GEREKSİNİMLER LİSTESİ

Pragmite

Java Refactoring & Complexity Assistant

Konu:

Java projelerinde kod kalitesini, performans verimliliğini ve bakım kolaylığını artırmak amacıyla geliştirilen, statik analiz, otomatik refactoring, karmaşıklık analizi (Big-O), kod kokusu tespiti, mikro benchmark ve profilleme özellikleri sunan bütünleşik bir analiz aracıdır.

Amacı:

- Kod kokularını ve teknik borcu tespit edip azaltmak.
- Fonksiyonların zaman/mekân karmaşıklıklarını etiketleyerek geliştiriciye içgörü sağlamak.
- Güvenli otomatik refactoring adımları sunmak.
- Gerçek çalışma zamanı profilini (JFR) ve mikro-benchmark sonuçlarını (JMH) toplayarak performans önerileri üretmek.
- Pragmatic Programmer ilkeleri (DRY, YAGNI, Orthogonality, Tracer Bullets) çerçevesinde analiz sonuçlarını değerlendirmek ve skorlama sunmak.

Proje Adı: Pragmite

Fonksiyonel Gereksinimler:

- Döngü ve stream yapılarından O(n) türü karmaşıklık etiketi çıkarılması.
- Kod kokularının (bad smells) otomatik olarak tespit edilmesi.
- Refactoring önerilerinin güvenli biçimde tek tıklamayla uygulanması.
- Her refactoring adımından sonra build ve test süreçlerinin otomatik işletilmesi.
- Başarısız değişikliklerin otomatik revert edilmesi.
- VS Code üzerinde LSP tabanlı analiz, etiket ve refactor önerileri gösterimi.
- Web UI üzerinden proje yükleme, analiz başlatma ve sonuç inceleme.
- Hotspot fonksiyonlar için JFR verisi toplanması.
- Kritik fonksiyonlar için JMH şablonu oluşturulup benchmark yapılması.
- Kod kalitesine ilişkin skorların DRY, Orthogonality, Correctness, Perf gibi başlıklarda gösterilmesi.
- CI/CD sistemleri için kalite kapıları sunulması.
- Sonuçların arşivlenmesi (JSON ya da opsiyonel DB).

Tasarım Gereksinimleri:

- **Modüler yapı**: analyzer/, rules/, engine/, runtime/, scoring/, ui-web/, connectors/, persistence/ klasörleri.
- Web arayüzü Spring Boot ile geliştirilmiş olmalı.
- Kod editörü entegrasyonu için VS Code uzantısı sağlanmalı.
- REST API yapısı ile backend-web arası iletişim.
- Tüm kod refactoring'leri önce dry-run ile kontrol edilmeli.

Veri Gereksinimleri:

- Statik analiz için: kaynak .java dosyaları, AST ve sembol bilgileri.
- Profiling için: JFR kayıt dosyaları.
- Benchmark için: JMH çıktıları.
- Skorlama için: karmaşıklık metrikleri, refactor istatistikleri, CI log'ları.
- Arşivleme için: analiz sonuçları (JSON, opsiyonel SQL veritabanı).

Kullanıcı Gereksinimleri

- Geliştiriciler VS Code üzerinden analiz başlatıp önerilere erişebilmelidir.
- Web UI üzerinden kod kalite panosu, skorlar, refactor diff'leri, JFR/JMH grafikleri görüntülenebilmelidir.
- CLI kullanıcıları otomasyon için komut satırı araçlarıyla analiz çalıştırabilmelidir.
- CI/CD entegrasyonu sağlayan DevOps kullanıcıları kalite eşikleri tanımlayabilmelidir.
- Teknik olmayan kullanıcılar Web UI'dan yalnızca skorları ve grafik analizleri görüntüleyebilir.

Fonksiyonel Olmayan Gereksinimler

- Performans: Statik analiz < 1 saniye / dosya, refactor işlem süresi < 2 saniye hedeflenir.
- Güvenlik: Uygulanan değişikliklerin build/test güvenliği garantilenmelidir.
- Taşınabilirlik: Linux/Windows/MacOS sistemlerde çalışmalıdır.
- Genişletilebilirlik: Yeni kural ve refactor şablonları kolayca eklenebilir olmalıdır.
- Ölçeklenebilirlik: 10.000+ satırlık projelerde kararlı çalışmalıdır.
- Loglama & Hata ayıklama: Tüm adımlar izlenebilir olmalıdır.

Yol Haritası (Zaman Çizelgesi)

5. Hafta: Geliştirme ortamı ve araçların kurulumu (SonarQube, statik analiz eklentileri), örnek kod taramaları.

6. Hafta: Kod kokusu tespiti ve ilk analiz çıktılarının değerlendirilmesi; Sprint 1 değerlendirme, planlama.

7. Hafta: Öncelikli kod kokularının seçimi ve refactoring planı. JDeodorant entegrasyonu, öneri analizi.

8. Hafta: Refactoring uygulamaları (God Class, Long Method vb.); testlerle doğrulama.

9. Hafta: Gelişmiş refactoring; kod kalitesinin gözle görülür şekilde iyileştirilmesi.

10. Hafta: Refactoring tamamlanır; birim testler, kod temizliği ve SonarQube ile kalite ölçümü yapılır.

11. Hafta: Kod kalitesi metriklerinin ölçümü (CK, cyclomatic complexity, maintainability index), teknik borç izleme.

12. Hafta: Kalan sorunların düzeltilmesi; ilerleme raporu ve kalite sonuçlarının belgelenmesi.

13. Hafta: Performans testleri (JFR, VisualVM), refactoring sonrası karşılaştırmalar; final hazırlıkları.

14. Hafta: Tüm dokümantasyon ve sunum materyalleri hazırlanır; proje sunumu ve teslim.

Hafta	Sprint	Hedefler	Çıktılar
5	Sprint 1 Başlangıcı	Geliştirme ortamının kurulumu, VS Code eklentileri, SonarQube, JFR/JMH araçlarının test entegrasyonu	Kurulu analiz ortamı, örnek kodlarda ilk tarama
6	Sprint 1 Devamı	Kod kokusu analizi, ön bulguların sınıflandırılması	Sprint 1 raporu, kod kokusu listesi
7	Sprint 2 Başlangıcı	Refactoring kurallarının seçimi, JDeodorant analizi, auto-fix stratejileri belirleme	Refactor öneri listesi, uygulanacak kurallar
8	Sprint 2 Devamı	Kod üzerinde refactoring uygulamaları, build/test süreçlerinin otomasyonu	Güncellenmiş kod, testleri geçen refactorlar
9	Sprint 3 Başlangıcı	İleri seviye refactor çalışmaları, DRY/Ortho analizleri	Skorlanmış refactored modüller
10	Sprint 3 Devamı	Otomatik revert ve kalite kapısı mekanizmalarının entegrasyonu	Refactor güvenlik mekanizması
11	Sprint 4 Başlangıcı	Kod kalite metriklerinin ölçülmesi (CK, cyclomatic, maintainability)	Metrik çıktıları, grafikler, karşılaştırmalı analiz
12	Sprint 4 Devamı	Teknik borç hesaplama, kalan kod kokularının ele alınması	Borç skoru ve kalan refactor önerileri
13	Sprint 5	Performans profili çıkarımı (JFR, JMH), hız farkı ölçümü	Önce/sonra performans karşılaştırması
14	Final	Dokümantasyon, sunum slaytları, test sonucu raporları ve teslim hazırlığı	Final proje teslimi, sunum, tüm belgelerin tamamlanması