Yazılım Depolarının Analizi ile Tasarım – Kod Uyumluluğunun Araştırılması

Kadriye ÖZBAŞ ÇAĞLAYAN[1], Ali H. DOĞRU[2]

[1] TÜBİTAK-BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü, ANKARA
[2] ODTÜ Bilgisayar Mühendisliği Bölümü, ANKARA
kadriye.caglayan@tubitak.gov.tr
dogru@ceng.metu.edu.tr

Özet. Yazılım projelerinde analiz aşamasında tanımlanan probleme çözüm üretme aşaması olan tasarım, geliştiriciler tarafından yazılımı gerçekleştirmek üzere yürütülen kodlama sürecine girdi sağlar. Yazılım geliştirilirken ya da güncellenirken, kod ve tasarımın uyumlu olarak idamesi uzun vadeli bir yatırımdır ve idame ettirilebilir bir proje sonucunu doğurur. Ancak gerçek hayatta tecrübesizlik, bilgi eksikliği ya da tasarımın atıl kalmış olması gibi sebeplerle geliştiricilerin tasarımla tam olarak tutarlı kod gerçekleştiremediği durumlar yaşanabilmektedir. Tasarım ile uyumsuzluk ise kalite açısından tahmin edilemez bir kod oluşmasına sebep olur. Tasarım ile kod arasındaki uyumluluğun kontrol edilmesi faydalı ancak iş gücü yoğun bir işlemdir. Uyumluluğun kontrolü amacı ile yazılım depolarının analizinin yapılması, tersine mühendislik ve manuel kontrol süreçlerinde karşılaşılan güçlüklerin üstesinden gelinmesini sağlayacak ve zamanda yazılım geliştirme / idame süreçlerinin yöneticilerine değerli bilgiler sunacaktır. Bu çalışmada kod – tasarım arasında uyumluluğun belirlenmesi amacı ile yazılım depolarının analiz edilmesi için geliştirilen yaklaşım ile bu yaklaşımın örnek bir sürüm üzerinde uygulaması anlatılmaktadır.

1 Giriş

Yazılım geliştirme projelerinde, analiz aşamasında belirlenen soruna çözüm tasarım aşamasında oluşturulduğundan ötürü, tasarım oldukça önemli bir aşamadır. Tasarım aşamasında alternatifler arasından en iyi tasarım seçimi ile birlikte çözüm kararları da alınır. Tasarımda belirlenen çözüm ve tasarım bileşenler, daha sonraki aşamada bir ve daha fazla geliştirici tarafından yürütülen kodlama aşaması ile gerçekleştirilerek tasarım kaynak koduna dönüştürülür. Bu nedenle, üretilen kaynak kodunun çözümü tanımlayan tasarım ile uyumlu olması beklenir.

Tasarım, yazılım geliştirilirken ya da bakımı yapılırken üretilen yazılımın karmaşıklığını azaltmaya yardımcı bir unsurdur. Kaynak kodu ve tasarım arasında uyumluluğu korumak yüksek kaliteli ve sürdürülebilir bir proje için uzun vadeli bir yatırım-

dır. Ancak gerçek hayatta tecrübesizlik, bilgi eksikliği ya da yazılımın yaşlanması sonucu tasarımın atıl kalmış olması [1] gibi sebeplerle geliştiricilerin tasarımla tam olarak tutarlı kod gerçekleştiremediği durumlar yaşanabilmektedir. Bazı araştırmacılar bu sorunları tasarım ilkelerine uyulmamasına [2] ve tasarım kalıplarının kullanılmamasına [3] dayandırır; özellikle de yazılım geliştirme ve bakım aşamalarında bu konuların dikkate alınması konusuna vurgu yapar [4]. Tasarım ile uyumsuzluk ise kalite açısından tahmin edilemez bir kod oluşmasına sebep olur.

Tasarım ile kod arasındaki uyumluluğun kontrol edilmesi faydalı ancak iş gücü yoğun bir işlemdir. Çünkü bu işlem kodun tersine mühendislik yaklaşımı ile tasarıma dönüştürülmesini ve hedeflenen tasarım ile manuel olarak uyumluluğunun kontrol edilmesini gerektirir. Kod – tasarım uyumluluğunun kontrolü amacı ile yazılım ürünlerinin saklandığı yazılım depolarında analizler yapılması, tersine mühendislik ve manuel kontrol süreçlerinde karşılaşılan güçlüklerin üstesinden gelinmesini sağlayacaktır. Benzer şekilde, kod – tasarım arasındaki uyumluluğun incelenmesi, yazılım geliştirme / idame süreçlerinin yöneticilerine değerli bilgiler sunacaktır.

Bu çalışmada, tasarım ve kod uyum düzeylerini analiz etmek yazılım deposu analiz yöntemi ve kısmen metin madenciliği teknikleri kullanılması hedeflenmektedir.

Makalenin geri kalan kısmı şu şekilde düzenlenmiştir: 2. bölümde ilgili literatür gözden geçirilmiştir. 3. bölümde tasarım-kod uyumluluğu analizi için izlenen yöntem açıklanmaktadır. 4. bölümde tasarım-kod uyumluluğunu değerlendirmek için yapılan örnek analiz çalışmasının sonuçları sunulmuştur. Son bölümde ise bu çalışmanın sonuçları ve gelecek çalışmalar özetlenmiştir.

2 İlgili Çalışmalar

Yazılım mühendisliği araştırmacıları, yazılımın gelişimi ve değişimi bağlamında yazılım depolarından ilişkileri ve eğilimleri ortaya çıkarmak için geniş bir yelpazede farklı yaklaşımlar geliştirdiler. Yazılım depolarından yapısal kaynak kodu değişiminin tespit edilmesi için geliştirilen bir araç [5]'te verilmiştir. Söz konusu araç ile parametre / değişken / metot ekleme / kaldırma; bir değişkenin / metodun gizlenmesi / açılması / isim değişikliği; gizlemek / açmak / yeniden adlandırma; özniteliğin / metodun / sınıfın yerinin değiştirilmesi; süper sınıf / arayüz / sınıf bilgisinin çıkartılması gibi değişim türleri tespit edilebilmektedir.

Kaynak kodu ile birlikte test kodunun değişiminin eşgüdümü, bir projenin sürümleri için kod kapsama oranları ve test / kaynak kodlarının boyutları incelenerek tespit edilmiştir [6].

Levenshtein mesafesi algoritması ile bilgi çekme teknikleri (Vektör Uzayı Modeli) birleştirerek CVS depolarındaki kaynak kodları analiz edilmiş, kod satırı düzeyinde değişiklikleri tespit etme yöntemi tanımlanmıştır [7].

Kodlama ve tasarım arasındaki uyumluluk kontrolleri ile ilgili olarak literatürde geçen eserler arasında yer alan Yazılım Yansıma Modeli [8] tekniği ile projenin bir iş ürününün (örneğin, tasarım modeli), bir başka iş ürünü (örneğin, kaynak kodları) ile uyumlu ya da uyumsuz olduğu noktaları özetleyerek yazılım mühendislerine yardımcı olmayı hedeflemektedir.

Kullanım durumları ile kaynak kodları arasındaki izlenebilirlik ilişkilerinin otomatik olarak kurulabilmesine ya da hatalı olarak kurulmuş izlenebilirlik ilişkilerinin tespit edilmesine yönelik olarak bir çözüm sunmak üzere geliştirilen LeanArt aracı, geliştirici tarafından örnek olarak kurulan izlenebilirlik ilişkilerinden derlenen bilgilerle diğer izlenebilirlik ilişkilerini tahmin etmeye çalışır [9]. LeanArt otomatik öğrenme yöntemi olan Çok Stratejili Öğrenme Yaklaşımı (Multistrategy Learning Approach) ile geliştirilmiş olup kullanım durumu adları ile kod elemanları isimleri arasında birebir uyum olmasa da izlenebilirlik ilişkilerini tespit edebilmeyi hedeflemektedir.

Nesne-yönelimli tasarımın ilgili kaynak kodu ile uyumunu kontrol etmek için Düzenleme Mesafesi'nin hesaplanması ve maksimum eşleşme algoritmasından faydalanan bir yöntem [10]'de sunulmuştur. Bu yöntemde C++ kaynak kodlarından mevcut tasarım çıkartılır ve gerçek tasarım ile karşılaştırılarak tutarsızlıklar bulunur. Önerilen yöntemin çıktısı, her eşleşen sınıf için benzerlik ölçütleri ve eşleşmeyen sınıflar kümesi şeklinde sunulmaktadır.

UML modellerinin yapısal farklılaşmalarını tespit etmeye yönelik olarak önerilen UMLDiff algoritması baz alınarak nesne yönelimli yazılım sistemlerinin değişimini analiz eden bir yöntemde [11] ise tasarım değişimlerinin analizi, kaynak kodunun ardışık olarak bir dizi anlık durumunu girdi alarak gerçekleştirilir.

3 Kod-Tasarım Uyumluluğu Analizi Yaklaşımı

Bu çalışmada, artırımlı yinelemeli yaşam döngüsü kullanılan ve toplamda 5 farklı sürüm yayınlanan bir projede ilk (v1.0) ve son sürümünden (v4.1) elde edilen veriler analiz edilmiştir. Artırımlı yinelemeli yaşam döngüsünün her bir yinelemesinde sisteme yeni özellikler eklenir ve mevcut özellikler kullanıcı geri bildirimleri doğrultusunda iyileştirilir. Her yineleme analiz, tasarım, kodlama ve sistem testleri aşamalarından oluşur ve projenin iş ürünleri her yinelemede bu aşamalarında sonunda dayanaklandırılır. Bu çalışmada, tasarım aşaması sonunda dayanaklandırılan tasarım modelinden elde edilen veriler ile ve sistem testlerinin sonunda dayanaklandırılan kaynak kodundan elde edilen veriler kullanılmıştır. Söz konusu veriler Enterprise Architect aracı ile UML 2.1 notasyonu kullanılarak oluşturulmuş tasarım modeli ve Java programlama dili kullanılarak geliştirilmiş kaynak kodlardan elde edilmiştir. Veriler elde edilirken, öncelikle tasarım modelinden sınıf tanımlarını içeren kod Enterprise Architect aracı kullanılarak üretilmiş, tasarımdan üretilen kodda ve gerçekte geliştirilen kodda yer alan sınıf tanımları, bu çalışma kapsamında geliştirilen bir metin ayrıştırıcı kullanılarak çıkartılmıştır.

Bu makale kapsamında tasarım ve kod arasında uyum incelenirken, tasarımda sadece statik özelliklerin modellendiği sınıf tanımları ve kalıtım (inheritance) ve gerçekleştirim (implementation) ilişkileri dikkate alınmıştır. Bu bağlamda yazarlar, dinamik özelliklerin modellendiği sıralama (sequence) ve aktivite diyagramları gibi tanımların yanı sıra birlik (association) ve birleştirme (composition) ilişkilerinin de incelenmesinin çalışmada sunulan yaklaşımı zenginleştireceğini ve sonuçları daha iyileştireceğini

değerlendirmekle birlikte söz konusu incelemeleri ileriki aşamalarda yapılacak çalışmalara bırakmışlardır. Tasarım ve sistem testleri dayanaklarından çıkartılan sınıf tanımlarını içeren ham veri işlenerek, sınıf tanımları aşağıdaki öznitelikleri içerecek şekilde yapısal bir duruma getirilmiştir:

- Adı (String) - Sınıfın adı
- ErişimTürü (Sayısal) – public / private / protected sınıf
- SınıfMı (Sayısal) - class veya interface olduğu
- SoyutMu (Sayısal) - soyut bir sınıf / arayüzü olup olmadığı
- FinalMi (Sayısal) - final bir sınıf / arayüzü olup olmadığı
- Extends (0 .. n) (String) - Sınıf tarafından extend edilen sınıflar
- Implements (0 .. n) (String) - Sınıf tarafından implement edilen sınıflar

Kodda yer alan bir sınıf tanımı ile tasarımda yer alan bir sınıf tanımının tam olarak eşleşme karşılaştırması, yukarıda listelenen özniteliklerin tümü karşılaştırılarak yapılmakta ve yalnızca tüm öznitelikler aynı ise kod ve tasarım sınıfları uyumlu olarak kabul edilmektedir. Özniteliklerden herhangi birisi kod ve tasarımda aynı değilse, kod ve tasarım sınıfları uyumsuz kabul edilmiştir. Veri analizleri RapidMiner (Text Processing özelliği ile) kullanılarak yapılmıştır.

4 Uyumluluk Analizi Sonuçları ve Tartışma

Veri analizinde ilk adım olarak, tasarım ve kodda yer alan sınıfların sayıları tespit edilmiş, bu sınıflardan sınıf isimleri birebir eşleşen sınıf sayıları belirlenmiştir. Kod ve tasarım arasında uyumu irdelemek amacı ile tam olarak eşleşen sınıf tanımları bulunmuştur. 3. Bölüm'de de açıklandığı gibi, tam olarak eşleşme analizi yukarıda belirtilen özniteliklerin tamamında eşleşme kontrolü yapar; yani sınıflar tüm özniteliklerinin eşleşmesi durumunda uyumlu, aksi takdirde uyumsuz olarak sayılır.

Tablo 1. Sürümlere Göre Tasarım ve Kod Ölçümleri

Proje Ölçümleri	Sürüm v1.0	Sürüm v4.1
Tasarımdaki Sınıf Sayısı	337	1144
Koddaki Sınıf Sayısı	2344	3118
Tasarım ve Kodda Aynı İsimli Sınıfların Sayısı	108	845
Tasarım ve Kodda Aynı İsimli Sınıfların Yüzdesi	53	74
Tasarım ve Kodda Tam Olarak Eşleşen Sınıfların Sayısı	95	336
Tasarım ve Kodda Tam Olarak Eşleşen Sınıfların Yüzdesi	4	11
Tasarım Kalıpları ile Uyumlu Kod Sınıflarının Sayısı	771	2279
Tasarım Kalıpları ile Uyumlu Kod Sınıflarının Yüzdesi	33	73

Tablo 1'de yer alan ölçüm sonuçları incelendiğinde, sınıf isimleri açısından tasarım ve kodlama arasında ~% 50 - % 75 arası birebir uyum söz konusudur. Tasarım ve kod

arasında tam olarak (tüm öznitelikleri birebir) eşleşen sınıf tanımlarına bakıldığında ise çok daha düşük seviyelerde eşleşme olduğu görülmektedir (~% 4 - % 11).

Bu sonuçlar ile ilgili proje ekibi ile yapılan görüşmeler üzerine, tasarımda birbirini tekrarlayan durumlar için tek bir örnek kalıba yer verildiği, bu tasarım kalıbına karşılık kodda pek çok sınıf bulunabildiği anlaşılmıştır (örneğin, "tüm XXXView sınıfları IProjectView sınıfını "implement" etmelidir" kalıbı tasarımda bir kez belirtilir ancak bu kalıp ile uyumlu olarak kodda 75+ XXXView sınıfı mevcuttur). Bu tespitin ardından, bahsi geçen tasarım kalıplarını temsil etmek üzere düzenli ifadeler (örneğin, sınıf adı *View şeklinde geçiyorsa bu sınıf tanımının devamında "implements IProjectView" ifadesi de geçmelidir kuralını ayrıştıran kurallar) tanımlanarak tasarım kalıbı ile örtüşen sınıf tanımları tespit edilmiştir. Bu düzenli ifadeler, öncelikle taraşım tanımında kontrol edilmiş, tasarımda bu kurala uygun bir sınıf tanımı varsa kodda kurala uyan tüm sınıflar tasarım ile uyumlu sayılmıştır. Söz konusu iyileştirmenin ardından tasarım ve kod sınıfları arasındaki eşleşme oranları öenmli ölçüde artmıştır (% 33 - % 73).

Tablo 1'de verilen sonuçların da özetlediği gibi kod ve tasarım arasındaki uyum düzeyleri proje ilerledikçe yükselmektedir. Bu sonuçlar arkasındaki nedenlerinden biri, iş kuralı kontrolü, uyarı ve hata mesajı sınıfları gibi birbirine çok benzer ancak son derece basit tanımı olan sınıflar kodda mevcut olmakla birlikte tasarımda mevcut değildir (ilk sürümdeki sınıfların yaklaşık olarak % 40'ı bu tür sınıflardır). Diğer bir neden de bir sınıfın tasarımdaki tanımına ilave olarak bir başka sınıfı da "extend" etmesi gibi durumlarda uyumun yok sayılması, sadece tam uyuma odaklanılması ve kısmi uyumun herhangi bir şekilde sayılmamasıdır.

5 Sonuçlar ve Gelecek Çalışmalar

Bu makalede, tasarım ve kod arasındaki uyumluluğu belirlemek için yazılım depolarında yer alan tasarım modeli ve kaynak kodlarının eşleşme analizi için bir yaklaşım sunulmuştur. Sınıf tanımları arasındaki tam eşleme ve sonrasında düzenli ifadeler kullanılarak tasarım kalıpları ile kod arasında eşleşme durumları incelenmiştir. Tasarım kalıplarının dikkate alındığı durumda tasarım ve kod arasındaki uyum düzeylerinin oldukça iyileştiği görülmüştür.

Daha sonraki aşamalarda yapılacak çalışmalarda tam uyuma ek olarak kısmi uyumun da dikkate alınmasına yer verilebilir. Düzenli ifadeler tanımlayarak tasarım kalıplarına uyumu incelemek yerine bu kalıpların veri madenciliği / metin madenciliği teknikleri ile tespit edilmesi de çalışmanın önemli bir açılımı olacaktır. Ayrıca, tasarım modeli ve kaynak kodların eşleşme analizlerinde sıralama (sequence) ve aktivite diyagramları gibi dinamik modelin yanı sıra birlik (association) ve birleştirme (composition) ilişkilerinin de incelenmesi çalışma sonuçlarının doğruluğunu artıracaktır.

Kaynaklar

1. Parnas D.L.: "Software aging," 16th Int. Conf. on Software Engineering, Soronto, Italy (1994) 279-287

2. Martin R. C.: Agile software development: principles, patterns and practices, Prentice Hall, (2003)
3. Gamma E., Helm R., Johnson R., Vlissides J.: Design patterns: elements of reusable object-oriented software, Addison Wesley (1995)
4. Chatzigeorgiou A., Manakos A.: "Investigating the evolution of bad smells in object-oriented code," 7th International Conference on the Quality of Information and Communications Technology (2010)
5. Gerlec C., Krajnc A., Heričko M., Božnik J.: "Mining source code changes from software repositories," 7th Central and Eastern European Software Engineering Conference in Russia (2011)
6. Zaidman A., Rompaey B. van, Demeyer S., van, Deursen A.: "Mining Software Repositories to Study Co-Evolution of Production & Test Code," 1st International Conference on Software Testing, Verification, and Validation (2008)
7. Canfora G., Cerulo L., Di Penta M.: "Identifying Changed Source Code Lines from Version Repositories," 4th International Workshop on Mining Software Repositories (2007)
8.Murphy G. C., Notkin D., Sullivan K. J.: "Software Reflexion Models: Bridging the Gap between Design and Implementation," IEEE Transactions on Software Engineering, Cilt No 27, 364-380 (2001)
9. Grechanik M., McKinley K., Perry D.: "Recovering And Using Use-Case-Diagram-To-Source-Code Traceability Links," 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (2007)
10.Antoniol G., Potrich A., Tonella P., Fiutem R.: "Evolving object oriented design to improve code traceability," 7th International Workshop on Program Comprehension (1999)
11.Xing Z., Stroulia E.: "Analyzing the Evolutionary History of the Logical Design of Object-Oriented Software," IEEE Journal on Software Engineering (2005) 850-868