Yazılım Davranışlarının Dinamik Olarak Değiştirilebilmesini Sağlayan Yeni Bir Yaklaşım

Çağdaş Evren Gerede

TOBB Ekonomi ve Teknoloji Üniversitesi, Ankara, Türkiye
Web Page: http://cegerede.etu.edu.tr

Özet. Günümüzde web ve mobil platformlar üzerinden sağlanan yazılımlar sık sık güncellenebilmektedir. Güncellemeler ile beraber yazılımların davranışları değişir. Bazı davranışlar istenmeyen hasarlara yol açabilir. Bu sebeple müşteriye ulaşan hatalı davranışların düzeltilmesi yeni güncellemeler yoluyla yapılmaktadır. Kaynak kodlarda düzeltmelerin yapılması, değişikliklerin test edilmesi ve kaynak kod denetimlerinden geçmesi, bu değişikliklerle yeni yazılım versiyonunun inşa edilmesi ve güncellenmiş versiyonun üretim ortamlarına konuşlandırılması saatler alabilir. Dolayısıyla birçok müşteriyi etkileyebilecek ve hatalı olduğunda müşterilerde büyük iş kayıplarına yolaçabilecek değişikliklerin saniyeler mertebesinde düzeltilebilmesi yazılım üreticilerinin prestijinin korunması ve müşteri memnuniyetinin devamı için önemli bir ihtiyaçtır. Bu çalışmada koşan yazılımların davranışlarını dinamik olarak değiştirilebilmesini sağlayacak bir yaklaşım ve bu yaklaşımı gerçekleştiren bir yazılım aracı sunmaktayız. Yaklaşımımız yazılım güncellemelerinin herhangi bir kod değişikliği gerektirmeden kademeli olarak devreye alınmasını veya saniyeler içerisinde tamamen devreden çıkarılmasını sağlamaktadır. Aynı zamanda yaklaşımımız deneysel sayılabilecek yeni yazılım özelliklerinin üretim ortamında ufak bir kullanıcı kitlesi üzerinde denenebilmesini mümkün kılmaktadır.

Anahtar Kelimeler: Yazılım güncellemesi, Yazılımların davranışlarının dinamik olarak değişitirilmesi, Yazılım konuşlandırması, Dönemeç

A New Approach Making It Possible To Change Software Behavior Dynamically

Çağdaş Evren Gerede

TOBB University of Economics and Technology, Ankara, Turkey
Web Sayfası: http://cegerede.etu.edu.tr

Abstract. Nowadays software provided over web and mobile platforms can be frequently updated. With updates the behavior of software changes. Certain behaviors can cause undesired damages. Therefore, the correction of incorrect behavior that reaches customers is done via new updates. Fixing software's source code, testing these new changes, getting peer approval through code reviews, building a new version of the software and deploying it to the production environment can take hours. As a result, being able to correct any changes that may severely affect a lot of customers in the matter of seconds is an important need for software developers to preserve developers' prestige and customer satisfaction. In this study, we propose an approach that enables developers to change software's behavior at run time and describe an implementation realizing this approach. With our approach, software updates can be deployed incrementally without making any code changes or they can be reverted completely within seconds. In addition, our approach makes it possible to try out experimental features on small subsets of users in production environments.

Keywords: Software update, Changing software behavior dynamically, Software deployment, Junction

1 Giriş

Günümüzde İnternet alt yapısının giderek iyileşmesi ile beraber web ve mobil platformlar üzerinden sunulan yazılımlar elektronik olarak sık sık güncellenebilmektedir[1, 2]. Firmaların web uygulamalarını haftada birkaç defa güncelleyerek üretim ortamlarındaki sunucularına aktarmaları standart hale gelmiştir. Mobil uygulamalar için App Store ve Google Play gibi uygulama dükkanlarının onay süreçleri güncelleme sürecini uzatsa da firmaların iki-üç haftada bir yeni güncellemeler yapabildiğini görmekteyiz[3]. Yazılım güncellemelerini genel olarak yeni özelliklerin eklenmesi, hataların düzeltilmesi, ve kaynak kodun bakımının kolaylaştırılması için iyileştirilmesi olarak guruplayabiliriz. Son kategori yazılımın davranışını aynı tutma amacı güder. İlk iki kategorideki güncellemeler ile beraber ise yazılımların davranışları kasıtlı olarak değiştirilir. Bu değişikliklerin bazılarında istenmeden hatalar yapılmış olabilir ve de bunlar yazılımın bazı fonksiyonlarının yanlış çalışmasına yol açabilir.

Müşteriye ulaşan hatalı davranışların düzeltilmesi genellikle yeni güncellemeler yoluyla yapılmaktadır. Kaynak kodlarda gerekli düzeltmelerin yapılması, değişikliklerin test edilmesi ve diğer geliştiriciler tarafından yapılan kaynak kod denetimlerinden geçmesi, bu değişikliklerle yeni yazılım versiyonunun inşa edilmesi ve güncellenmiş versiyonun üretim ortamlarına konuşlandırılması saatler alabilir[4]. Dolayısıyla birçok müşteriyi etkileyebilecek ve hatalı olduğunda müşterilerde büyük iş kayıplarına yolaçabilecek değişikliklerin saniyeler içerisinde düzeltilebilmesi yazılım üreticileri için önemli bir ihtiyaçtır. Çünkü aksi halde müşterinin hem üreticiye karşı güveni sarsılacak hem de ürün ile ilgili memnuniyeti olumsuz olarak etkilenecektir[5].

Üreticiler bu tür sorunlarla karşılaşmamak için geleneksel olarak kullanıcı trafiğinin en aza indiği saatlerde güncellemelerini üretim ortamlarına konuşlandırmayı tercih etmektedir. Böylece eğer güncellemelerde bir sorun ortaya çıkarsa en az sayıda kullanıcı bu durumdan etkilenecektir. Fakat bu geleneksel yaklaşımın önemli problemleri vardır. Birincisi, örneğin, bir Cumartesi günü sabaha karşı saat 3:00'da çalışıyor olmak çalışanların iş memnuniyetlerini olumsuz olarak etkiler. İkincisi, kullanıcı trafiğinin azaldığı saatlerde güncellemelerin etkilerini yeterli oranda görmek mümkün olmamaktadır. Örneğin yalnızca gün içerisinde kullanılmakta olan bir özelliğin güncellemeden etkilenip etkilenmediğini test edebilmek mümkün değildir. Dolayısıyla ilerleyen saatlerde kullanıcı trafiğinin artmasıyla beraber birçok sorun gün yüzüne çıkabilir. Güncellemelerin yapılmasından sonra uzun bir zaman geçtiği için sorunlara müdahale edebilecek ekipler çoktan evlerine dağılmış olabilir. Bu yaklaşımın üçüncü problemi ise birçok farklı coğrafyada kullanılan yazılımlar için kullanıcı trafiğinin azaldığı bir zaman aralığının mevcut olmamasıdır. Bir bölgenin mesai sonrası zamanı bir başka bölgenin mesai saatleriyle çakışabilir. Bu durumlarda da bu yaklaşımı kullanmak mümkün değildir.

Bu çalışmada koşan yazılımların davranışlarının dinamik olarak yönetilmesini sağlayacak bir yaklaşım önermekteyiz. Yaklaşımımız "bilgi saklama" (İng. information hiding) ve "korunmuş varyasyonlar" (İng. protected variations) prensipleri ile uyumludur[6, 7]. Bu yaklaşım sayesinde günün herhangi bir saatinde

sunucular üzerinde güncellemeler yapılabilir ve bu güncellemeler kullanıcıların karşısına kademeli olarak çıkartılabilir. Böylece kullanıcıların hangi yeni davranışlardan etkilenebileceği kontrol edilebilecek ve gerektiğinde güncellemeler devreden çıkartılabilecektir.

Yaklaşımımız herhangi bir yazılım kaynak kodu güncellemesi gerektirmez. Dolayısıyla davranışlar yukarıda bahsettiğimiz uzun sürecek geliştirme-test etme-inşa etme-konuşlandırma sürecinden bağımsız olarak değiştirilebilir. Aynı zamanda yaklaşımımız deneysel sayılabilecek yeni yazılım özelliklerinin üretim ortamında ufak bir kullanıcı kitlesi üzerinde denenebilmesini mümkün kılmaktadır. Böylece yeni bir fikir tüm kullanıcıları olumsuz olarak etkilemeden ama aynı zamanda da kullanıcıların bir alt kümesinden geribildirimler alınarak olgunlaştırılabilir.

2 Yaklaşım

Yaklaşımımızda kaynak kod içerisinde eski ve güncellenmiş kod bölgeleri aynı anda bulundurulur. Bir istek için hangi kaynak kod bölgesinin çalıştırılacağı *dönemeç* (İng. junction) diye tanımladığımız değişken değerlerine bakılarak çalışma zamanınında dinamik olarak belirlenir. Dönemeçlerin değerleri yazılımın dışında bir başka sistem içerisinde saklanır. Bu sistem üzerinden dönemeç değerleri güncellenebilir. Uygulama ise periyodik olarak yeni dönemeç değerlerini indirir. Bu sayede yazılımın davranışları dinamik olarak değişmiş olur.

Tablo 1'de örnek dönemeç kullanımları gösterilmektedir. Örneğin, bir loglama fonksiyonu çağrıldığında log mesajının yeni sisteme mi yoksa var olan sisteme mi kaydedileceğine bir if-else bloğu ile karar verilmektedir. Karar şartının değeri bir fonksiyon çağrılarak elde edilir. Örnekte *bool* fonksiyon ismi dönemecin *boolean* veri tipinde olduğunu işaret eder. "LOG_TO_NEW_SYSTEM" kullanılan dönemecin ismini içeren *String* tipinde bir sabit değişkendir. Fonksiyonun ikinci parametresi verilen isimde bir dönemeç bulunamadığında dönecek varsayılan değeri gösterir. Dönemeç kütüphanesinin fonksiyonları "d" değişkeni tarafından sağlanır. Dolayısıyla, satır 3'e göre o an dönemeç değeri "true" ise mesaj yeni sisteme loglanır. Aksi halde varolan sistem kullanılmaya devam eder.

Yine Tablo 1'de ikinci satırda bir özelliğin yalnızca bir grup kullanıcıya kademeli olarak sağlanmasına örnek gösterilmiştir. Burada "fraction" tipinde bir veri kullanılmıştır. Dönemeç tanımlanmadığında ekran arayüzü oluşturulurken deneysel olarak eklenmiş yeni bir diyalog penceresi gösterilmemektedir çünkü dönemecin varsayılan değeri sıfırdır ve çalışma zamanında karar şartının sol tarafında sıfır ile bir arasında rastgele bir sayı seçilir. Bu sayı hiçbir zaman sıfırın altında olmayacağı için uygulama birinci if-bloğu içerisine giremez. Dönemeç değeri 0.10 olarak güncellendiğinde ise kullanıcı trafiğinin rastgele %10'u için karar şartı doğru olacaktır ve böylece kullanıcıların yalnızca bir alt kümesi uygulamayı kullanırken yeni diyalog ile karşılaşacaktır.

Üçüncü kullanım örneğinde ise deneysel olarak başlangıçta yalnızca bazı kullanıcılara göndermeyi düşündüğümüz bildirimler için bir kod bölgesi görülmektedir. Bu defa dönemecimizin tipi bir String veri tipi dizisidir. Dönemeçten dönen

```
1 public void log(String msg) {
2   // default: false
3   if (d.bool(LOG_TO_NEW_SYSTEM, false)) {
4     ...// logs to new system
5   } else {
6     ...// logs to old system
7   }
8 }
```

```
1 public void renderUI(String msg) {
2   // default: 0
3   if (Math.random() < d.fraction(SHOW_NEW_DIALOG, 0)) {
4     ...
5   }
```

```
1 public void sendNotification() {
2   if (contains(d.stringArray(WHITELISTED_IPs), CUSTOMER_IP) {
3       // only accessible to certain ip addresses.
4       ...
5   }
6 }
```

Tablo 1. Örnek dönemeç kullanımları

IP adresleri kullanıcının IP adresini içeriyorsa bildirimler gönderilir. Seçilmiş kullanıcılar haricindeki kullanıcılar bu deneysel özelliği henüz göremezler.

Tablo 2 olası dönemeç veri tipleri ve örnek kullanımlar görülmektedir. Çağrılan fonksiyonların ikinci parametreleri verilen dönemeç bulunamadığında dönecek varsayılan değeri gösterir.

Dönemeç Tipi	**Örnek kullanım**
bool	d.bool("new_optimization_on", false)
integer	d.integer("threshold", 20)
fraction	d.fraction("p", 3.14)
string	d.string("name", "jack")
integerArray	d.integerArray("myintarray", [120, 20, 30])
stringArray	d.stringArray("mystringarray", ["jack", "jill"])
fractionArray	d.fractionArray("myFractionArray", [3.1, 2.7])
struct	d.struct("mystruct", "sendToA" : 100, "sendToB" : 0)

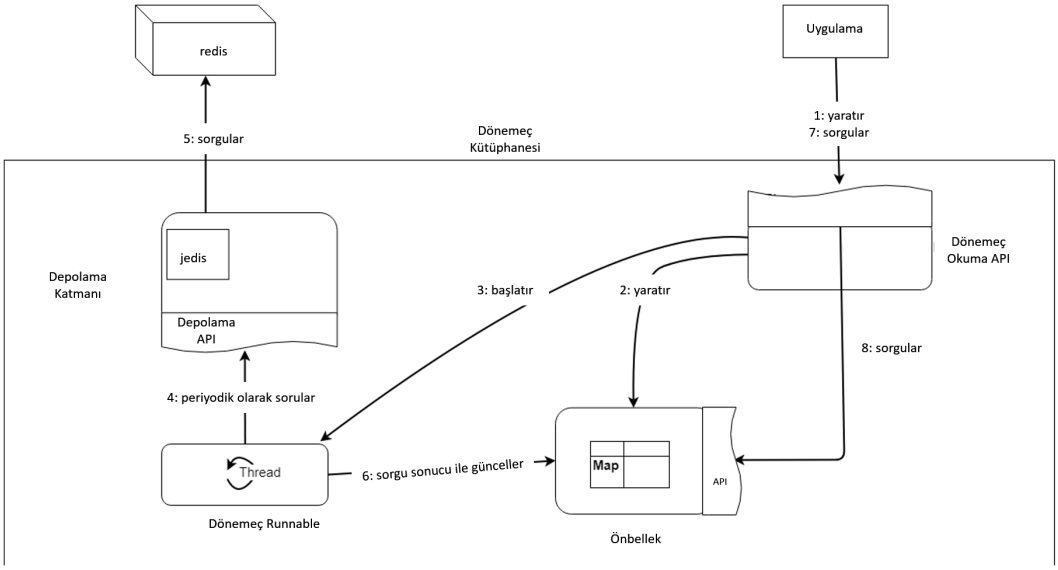
Tablo 2. Desteklenen dönemeç tipleri.

2.1 Sistem Mimarisi

Önerdiğimiz sistemin mimarisini iki bölümde anlatacağız. Birinci bölümde sistemi kullanan uygulamanın dönemeçleri nasıl sorguladığını göstereceğiz. İkinci bölümde ise bir yönetim arayüzü vasıtasıyla dönemeçlerin nasıl yönetildiğinden (yaratılma, güncellenme, zaman aşımına uğrama, silinme) bahsedeceğiz.

Dönemeçlerin uygulama tarafından sorgulanması: Şekil 1'de bir uygulamanın dönemeç kütüphanesi ile nasıl etkileştiği görülmektedir. Dönemeç kütüphanesinin sorgulama ile ilgili kısımları ana hatlarıyla dört bölümden oluşmaktadır. Birinci bölüm uygulamanın doğrudan erişimine açılmış sorgulama katmanıdır. Burada dönemeçlerin okunmasına yönelik çeşitli metotlar bulunmaktadır. İkinci bölüm dönemeçlerin tutulduğu bir anahtar-değer deposu sistemine erişimi ve aradaki iletişim protokolünü gerçekleştirebilen depolama katmanıdır. Anahtar-değer depolama sistemi olarak *Redis* ve *Memcached* gibi birçok alternatif teknolojiler mevcuttur. Bu çalışmada depolama sistemi olarak Redis[8] kullanılmış ve Redis ile iletişimi sağlamak için ise Java dilinde yazılmış *Jedis*[9] kütüphanesi seçilmiştir. Bu kütüphanenin en önemli avantajı birden fazla Redis sisteminin küme (İng. cluster) olarak çalıştırıldığı durumları da destekliyor olmasıdır.

Bir başka bölüm ise dönemeç depolama sisteminden sık periyotlarla dönemeçlerin en son değerlerini indirmektedir. Bu sayede dönemeçlerde olan değişiklikler kısa sürede fark edilebilmektedir. Bu bölümün indirdiği değerler önbellek biriminde saklanır. Uygulama dönemeç sorguları yaptığında ön bellekte olan

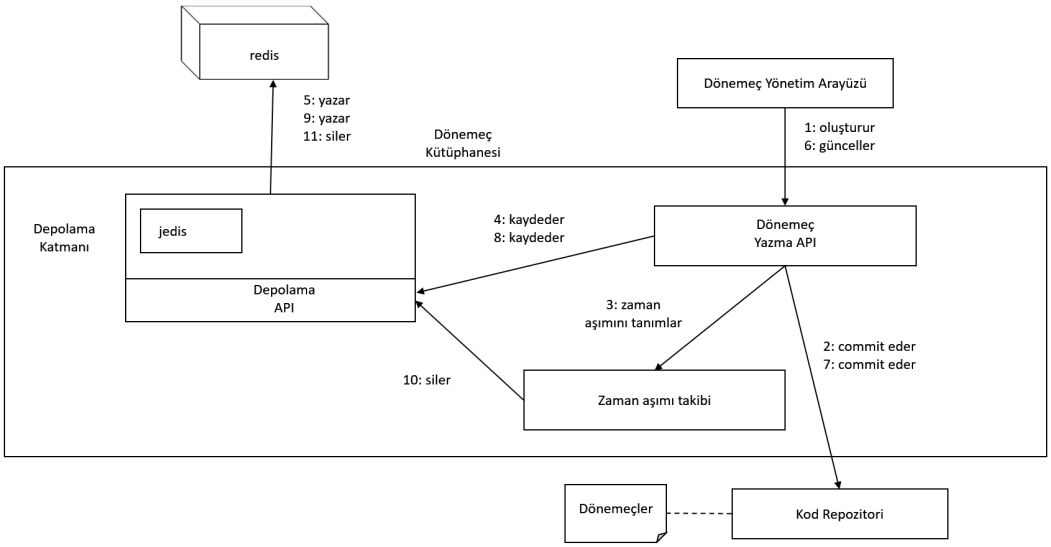


Şekil 1. Dönemeç sorgulama mimari diyagramı

dönemeç değerleri uygulamaya dönülür. Önbellek sayesinde uygulama dönemeç değerlerine bloke olmadan ulaşabilmektedir. Böylece dönemeç kütüphanesi kendisini kullanan bir uygulamaya herhangi kayda değer bir performans maliyeti getirmemesi sağlanır. Uygulama açısından herhangi bir dönemecin değerine ulaşmak hafızadaki herhangi bir değişkenin değerine ulaşmakla aynı maliyettedir.

Şekil 1'de görüldüğü üzere uygulama öncelikle önbellek ve periyodik dönemeç indirme bölümlerini oluşturur (1, 2, 3). Bu noktadan sonra dönemeçler dönemeç depolama sisteminden periyodik olarak indirilir ve önbellek nesnesinde saklanır (4, 5, 6). Uygulama bir dönemecin değerini sorgularsa dönemecin değeri yerel olarak önbellek nesnesi vasıtasıyla cevaplanır (7, 8).

Dönemeçlerin yönetimi: Şekil 2'de bir dönemeçlerin nasıl yönetildiği görülmektedir. Dönemeçler oluşturuldukça dönemeçlerin neler olduğu ve hangi değerleri içerdiği depolama sistemi sorgulanarak bulunabilir. Fakat bu bilgilerin yazılım kaynak kodları gibi bir versiyon kontrol sistemi içerisinde tutulması dönemeçlerin ve dönemeçleri kullanan yazılımların bakımını kolaylaştırır. Bu sebeple bir dönemeç yaratılmak istendiğinde bir versiyon kontrol sistemine ulaşılır ve bu sistemde dönemeç tanımlamalarını içeren dosyalar üzerinde gerekli değişiklikler otomatik olarak yapılır. Bu sayede birden fazla kişinin aynı dönemeçleri birbirine çatışır şekilde değiştirmesi önlenmiş olur. Aynı süreç var olan bir dönemeç güncellenmek istendiğinde de izlenir.



Şekil 2. Dönemeç yönetimi mimari Diyagramı

Ayrıca bir dönemeç için bir son kullanma tarihi belirlenebilir. Bu tarihten sonra dönemeç otomatik olarak anahtar-değer depolama sisteminden silinir. Silinmiş bir dönemeç sorgulandığı durumda dönemecin veri tipine karşılık gelen varsayılan değer dönülür. Son kullanma tarihi yazılımdaki deneysel özelliklerin belirli bir tarihe kadar kullanımda kalmasını sağlar. Tarih geçtikten sonra deneysel özelliklerin etkileri otomatik olarak ortadan kalkar. Bu da bu tür denemelerin yönetimini kolaylaştırır.

Dönemeçlerin Saklanması ve Ağ Üzerinden İletilmesi: Bu çalışmada anahtar-değer depolama sistemi olarak Redis kullanılmaktadır. Dönemeç kütüphanesi ile Redis arasında dönemeçlerin iletilebilmesi ve Redis'te saklanılması için birçok format alternatifi kullanılabilir. Biz bu gerçekleştirmemizde JSON formatını tercih ettik. Tablo 3'te JSON formatında serileştirilmiş dönemeç bilgileri gösterilmiştir. Burada "value" dönemecin değerini ve "type" tipini göstermektedir. Bunun yanında dönemecin zaman aşımına uğrayarak Redis üzerinden silinmesi için bir zaman aşım tarihi görülmektedir ("expiration_date'). Bunlara ek olarak bir dönemecin hangi maksatla oluşturulmuş olduğunun takibi için mesele takip sistemini referanslayan mesele numarası ("issue_number') ve dönemecin değerinin en son ne şekilde güncellendiğinin takibi için kod versiyon sistemini referanslayan güncelleme numarası ("commit") da kaydedilen bilgiler arasındadır.

Dönemeç Tipi	JSON

fraction	{
	"value": 1000,
	"type": "fraction ",
	"commit":"70dbf07ad0a6a28d4aa19b26def4abb451de46a8",
	"issue": "41920"
	"expiration_date": "2018-10-30"
	}

fractionArray	{
	"value": [10.01, 20.02, 30.03],
	"type": "fractionArray",
	"commit":"26def4abb451de46a870dbf07ad0a6a28d4aa19b",
	"issue": "41930",
	"expiration_date": "2018-09-04"
	}

struct	{
	"value": "username" : "john", "counts": [2, 7, 9],
	"type": "struct",
	"commit":"4abb451de46a870dbf07ad0a26def6a28d4aa19b",
	"issue": "42001",
	"expiration_date": "2018-11-02"
	}

Tablo 3. Dönemeçlerin JSON olarak serileştirilmiş hallerine örnekler.

2.2 Dönemeçlerin Kullanımı

Öncelikle dönemeçler bir dönemeç yönetim arayüzü vasıtasıyla tanımlanır. Daha sonra tehlike arz eden veya deneysel olan kod değişiklikleri if-else bloğu içerisine alınır ve if-bloğunun karar şartının doğruluğuna dönemeç kütüphanesi sorgulanarak karar verilir. Burada dikkat edilmesi gereken husus sorgulamadan dönemecin varsayılan değeri döndüğünde güvenli ve kararlı kod bölgesinin çalıştırılıyor olmasıdır. Örneğin aşağıda bu kuralın yanlış bir uygulanışı görülmektedir.

```
if (d.bool(ENABLE_DANGEROUS_CHANGE, true)) { // false olmalıydı.
...//experimental
} else {
...//stable
}
```

Aynı dönemeç, kaynak kodun farklı bölgelerini aktif hale getirmek için kullanılabilir. Bu sayede kaynak kodun birçok bölgesine yayılmış olan bir güncelleme kontrol altına alınabilir. Daha sonra dönemeçlerle etiketlenmiş yazılım, üretim ortamlarına konuşlandırılır. Dönemeç yönetim arayüzü vasıtasıyla dönemeç değerleri güncellenir. Saniyeler mertebesinde, yazılım, güncellenen dönemeç değerlerinin etkisiyle yeni davranışlar göstermeye başlar. Herhangi bir sorunla

karşılaşıldığında dönemeç değerleri yeniden arayüz vasıtasıyla değiştirilerek ilgili güncelleme devreden çıkarılır.

Güncellemelerin hatalara yol açmadığı veya deneysel fikirlerin yeteri kadar olgunlaştırıldığı kararına varıldığında kaynak kodlar içerisinde dönemeç değerleri aranır. if-else blokları içerisine alınmış olan bölgeler temizlenerek yeni özelliklerin varsayılan davranışlar olması sağlanır. Son olarak dönemeçler henüz zaman aşımına uğramadıysa dönemeçler dönemeç yönetim arayüzü vasıtasıyla depolama sisteminden silinebilir.

3 İlgili Çalışmalar

Craig Larman yazılımlarda meydana gelecek değişikliklerle başa çıkmanın yolu olarak "korunmuş değişimler" (İng. protected variations") tasarım desenini önermiştir[10]. Bu desende konfigürasyon değerlerinin dış bir kaynaktan okunması sağlanarak, istendiğinde yazılımların davranışlarının dinamik olarak değiştirilmesi sağlanabilir. Bu çalışmadaki yaklaşım bu desen ile uyumludur. Martin Fowler yazılım davranışlarının kontrolü için özellik bayrakları veya anahtarları (İng. feature flags/toggles) kavramlarını ortaya koymuştur[11]. Bu tür yapıları dört kategoriye ayırmıştır: izin anahtarları (İng. permission toggles), operasyon anahtarları (İng. operations toggles), deney anahtarları (İng. experiment toggles), salıverme anahtarları (İng. release toggles). Bu çalışmadaki yaklaşımımızda bu kategorilerin tamamında özellik anahtarları tanımlamak mümkündür.

Açık kaynak kodlu bir proje olan Togglz, bu çalışmada sunduğumuz sistem ile birçok yönden benzeşmektedir[12], sunduğumuz sistemde birçok farklı tipte dönemeç konfigüre edilebilmektedir. Bu sebeple Togglz yaklaşımına göre Dönemeçler daha esnek bir karar verme stratejisini mümkün kılar. Yaklaşımızla benzeşen LaunchDarkly[13] ve Rollout.io[14] gibi ticari servisler de mevcuttur. Fakat operasyonel gizliliğin korunmasının tercih edildiği ortamlarda veya üçüncü parti bir hizmete bağımlılığın oluşturulmak istenmediği sistemlerde bu ticari ürünlerin kullanımı mümkün değildir.

Yazılımlarda oluşan hataların otomatik olarak tespit edilmesi ve tamir edilmesi ile ilgili Gazzola vd. son yıllarda yapılan çalışmaları derlemişlerdir[15]. Bu çalışmalarda yazılımlar üzerinde statik ve dinamik analizler yaparak otomatik veya yarı otomatik olarak tamir edici yamaların üretilmesi ve sisteme entegre edilmesi hedeflenmektedir. Bu çalışmalardan farklı olarak bizim çalışmamızda hataların tespitinin otomatikleştirilmesi konusunda herhangi bir efor bulunmamaktadır. Fakat çalışmamızda bu çalışmaları tamamlayacak şekilde hatalı olduğunda ciddi hasarlara yol açabilecek değişikliklerin kademeli olarak devreye alınması veya devreden çıkarılmasını sağlayacak bir yaklaşım hedefledik.

Tüm hasarların tamirini tamamen otomatik hale getirmek teknik olarak mümkün değildir. Dolayısıyla üretim ortamlarında ortaya çıkan hataların geliştiriciler tarafından tamir edilmesi uzun süreler alabilir. Bu sebeple rapor edilen hataların arasında doğru biçimde önceliklendirme yaparak en fazla müşteriyi etkileyen hataların belirlenmesi büyük önem arz eder. Kim vd., Firefox ve Thunderbird ürünlerinin hata raporu veritabanları üzerinde çalışan otomatik bir analiz

geliştirmişler ve yüksek başarıyla hangi hataların öncelikle çözülmesi gerektiğini tahmin edebilmişlerdir[5]. Bizim çalışmamız bu çalışmaları da tamamlayıcı niteliktedir. Bazı hataların daha sonra tamir edilmesine karar verilse dahi tüm hatalı davranışlar önerdiğimiz yaklaşım sayesinde hızlıca devreden çıkarılabilir.

Yazılımların otomatik olarak konuşlandırılması üzerine birçok çalışma yapılmıştır. Benson vd. bu çalışmaları derleyen bir çalışma yayınlamıştır[16]. Yazılım sistemlerinin, bulut servis sağlayıcıları üzerinde, alana özgü programlama dilleri vasıtasıyla konfigüre edilmesi ve güncellenmesi için Chef, SaltStack ve Ansible gibi birçok teknoloji geliştirilmiştir[17]. Bu teknolojilerle yazılım unsurlarının nasıl ve nereye konuşlandırılacağı ile birbirleriyle nasıl iletişime geçecekleri belirlenmektedir. Böylece el yordamıyla yapılan birçok adımlar alana özgü dillerle yazılmış programlar çalıştırılarak yapılabilmektedir[18]. Bu çalışmalardan farklı olarak buradaki çalışmamızdaki amacımız sistemlerin konuşlanma konfigürasyonunu kolaylaştırmak değil, sistemler üzerinde çalışan yazılımların davranışlarını dinamik olarak değiştirebilmektir.

4 Sonuçlar ve Gelecek Çalışmalar

Bu çalışmada yazılımların dinamik olarak davranışlarının değiştirilebilmesini ve dolayısıyla tehlikeli veya deneysel sayılabilecek kaynak kod güncellemelerinin yönetimini kolaylaştıracak bir yaklaşım sunduk. Aynı zamanda bu yaklaşımı uygulama tarafından kullanılacak bir kütüphane, dönemeç depolama sistemi ve dönemeç yönetim arayüzü olarak pratiğe dönüştürdük. Bu sistemi finans alanında çalışan bir yazılım firmasında kullandık. Sistem sayesinde hatalı yapıldığında ciddi mali kayıplara yol açabilecek tehlikeli değişikliklerin kademeli olarak devreye alınması sağlanmıştır. Aynı zamanda bu sistemle deneysel sayılabilecek ve müşterinin geri bildirimine muhtaç olan yeni fikirlerin tüm kullanıcıları etkilemeden hayata geçirilmesi ve test edilmesi mümkün kılınmıştır. Önümüzdeki dönemde geliştirdiğimiz sistemimizi açık kaynak kod olarak diğer firmaların da kullanımına sunmayı planlamaktayız. Aynı zamanda sistemimiz içerisine istatiksel analiz yapabilecek bir başka alt sistem geliştirerek, dönemeçlerle yapılan deneysel çalışmalara kullanıcı davranışlarının nasıl etkilendiğini gösteren istatiksel geri bildirimler vermeyi hedefliyoruz.

Kaynakça

1. Tony Savor, Mitchell Douglas, Michael Gentili, Laurie Williams, Kent Beck, Michael Stumm. 2016. Continuous Deployment at Facebook and OANDA. In Proceedings of the IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C).
2. Giovanni Grano, Adelina Ciurumelea, Sebastiano Panichella, Fabio Palomba, Harald C. Gall. 2018. Exploring the integration of user feedback in automated testing of Android applications. In Proceedings of the IEEE 25th Software Analysis Evolution and Reengineering (SANER) International Conference on, pp. 72-83.

3. Simone Scalabrino, Gabriele Bavota, Barbara Russo, Rocco Oliveto, Massimiliano Di Penta. 2017. Listening to the Crowd for the Release Planning of Mobile Apps. IEEE Transactions on Software Engineering (Early Access).
4. Erişim Adresi:
https://dougseven.com/2014/04/17/knightmare-a-devops-cautionary-tale/
Son erişim: Ağustos 2018.
5. Dongsun Kim, Xinming Wang, Sunghun Kim, Andreas Zeller, S.C. Cheung, Sooyong Park. 2011. Which Crashes Should I Fix First?: Predicting Top Crashes at an Early Stage to Prioritize Debugging Efforts. 2011. IEEE Transactions on Software Engineering, Volume: 37, Issue: 3.
6. J. Vlissides vd. Patterns Languages of Program Design. 1996. Reading, MA, Addison-Wesley.
7. D. Parnas. On the criteria To Be Used in Decomposing Systems Into Modules. 1972. Communications of the ACM, Volume: 5, No: 12.
8. Erişim Adresi: https://redis.io/
Son erişim: Haziran 2018.
9. Erişim Adresi: https://github.com/xetorthio/jedis
Son erişim: Haziran 2018.
10. C. Larman. Protected variation: the importance of being closed. 2001. IEEE Software, Volume: 18, Issue: 3.
11. Erişim Adresi: https://martinfowler.com/articles/feature-toggles.html
Son erişim: Ağustos 2018.
12. Erişim Adresi: https://www.togglz.org
Son erişim: Ağustos 2018.
13. Erişim Adresi: https://https://launchdarkly.com/
Son erişim: Ağustos 2018.
14. Erişim Adresi: https://rollout.io
Son erişim: Ağustos 2018.
15. Luca Gazzola, Daniela Micucci, Leonardo Mariani. Automatic Software Repair: A Survey. 2017. IEEE Transactions on Software Engineering (Early Access).
16. James O. Benson, John J. Prevost, Paul Rad. 2016. Survey of automated software deployment for computational and engineering research. In Proceedings of Annual IEEE Systems Conference (SysCon).
17. Liming Zhu, Donna Xu, An Binh Tran, Xiwei Xu, Len Bass, Ingo Weber, Srini Dwarakanathan. Achieving Reliable High-Frequency Releases in Cloud Environments. 2015. IEEE Software Volume: 32, Issue: 2.
18. Matt Callanan, Alexandra Spillane. DevOps: Making It Easy to Do the Right Thing. 2016. IEEE Software, Volume: 33, Issue: 3.