MS-QDR: Mikroservis Kod Tabanları için Açıklanabilir Sınıf Düzeyinde Kalite Kararı ve Yeniden Yapılandırma Modeli

Hasan Şenyurt • BLG625/Bilgisayar Mühendisliği • Istanbul Teknik Üniversitesi

***Özet*—Mikroservis sistemlerinde tasarım kalitesini korumak zordur çünkü mimari sınırlar, hızlı evrim ve servisler arası bağımlılıklar yüksek riskli kod noktalarını gizleyebilir. Bu makale, raporlama için mikro servis bağlamını korurken sınıf düzeyinde kaliteyi değerlendiren, statik analiz tabanlı bir karar destek modeli olan MS-QDR'yi (Microservice Quality Decision & Refactor Model) sunmaktadır. MS-QDR, boyut (size), karmaşıklık (complexity), bağımlılık (coupling) ve mimari (architecture) olmak üzere dört risk kanalını birleştirerek normalize edilmiş risk puanları oluşturur ve yorumlanabilir bir Sınıf Kalite Puanı (CQS - Class Quality Score) ve aksiyon alınabilir bir karar etiketi {OK, WATCH, REFACTOR} üretir. HVL-Shopping .NET mikroservis çözümü (7 servis genelinde 279 sınıf) üzerine yapılan bir çalışmada, MS-QDR 66 yeniden düzenleme adayı belirledi ve yeniden düzenleme ihtiyacı olan sınıfların hangi servislerde yoğunlaştığını gösterdi. Çıktılar, yeniden üretilebilir CSV dosyaları olarak sunuldu ve bu dosyalar, yeniden düzenleme iş listesinin oluşturulmasını ve risk analiziyle beraber projenin ne durumda olduğu konusundaki bilgilerin elde edilmesini sağladı.**

Anahtar Kelimeler—mikro servisler, yazılım kalitesi, static analiz, yeniden düzenleme önerisi, kod metrikleri, açıklanabilir karar modeli.

I. GİRİŞ

Mikroservis mimarileri, dağıtım bağımsızlığını ve ölçeklenebilirliği artırır, ancak aynı zamanda tasarım kalitesi yönetimini de karmaşıklaştırır. Tek bir iş yeteneği birden fazla hizmeti kapsayabilir ve kod, teslimat baskısı altında hızla gelişir. Sonuç olarak, kalite sorunları genellikle yerel odak noktaları (örneğin, karmaşık handler'lar, aşırı bağımlı repository'ler) şeklinde birikir ve yalnızca servis düzeyi göstergeleri izlendiğinde fark edilmesi zordur.

Geleneksel statik metrikler boyut veya karmaşıklığı ölçebilir, ancak mühendislik sorusuna doğrudan cevap vermezler: Bu sınıf şimdi yeniden düzenlenmeli mi, sadece izlenmeli mi yoksa olduğu gibi mi bırakılmalı? Ayrıca, mikroservis projelerinde sıklıkla kullanılan servis düzeyindeki toplulaştırılmış skorlar, bakım maliyetini büyük ölçüde belirleyen uç değerli (aykırı) sınıfların gizlenmesine neden olabilir. Bu çalışma, raporlama için mikroservis yapısını korurken aksiyon alınabilir yeniden yapılandırma önerileri üreten, açıklanabilir bir sınıf düzeyinde karar modeli olan MS-QDR'yi (Microservice Quality Decision & Refactor Model) önermektedir [7]. Başlıca katkılar şunlardır:

- Servis düzeyindeki maskeleme etkisini önlerken, mikroservis bazlı dağılımları raporlamaya devam eden sınıf seviyesinde bir karar mekanizması.
- Normalizasyon yapılmış bileşen riskleri ve 0-100 Sınıf Kalite Puanı (CQS) içeren dört kanallı bir risk modeli (boyut, karmaşıklık, bağımlılık, mimari).
- Yeniden düzenleme işleri için her sınıf için açıklanabilir çıktılar (Karar, Birincil Sebep, Önerilen Eylem) CSV dosyaları olarak sunulur.
- Gerçek bir .NET mikroservis çözümü (HVL-Shopping) üzerine bir vaka çalışması; kararların ve risklerin servisler arasında nasıl dağıldığını göstermektedir.

II. İLGİLİ ÇALIŞMALAR

Yazılım kalitesi değerlendirmesi ve yeniden düzenleme önerisi, nesne yönelimli ve servis yönelimli sistemler bağlamında geniş çapta incelenmiştir. Bu bölümde, en ilgili araştırma alanları gözden geçirilmekte ve MS-QDR'nin mevcut yaklaşımlardan nasıl farklılaştığı ve bunları nasıl genişlettiği açıklanmaktadır.

A. Metrik Tabanlı Yazılım Kalitesi Modelleri

Yazılım kalitesi üzerine yapılan ilk araştırmalar, sürdürülebilirlik, karmaşıklık ve değişim eğilimini ölçmek için büyük ölçüde nesne yönelimli metriklerden yararlanmıştır. CK metrik paketi, nesneler arası bağımlılık (CBO - coupling between objects), bir sınıf için yanıt (RFC - response for a class) ve sınıf başına ağırlıklı metotlar (WMC - weighted methods per class) gibi temel ölçütleri tanıttı ve bunlar o zamandan beri iç tasarım kalitesinin standart göstergeleri haline geldi [1]. McCabe'in döngüsel karmaşıklığı (cyclomatic complexity), kontrol akışı karmaşıklığı (control-flow complexity) ve karar yoğunluğunun (decision density) iyi bilinen bir ölçüsünü daha sunmaktadır [4].

Bu metrikler bireysel kalite özelliklerini yakalamada etkili olsa da, genellikle tek başına veya sabit eşik değerler üzerinden yorumlanırlar. Bu tür yaklaşımlar doğrudan uygulanabilir yeniden düzenleme kararları üretmez ve genellikle birden fazla kalite boyutu genelinde birleşik bir yorumdan yoksundur.

B. Yeniden Düzenleme Tespiti ve Öneriler

Kodu yeniden düzenleme araştırmaları, kural tabanlı kod kokusu (code smell) tespiti, tasarım anti-kalıplarının (antipattern) belirlenmesi ve öneri sistemlerini kapsar. Fowler'ın kod yeniden düzenlemeleri ve kod kokusu (code smell) bulguları, tasarım kalitesini iyileştirmek için

kavramsal rehberlik sağlar [2]. Daha sonraki çalışmalarda, genellikle metrik eşik değerlerine dayanan, Büyük Sınıf, Tanrı Sınıfı (God Class) veya Özellik Kıskançlığı (Feature Envy) gibi sorunları tespit etmeye yönelik otomatik detektörler önerilmiştir.

Daha yeni yaklaşımlar, yeniden düzenleme eylemleri önermek için arama tabanlı optimizasyon veya makine öğrenimini kullanmaktadır. Bu teknikler güçlü olmalarına rağmen, sıklıkla geçmiş sürüm verilerine, eğitim etiketlerine veya karmaşık optimizasyon süreçlerine bağlıdır; bu da şeffaflığı ve endüstriyel ortamlarda benimsenmeyi azaltabilir.

MS-QDR, geçmiş verilere veya öğrenmeye dayalı çıkarımlara ihtiyaç duymadan açık sınıf düzeyinde kararlar üreten, hafif, statik analiz tabanlı bir karar modeli sunarak diğerlerinden ayrılır.

C. Mikroservis Mimarilerinde Kalite Ölçümü

Mikroservis mimarileri, dağıtılmış bağımlılıklar, servis sınırlarının aşınması (service boundary erosion) ve servisler arası bağımlılık gibi kalite değerlendirmesi için ek zorluklar ortaya çıkarır. Newman [3] ve Dragoni et al., mikroservislerin karmaşıklığını dağıtımdan tasarım zamanı koordinasyonuna ve bağımlılık yönetimine kaydırdığını vurgulamaktadır.

Çeşitli çalışmalar, mikroservislerin sağlığını değerlendirmek için servis düzeyi metrikleri veya toplu kalite göstergeleri önermektedir. Ancak, servis düzeyinde yapılan toplulaştırma, bakım maliyetinin büyük kısmını oluşturan az sayıdaki sınıfın yerel tasarım sorunlarını gizleyebilir. MS-QDR, raporlama ve önceliklendirme için mikroservis bağlamını korurken sınıf düzeyinde karar verme hassasiyetini muhafaza ederek bu sınırlamayı giderir.

D. Açıklanabilir Kalite ve Karar Desteği

Son çalışmalar, özellikle teknik borç (technical debt) yönetimi açısından, kalite değerlendirme araçlarında açıklanabilirliğin önemini vurgulamaktadır. Mühendisler, bir sınıfın neden işaretlendiğini ve risk açısından hangi kalite boyutunun baskın olduğunu anlamalıdır. MS-QDR, kaliteyi yorumlanabilir risk kanallarına ayırarak ve her sınıf için baskın nedeni ve önerilen eylemi açıkça raporlayarak bu yöne uyum sağlamaktadır.

Özetle, mevcut yaklaşımlar ya eyleme geçirilebilir kararlar olmadan ölçümler hesaplıyor, sınırlı şeffaflıkla yeniden düzenlemeler öneriyor ya da servis düzeyinde toplama yöntemine dayanıyor. MS-QDR, mikroservis kod tabanlı projelere özel olarak tasarlanmış, açıklanabilir, sınıf düzeyinde bir karar modeliyle bu boşluğu dolduruyor.

III. MS-QDR: Microservice Quality Decision & Refactor Model

MS-QDR, sınıf düzeyindeki metrikleri hesaplayan, bunları normalize edilmiş risk puanlarına dönüştüren ve her sınıf için yorumlanabilir bir kalite puanı ve karar etiketi üreten statik analiz tabanlı bir karar destek modelidir.

A. Girdiler ve Analiz Kapsamı

MS-QDR'ye girdi olarak bir veya daha fazla mikroservis projesi içeren bir .NET çözümü dosyası(.NET solution) verilir. Analiz, arka uç (backend) servislerini (örneğin, API'ler) hedef alır ve tamamen statik kaynak kod üzerinde çalışır. Analiz birimi, kaynak dosya yoluna eşlenmiş bir sınıftır.

MS-QDR, her bir c sınıfı için ham ölçümlerden oluşan bir vektör çıkarır:

M(c) = {LOC (Lines of Code), NOM (Number of Methods), NOF (Number of Fields), WMC (Weighted Methods per Class), RFC (Response for a Class), Cyclomatic, CBO (Coupling Between Objects), FanIn (bir sınıfa bağımlı olan diğer sınıfların sayısı), FanOut (bir sınıfın bağımlı olduğu diğer sınıfların sayısı), Instability (bir sınıfın değişime ne kadar açık olduğu), LayerViolations (bir sınıfın tanımlı mimari katman kurallarını ihlal eden bağımlılıkları), CycleInvolvement (bir sınıfın döngüsel bağımlılıkların (dependency cycles) parçası olup olmadığı)}

Aşağıda her bir metriğin nasıl elde edildiği ayrıntılı olarak açıklanmaktadır.

1. LOC (Lines of Code – Kod Satırı Sayısı)

Tanım:

LOC, bir sınıfa ait boş olmayan ve yorum satırı içermeyen kaynak kod satırlarının sayısını ifade eder.

Hesaplama:

Sınıf tanımı içerisinde yer alan tüm yürütülebilir satırlar sayılır; boş satırlar ve yorumlar hariç tutulur.

Yorum:

LOC, sınıfın büyüklüğünü ve implementasyon hacmini gösteren temel bir ölçüttür.

2. NOM (Number of Methods – Metot Sayısı)

Tanım:

NOM, bir sınıf içerisinde tanımlanan metotların toplam sayısını ifade eder.

Hesaplama:

$$NOM(c) = |\{m | m \text{ sınıf } c \text{ içinde tanımlı bir metottur}\}|$$

Yorum:

Yüksek NOM değeri, bir sınıfa aşırı sorumluluk yüklendiğine işaret edebilir.

3. NOF (Number of Fields – Alan Sayısı)

Tanım:

NOF, bir sınıf içinde tanımlanan alanların (özelliklerin) sayısını ifade eder.

Hesaplama:

$NOF(c) = |\{f | f \text{ sınıf } c \text{ içinde tanımlı bir alandır }\}|$

Yorum:

NOF metriği, sınıfın tuttuğu durum (state) miktarını ve sorumluluk yoğunluğunu gösterir.

4. WMC (Weighted Methods per Class – Ağırlıklı Metot Sayısı)

Tanım:

WMC, bir sınıftaki metotların toplam karmaşıklığını ölçer.

Hesaplama:

$$WMC(c) = \sum_{m \in c} Cyclomatic(m)$$

Yorum:

Yüksek WMC değeri, sınıfın davranışsal karmaşıklığının ve bakım zorluğunun arttığını gösterir.

5. RFC (Response for a Class – Sınıfın Yanıt Kümesi)

Tanım:

RFC, bir sınıfa gelen bir isteğe yanıt olarak çalıştırılabilecek metotların sayısını ifade eder.

Hesaplama:

$$RFC(c) = |\{m | m \text{ sınıf } c \text{ içinde tanımlı veya } c \text{ tarafından çağrılan bir metottur}\}|$$

Yorum:

Yüksek RFC değeri, sınıfın geniş bir etkileşim yüzeyine sahip olduğunu ve test maliyetinin arttığını gösterir.

6. Cyclomatic Complexity (Çevrimsel Karmaşıklık)

Tanım:

Çevrimsel karmaşıklık, bir programın kontrol akışındaki bağımsız yürütme yollarının sayısını ölçer.

Hesaplama:

$$Cyclomatic(G) = E - N + 2P$$

Burada;

E: kenar sayısı

N: düğüm sayısı

P: bağlı bileşen sayısı (genellikle 1)

Yorum:

Yüksek çevrimsel karmaşıklık, kodun anlaşılabilirliğini ve test edilebilirliğini azaltır.

7. CBO (Coupling Between Objects – Nesneler Arası Bağımlılık)

Tanım:

CBO, bir sınıfın doğrudan bağımlı olduğu farklı sınıfların sayısını ölçer.

Hesaplama:

$$CBO(c) = |c' | c \to c'|$$

Yorum:

Yüksek CBO değeri, değişikliklerin sistem genelinde yayılma riskini artırır.

8. FanIn

Tanım:

FanIn, bir sınıfa bağımlı olan diğer sınıfların sayısını ifade eder.

Hesaplama:

$$FanIn(c) = |c' | c' \to c|$$

Yorum:

Yüksek FanIn değeri, sınıfın sistem içinde merkezi bir rol oynadığını gösterir.

9. FanOut

Tanım:

FanOut, bir sınıfın bağımlı olduğu diğer sınıfların sayısını ifade eder.

Hesaplama:

$$FanOut(c) = |c' | c \to c'|$$

Yorum:

Yüksek FanOut, sınıfın çok sayıda dış bileşene bağlı olduğunu ve kırılganlığının arttığını gösterir.

10. Instability (Kararsızlık)

Tanım:

Instability, bir sınıfın değişime ne kadar açık olduğunu ölçer.

Hesaplama:

$$Instability = \frac{FanOut(c)}{FanIn(c) + FanOut(c)}$$

$$0 \leq \text{Instability} \leq 1$$

Yorum:

- 0'a yakın değerler: kararlı sınıflar
- 1'e yakın değerler: değişime açık sınıflar

11. LayerViolations (Katman İhlalleri)

Tanım:

LayerViolations, bir sınıfın tanımlı mimari katman kurallarını ihlal eden bağımlılıklarını ifade eder.

Hesaplama:

$$LayerViolations(c) = |\{d | d \text{ katman kurallarını ihlal eden bir bağımlılıktır }\}|$$

Yorum:

Katman ihlalleri, mimari bozulmanın ve teknik borcun önemli göstergeleridir.

12. CycleInvolvement (Döngüsel Bağımlılığa Katılım)

Tanım:

CycleInvolvement, bir sınıfın döngüsel bağımlılıkların parçası olup olmadığını gösterir.

Hesaplama:

$$CycleInvolvement(c) = \begin{cases} 1, & \text{eğer c bir bağımlılık döngüsünün parçasıysa} \\ 0, & \text{aksi halde} \end{cases}$$

Yorum:

Döngüsel bağımlılıklar modülerliği azaltır ve refactor maliyetini artırır.

B. Risk Kanalı Modellemesi

MS-QDR'de kullanılan ham metrikler (ör. LOC, WMC, CBO), farklı ölçeklere ve dağılımlara sahiptir. Bu metriklerin doğrudan birleştirilmesi, uç değerlerin (outliers) toplam risk üzerinde orantısız etki yaratmasına neden olabilir. Bu nedenle MS-QDR, her bir metriği robust (sağlam) istatistiklere dayalı bir normalizasyon sürecinden geçirir.

Her bir metrik m için öncelikle tüm sınıflar üzerinde medyan ve Medyan Mutlak Sapma (MAD) hesaplanır:

$$MAD_m = median(|m(c) - median(m)|)$$

Ardından, her sınıf için metrik değeri robust z-score kullanılarak normalize edilir:

$$z_m(c) = \frac{\big(m(c) - median(m)\big)}{(1.4826 * MAD_m)}$$

Buradaki 1.4826 katsayısı, MAD değerini standart sapmaya ölçeklendirmek için kullanılan sabit bir çarpandır. Bu yaklaşım, klasik z-score yöntemine kıyasla uç değerlere karşı daha dayanıklıdır.

Elde edilen z değeri, sigmoid fonksiyonu aracılığıyla [0,1] aralığında bir risk katkısına dönüştürülür:

$$r_m(c) = \frac{1}{\big(1 + e^{(-z_m(c))}\big)}$$

1) Boyut Riski:

Boyut riski, bir sınıfın aşırı sorumluluklar biriktirme olasılığını ifade eder.

$$R_{size}(c) = \frac{r_{LOC}(c) + r_{NOM}(c) + r_{NOF}(c)}{3}$$

2) Karmaşıklık Riski:

Karmaşıklık riski, kontrol akışı ve davranışsal karmaşıklığı yansıtır.

$$R_{complexity}(c) = \frac{r_{WMC}(c) + r_{RFC}(c) + r_{Cyclomatic}(c)}{3}$$

3) Bağımlılık Riski:

Bağımlılık riski, bağımlılık yoğunluğunu ve değişikliklerin sistem genelinde yayılma baskısını modellendirir.

$$R_{coupling}(c) = \frac{r_{CBO}(c) + r_{FanIn}(c) + r_{FanOut}(c) + r_{Instability}(c)}{4}$$

4) Mimari Risk:

Mimari risk, amaçlanan katmanlama ve döngüsel bağımlılıkların ihlallerini kapsar.

$$R_{architecture}(c) = \frac{r_{LayerViolations}(c) + r_{CycleInvolvement}(c)}{2}$$

C. Risk Birleştirme ve Sınıf Kalite Puanı

Bir sınıfın genel riski, kanal risklerinin ağırlıklı toplamı olarak hesaplanır:

$$R_{total}(c) = w_s * R_{size}(c) + w_c * R_{complexity}(c) + w_k * R_{coupling}(c) + w_a * R_{architecture}(c)$$

$$w_s + w_c + w_k + w_a = 1$$

Ağırlıklar varsayılan olarak eşit seçilebilir. Fakat projeden projeye veya isterden istere bu ağırlıklar istenildiği gibi ayarlanıp eşik değerleri de buna göre seçilebilir. Bu yöntemde ağırlıklar sezgisel olarak aşağıdaki gibi seçilmiştir. Daha önceki deneyimlere dayanarak sırayla

bağımlılık, karmaşıklık, boyut ve mimariye önem verilmiştir.

$$w_s = 0.20$$

$$w_c = 0.25$$

$$w_k = 0.35$$

$$w_a = 0.20$$

Sınıf Kalite Puanı (CQS – Class Quality Score) şu şekilde tanımlanır:

$$CQS(c) = 100 * \left(1 - R_{total}(c)\right)$$

Daha yüksek CQS değerleri, daha iyi yapısal kaliteyi gösterir.

D. Karar Eşik Değerleri

MS-QDR, sabit eşik değerleri yerine veriye bağlı eşikler kullanır. Bu yaklaşım, farklı büyüklükteki projelerde yöntemin daha tutarlı çalışmasını sağlar.

$$T_{ref} = P_{20}(CQS)$$

$$T_{watch} = P_{50}(CQS)$$

- P_{20}: CQS değerlerinin %20 yüzdelik dilimi
- P_{50}: CQS değerlerinin medyanı

Bu eşiklere ek olarak, kritik risk kanalları için sert sınırlar tanımlanmıştır. Nihai karar mantığı aşağıdaki gibidir:

$$Decision(c) = \begin{cases} REFACTOR, & \text{eğer } CQS(c) \leq T_{ref} \\ & \text{veya } R_{coupling}(c) \geq 0.75 \\ & \text{veya } R_{architecture}(c) \geq 0.70 \\ WATCH, & \text{eğer } CQS(c) \leq T_{watch} \\ & \text{veya } R_{complexity}(c) \geq 0.70 \\ OK, & \text{aksi halde} \end{cases}$$

Bu yapı sayesinde:

Kritik bağımlılık veya mimari problemler, toplam skor iyi olsa dahi maskelemez ve daha orta seviyedeki riskler WATCH kategorisi ile izlenebilir hale getirilir.

E. Açıklanabilir Çıktılar

MS-QDR, yalnızca sayısal bir kalite skoru üretmekle kalmaz; her bir sınıf için verilen kararın nedenlerini açık ve yorumlanabilir biçimde raporlar. Bu amaçla model, risk kanalları arasındaki baskın faktörü belirleyerek kararın temel gerekçesini ortaya koyar.

$$PrimaryCause(c) = argmax\{R_{size}, R_{complexity}, R_{coupling}, R_{architecture}\}$$

Bu ifade, sınıfın kalite açısından en baskın problem kaynağını temsil eder.

PrimaryCause (yeniden düzenleme için birincil sebep) belirlendikten sonra RecommendedAction (yeniden düzenlemenin nasıl yapılabileceği ile ilgili aksiyon önerisi) tanımlanır. Boyut için önerilen aksiyon ‘Büyük sınıfların bölünmesi, tek sorumluluk ilkesine uygun hale getirilmesi ve alan/metot sayısının azaltılması’, karmaşıklık için önerilen aksiyon ‘Karmaşık metotların ayrıştırılması, koşul sayısının azaltılması ve uygun durumlarda polimorfizm veya strateji desenlerinin kullanılması.’, bağımlılık için önerilen aksiyon ‘Bağımlılıkların arayüzler aracılığıyla gevşetilmesi, bağımlılık enjeksiyonu kullanımı, facade veya adapter desenleriyle bağımlılık sayısının azaltılması’ ve mimari için önerilen aksiyon ‘Katman ihlallerinin giderilmesi, kodun doğru katmana taşınması ve döngüsel bağımlılıkların arayüzler veya olay tabanlı iletişim yoluyla kırılması’dır.

MS-QDR’nin açıklanabilir çıktı yapısı, yalnızca “hangi sınıf refactor edilmeli?” sorusuna değil, aynı zamanda “neden?” sorusuna da açık ve yorumlanabilir yanıtlar sunar. Risk kanalları arasındaki baskın faktörün argmax yaklaşımıyla belirlenmesi, kararların sezgisel ve mühendislik pratiğiyle uyumlu biçimde gerekçelendirilmesini sağlar.

F. Yöntemi Uygulamak İçin İzlenmesi Gereken Adımlar

- .NET 10.0.101 sürümü kurulu olmalıdır.
- Python 3.12.5 sürümü kurulu olmalıdır.
- Aşağıdaki dosyalar aynı konumda bulunmalıdır:
  - hvl-shopping (incelenen çalışma)
  - roslyn_analyzer (metrik toplayabilen statik kod analiz altyapısı)
  - run_ms_qdr.py (toplanan metrikleri kullanarak MS-QDR yöntemiyle anlamlı sonuçlar çıkaran script)
- Dosyaların bulunduğu konumda aşağıdaki komut satırı çalıştırılmalıdır:
  - python run_ms_qdr.py --solution hvl-shopping/HavelsanShoppingApi.sln --repo-root hvl-shopping
- Analiz çıktıları out klasöründe aşağıdaki dosyalar içerisinde olacaktır:
  - class_scores_detailed.csv (Sınıfların MS-QDR’nin çıkardığı bütün sayısal değerleri, karar, birincil sebep ve önerilen aksiyon bilgileri bulunmaktadır.)
  - refactor_candidates.csv (Yeniden düzenleme gereken sınıflar bulunmaktadır.)
  - service_decision_summary.csv (Her mikroservis için yeniden düzenlenecek

sınıf sayısı ve yoğunluğu analizleri bulunmaktadır.)

- dependencies.csv

IV. ÇALIŞILAN PROJE: HVL-SHOPPING E-TICARET UYGULAMASI

MS-QDR'yi HVL-Shopping .NET çözümü (HavelsanShoppingApi.sln) [6] üzerinde değerlendirdik. Bu proje, 7 mikroservise dağılmış 279 analiz edilmiş sınıf içermektedir: WebApi, AuthServer, DiscountService, NotificationService, OrderService, ReviewService ve Shared bileşenleri. Aşağıdaki analiz dosyaları üretilmiştir:

(i) class_scores_detailed.csv
(ii) refactor_candidates.csv
(iii) service_decision_summary.csv
(iv) dependencies.csv

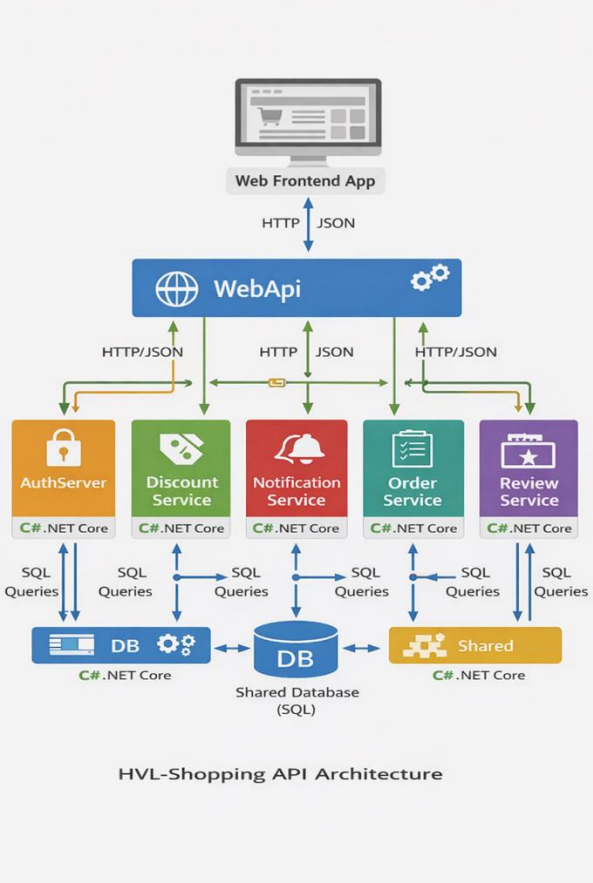

Fig. 1. HVL-Shopping Projesinin Mimarisi

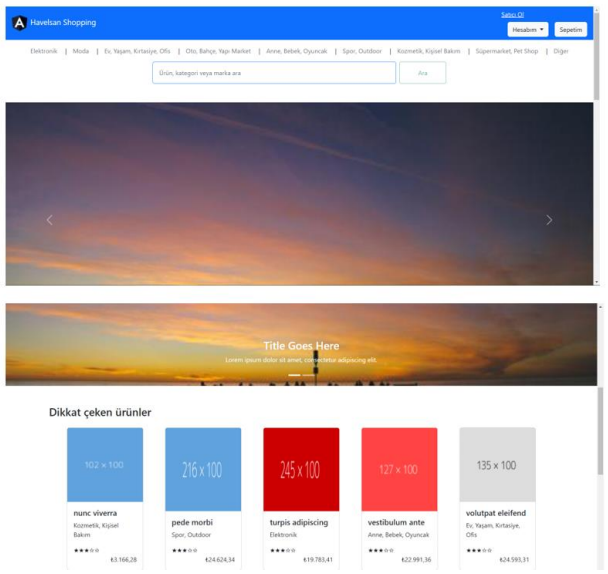

Fig 2. HVL-Shopping Uygulamasının Ana Sayfası

V. SONUÇLAR

Bu bölüm, vaka çalışmasında gözlemlenen karar dağılımını, servis düzeyindeki yeniden düzenleme yoğunluğunu ve baskın risk faktörlerini özetlemektedir. Tüm sonuçlar, tekrarlanabilirliği sağlamak amacıyla üretilen CSV çıktılarından elde edilmiştir.

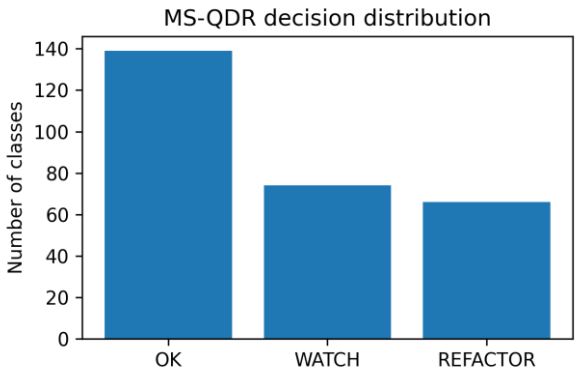

Fig. 3. 279 Sınıfın Karar Dağılımları (OK=139, WATCH=74, REFACTOR=66)

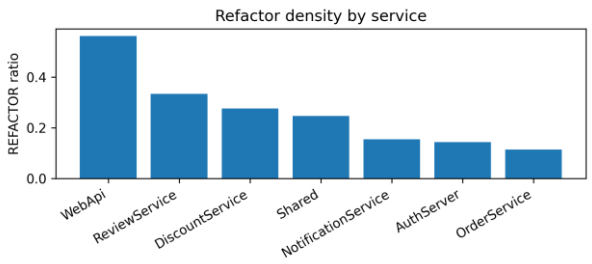

Fig. 4. Servis Bazında Yeniden Düzenleme Yoğunluğu (REFACTOR_Count / TotalClasses)

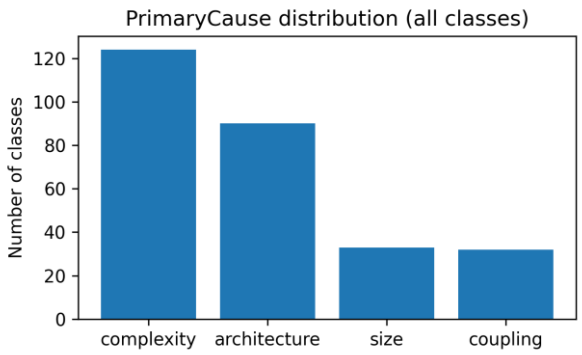

Fig. 5. Tüm sınıflarda baskın risk kanalı (Birincil Sebep) dağılımı

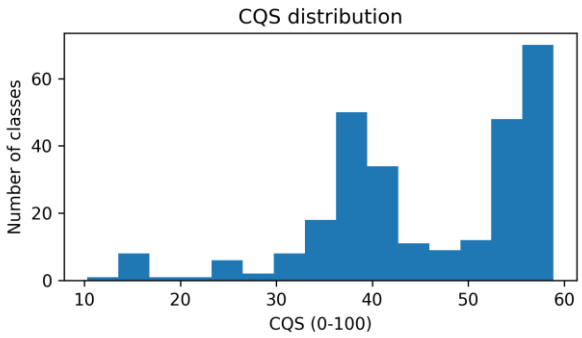

Fig. 6. Sınıf Kalite Puanı (CQS) dağılımı. CQS değerleri 10,3 ile 58,9 arasında değişmektedir (ortalama 45,6)

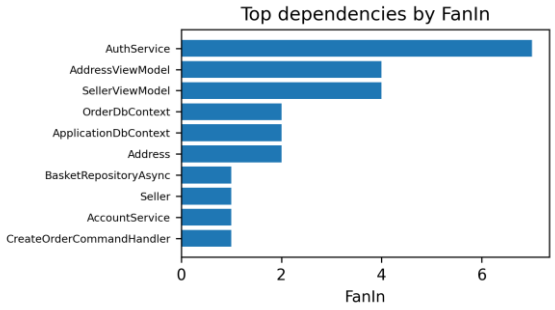

Fig. 7. Çıkarılan bağımlılık grafiği temel alınarak, FanIn (giriş derecesi) metriğine göre sıralanan en kritik bağımlılık odakları

A. Genel Karar Dağılımı

279 sınıftan 139'u OK, 74'ü WATCH ve 66'sı REFACTOR olarak etiketlendi. Bu, kod tabanının yaklaşık dörtte birinin, eşiklerin altında olduğunu ve yeniden düzenleme gerektiren sınıflar içerdiğini gösterir.

B. Service Seviyesinde Yeniden Düzenleme Yoğunluğu

MS-QDR bir servis puanı hesaplamasa da, yeniden düzenleme baskısının nerede biriktiğini ortaya çıkarmak için servis başına alınan kararları bir araya getirir. WebApi en yüksek yeniden düzenleme oranına (0,562) sahip olup, sınır arayüzü niteliğindeki API kodunun daha fazla risk noktası içerdiğini göstermektedir. Paylaşılan bileşenler (Shared), daha büyük boyutları ve merkezi olarak yeniden kullanılmaları nedeniyle en fazla yeniden düzenleme adayı içerir.

C. Risk Etkenleri ve Açıklanabilirlik

Tüm sınıflarda en sık görülen baskın kanal karmaşıklık olup, bunu mimari takip etmektedir. Ancak yeniden düzenleme adayları arasında baskın nedenler karmaşıklık ve bağımlılıktır; bu da eşik değeri koymanın, yüksek karar noktalarına ve güçlü bağımlılık baskısına sahip sınıfları en güçlü şekilde işaretlediğini yansıtmaktadır. Bu açıklanabilirlik, mühendislerin yöntem ayrıştırmasına, bağımlılık ayrıştırmasına veya her ikisine birden öncelik verip vermeyeceklerine karar vermelerine yardımcı olur.

D. Yeniden Düzenleme Adayları Örnekleri

Tablo I, temel metrikleriyle birlikte temsilî yeniden düzenleme adaylarını listelemektedir. Bu tablo, uygulanabilir bir yeniden düzenleme iş listesi oluşturmak için başlangıç noktası olarak tasarlanmıştır.

Service	ClassName	LOC	Cyclomatic	CBO	CQS
Shared	RabbitMQEventBus	282	24	9	10.30
Shared	ProductRepositoryAsync	133	15	5	14.08
OrderService	OrderRepositoryAsync	114	11	5	14.13
AuthServer	AccountService	155	16	4	14.56
Shared	CreateOrderCommandHandler	165	23	9	15.34
NotificationService	GenericRepositoryAsync	78	11	5	15.39
OrderService	GenericRepositoryAsync	78	11	5	15.39
Shared	GenericRepositoryAsync	78	11	5	15.39

Tablo I. Yeniden Düzenleme Adayları (en düşük CQS'ten en yükseğe)

VI. YORUMLAR

Mikroservis tabanlı sistemlerde sınıf düzeyinde karar üretimi kritik öneme sahiptir; zira servis düzeyinde yapılan toplulaştırmalar, bakım maliyetinin büyük kısmını oluşturan az sayıdaki uç sınıfı gizleyebilmektedir. MS-QDR, mikroservis bağlamını yalnızca raporlama amacıyla korumakta, böylece değerlendirme sonuçları paydaşlar için anlaşılır kalırken sınıf düzeyindeki ayrıntı kaybı önlenmektedir.

Vaka çalışması, literatürde sıkça gözlemlenen bir örüntüyü ortaya koymaktadır: WebApi gibi sistem sınırında konumlanan servisler, orkestrasyon mantığı ve istek doğrulama sorumlulukları nedeniyle genellikle daha yüksek karmaşıklık biriktirmektedir. Öte yandan, paylaşılan bileşenler yeniden kullanım merkezi olarak

işlev gördükleri için çok sayıda yeniden düzenleme adayına sahip olabilmektedir. Bu tür bileşenlerin iyileştirilmesi sistem genelinde önemli faydalar sağlayabilse de, kapsamlı geriye dönük testlerin dikkatle planlanmasını gerektirmektedir.

Çalışmanın bazı sınırlılıkları bulunmaktadır. Bunlar arasında, FanIn/FanOut ve döngü tespiti gibi metrikleri doğrudan etkileyen bağımlılık çıkarımının doğruluğu ile kullanılan eşik değerleri ve ağırlıkların seçimi yer almaktadır. Gelecek çalışmalar kapsamında, eşik değerlerinin geçmiş hata verileri kullanılarak kalibre edilmesi, sürüm kontrol sistemlerinden elde edilen değişiklik sıklığı metriklerinin modele dahil edilmesi ve MS-QDR'nin sürekli kalite takibi amacıyla CI boru hatlarına(pipeline) entegre edilmesi değerlendirilebilir.

Sonuç olarak MS-QDR, mikroservis kod tabanları için açıklanabilir ve sınıf düzeyinde çalışan bir karar modeli sunmaktadır. Birden fazla risk kanalını tekil ve yorumlanabilir bir kalite skorunda birleştirmesi ve uygulanabilir etiketler ile yeniden düzenleme önerileri üretmesi sayesinde, tekrarlanabilir yeniden düzenleme önceliklendirmesini ve teknik borç yönetimini desteklemektedir. HVL-Shopping proje çalışmasında MS-QDR, 66 adet yeniden düzenleme adayı belirlemiş ve yeniden düzenleme baskısının servisler arasında homojen dağılmadığını, özellikle sistem sınırına bakan servislerde daha yoğunlaştığını ortaya koymuştur.

REFERANSLAR

[1] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476–493, 1994.

[2] M. Fowler, Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999.

[3] S. Newman, Building Microservices. O'Reilly Media, 2015.

[4] T. J. McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, no. 4, pp. 308–320, 1976.

[5] E. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, 2003.

[6] https://github.com/fatihfurkanaydemir/hvl-shopping

[7] https://github.com/yhasansenyurt/MS-QDR-Mikroservis-Kod-Tabanlari-icin-Aciklanabilir-Sinif-Duzeyinde-Kalite-Karari-ve-Refaktor-Modeli