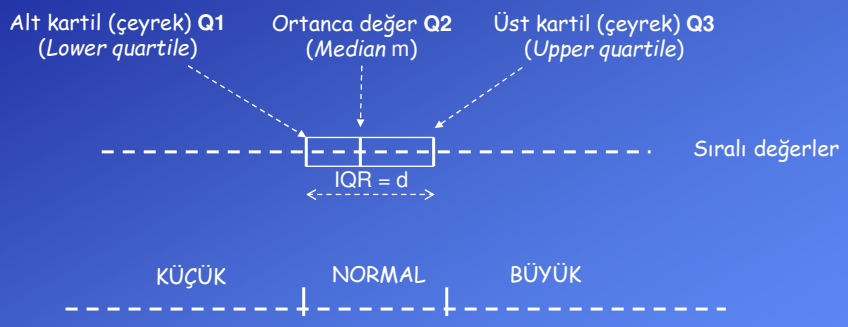
- Ortalamanın kullanıldığı bu tür eşikler daha çok boyutla ilgili ve sayma yoluyla elde edilen metrikler için uygundur.

  Örneğin satır sayısı, metot sayısı gibi.

b) Eşik değerlerinin belirlenmesinde ortanca değer (medyan) kullanımı:

Sırasal ölçekte değer üreten metrikler için ortalama yerine medyan kullanılmalıdır.

Örneğin **Kutu Çizimi (*Box plot*)**:



Q2 (m): Ortanca değer (*median*). Değerler sıralandığında ortadaki elemanın değeri

Q1: Alt çeyrek. Ortanca değerden düşük değerdeki elemanların ortanca değeri

Q3: Üst çeyrek. Ortanca değerden büyük değerdeki elemanların ortanca değeri

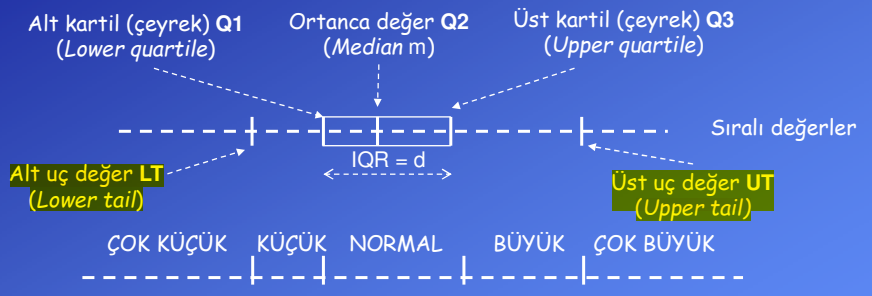
IQR: InterQuartile Range

b) Ortanca değer (medyan) kullanımı (devamı):

En az aralık ölçeğinde değer üreten metrikler için de medyan kullanılabilir.

Böylece aşırı değerlere sahip birimler (*outlier*) değerlendirme dışı bırakılabilir.

Örneğin **Kutu Çizimi (*Box plot*)**:



Q2 (m): Ortanca değer (*median*). Değerler sıralandığında ortadaki elemanın değeri

Q1: Alt çeyrek. Ortanca değerden düşük değerdeki elemanların ortanca değeri

Q3: Üst çeyrek. Ortanca değerden büyük değerdeki elemanların ortanca değeri

LT: Alt uç değer. LT = Q1 - 1.5*d .

UT: Üst uç değer. UT = Q3 + 1.5*d .

Metrik en az aralık ölçeğinde olduğu için bu şekilde aritmetik işlemler yapılabilir.

Kural tabanlı örnek çalışma 1 (M. Lanza, R. Marinescu):

- Bu konuda bir çok çalışma yapılmış ve yayımlanmıştır.
- Ders kapsamında ilk olarak aşağıdaki kitaptaki çalışma ele alınacaktır.

M. Lanza, R. Marinescu, *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer, 2006.

- Bu kitapta tasarım kusurlarına tasarım uyumsuzlukları (*disharmonies*) adı verilmiştir.
- Kitapta 11 adet kusur tanımlanmış ve her birinin belirlenmesi için metrikler kullanan kural tabanlı bir model önerilmiştir.
- Ayrıca bu kusurların düzeltilmesi ile ilgili önerilerde kitapta yer almaktadır.

Eşik değerlerini belirlemek için istatiksel yöntemin kullanılması:

- 45 Java ve 37 C++ projesi incelenerek istatistik değerleri elde edilmiş ve bazı metrikler için eşik değerleri elde edilmiştir.
- Projeler değişik uygulama alanlarından ve değişik boyutlarda (satır sayısı: 20,000 - 2,000,000) seçilmiştir.
- Yazılımların bir kısmı ücretli, ticari yazılımlar, bir kısmı ise açık kaynaklı yazılımlardır.

Tartışma:

- Mantıklı ve kabul edilebilir referans (eşik) değerleri elde etmek için veri toplanacak referans projeler nasıl seçilmeli?
  - Fazla hata içermediği ve güvenilir olduğu düşünülen belli yazılım projeleri mi incelenmeli?
  - Çok hata içeren projeler mi incelenmeli?
  - Yazılımların genel karakterini yansıtan ortalamada uygun değerlere sahip ancak bazı kusurları da olan projeler olmalı.
- Eşik değerleri uygulama alanına bağlı olabilir. Eşik değerlerinin belli alanlardaki projelerde ayrı ayrı toplanması gerekmez mi?

Eşik değerlerini belirlemek için istatiksel yöntemin kullanılması (devamı):

İncelenen metrikler:

- Bir sınıftaki ortalama metot sayısı (NOM/Class)
- Bir metottaki ortalama kod satırı sayısı (LOC/Method)
- Kod satırı başına düşen dallanma noktası (CYCLO/LOC)

Seçilmelerinin nedenleri:

- Projenin boyutu ve karmaşıklığı hakkına bilgi veren temel metriklerdir.
- Birbirlerinden bağımsızdırlar; farklı bilgiler verirler.
- Projenin boyutundan bağımsızdırlar.

İstatistiksel değerler ve eşikler:

- Ortalama (AVG) metriklerin tipik değerlerini verir.
- Standart sapma (STDEV) değerlerin ne kadar dağıldığını gösterir.
  Not: Standart sapma, varyansın kare köküdür.
- Alt sınır: AVG - STDEV
- Yüksek sınır: AVG + STDEV
- Çok Yüksek: (AVG + STDEV)·1.5

Elde edilen eşik değerleri:

	Java				C++			
Metric	Low	Ave-rage	High	Very High	Low	Ave-rage	High	Very High
CYCLO/Line of Code	0.16	0.20	0.24	0.36	0.20	0.25	0.30	0.45
LOC/Method	7	10	13	19.5	5	10	16	24
NOM/Class	4	7	10	15	4	9	15	22.5

Bu eşik değerleri ilgili metriklerden türetilebilen diğer metriklerin eşik değerlerini hesaplamak için de kullanılabilir.

Örnek: CK WMC (*Weighted methods per class*) metriği aşağıdaki gibi hesaplanır:

$$WMC = \frac{CYCLO}{LOC} \cdot \frac{LOC}{Method} \cdot \frac{NOM}{Class}$$

Türetilen eşik değerleri:

	Java				C++			
Metric	Low	Ave-rage	High	Very High	Low	Ave-rage	High	Very High
WMC	5	14	31	47	4	23	72	108
AMW	1.1	2.0	3.1	4.7	1.0	2.5	4.8	7.0
LOC/Class	28	70	130	195	20	90	240	360
NOM/Class	4	7	10	15	4	9	15	23

AMW: *Average Method Weight* *CYCLO /Method veya WMC/NOM*

2. Anlamsal Eşik (Referans) Değerleri:

- Deneyimler (gözlemler) sonucu oluşmuş, genelde kabul gören eşik değerleri.
- Bu değerlere uzamanlar (insanlar) karar verir.

  Örneğin:
  - Üç ve daha az katı olan bina kısadır. 10'dan fazla katı olan bina yüksektir. 20'den fazla katı olan bina çok yüksektir.
  - Yüz metre mesafedeki bir yer yürümek için yakındır. İki kilometre mesafedeki bir yer yürümek için uzaktır.
- Aslında bunlar da uzun süreler içinde yapılan gözlemlerin istatistiksel değerlendirmelerine dayanır.
- Yazılımlar için de çeşitli algılar vardır.

  Örneğin:
  - İç içe 3 veya daha az döngü "normal", daha çoğu "fazla" olarak nitelendirilebilir.
  - Bir metodun üç veya daha az parametresi olması normaldir. Üç, altı arası parametre fazladır. Altıdan çok parametre çok fazladır.
- Yazılım dünyasında bu referans değerler genelde 10'dan küçüktür. Bunları bulmak için istatistiksel analize gerek duyulmamaktadır.

2. **Anlamsal Eşik Değerleri** (devamı):
   - Anlamsal eşikler ikiye ayrılabilir:

A. Genel kabul gören kesirler: 1/2, 1/3, 2/3, 3/4 veya 0.5, 0.33, 0.75 gibi

Değerleri 0 - 1 arası değişen normalize metriklerle çalışıldığında bu tür kesirlerin eşik değeri olarak kullanılması önerilir.

B. Genel kabul gören mutlak değerler:

0 - 7 arası değişen tamsayılar da genel kabul gören anlamlara sahiptirler.

İnsanın kısa süreli belleğinin üst sınırının 7 olduğu gösterilmiştir.

Çalışmada kullanılan bazı anlamsal değerler.

Sayısal değer	Anlam
0	Yok (*None*)
1	Bir/Sığı (*Shallow*)
2 - 5	İki, Üç, Birkaç (*Few, Several*)
7 - 8	Kısa süreli bellek kapasitesi

Yazılımın Uygunluğu (veya Uyumu) (*Harmony*)

Bir nesneye dayalı yazılımın tasarımının uygunluğu, sınıf temelli olarak ele alınan üç uyum (*harmony*) bileşeninden oluşur.

1. Kimlik Uyumu (***Identity Harmony***): Bir birimin (sınıf veya metot) var olma nedeni gerçekçi midir? Belli bir işi var mı? Çok fazla (çok az) iş mi yapıyor?
2. İşbirliği Uyumu (***Collaboration Harmony***): Bir sınıfın diğer sınıflarla etkileşimi nasıldır? Tüm işi kendi mi yapıyor? Diğer sınıflarla çok mu iletişimde bulunuyor?
3. Sınıflandırma (Hiyerarşik) Uyumu (***Classification Harmony***): Bir sınıfın kalıtım/türetim uyumu nasıldır? Kalıtımla aldığı tüm özellikleri kullanıyor mu, çoğunu değiştiriyor mu?

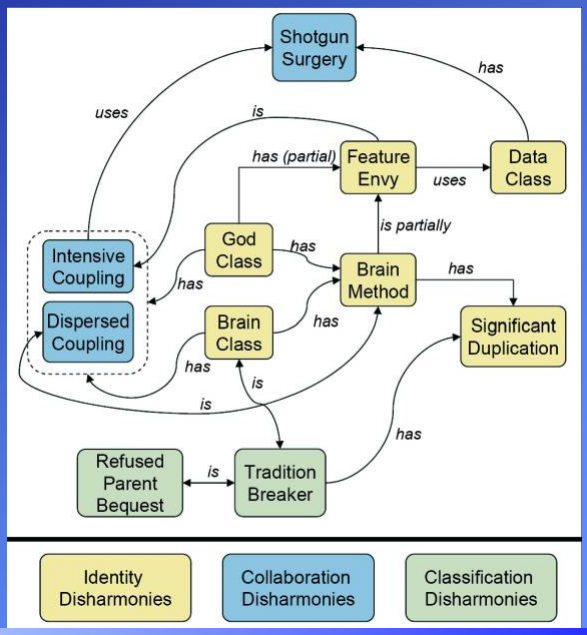
Bir yazılımdaki her birim (örneğin her sınıf);

1. Kendisiyle uyumlu olmalıdır: Çok küçük, çok büyük, çok karmaşık, çok basit olmamalıdır.
2. İşbirliği yaptığı birimlerle uyumlu olmalıdır: Çok fazla birimle iletişimde olmamalı, hiç kimse ile iletişimde bulunmadan her işi kendi yapmamalı.
3. Taban sınıfları (ataları) ve ardıllarıyla uyumlu olmalı.

Örnek çalışmada ele alınan kusurlar ve ilişkileri:

Bu çalışmada kusurlar uyumsuzluk (***disharmony***) olarak adlandırılmıştır.

M. Lanza, R.Marinescu, *Object-oriented metrics in practice : using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer, 2006.



Kimlik Uyumsuzlukları (*Identity Disharmonies*)

- Kimlik uyumsuzlukları tek bir birimi (sınıf, metot) etkilerler.
  Bu nedenle bu tür kusurlara birimlerin bağımsız olarak incelenmesiyle belirlenebilir.
- Bir birimin kimlik uyumunu belirleyen üç bileşen vardır:
  1. Boyut/Orantı (*Proportion*) Bileşeni:
     - Sınıf ve metotların boyutları uygun olmalı (çok küçük ya da çok büyük olmamalı).
     - Yazılımın karmaşıklığı (kod) sadece belli birimlere toplanmamalı.
  2. Sunum (*Presentation*) Bileşeni:
     - Her sınıf kendi kimliğini dışarıya uygun hizmetler içeren bir arayüz üzerinden yansıtmalı.
     - Sınıfın tek bir konuda sorumluluğu olmalı.
     - Sınıftaki metotlar belli bir sorumluluk konusunda yoğunlaşmalı.
     - İşlevsel olmayan (*setter/getter, delegator*) metotların oranı belli bir değeri aşmamalı, sınıfa özgü sorumlukları yerine getiren metotlar da olmalı.
       - Sınıfın bazı metotları veri erişimi için (set/get) kullanılabilir.
       - Ayrıca başka sınıflara görev aktaran metotlar (*delegator*) bulunabilir.

Kimlik uyumunu belirleyen üç bileşen (devamı):

2. Sunum (*Presentation*) Bileşeni (devamı):
   - Sınıfın kendine özgün (diğer birimlerden farklı) davranışı olmalı.
   - Kod (görev) kopyaları oluşmamalı.
   - Veriler ve içişler gizli, hizmetler açık olmalı.
3. Gerçekleme (*Implementation*) Bileşeni:
   - Sınıftaki veriler ve işlemler birbirleri ile ilgili ve sınıfla anlamsal bütünlük içinde olmalı.
   - Tüm metotlar sınıfların niteliklerini büyük oranda kullanmalı.
   - Birbirleriyle ilgileri olmayan veri ve metot kümelerini aynı sınıfta barındırmaktan kaçınılmalı.

Örnek kimlik uyumsuzlukları:

God Class:

- "God Class" çok fazla iş yapar ve diğer sınıfların verilerini kullanır.
- Sistemdeki bir çok sorumluluğun tek bir sınıfa toplanması durumda oluşurlar.
- Tekrar kullanılabilirliği ve anlaşılırlığı azaltır.

"God class" belirlemek için kullanılan metrikler:

- **ATFD** (*Access to Foreign Data*):
  - Bir sınıftan, niteliklerine doğrudan ya da dolaylı olarak erişilen diğer sınıfların sayısı
  - Bu metrik, örnek çalışmanın yazarlarından Radu Marinescu tarafından tanımlanmıştır.

  Radu Marinescu, **Detection Strategies: Metrics-Based Rules for Detecting Design Flaws**, Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM 2004).

  Radu Marinescu, **Detecting Design Flaws via Metrics in Object-Oriented Systems**, Proceedings of 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39), 2001.

"God class" belirlemek için kullanılan metrikler (devamı):

- **WMC** (*Weighted Method Count*):
  CK Metriği . Metotların karmaşıklığı (sayısı) hakkında bilgi verir.
- **TCC** (*Tight Class Cohesion*):
  Bir sınıftaki doğrudan (sıkı) bağlı metot çiftlerinin sayısı, tüm olası metot çiftlerinin sayısına bölünür.
  İki metodun doğrudan (sıkı) bağlı olması, ait oldukları sınıfın en azından bir tane niteliğini ortak kullandıkları anlamına gelir.

  TCC = NDC(C) / NP(C)

  NDC: Number of directly connected methods
  NP: Number of possible pairs

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

  J.M. Bieman and B.K. Kang. "Cohesion and reuse in an object oriented system", In Proceedings ACM Symposium on Software Reusability, April 1995.

"God class" belirlemek için kullanılan kural tabanlı yöntem:

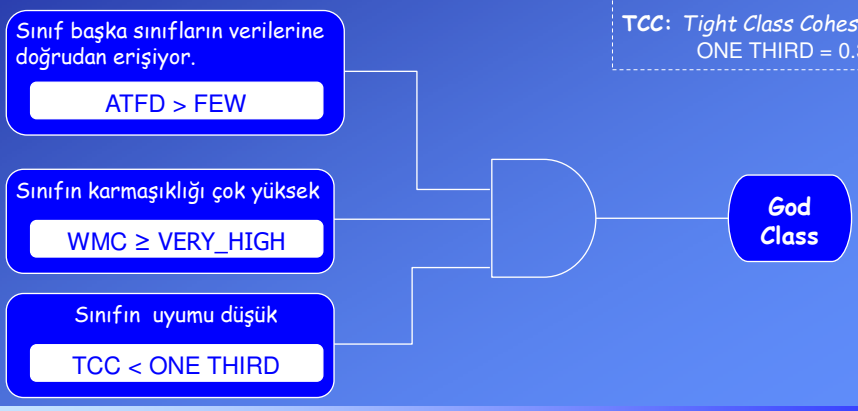
- Eğer sınıf başka sınıfların verilerine erişiyorsa (ATFD) ve
- Çok sayıda (karmaşık) metodu varsa (WMC) ve
- Metot uyumu düşük ise (TCC)

"God class" olmaya adaydır.

ATFD: *Access to Foreign Data*
FEW = 2 - 5

WMC: *Weighted Method Count*
VERY HIGH = 47 (Java)
VERY HIGH = 108 (C++)

TCC: *Tight Class Cohesion*
ONE THIRD = 0.33



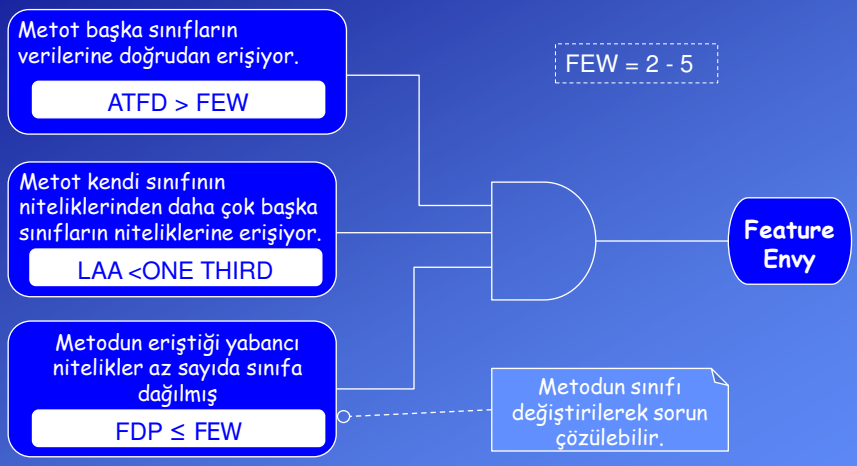
Feature Envy (Başka sınıfların verilerine imrenmek):

- Metotlarla ilgili bir uyumsuzluktur.
- Üyesi olmadığı sınıfların niteliklerine doğrudan veya metotlar üzerinden (get/set) çok erişen metotları ifade eder.
- Metodun yanlış sınıfta olduğunu veya sınıf organizasyonunun kötü yapıldığını gösterir.

Belirlemek için kullanılan metrikler:

- **ATFD** (*Access to Foreign Data*):
  Bir metottan niteliklerine doğrudan ya da dolaylı olarak erişilen diğer sınıfların sayısı. (Buradaki ATFD metodlarla ilgili olup, tanımı "God class" belirlemede kullanılandan farklıdır.)
- **LAA** (*Locality of Attribute Accesses*):
  Metodun kendi sınıfından eriştiği niteliklerin sayısının, erişilen tüm niteliklerin (doğrudan ya da dolaylı) (kendi sınıfı ve başka sınıfa ait) sayısına oranı.
- **FDP** (*Foreign Data Providers*):
  Erişilen yabancı niteliklerin ait olduğu farklı sınıf sayısı.
  "Feature Envy" sorununda bu değer küçüktür. (Daha çok belli bir sınıfa erişiyor.) Bu da metodun yanlış sınıfta olduğuna işaret eder.
  Eğer FDP değeri yüksekse bu metot bir denetçi (*controller*) ya da "brain method" olabilir.

"Feature Envy" belirlemek için kullanılan yöntem:



ATFD (*Access to Foreign Data*) Metot için tanımlı
LAA (*Locality of Attribute Accesses*)
FDP (*Foreign Data Providers*)

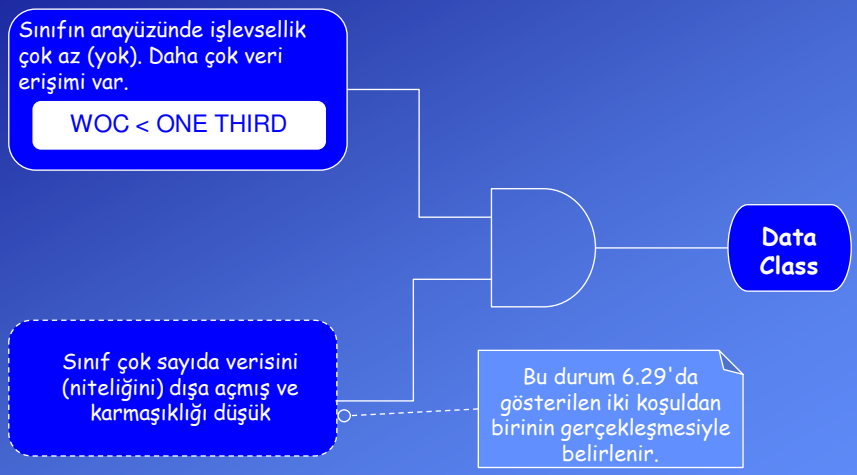
Data Class:

- Bu tür sınıflar diğer sınıfların çok eriştiği verilere sahiptirler ancak kendileri bu veriler üzerinde işlem yapmazlar.
- İlgili verilerin ve fonksiyonların birlikte tasarlanmadığını (*encapsulation problem*) göstergesi olabilir.

Belirlemek için kullanılan metrikler:

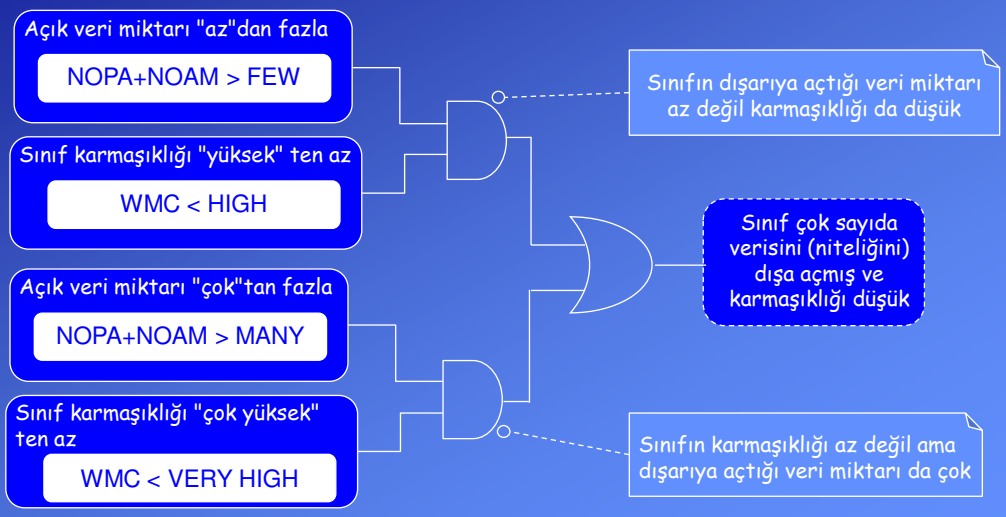
- **WOC** (*Weight Of Class*):
  - Bir sınıfın ara yüzünde yer alan (*public*), sırf erişim amaçlı olmayan (*non-accessor*) metotların sınıftaki tüm açık (*public*) metotlara oranı.
  - Sınıfın belli bir hizmet vermesi için bu değerin 1.0'a yakın olması tercih edilir.
  - Bu değerin küçük olması (WOC<0.33) sınıfın veri sınıfı olduğu şüphesi uyandırır.
- **NOPA** (*Number Of Public Attributes*):
  Verilerin (niteliklerin) açık (*public*) olması istenmeyen bir özelliktir.
- **NOAM** (*Number Of Accessor Methods*):
  Erişim (*accessor*) metotlarının sayısı

"Data Class" belirlemek için kullanılan yöntem:



"Sınıf çok sayıda verisini (niteliğini) dışa açmış ve karmaşıklığı düşük" koşulunun belirlenmesi:

Bir sınıf dış erişime açtığı veri miktarının kendi karmaşıklığı (yaptığı iş miktarı) ile uyumlu (orantılı) olması beklenir.



İşbirliği Uyumsuzlukları (*Collaboration Disharmonies*)

Nesneye dayalı tasarımın sınıflar arası işbirliği konusundaki önerileri:

- Bir sınıf iyi tanımlanmış bir sorumluluğu olmalı, kendisi ile ilgili olmayan işleri başka sınıflardan hizmet alarak (işbirliği ile) (*delegation*) yerine getirmeli.
- İyi bir nesneye dayalı yazılımda sınıflar arası işbirliği metotlar üzerinden olmalı.
- Bir metot (sınıf) başka sınıflardan sınırlı sayıda hizmet almalı.
- Bir metot (sınıf) sınırlı sayıda yabancı sınıfa bağımlı olmalı.
- Bir metodun bağımlı olduğu (hizmet aldığı) diğer işlemlerin yeri aşağıdaki sıraya göre tercih edilir.
  (a) Aynı sınıf içinde, (b) aynı türetim zincirinde, (c) aynı pakette veya alt sistemde.
- Sınıflar (metotlar) arası bağımlılık zincirleme olarak çok yayılmamalı.

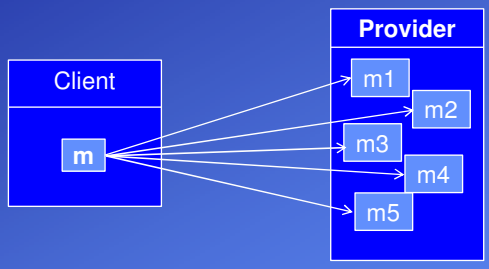
Kitapta tanımlanan uyumsuzluklar:

Intensive Coupling, Dispersed Coupling, Shotgun Surgery

M. Lanza, R.Marinescu, *Object-oriented metrics in practice : using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer, 2006.

Intensive Coupling (Yoğun Bağımlılık):

- Bir metot, çok sayıda başka metodu çağırmaktadır ve
- bu yabancı metotlar aynı sınıfta (veya çok az sayıda sınıfta) toplanmıştır.



Kusurun belirlenmesi:

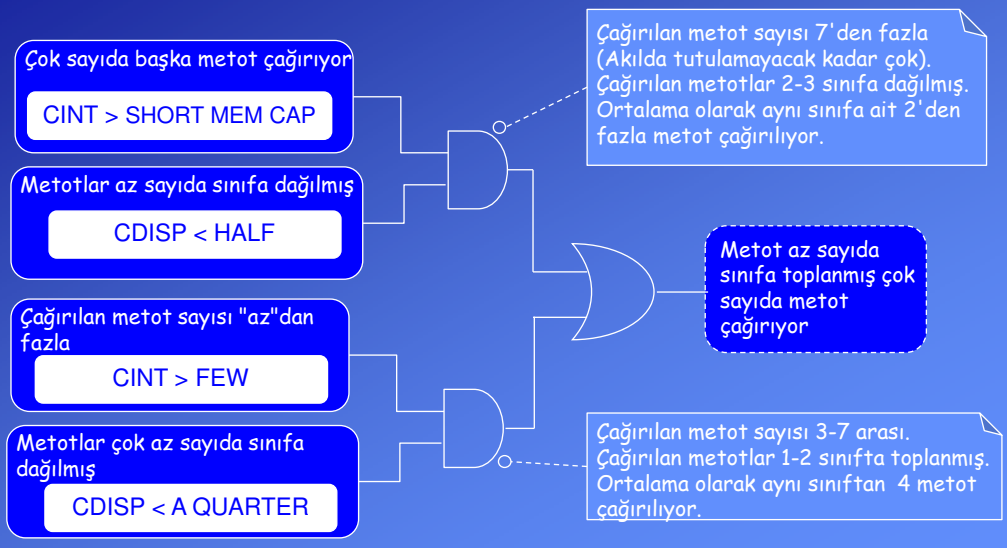
- İki koşulun geçerliliği sınanacak.
  1. Metot az sayıda başka sınıfa toplanmış çok sayıda metot çağırıyor mu?
  2. Metodun karmaşıklığı "az" değil mi ("az"dan fazla mı)?
     İkinci koşulun nedeni bilerek yazılmış bazı zararsız metotları elemektir.
     Bu metotlar belli sınıfları uygun şekilde başlatmak ya da koşullamak için yazılmış olabilir.

"Intensive Coupling" belirlemek için kullanılan yöntem:

Kullanılan metrikler:

- **CINT** (*Coupling Intensity*):Bir metottan çağrılan farklı metotların sayısı
- **CDISP** (*Coupling Dispersion*):
  Bir metottan çağrılan diğer metotların ait olduğu sınıfların sayısı CINT'e oranı
  Bağımlılığın farklı sınıflara nasıl yayıldığını gösterir.
  Bu değerin küçük olması bağımlılığın daha çok aynı sınıfa olduğu anlamına gelir.

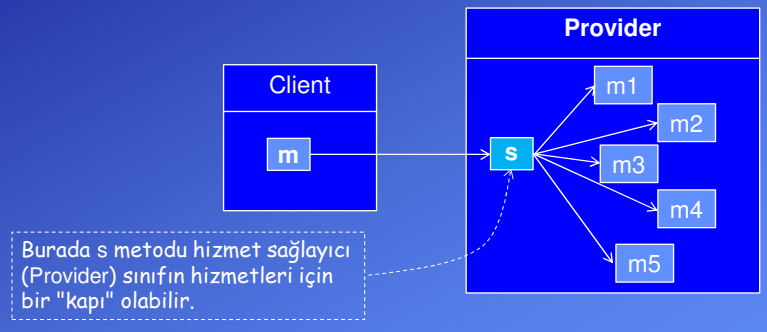
"Metot az sayıda sınıfa toplanmış çok sayıda metot çağırıyor" koşulunun belirlenmesi:



"Intensive Coupling" sorununu gidermek için olası düzenlemeler (*refactoring*):

A. "Intensive Coupling" sorunu bir metodun yanlış sınıfta olmasından kaynaklanabilir.
Bu durumda metodun yerini veya sınıfların sorumluluklarını değiştirmek gerekir.

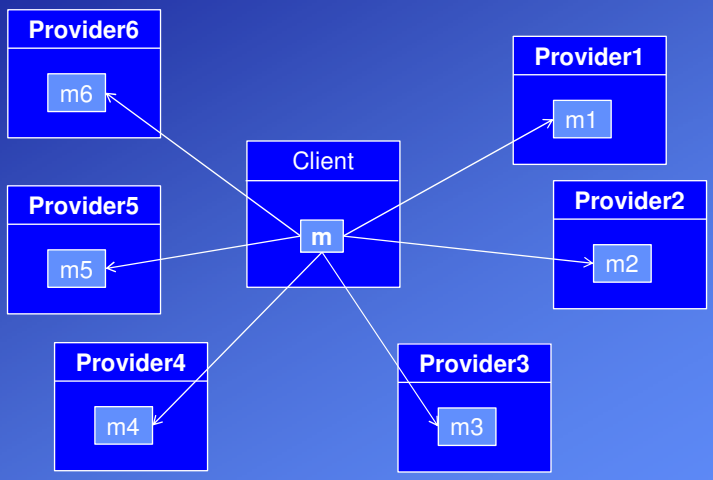
B. Bazı durumlarda ise sorun hizmet sağlayan sınıfın yapısından kaynaklanır.
Bu durumda aşağıdaki düzenleme yapılabilir:



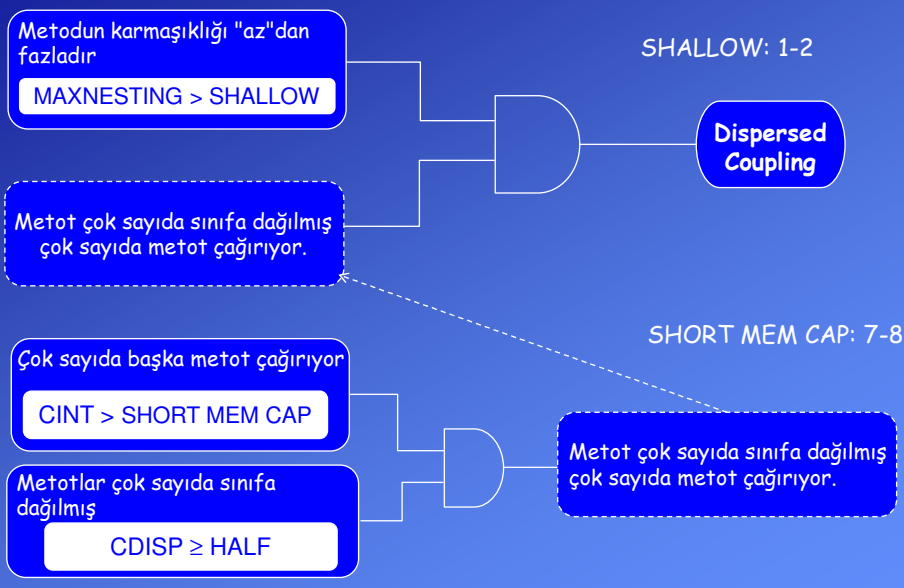
- Hizmet odaklı (*service-oriented*) yapılarda sınıfların (hizmetlerin) bu şekilde tasarlanması gerekir.
- Diğer bir çözüm ise araya bir cephe sınıfı (*Facade*) koymak olabilir.

Dispersed Coupling (Dağılmış Bağımlılık) :

- Bir metot çok sayıda başka metodu çağırmaktadır ve bu yabancı metotlar çok sayıda farklı sınıfa dağılmışlardır.



"Dispersed Coupling" belirlemek için kullanılan yöntem:



Sınıflandırma (Hiyerarşi) Uyumsuzlukları (*Classification Disharmonies*)

Nesneye dayalı tasarımda kalıtımın (türetimin) iki temel amacı vardır:

1. Daha genel yapılardan daha özel yapılar türetmek.
   Özel yapılar oluşturulurken genel yapıların ortak özellikleri tekrar kullanılmış (*reusability*) olur.
2. Aynı arayüze (*interface*) sahip sınıflar oluşturmak.
   Bu sınıflar ortak bazı sorumlulukları yerine getirirler ve birbirlerinin yerine geçebilirler (Örneğin GoF stratejileri).

Ancak türetimin yanlış kullanılması tasarım kusurlarına neden olabilir.

Özellikle tekrar kullanımı arttırmak için sadece kalıtımın (türetimin) kullanılması, sahip olma ilişkisinin göz ardı edilmesi sorunlara neden olur.

Uygun kalıtım (türetim) ile ilgili kurallar:

- Türetim ağaçlarının boyutları uygun olmalı (fazla geniş, fazla uzun değil).
  - Türetim zincirlerinin hiç bulunmaması da sorun göstergesi olabilir.
  - Çok geniş ağaçlar alt sınıfların kopyala yapıştır yöntemiyle yaratıldığının işareti olabilir.
  - Çok derin (uzun) ağaçlar yazılımın anlaşılmasını ve bakımını zorlaştırır.
- Sınıflar türetim zinciri içinde kendilerinden önce ve sonra gelen sınıflar ile uyum içinde olmalıdır.
  - Üst sınıflardan alınan ve yeniden tanımlanan üyelerin oranı uygun olmalı.
  - Üst sınıfın üyelerinin büyük çoğunluğu reddedilmemeli.
- Üst (taban) sınıflar kendilerinden sonra gelen (türetilen) sınıflara bağlı olmamalı.
- Alt sınıf (türeyen), üst sınıfın metotlarını sadece çağırarak kullanmamalı.
  - Bu tür kullanım "has-a" ilişkisi için daha uygun olur.
  - Alt sınıf üst sınıfın metotlarını yeniden tanımlamalı, daha özel metotlar yaratmak için gerektiğinde çağırmalı.

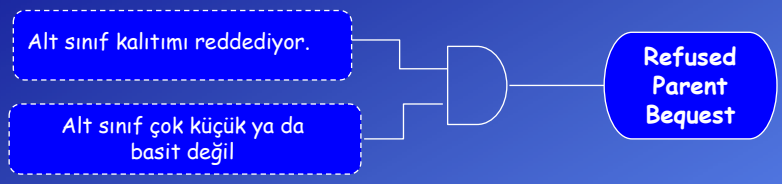
Refused Parent Bequest (Kalıtımın Reddi):

- Türeyen sınıf üst sınıftan gelen kalıtımdan (ortak üyeler) yararlanmalıdır.
- Kalıtım (*inheritance*) özelliğinin temel amaçlarından biri tekrar kullanımdır (*reusability*) diğeri ise ortak ara yüz oluşturmaktır.
- Eğer alt sınıf üst sınıftan gelen özelliklerden yararlanmıyorsa türetim ağacının yapısında sorun olduğu şüphesi oluşur.

Kusurun belirlenmesinde varsayımlar:

1. İncelenen sınıfın üst (taban) sınıfı vardır.
2. Üst sınıf üçüncü parti bir sınıf değildir (örneğin bir arşiv sınıfı), veya bir ara yüz (*interface* "Java") değildir.

Kalıtım reddi uyumsuzluğunun belirlenmesi için iki koşulun oluşması gerekir:



Bir alt sınıfın üst sınıftan gelen kalıtımı kullanması üç şekilde olur:

- Üst sınıfın korunan (*protected*) niteliklerine (verilerine) erişir.
- Üst sınıfın korunan (*protected*) metotlarını çağırır.
- Üst sınıfın metotlarını örterek günceller (*overriding*).

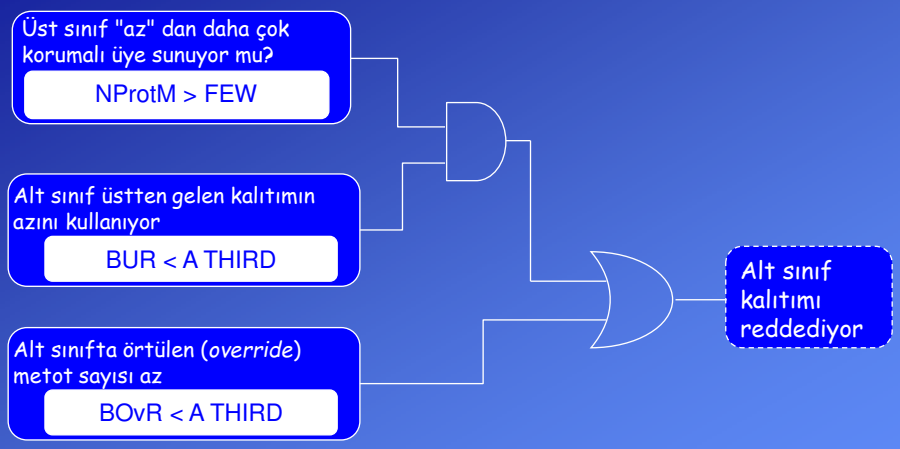
Bunu ölçmek için kullanılan metrikler:

BUR (Base class Usage Ratio) : Alt sınıfta erişilen korunan (*protected*) üyelerin oranı.

BOvR (Base-class Overriding Ratio): Örtülen ve güncellenen üst sınıf metot oranı

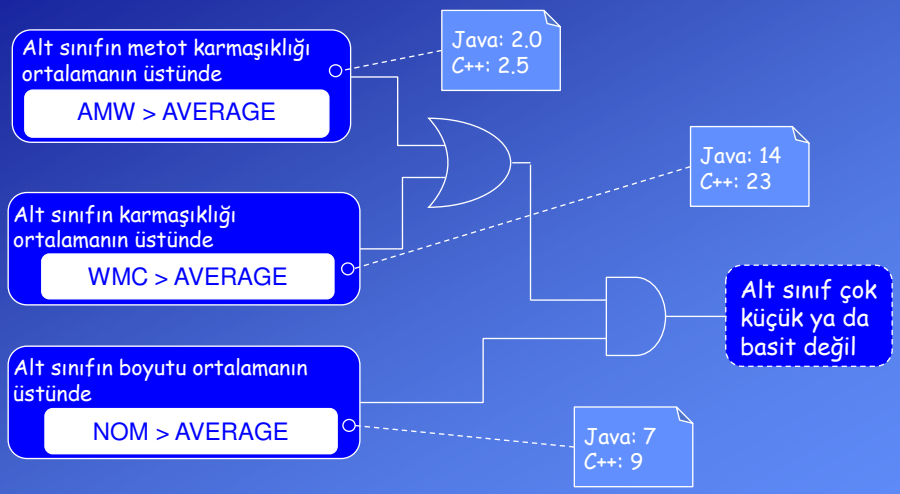
NPrM (Number of Protected Members): Üst sınıftaki korunan (*protected*) üye sayısı

1. Koşul: Üst sınıftan gelen kalıtım göz ardı ediliyor mu?



NProtM (Number of Protected Members): Üst sınıftaki korunan (*protected*) üye sayısı
BUR (Base class Usage Ratio) : Alt sınıfta erişilen korunan üyelerin oranı.
BOvR (Base-class Overriding Ratio): Örtülen ve güncellenen üst sınıf metot oranı

2. Koşul: Alt sınıf yeteri kadar büyük mü?

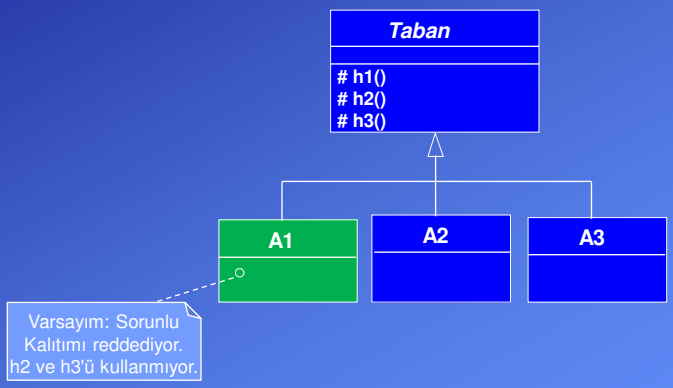


AMW: *Average Method Weight* *WMC/NOM*
WMC: *Weighted Method Count*
NOM: *Number Of Methods*

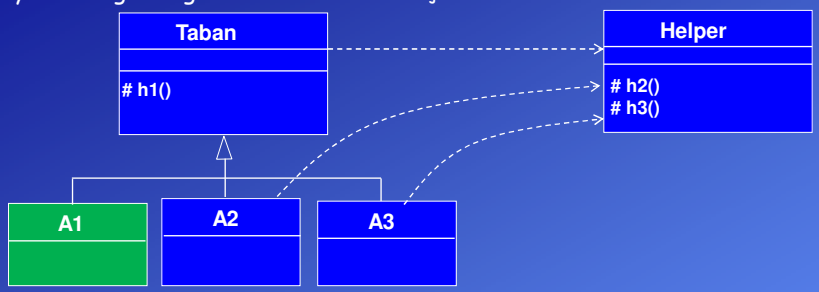
Kalıtım reddi uyumsuzluğunun giderilmesi:

- Bu sorunun oluşmasının çeşitli nedenleri olabilir.
- Eğer türetim zinciri hatalıysa yapı yeniden oluşturulmalıdır.
- Diğer bir neden ise taban sınıfın çok sayıda alt sınıfı olması ve taban sınıftaki bazı özelliklerin bazı sınıfları çok ilgilendirmemesi.

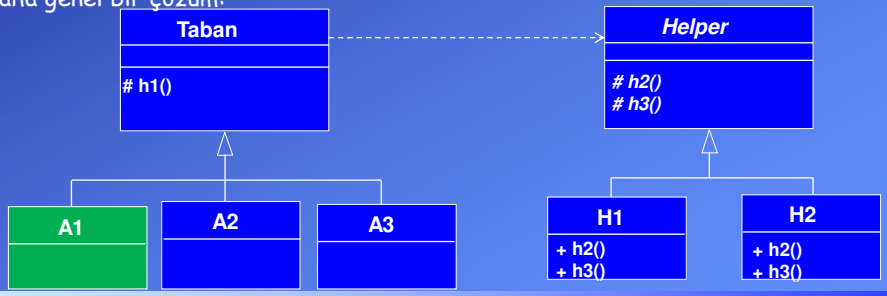
  Bu durumda GoF strateji (veya köprü) kalıbı kullanılarak sorun çözülebilir.



Kalıtım reddi uyumsuzluğunun giderilmesi: "has-a" ilişkisinin kullanılması



Daha genel bir çözüm:



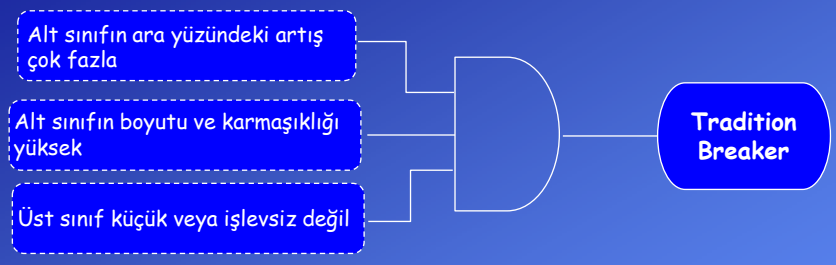
Tradition Breaker (Geleneğin Sürdürülmemesi):

- Normalde bir sınıfın ara yüzü (dışarıya verdiği hizmetler) evrimsel olarak alt sınıflarda güncellenir ve genişletilir.
- Alt sınıf üst sınıfın "geleneğini sürdürmeli", bir anda yukarıdan aldığı tüm hizmetleri değiştirip tamamen farklı hizmetler sunmamalı.

Kusurun belirlenmesinde varsayımlar:

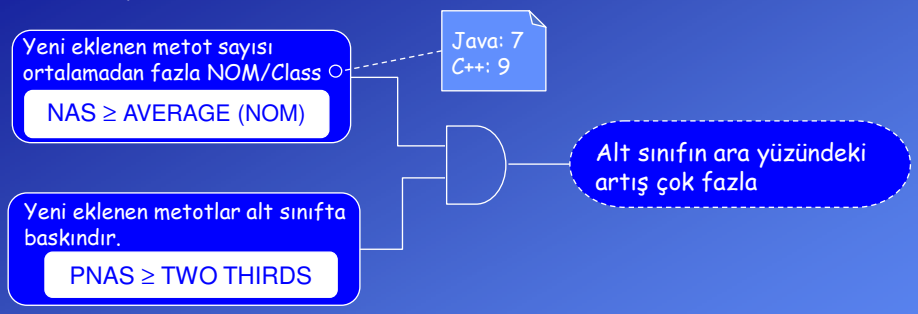
1. İncelenen sınıfın üst (taban) sınıfı vardır.
2. Üst sınıf üçüncü parti bir sınıf değildir (örneğin bir arşiv sınıfı), veya bir ara yüz (*interface* "Java") değildir.

Tradition Breaker uyumsuzluğunun belirlenmesi için üç koşulun oluşması gerekir:



1. Alt sınıfın ara yüzü, üst sınıfla karşılaştırıldığında çok büyümüştür.
2. Alt sınıf tek başına da büyük ve karmaşıktır.
3. Üst sınıfta yeteri kadar işlev vardır. Aksi durumda bir "gelenekten" söz edilemez.

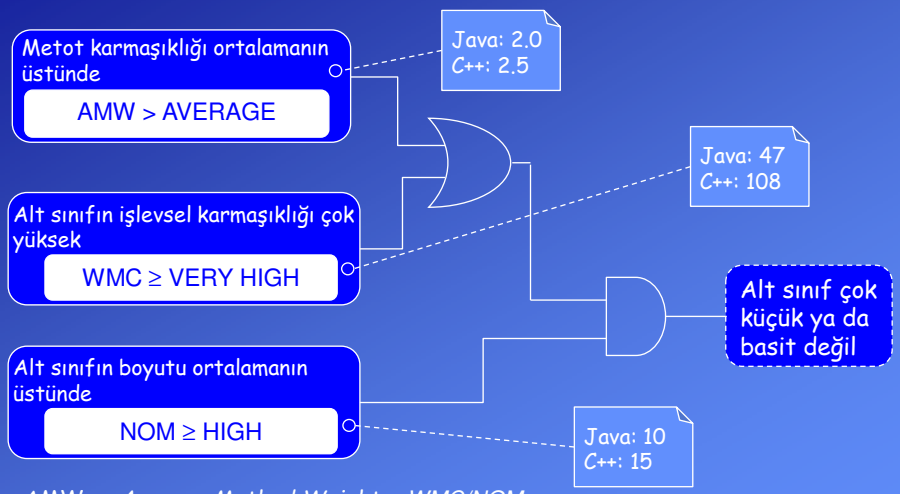
1. Koşul: Alt sınıfın ara yüzündeki artış çok fazla



NAS (*Number of Added Services*) : Alt sınıfta yeni eklenen "public" metotların sayısı. Bunlar üst sınıfta olmayan metotlardır.

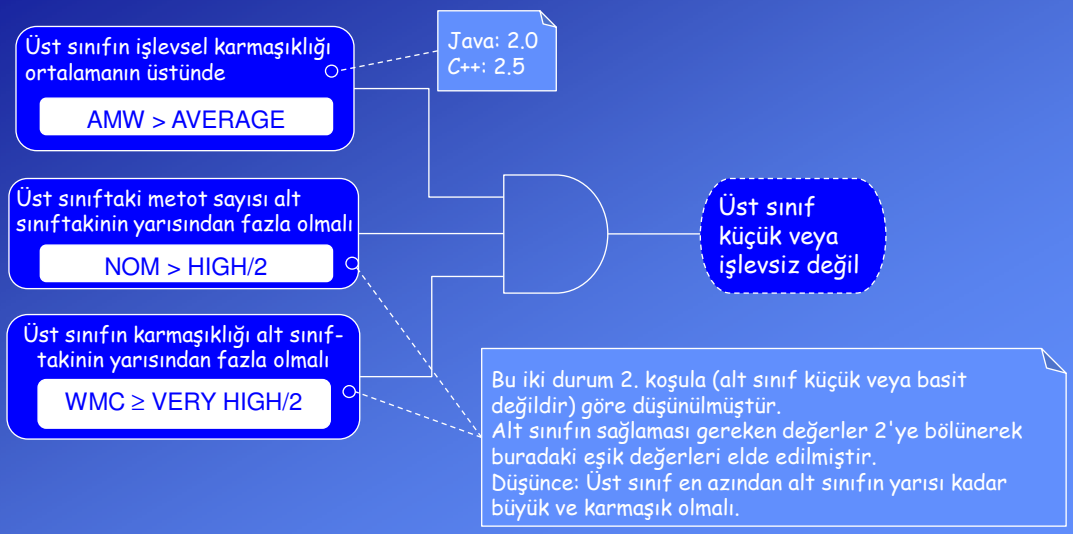
PNAS (*Percentage of Newly Added Services*): Alt sınıfta eklenen "public" metotların sayısının toplam "public" metotların sayısına oranı

2. Koşul: Alt sınıf yeteri kadar büyük ve karmaşıktır



AMW: Average Method Weight WMC/NOM
WMC: Weighted Method Count
NOM: Number Of Methods

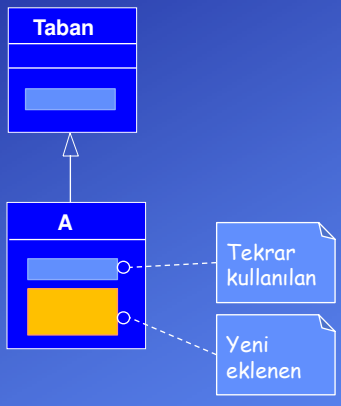
3. Koşul: Üst sınıf küçük veya işlevsiz değil



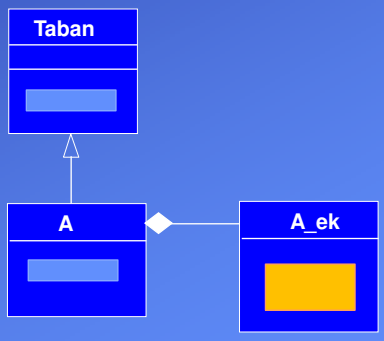
Geleneğin sürdürülmemesi sorununun giderilmesi:

Bu sorunun oluşmasının da temel nedeni türetim zincirinin hatalı kurulması olabilir.
Çözüm yöntemlerinden biri gerekli yerlerde "has-a" ilişkisinin kullanılmasıdır.

Sorun:



Düzenleme:



Kural tabanlı örnek çalışma 2:

Rahime Yılmaz, Feza Buzluca, "*A fuzzy logic-based quality model for identifying microservices with low maintainability*", Journal of Systems and Software, 2024.

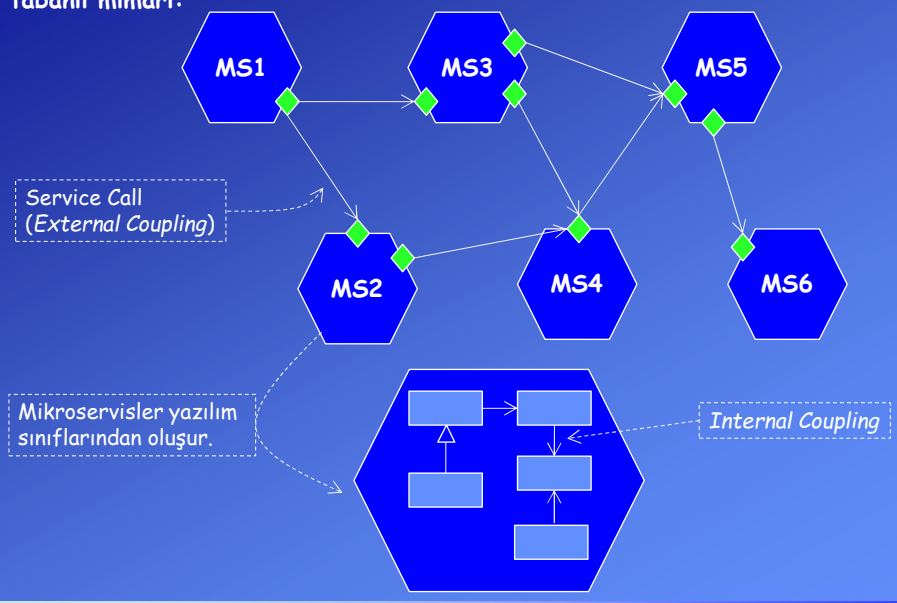
Amaç:

- Sürdürülebilirliği (*maintainability*) kabul edilebilir düzeyin altında olan mikroservisleri kod/tasarım metriklerini kullanarak öngörmek (dolaylı olarak ölçmek).

Geliştirilen yaklaşım ve yapılanlar:

- Üst düzey kalite karakteristiği olan sürdürülebilirliği doğrudan ölçülebilen yazılım niteliklerine bağlayan hiyerarşik bir model oluşturulmuştur.
- Yazılım niteliklerine nicel değerler atamak için metrikler belirlenmiştir.
- Metrikleri "düşük", "orta", "yüksek" olarak sınıflandırabilmek için referans (eşik) değerleri istatistiksel olarak belirlenmiştir.
- Metrikleri sınıflandıran eşik değerleri gerçek yazılımlarda çok keskin olamayacağından sınıflandırmada bulanık mantık (*fuzzy logic*) kullanılmıştır.
  Bulanık mantıkta bir metrik değeri aynı anda birden fazla sınıfa ("düşük", "orta", "yüksek") ait olabilmektedir.
- Bulanık mantık kullanımı modelin sonuçların başarımını artırmaktadır.

Mikroservis tabanlı mimari:



Oluşturulan hiyerarşik model:

Ana karakteristik:

- Sürdürülebilirlik (*maintainability*)

Alt karakteristikler:

- Değiştirilebilirlik (*modifiability*)
- Test edilebilirlik (*testability*)

Mikroservislerin özellikleri:

- İç bağımlılık (*internal coupling*)
- Dış (hizmetler arası) bağımlılık (*service coupling*)
- Boyut (*size*)
- Karmaşıklık (*complexity*)

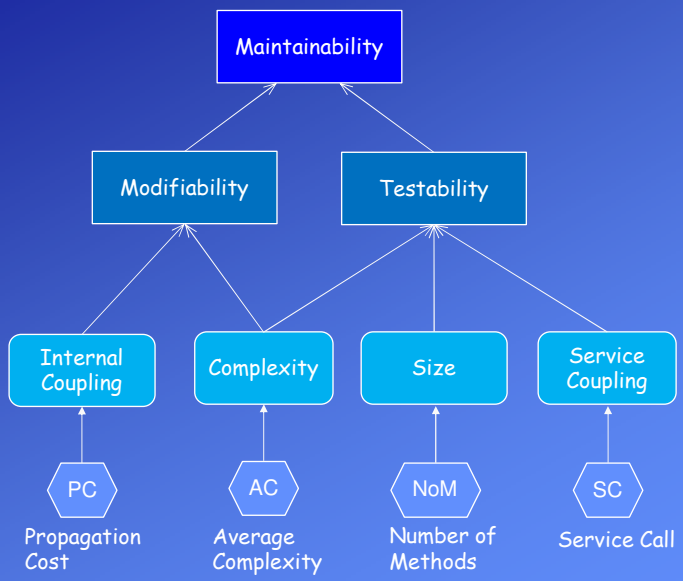
Metrikler:

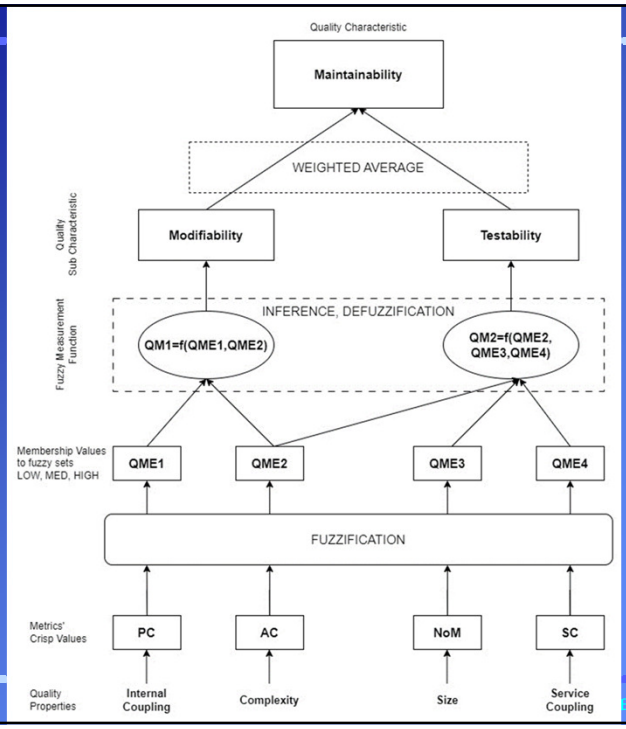
- Propagation Cost (PC) for internal coupling
- Service Call (SC) for service coupling
- Average Complexity (AC) for complexity
- Number of Methods (NoM) for size

Ana karakteristik ve alt karakteristikler ISO 25010 standardı temel alınarak belirlenmiştir.

Alt karakteristikleri etkileyen mikroservis özellikleri ve onlara nicel değerler atanmasını sağlayan metrikler, deneyimle ve GQM yöntemi ile belirlenmiştir.

Oluşturulan hiyerarşik model:





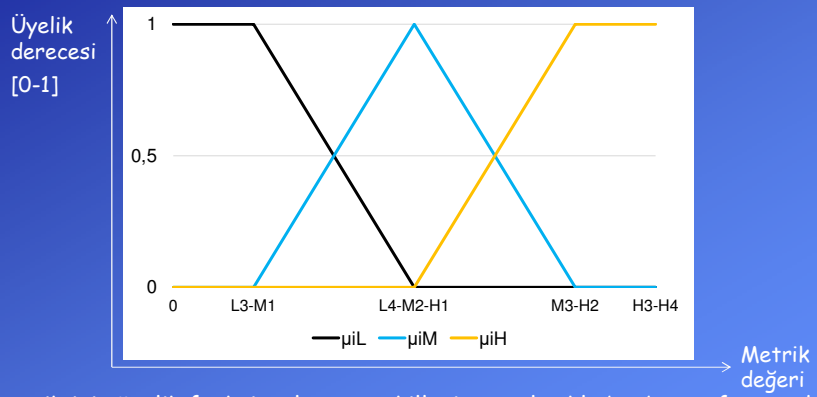
Bulanık mantık tabanlı ölçme sistemi:

- Standart mantıkta (lojikte) bir değer sadece bir kümede (High/Low) yer alırken bulanık mantıkta bir değer aynı anda farklı kümelerde (belli derecelerde olmak üzere) yer alabilir.
- Bulanık mantık sistemleri üç aşamadan oluşur:

1. Fuzzification:
   - İnsan dili ile ifade edilebilen belli sayıda değerlendirme kümesi seçilir.
     Örneğin; LOW, MEDIUM, HIGH. Farklı sayılarda küme olabilir. Örneğin beş küme.
   - Bir metrik değerinin hangi kümeye denk düşeceğini belirleyen üyelik fonksiyonları (*membership function*) oluşturulur.
   - Bulanık mantıkta bir metrik değeri aynı anda birden fazla kümeye girebilir.
   - Üyelik fonksiyonu, bir değerin farklı kümlere ait olma derecesini belirler.
2. Inference Rules:
   - Ölçülmek istenen kalite niteliğinin düzeyi metriklerin düzeyleri cinsinden **lojik kurallarla** ifade edilir.
     Örneğin; M1 = LOW and M2 = LOW then R = HIGH
   - Bir metrik değeri aynı anda birden fazla kümede yer alabildiğinden aynı metrik değerleri birden fazla kuralı sağlayabilirler.
   - Aynı anda geçerli olan ve farklı sonuçlar üretebilen kuralların uygun şekilde birleştirilmesi gerekir.
3. Deffuzification: Kurallardan elde edilen düzeyler kesin sayısal değerlere dönüştürülür.

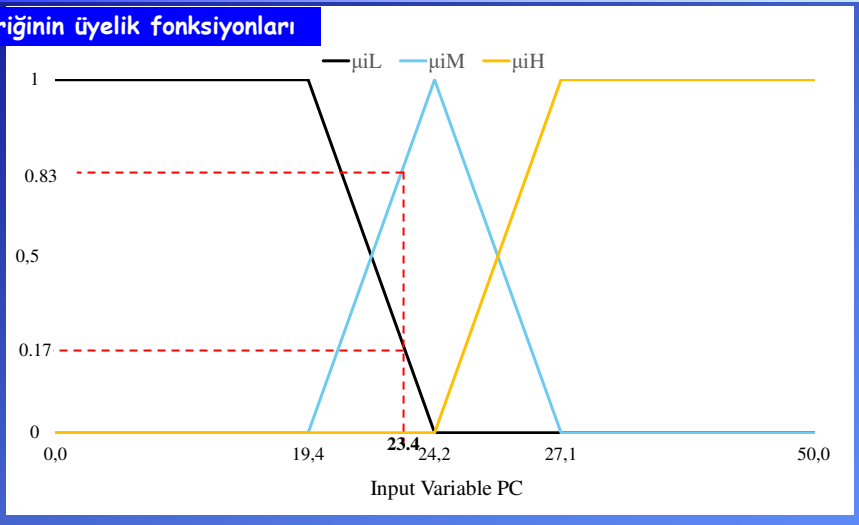
Fuzzification:

- Metrik değerleri üç kümeye ayrılmıştır: LOW, MEDIUM, HIGH
- Bir metrik değerinin hangi kümeye, hangi derecede ait olduğunu belirleyen üyelik fonksiyonları (*membership function*) oluşturulmuştur.
- Her küme (L, M, H) için ayrı bir üyelik fonksiyonu (μ_L, μ_M, μ_H) vardır.



- Her metrik için üyelik fonksiyonlarının şekilleri aynı olmakla berber referans değerleri (M1, M2, M3 gibi) farklıdır.

Örnek: PC metriğinin üyelik fonksiyonları



- PC = 23.4 olması PC'nin 0.17 derecesinde LOW, 0.83 derecesinde MEDIUM, 0 derecesinde HIGH olduğu anlamına gelir.
- Fonksiyonların şekilleri belirlenirken bir metrik değerinin tüm üyelik derecelerinin toplamının 1 olması sağlanır.

$$\mu_L^i(x) + \mu_M^i(x) + \mu_H^i(x) = 1\ ,\ \forall\ i \in \{PC, AC, NoM, SC\}$$

The trapezoidal membership function $\mu_L^i(x)$ for the fuzzy set "LOW":

$$\mu_L^i(x) = \begin{cases} 1 \ , & L1^i = L2^i = 0 \leq x \leq L3^i \\ \dfrac{(L4^i - x)}{(L4^i - L3^i)}, & L3^i < x \leq L4^i \\ 0 \ , & x > L4^i \end{cases}$$

The triangular membership function $\mu_M^i(x)$ for the fuzzy set MED":

$$\mu_M^i(x) = \begin{cases} \dfrac{(x - M1^i)}{(M2^i - M1^i)}, & M1^i \leq x \leq M2^i \\ \dfrac{(M3^i - x)}{(M3^i - M2^i)}, & M2^i < x \leq M3^i \\ 0 \ , & x < M1^i \text{ or } x > M3^i \end{cases}$$

Her metrik (i = PC, AC, NoM, SC) için üyelik fonksiyonlarının şekilleri (fonksiyon ifadeleri) aynı olmakla berber referans değerleri (L, M, H) farklıdır.

The trapezoidal membership function $\mu_H^i(x)$ for the fuzzy set "HIGH":

$$\mu_H^i(x) = \begin{cases} 0 \ , & H1^i \leq x \\ \dfrac{(x - H1^i)}{(H2^i - H1^i)}, & H1^i < x \leq H2^i \\ 1 \ , & H2^i < x \leq H3^i = H4^i \end{cases}$$

$$\mu_L^i(x) + \mu_M^i(x) + \mu_H^i(x) = 1 \ , \ \forall \text{ i} \in \{\text{PC, AC, NoM, SC}\}$$

Referans (eşik) değerlerin belirlenmesi:

- Referans değerleri açık kaynaklı referans projelerden elde edilen istatistiksel değerler ile belirlenmiştir.
- First quartile (Q1), median (Q2), third quartile (Q3) kullanılmıştır.

	L1=L2	L3=M1	L4=M2=H1	M3=H2	H3=H4
SC	0	0.125	0.25	0.5	1
PC	0	19.4	24.2	27.1	50
AC	0	2.81	4.78	5.63	11
NoM	0	9	16	30	90
SC	0	0.125	0.25	0.5	1

Inference Rules:

- Her bir alt karakteristik için üç kümeli karar kuralları belirlenmiştir.
- Kurallar deneyimle, yazılım dünyasında bilenen bilgilere dayanılarak oluşturulmuştur.

  Örnek:

  IF internal coupling (PC) = HIGH AND Complexity (AC) = HIGH THEN Modifiability = LOW

- Modifiability için oluşturulan kurallar:

Rule	Internal Coupling (PC)	Complexity (AC)	Modifiability
RM1	LOW	LOW	HIGH
RM2	LOW	MED	HIGH
RM3	LOW	HIGH	MED
RM4	MED	LOW	HIGH
RM5	MED	MED	MED
RM6	MED	HIGH	LOW
RM7	HIGH	LOW	MED
RM8	HIGH	MED	LOW
RM9	HIGH	HIGH	LOW

- Benzer şekilde Testability için de kurallar oluşturulmuştur.

Değerlerin aynı anda sağladığı (uyduğu) kuralların birleştirilmesi:

- Bir metrik değeri birden fazla kümeye üye olabildiğinden aynı metrik değerleri birden fazla kuralı sağlayabilmektedir.
- Sağlanan kuralları birleştirmek için *Mamdani's Max-Min Inference Method* kullanılmıştır.

Örnek: PC = 23.4 ve AC = 6.53 için Modifiability değeri

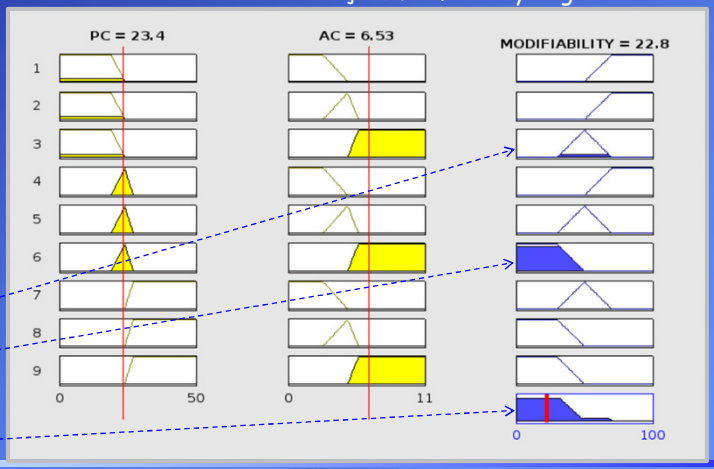
PC = 23.4 ve AC = 6.53 için Modifiability değeri

PC = 23.4
- 0.17 L
- 0.83 M
- 0 H

AC = 6.53
- 0 L
- 0 M
- 1 H

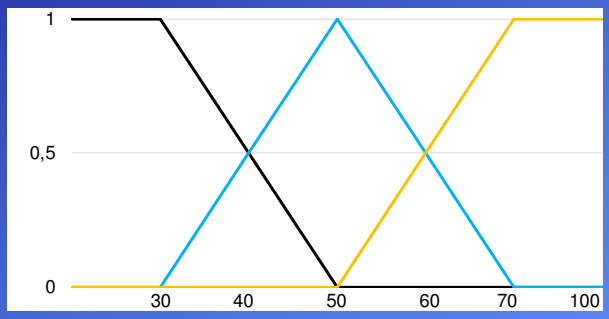
Sağlanan kurallar:
- 0.17 RM3
- 0.83 RM6

Sonuç:
- İkisinin birleşimi



Defuzzification:

- Kuralların sonuçları yeni üyelik fonksiyonları ile sayısal değerlere dönüştürülür.
- Aynı anda birden fazla kural geçerli ise "*centroid defuzzification*" yöntemi ile birleştirilen sonucun ağırlık noktası bulunur.
- Ağırlık noktasına ilişkin sayısal değer sistemin ürettiği sonuç olur.
- Bu yöntemle her mikroservis için modifiability ve testability değerleri elde edilir.



Score	Meaning
0 - 30	Low
30 - 40	Low - Medium, closer to Low
40 - 50	Low - Medium, closer to Medium
50	Medium
50 - 60	Medium - High closer to Medium
60 - 70	Medium - High closer to High
70 - 100	High

Sürdürülebilirlik (*maintainability*) hesabı ve değerlendirme :

- Sürdürülebilirlik, iki alt karakteristiğin ağırlıklı ortalaması hesaplanarak elde edilir:

$$MNT_m = w_{mod} \times MOD_m + w_{tst} \times TST_m$$

MNT_m : m mikroservisinin sürdürebilirlik değeri.
MOD_m : m mikroservisinin değiştirilebilirlik değeri
w_{mod} : Değiştirilebilirlik alt karakteristiğinin ağırlığı
TST_m : m mikroservisinin test edilebilirlik değeri
w_{tst} : Test edilebilirlik alt karakteristiğinin ağırlığı

- Bu çalışmada ağırlıklar eşit olarak seçilmiştir. Yöntemi kullanan yazılım geliştirme takımları alt karakteristiklerin projedeki önemine göre farklı ağırlıklar seçebilirler.

Düzeltilmesi (*refactoring*) gereken mikroservislerin belirlenmesi:

- MNT_m değeri önceden belirlen bir eşik değerinin (T_{REF}) altında olan mikroservislerin düzeltilmesi gerektiği kararı verilmektedir.

  If $MNT_m \leq T_{REF}$, *the microservice m needs refactoring.*

- Bu çalışmada $T_{REF} = 40$ olarak seçilmiştir.

  Bir mikroservis için elde edilen sürdürülebilirlik puanının 40'tan küçük olması bu karakteristiğin MEDIUM'dan az, LOW'a yakın olduğunu gösterir.

- Yöntem kullanılırken T_{REF} değeri farklı (örneğin 30 veya 50) seçilerek daha az veya daha çok sayıda mikroservisin düzeltilmesine karar verilebilir.

Deneyler ve Sonuçlar:

- Geliştirilen yöntem açık kaynaklı bir proje olan Train Ticket üzerinde denenmiştir.
- Train Ticket projesinde yer alan 36 adet mikroservis üç deneyimli yazılım uzmanına önceden LOW, MED ve HIGH olarak etiketlettirilmiştir.
  - LOW: Sorunlu, düzeltilmeli
  - MED: Ufak sorunlar var ama düzeltmeye gerek yok
  - HIGH: Sorunsuz
- Uzmanların LOW olarak etiketlediklerini önerilen yöntemin bulma başarısı değerlendirilmiştir.

		Evaluators	
		LOW	Not LOW
Predicted (Model)	LOW	TP = 7	FP = 2
	Not LOW	FN = 0	TN = 27
	Total	7	29

Recall = TP / (TP + TN)
Precision = TP / (TP + FP)
F-Measure = 2 * Precision * Recall / (Precision + Recall)
Accuracy = (TP + TN) /(TP + TN + FP + FN)

Recall	Precision	F-Measure	Accuracy
100%	77.78%	87.5%	94.44%

Label by the Evaluators	Average	Standard Deviation
HIGH	64	8.75
MED	50.33	16
LOW	31.6	4.2

Çalışmanın geçerliliğini tehdit eden unsurlar (*Threats to Validity*) :

- Üyelik fonksiyonlarında kullanılan referans değerleri az sayıda projeden elde edilmiştir.
  Bu değerler projelere bağımlıdır.
  Daha geniş yelpazede ve daha çok sayıda projeden veri toplanabilir.
- Deneyler tek bir proje üzerinde yapılmıştır.
- Sadece Java dili ile yazılmış projeler üzerinde çalışılmıştır.