Dağıtık Kalite Güvencesi

Cemal YILMAZ[1] Adam PORTER[4]
[1, 4]Bilgisayar Bilimleri Bölümü, Maryland Üniversitesi, College Park
[1]cyilmaz@cs.umd.edu [4]aporter@cs.umd.edu

Çağatay ÇATAL[2]
[2]TÜBİTAK, Marmara Araş. Merk., Bilişim Teknolojileri Enstitüsü, Gebze,Kocaeli
[2]cagatay.catal@bte.mam.gov.tr

Oya KALIPSIZ[3]
[3]Bilgisayar Mühendisliği Bölümü, Yıldız Teknik Üniversitesi, Beşiktaş, İstanbul
[3] kalipsiz@yildiz.edu.tr

Özet

Kalite güvencesi aktiviteleri, çoğu durumda geliştirici ekip tarafından sadece geliştirme ortamında gerçekleştirilmektedir. Birçok sistem için bu yaklaşım, çeşitli sorunlara neden olduğundan kullanıcıyı da kalite güvencesi aktiviteleri içine alan araçlar ve süreçler geliştirilmektedir. Ancak; yapılan bu çalışmalar sadece belirli koşullar altında çalışmakta ve kalite güvence sürecinin etkin, ölçeklenebilir olmasını sağlayacak olan denetim, işbirliği, koordinasyon işlemlerini içermemektedir. Bu nedenle; genel amaçlı kalite güvence sürecini sağlayacak olan bir yapının kurulması, bu yapı üzerinde kalite güvence senaryolarının işletilmesi, çeşitli araçlarla bu yapının desteklenmesi gerekmektedir. Açıklanan gereksinimler neticesinde, Skoll projesi başlatılmıştır. Bu çalışmada; Skoll projesi kapsamında yapılan analizlerin özeti verilmeye çalışılmış, Türkiye'deki araştırmacıların bu alandaki teknolojiler ve araştırma konularında bilgilendirilmesi amaçlanmıştır. Bunun yanı sıra, Skoll mimarisine 3. parti araçların eklenmesi durumunda elde edilecek kazanımlar bu çalışma neticesinde ortaya konulmuştur.

Abstract

Quality assurance activities are mainly performed by developer team in only developer site. Since this approach produces various problems for many systems, tools and processes are being developed by including the user in the QA process. However; these tools work on the limited conditions and do not include control, cooperation, and coordination policies to make the QA process effective and scalable. For this reason; it is necessary to create a general purpose QA framework, operate QA scenarios on this framework and support it with various tools. Skoll project has been started for these reasons. In this study; it is tried to summarize the analysis of Skoll project and it is intended to inform Turkish researchers on these technologies and research subjects. Furthermore; as a result of this study, it is showed that there will be some benefits when 3rd party tools are integrated to this framework.

1. Giriş

Yazılım kalitesi; yazılım proje planı ile kalite kontrolü yöntem ve araçlarının belirlenmesi, geliştirme süreci durak noktalarında gözden geçirme, kaynak programın test edilmesi gibi üretim

sürecinde kalite kontrolü ve ürünün kullanıcıya teslimi sırasında da kabul muayenesi ile gerçekleştirilir [1].

Test yoluyla hatalar bulunup düzeltildiği gibi, yazılımın spesifikasyonlara uygun olarak işlemlerini yerine getirdiği ve performans gereksinimlerini gerçekleştirdiği kanıtlanmış olmaktadır. Ayrıca; bir bütün olarak yazılımın güvenilirliği ve kalite özellikleri ortaya konmaktadır [2].
Geliştirilen sistem için kalite faktörlerinin değerlendirilmesi zor bir problemdir. Bu amaçla farklı metrikler tanımlanarak, her metriğe bir not verilebilir [3].
Kalite güvence kapsamındaki performans analizleri, testler, modüllerin zamanlama profillerinin çıkarılması gibi aktiviteler çoğu durumda geliştirici ekip tarafından sadece geliştirme ortamında gerçekleştirilmektedir.
Bu yöndeki etkinliklerin faydası; kalite güvence ekipleri tarafından bulunan hataların kolayca anlaşılarak, gerekli çözümlerin kısa sürede geliştirici ekip tarafından oluşturulabilmesidir. Geliştirici ekibin koda tam olarak sahip olması, gereksinim analizinden testlere kadar tüm yazılım mühendisliği süreçlerinin aynı ekip tarafından gerçekleştirilmesi, tespit edilen hatanın aynı koşullarda oluşturulabilir olması geliştirici ortamında kalite güvencesi aktivitelerini kolaylaştırmaktadır. Ancak; testlerin, performans analizlerinin ve diğer aktivitelerin geliştirici ortamında yapılması, harcanan zamana bağlı olarak kalite güvence maliyetlerinde artışa neden olmaktadır. Geliştirilen yazılımın farklı platformlar için çalışması gerektiği durumlarda, yapılacak olan testleri, analizleri göz önüne alırsak yapılan masrafların göz ardı edilemeyecek derecede olduğunu görebiliriz [4, 5, 6]. Bununla birlikte; geliştirme ortamında kullanılan test senaryoları, işletim sistemleri, giriş verilerindeki iş yükü, yazılım versiyon farklılıkları kullanıcıdaki yazılım üzerinde oluşabilecek tüm durumları içermeyebilir. Bunun neticesinde; hiç beklenmedik şekilde yazılımda sorunlar çıkabilmektedir.
Örneğin; son yıllarda performans kriterinin temel alındığı, dağıtık sistemler, gerçek zaman sistemleri, ara katman yazılımları, gömülü sistemler, doğal dil işleme araçları gibi yazılımlar farklı donanımlar, derleyeciler kullanılarak oluşturulmakta ve bu yazılımların farklı işletim sistemlerinde çalışması beklenmektedir [6]. Performansın temel ölçüt olduğu bu sistemlerde, hizmet kalitesini arttırmak için birçok konfigürasyon seçeneği sunulmaktadır. Kullanıcı; iş yükünü saptama, işletim sisteminin özel desteklerini etkinleştirme, kullanmadığı özellikleri kapatma gibi kendi ihtiyaçlarına göre gerekli konfigürasyon parametrelerini belirleyerek yazılımı etkin şekilde çalıştırabilir.
Yüksek seviyede değişikliğe izin veren bu sistemler sayesinde, belirlenen özellikler neticesinde yazılımın kullanıldığı ortamda yüksek performans alınması mümkün olmaktadır. Ancak; gözden kaçan bir parametrenin yazılımın çalışmasını bozabileceği de açık bir gerçektir.
Son yıllarda yazılım sektöründeki değişimlerle birlikte, kullanıcının da kalite güvencesi etkinliklerine katılması gereklidir. Bu değişimler aşağıda açıklanmaktadır [5]:

- *Maliyet ve zaman kısıtları:* Günümüz yazılım endüstrisinde, artan rekabete bağlı olarak ürünlerin en kısa zamanda piyasaya sunulması gerekmektedir. Bunun yanında; geliştirme ve kalite güvencesi için yapılan masrafların da azaltılması beklenmektedir. Kullanıcıların parametrelerin seçimi gibi üzerinde değişiklik yapmak durumunda olduğu yazılımlara para ödemekteki isteksizliği göz önüne alınırsa, kalite güvencesi adımlarının kullanıcı tarafından da yapılması gerektiği söylenebilir.
- *Kullanıcıya özel değişiklik:* Performansın temel ölçüt olduğu sistemler, yazılımın çalıştığı ortama, beklentilere göre değiştirilebilmelidir. Bu ortamı işletim sistemi olarak ele alırsak; yapılacak olan testlerin fazlalığını ve getireceği maliyet artışını hesaba katarak kullanıcıyı bu aktivitelerin içine almamız gerektiği sonucunu çıkarabiliriz.

- *Dağıtık ve artımlı geliştirme:* Geliştirme süreçleri günümüzde, eş zamanlı çalışabilmeyi sağlamak için dağıtık olarak gerçekleştirilebilmektedir. Kimi zaman coğrafi olarak dağıtıklık söz konusu iken, farklı işletme birimleri arasında da bu dağıtıklık kavramından yararlanılmaktadır. Artımlı yazılım geliştirmede ise, her gün yazılıma birkaç parça eklendiği için hata olasılığı artmaktadır.

Oracle, Apache gibi çoğu yazılımda yüzlerce konfigürasyon parametresi bulunmaktadır. Bu parametrelerin tüm işletim sistemlerinde denenmesi durumundaki iş gücünü dikkate alırsak, kullanıcıyı da içine alan kalite güvencesinin vazgeçilmez olduğu sonucunu çıkarabiliriz. Kullanıcının da kalite güvencesi aktiviteleri içinde yer aldığı veya bu aktivitelerin kullanıcı tarafına kaydığı kalite güvencesi türü "*dağıtık kalite güvencesi"* olarak tanımlanmaktadır. Asıl amaç; kullanıcıya yük olmadan kullanıcının kaynaklarından yararlanarak yazılım kalitesini arttırmaktır. Değişen dünyada; bir yazılım sisteminin milyonlarca kullanıcısı olabilmekte ve bu kullanıcıların çoğu internet'e bağlı durumdadır. Bu kullanıcı kaynaklarından yararlanarak yazılım kalitesi arttırılabilir mi sorusuna yanıt aranmaya çalışılmakta ve bu yönde çalışmalar sürdürülmektedir.

2. Mevcut Yaklaşımlar ve Araçlar

Dağıtık kalite güvencesini gerçekleştirebilmek için şimdiye kadar birçok çalışma yapılmış ve bazı araçlar geliştirilmiştir. Bu kapsamda; Netscape Kalite Geribesleme Etmeni, Microsoft XP hata raporlama sistemi gibi çevrimiçi hata raporlama sistemleri oluşturulmuş ve sistemde sorun olması durumunda sistemin içinde bulunduğu durum bilgisini otomatik olarak alabilmeleri sağlanmıştır.
Bu tür araçlar yazılım kalitesini arttırmasına rağmen, sınırlı çözümler sunmaktadır:

- Kalite güvencesinde kullanıcı katılımı sağlanarak otomatik olarak hataların raporlanabildiği araçlar, kalite güvencesinin sadece bir kısmını gerçekleştirebilmektedir. Örneğin; bu tür araçlarla sadece programın çalışmasının çöktüğü andaki hata alınabilmektedir.
- Kalite güvencesine ilişkin tüm aktivitelerin bu tür araçlarla belgelenmesi mümkün olmamaktadır. Bunun sonucunda, geliştiriciler kalite güvencesi sürecinde tam olarak denetimi sağlayamamaktadır.
- ACE+TAO, Mozilla gibi birçok bilinen projede de son kullanıcılara sunulan testler sayesinde, kullanıcıların kendi ortamlarında bu testleri yapması sağlanarak kurulumun başarılı olup olmadığı anlaşılmaktadır. Hatalar, Bugzilla izleme sistemi ile raporlanmaktadır ve bu sistem, CVS gibi konfigürasyon yönetim araçları ile tümleşik çalışabilmekte, ileri raporlama yetenekleri sunmaktadır. Bu sistemin, kalite güvence sürecini belgelemede katkısı olmasına rağmen, sistemin hangi bölümlerinin test edileceği son kullanıcıya bırakılmaktadır. Ancak çoğu durumda, kullanıcı bu test sonuçlarını hata çok kritik değilse geliştiricilere bildirmezler.
  Türkiye'de de birçok kurum ya da firma Bugzilla hata takip sistemini kullanmaktadır. *TÜBİTAK-UEKAE* tarafından başlatılmış olan ulusal dağıtım projesi (*ULUDAĞ*) kapsamında geliştirilen *PARDUS* ürününün de bir hata takip sistemi mevcuttur [7]. Temelinde Bugzilla uygulamasının çalıştırıldığı bu takip sistemi, *Uluzilla* olarak adlandırılmaktadır. Kullanıcı hesabı üzerinden; geliştirme istekleri yapılabilir, hata raporlanabilir, oy kullanılabilir ve yorum yapılabilir. Hata veya diğer konulara ilişkin haberler, kayıt sırasında verilen e-posta adresine gelmektedir.
- Mevcut araçlarla yapılan testlerin sonuçlarına göre, sistemin adaptif olarak test edilmesi veya testlerden öğrenme mekanizmaları ile sonuçların çıkarılması mümkün olmamaktadır. Örneğin; belirli konfigürasyon parametresinde hata veren bir sistem, bu parametrelere yakın değerlerde de hata vermesi olasıdır. Ancak; mevut araçlarla bu tür işlemler gerçekleştirilememektedir.

Yukarıda açıklanan eksikliklerin düzeltilmediği bir kalite güvence sürecinin, etkin olarak çalışması mümkün değildir. Bu nedenle; daha genel amaçlı bir yapı kurularak, bu yapı üzerinde yukarıda açıklanan gereksinimlerin karşılandığı araçlarla dağıtık kalite güvencesi gerçekleştirilmelidir.

3. SKOLL

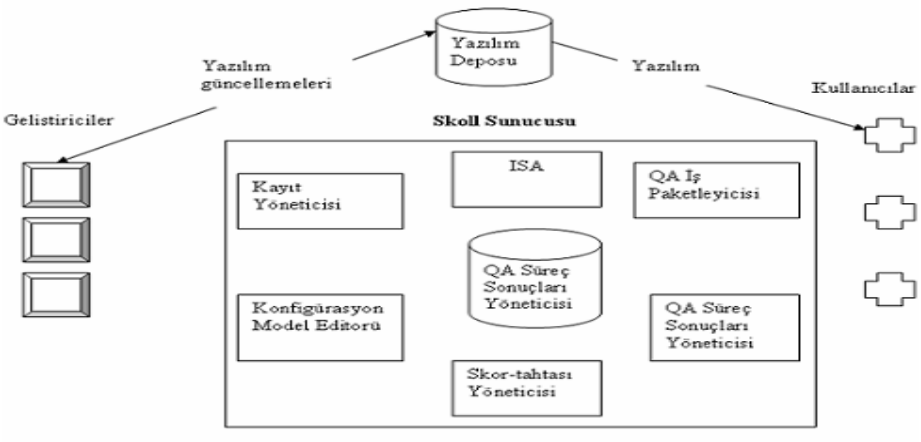
3.1 Giriş

Skoll projesi [8], genel amaçlı kullanılabilecek dağıtık kalite güvence sürecinin genel yapısını oluşturmak için başlatılmıştır. Temel amaç; dünya üzerindeki kullanıcıların hesaplama kaynaklarından yararlanarak yazılım kalitesini dağıtık ve sürekli bir biçimde geliştirebilmektir. Bu kapsamda; dağıtık, sürekli sürecin (DCQA-distributed continuous quality assurance) tüm dünyada binlerce kullanıcı makinesi üzerinde yönetildiği fikri, Skoll projesinin başlangıç noktası olmuştur.
Skoll içerisinde görevler; test, performans değerlendirme gibi kalite güvence aktiviteleridir. Bu aktiviteler, alt görevlere ayrılır. Örneğin; bir alt görev regresyon testleri yaparken, başka bir alt görev zorlama (stress) testlerini gerçekleştirebilir. Skoll içinde, hesaplama düğümleri gönüllü son kullanıcıların makinelerine karşılık gelmektedir. Böylece; farklı işletim sistemlerinde sistemin test edilme imkanı ortaya çıkmaktadır [8]. Buraya kadar açıklanan kısımlar, Skoll'un ana fikrini ortaya koymaktadır. Bu bölümün geri kalan kısmında, daha detaylı bilgi verilecek, Bölüm 3.2'de Skoll mimarisi, Bölüm 3.3'de Skoll konfigürasyon modeli ve Bölüm 3.4'de Akıllı Yönlendirme Etmeni (Intelligent Steering Agent-ISA) açıklanacaktır.

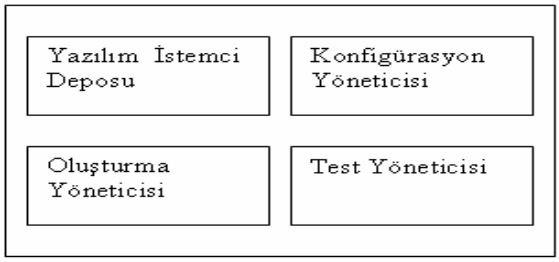
3.2 SKOLL Mimarisi

Dağıtık sürekli kalite güvence sürecini gerçekleştirmek için, Skoll istemci/sunucu mimarisi [8] kullanır. İstemciler, web tabanlı kayıt formunu kullanarak sunucunun içinde bulunan "Sunucu Kayıt Yöneticisi" ile sunucuya kayıt olurlar. Kullanıcılar; işletim sistemi, derleyici gibi istemci platformunun özelliklerini kayıt formu ile belirtirler. Bu bilgi sunucu veritabanında saklanır ve "Akıllı Yönlendirme Etmeni" tarafından kullanılır. ISA bölüm 3.4'de daha detaylı olarak açıklanacaktır.
Kayıt işleminin ardından istemciye sistem üzerinde tek olan bir numara ve konfigürasyon şablonu verilir. Şablon son kullanıcılar tarafından değiştirilerek sunucudan gelecek iş konfigürasyonları için sınırlandırma getirilebilir. Temel olarak iş konfigürasyonu; bir yazılım projesi ile ilişkili olan uygulama kodu, konfigürasyon parametreleri, oluşturma (build) komutları ve regresyon/performans testleri gibi kalite güvencesine özgü kodları içeren bir betiktir (script). İş konfigürasyonu aynı zamanda; istemcinin kendisine atanmış olan kalite güvencesi alt görevini yerine getirebilmesi için, ortam değişkenlerinin atanması, CVS'den yazılımın çekilmesi, loglamanın başlatılması/durdurulması, sistem komutlarının çalıştırılması gibi yardımcı olacak komutları içermektedir. Skoll istemcileri periyodik olarak, sunucudan iş konfigürasyonu talebinde bulunurlar. Sunucu ise istemci platformu, geçerli konfigürasyon uzayının bilgisi ve önceki kalite güvencesi aktivitelerinin sonuçlarına göre düzenlenmiş bir iş konfigürasyonu ile cevap verir.
Skoll istemcisi iş konfigürasyonu sunucudan alınca, komutları aldığı sırada birer birer işletir.
Skoll istemci/sunucu mimarisi Şekil 1'de gösterilmektedir [8]. Skoll'un istemci mimarisi ise, 4 bileşenden oluşmaktadır. Bu bileşenler; Yazılım İstemci Deposu (Software Repository Client), Oluşturma Yöneticisi (Build Manager), Konfigürasyon Yöneticisi (Configuration Manager) ve Test Yöneticisidir (Test Manager). CVS istemci bileşeni, yazılımı CVS deposundan indirmekten sorumludur. Versiyon ve modül ismi gibi bilgiler iş konfigürasyonu içinde gönderilir. Sunucudan gelen komutlar içerisinde, konfigürasyon parametresi ve bu parametrenin tanımlı olduğu başlık

dosyası yer almaktadır. Oluşturma yöneticisi, kayıt sırasında formda belirtilen derleyiciyi kullanarak yazılımı oluşturur. Bu işlemin ardından, Test Yöneticisi gerekli test işlemlerini başlatır. Her iş konfigürasyonu için, Skoll istemcisi tüm aktiviteleri bir günlük (log) dosyasına yazar. Test işleminin bitmesinden sonra bu dosya, sunucuya gönderilir ve sunucu içindeki "Sunucu Kalite Güvence Süreç Sonuçları Yöneticisi" (Server QA process results acquisition manager) tarafından dosya incelenerek veritabanına kayıt edilir. Sunucu Skor Tahtası Yöneticisi (Server Scoreboard Manager) ise web tabanlı bir form ile belirli bir iş konfigürasyonundaki sonucu istemciye gösterebilmektedir. Skoll istemcisi de Şekil 2'de gösterilmektedir [8].



Şekil 1: Skoll İstemci/Sunucu Mimarisi Bileşenleri



Şekil 2: Skoll İstemci Bileşenleri

3.3 Konfigürasyon Uzayı Modeli

Skoll projesinin en önemli adımlarından birisi de, kalite güvence sürecine ilişkin konfigürasyon uzayının bir modelinin oluşturulmasıdır [5]. Model, kalite güvencesi alt görevleri için tüm geçerli konfigürasyonları kapsamaktadır. Seçenekler ve bu seçeneklere izin verilen değerler arasında oluşturulan ikili çiftlerin kümesi konfigürasyonları oluşturmaktadır.
Vi: konfigürasyon seçeneği Ci: konfigürasyon seçeneğinin değeri
{(V1,C1), (V2,C2),...,(Vn,Cn)} kümesi ise olası tüm konfigürasyonları vermektedir. Uygulamada tüm konfigürasyonlar anlamlı olmayabilir. Örneğin; bir konfigürasyon seçeneği bir derleyicide desteklenmezken başka bir derleyici ise bu desteği sunabilmektedir. Bu nedenle; bir seçeneğin başka bir seçeneğe bağlı olarak kısıtlanmasını sağlayan "seçenekler arası kısıtlar" adı verilen

kavram ortaya konulmuştur. Bu kısıtlar (Pi → Pj) olarak gösterilmekte ve Pi doğru ise Pj de doğru olması gerekir şeklinde değerlendirilmektedir. Geçerli bir konfigürasyon, seçenekler arası kısıtlardan hiçbirini ihlal etmemesi gerekir. Skoll projesi kapsamında; geçerli konfigürasyon uzaylarını tanımlamak için bir araç oluşturulmuştur. Tablo 1'de görüldüğü gibi, bazı seçenekler arasında kısıtlar mevcut olabilir. Örneğin; tablodaki gibi AMI seçili iken, CORBA_MSG'de seçili olmalıdır. Bu tablo, ACE+TAO projesi üzerine yapılan çalışmadan alınmıştır [8].

Tablo1: Seçenekler ve Kısıtlar

Seçenek	***Seçenek Değeri***	***Yorum***
DERLEYİCİ	{gcc2.96, SUNCC5_1}	Derleyici
AMI	{1=Evet, 0=Hayır}	Etkinleştirme özelliği
CORBA_MSG	{1=Evet, 0=Hayır}	Etkinleştirme özelliği
Kısıtlar		
AMI=1 → CORBA_MSG=1		

3.4. Akıllı Yönlendirme Etmeni (Intelligent Steering Agent-ISA)

Skoll projesinin en önemli özelliklerinden birisi de ISA'dır. Bir önceki alt bölümde, geçerli konfigürasyon uzayını tanımlamak için bir araçtan bahsetmiştik. Bu bölümün ana teması da, bu uzayda nasıl hareket etmemiz gerektiğini belirlemektir. Bu nedenle; Akıllı Yönlendirme Etmeni (ISA) oluşturulmuştur.
ISA, istemci makinelere alt görev atamasında bulunmaktadır. ISA yeni bir konfigürasyon seçince; istemciye bu iş konfigürasyonunu gönderir ve istemci işlemi gerçekleştirdiği zaman oluşturduğu sonuçları ISA'ya gönderir. Varsayılan durumda; ISA bu sonuçlarla ilgilenmez. Ancak; çoğu kez gelen bu sonuçlardan öğrenme işleminin gerçekleştirilmesi istenir. Skoll sürecinin en önemli özelliklerinden birisi; kullanıcılardan gelen bilgilere göre ISA'nın öğrenebilmesi ve global Skoll sürecini adapte ederek daha etkin hale getirmesidir.
ACE+TAO projesinin incelenmesi sırasında 3 farklı adaptasyon stratejisi kullanılmıştır [8].

- *En Yakın Komşu Stratejisi:* Örneğin; bir test senaryosunun başarısız olduğu durumda, geliştiriciler benzer konfigürasyonların başarısız olup olmayacağını belirlemek isteyeceklerdir. Bir konfigürasyon seçeneği başarısız olduğu durumda, ilgili konfigürasyon kayıt edilir ve başarısız olan konfigürasyondan örneğin 1 birim uzaklıktaki konfigürasyonların test edilmesi sağlanır. 1 birim uzaklik ile belirtilen durum; tüm konfigürasyonlar içerisinden, başarısız olan konfigürasyona göre, sadece 1 seçeneğin değerinin farklı olduğu konfigürasyonların seçilmesidir. Amaç; başarısız olan konfigürasyonların oluşturduğu alt uzayların sınırlarındaki konfigürasyonları test etmektir. Bu bilgi daha sonra, hata karakterizasyon süreci (fault characterization process) tarafından, hataya neden olan seçenekleri belirlemek için kullanılır. Farklı açık kaynak kodlu projelerde yapılan incelemeler sonucu, bu stratejinin oldukça etkin çalıştığı ve olası hataları bulabildiği ortaya konulmuştur [8].
- *Geçici Kısıtlar:*Testler sırasında hataya yol açan bir konfigürasyon tespit edildiği ve bu hatanın kısa sürede çözülemediği durumlarda geçici kısıtlar tanımlamak mümkündür. Problem çözüldüğü zaman, bu kısıtlar ortadan kaldırılır.

- *Alt görevleri Bitirme/Değiştirme:* Bir test senaryosunun başarısız olduğu durumda, sürekli test etmeye devam etmek gereksiz yere zaman kaybına yol açacaktır. Ayrıca çok fazla bilgi de elde edilemeyecektir. Bu nedenle o alt göreve geçici olarak ara vermekte fayda vardır.

4. SKOLL Çalışma Adımları

Üst seviyede, Skoll süreci aşağıdaki şekilde çalışmaktadır [8, 9].

- Geliştiriciler konfigürasyon modelini ve adaptasyon stratejilerini oluşturur. ISA, otomatik olarak modeli planlama problemi haline dönüştürür. Ayrıca; geliştiriciler genel kalite güvencesi alt görev kodunu oluşturur ve bu kod iş konfigürasyonları oluşturulurken özelleştirilebilir.
- Kullanıcı kayıt formunu kullanarak sunucudan, Skoll istemcisini ve konfigürasyon şablonunu elde eder.
- İstemci periyodik olarak veya istek üzerine sunucudan bir iş konfigürasyonu talebinde bulunur.
- Sunucu; içerisinde ISA bir plan oluştururarak iş konfigürasyonlarını istemciye döndürür.
- İstemci iş konfigürasyonunu yürütür ve sonuçları sunucuya döndürür.
- Sunucu bu sonuçları inceler ve tüm adaptasyon stratejilerini çağırarak global sürecin adaptasyonunu sağlar.
- Periyodik olarak sunucu sanal skor tahtası oluşturur ve bu tahta alt görev sonuçlarını gösterir.

5. Yapılabilirlik Çalışması

Bu bölümde; Skoll üzerinde geliştirilmiş olan çeşitli süreçler tanımlanmaktadır. 5.1 alt bölümü ACE+TAO kütüphanesinin Skoll ile analizini, 5.2 hata karakterizasyon sürecini, 5.3 temel etkileri sahneleme sürecini açıklamaktadır.

ACE+TAO kütüphanesi, performansa duyarlı dağıtık yazılım uygulamaları için kullanılmaktadır. ACE, platformdan bağımsız ağ uygulaması gerçekleştirmek için kullanılırken TAO da ACE üzerine oluşturulmuş CORBA ORB gerçeklemesidir [7, 8, 10]. İki proje, 2 Milyondan fazla kod satırı içermektedir. Ayrıca, kütüphane içinde test amaçlı kullanılacak projeler mevcuttur. ACE+TAO projelerinin bakımı, coğrafi olarak farklı yerlerde bulunan 140 geliştirici tarafından gerçekleştirilmektedir. ACE+TAO kütüphaneleri bulunan hatalarla birlikte her gün iyileştirilmekte ve güncellemeler yapılamaktadır. Regresyon testleri sayısı 100'ü aşan makinede yapılmaktadır. ACE'in platformdan bağımsız ağ uygulaması geliştirilmede kullanılması sebebi ile, tüm platformlarda tüm konfigürasyon değerlerini denemek çok zordur. Bu nedenle, Skoll projesinin bu noktada fayda sağlayacağı düşünülmüştür.

5.1. Başlangıç Çalışması

Skoll projesi, ACE+TAO projesi üzerinde denenerek mevcut durumu analiz edilmiştir. ACE+TAO kurulumu için 17 tane derleme zamanı seçeneği saptanmış ve 35 seçenekler arası kısıt ortaya konmuştur. Bu konfigürasyon uzayı, 82.000 geçerli konfigürasyon içermektedir. Konfigürasyonun çok geniş olması sebebi ile, en yakın komşu adaptasyon stratejisi kullanılmıştır [8].

500 konfigürasyon testinden sonra, alt görevi bitirme stratejisi her konfigürasyonun derleme için başarısız olduğunu ortaya koymuştur. ACE+TAO geliştiricileri ile iletişime geçtikten sonra CORBA mesajlaşması ile ilgili 7 seçeneğin probleme yol açtığı saptanmıştır. Bu seçeneklerin kontrolünün derleme anından bağlama aşamasına taşınmasına karar verilmiştir. Bu nedenle, modelden 7 seçenek çıkarılarak 10 seçenek ve 7 kısıttan oluşan yeni bir model elde edilmiştir. Bu model de, 89 geçerli konfigürasyon içermektedir. Yapılan analiz sonunda bu 89

konfigürasyonundan sadece 29'unun hatasız derlendiğini, 60 tanesinde hata olduğunu ortaya koymuştur. Bu çalışmanın yapılmasının nedeni; kütüphaneyi geliştiren ekibin, tüm parametrelerin kütüphanenin kurulumunda etkisini incelememiş olması ve kütüphaneyi kullanacakların bu parametreleri istedikleri gibi değiştirmesi durumunda istenmeyen sonuçların önceden tespit edilmesini sağlamaktır.
Derleme aşamasında olduğu gibi benzer testler, çalışma anındaki seçenekler için de gerçekleştirilmiştir. Skoll projesinin, ACE+TAO gibi büyük kütüphanelerin analizinde kullanılabileceği sonucuna varılmıştır. Birçok hata tespit edilerek geliştiriciler tarafından düzeltilmiştir [11].

5.2. Hata Karakterizasyon Süreci (Fault Characterization Process)

Sistem üzerinde hataya yol açan konfigürasyon seçeneklerinin saptanması ve hatanın tam olarak belirlenebilmesi için hata karakterizasyonu işlemi gerçekleştirilmelidir. Bu amaçla; hata karakterizasyon süreci oluşturulmuştur. Bu sürecin amacı; hataya yol açan konfigürasyon alt uzaylarını geliştiriciler için doğru olarak tanımlamaktır. Örneğin; hangi ayarların spesifik hatalara neden olduğunu ortaya koymak için kullanılabilir. Bu işleme, hata karakterizasyonu (fault characterization) adı verilir.
Örneğin; hata karakterizasyon sürecini CORBA gerçeklemesinde kullanarak, yürütebilir dosyanın Linux'da çalışması durumunda (CORBA Mesajlaşma Desteğinin etkin fakat Asenkron Mesajlaşma Optimizasyonu etkin değilse) soket bağlantılarının zaman aşımına uğradığı tespit edilmiştir. Bu bilgi sistem geliştiricilerine verilerek hatanın gerçek sebebi hızlıca belirlenmiştir. Otomatik olarak hatayı karakterize eden model ile, arama uzayını inceleyerek geliştiricilerin işi oldukça kolaylaştırılmıştır. Otomatik oluşturulan hata karakterizasyonu modelleri yararlı olmasına karşın, tüm konfigürasyon uzayının test edilmek zorunda olması bu yaklaşımın ölçeklenebilirliğini bozmaktadır. Bu bölümde, alternatif bir strateji önerilmekte ve değerlendirilmektedir. Temel fikir konfigürasyon uzayını sistematik olarak örneklemek, sadece seçilen konfigürasyonları test etmek ve sonuç verisi üzerinde hata karakterizasyonunu gerçekleştirmektir.
Kullanılan örnekleme yaklaşımı, "kapsayan dizi" (covering array) adı verilen matematiksel bir nesnenin hesaplanmasına dayanmaktadır [6]. Kapsayan diziler, sıkça programların test giriş kombinasyonları için kullanılmıştır. Yapılan araştırmada, bu diziler sistem konfigürasyon seçeneklerine uygulanmıştır [6]. Kapsayan dizileri; hata karakterizasyon modellerinde kompleks konfigürasyon uzaylarını incelemek için örnekleme stratejisi olarak kullanmak; tüm testleri yaparak elde edilen sonuçlar kadar doğru sonuçlar üretebilir ancak daha az sayıda konfigürasyonla test yapılacağı için daha az zaman harcanacaktır. Bu sayede %50-99 oranında konfigürasyon sayısında azalma sağlanmıştır.

5.3. Temel Etkileri Sahneleme Süreci (Main effects screening process)

Performansa duyarlı sistemler geliştirirken, yapılan değişikliklerin sistemin performansını ne ölçüde etkilediği önemli bir problemdir. Bu problemi çözebilmek için; öncelikle sistemin performansını etkileyen parametreler belirlenmelidir.
Temel etkileri sahneleme; sistem değişikliği sonucunda geniş konfigürasyon uzayı karşısında performans azalmasının hızlıca tespit edildiği bir tekniktir. Bu tekniği; performansı yüksek oranda etkileyen konfigürasyon uzayının küçük bir alt kümesini belirleme şeklinde deneysel bir tasarım sorusuna dönüştürmek mümkündür. Bu noktadan; sistemin değiştiği her zaman, bu önemli seçeneklerin tüm kombinasyonlarını test eder ve böylece tüm konfigürasyon uzayında performansın güvenilir tahmini yapılır. Önemli seçenekler zamanla değişebileceği için süreci yeniden başlatarak önemli seçenekler tekrar ayarlanabilir [12]. Çok geniş kullanıcı topluluğu tarafından kullanılan ACE+TAO projesi üzerinde bu süreç denenmiştir. Bu sayede; temel etkileri sahneleme sürecinin,

güvenilir şekilde performans azalmasında anahtar kaynakları, geleneksel yöntemlerden daha az çabayla ortaya koyabileceği gösterilmiştir [12].

6. Yapılan Çalışmalar ve Değerlendirme

Skoll araştırma projesi olduğu için, yapılabilirlik çalışması da ileride yapılabilecek olan çalışmalar için yardımcı olacaktır. İleriki çalışmalar, daha çok deney ve alt yapının izlenmesi, fonksiyonelliğin arttırılması üzerine yoğunlaşacaktır. Bu kapsamda;

- ACE+TAO projesi ile çalışmaya devam edilecektir. 2 kıtada bulunan temel ACE+TAO geliştiricilerinin sağlayacağı yüzlerce makine ile farklı testler gerçekleştirilecektir.
- Skoll konfigürasyon modelleri zenginleştirilmektedir. Yeni hiyerarşik modeller eklenerek, modele öncelik bilgisi katılacaktır.
- ISA'yi iyileştirme çalışmaları yapılmaktadır. Bu sayede; ISA maliyet modelleri ve olasılık bilgisine göre planlama yapmaya izin verecektir. Örneğin; bir kullanıcının belirli platformlarda belirli sıklıkla istek gönderdiği bilgisi oluşturulabilirse, ISA bu bilgiyi iş konfigürasyonu ayırırken kullanabilir.

Bu çalışmaların yanında, Skoll'un otomatik testleri yapabilecek yetenekte bir mimariye sahip olması nedeni ile bu yönde de çalışmalar yapılabilir:

- İlk aşamada, test amaçlı kullanılan mevcut araçların (3. parti araçlar) analizi yapılarak bu yapıya nasıl entegre edileceği saptanabilir.
- Arayüzler doğru şekilde tanımlanarak, sistemdeki araçların değişmesi durumunda da yeni araçlarla çalışabilmesi sağlanmalıdır.
- Bellek kaçakları (memory leak), modüllerin zamanlama profillerinin çıkarılması (profiling), kapsama analizi (pure coverage) gibi testleri yapan araçlar mevcut yapıya dahil edilerek tüm testlerin otomatik olarak yapılması yoluna gidilebilir.
- 3. parti araçlarının yapıya eklenmesi sayesinde, en basit uygumalardan en karmaşık uygulamalara kadar testlerin otomatikleştirilmesi söz konusu olacaktır.
- Bu sayede; kalite güvencesi noktasında firma içinde önemli aşamalar kaydedilecektir.

7. Sonuç

Bu çalışmada; dağıtık kalite güvencesi ele alınarak gerekliliği ortaya konulmuştur. Mevcut sistemlerdeki yetersizlikler, genel amaçlı bir kalite güvence sistemi için ihtiyaçlar incelenmiştir. Bu kapsamda geliştirilmekte olan Skoll projesinin mimarisi, yararları, yapılabilirlik çalışması analiz edilmiştir. Yakın zamanda Skoll projesinin çıktıları, geliştirilecek olan dağıtık sistemlerde kolayca kullanılabilecek ve sistemlere olumlu katkılar sağlayacaktır. Kullanıcıyı da kalite güvencesinin içine alan bu yaklaşım sayesinde daha kaliteli yazılımlar ortaya çıkması beklenmektedir.
Bu çalışma neticesinde; Skoll projesi kapsamında şimdiye kadar yapılmış olan çalışmalar özetlenmiş ve bu alanda Türkiye'de çalışan araştırmacılara gerek teknolojik gerekse de kavramsal düzeyde bilgilendirme yapılmıştır.
Bunun yanı sıra; 3. parti araçların Skoll mimarisine entegre edilerek, bu sayede basit uygulamalardan karmaşıklara kadar her türlü sistemin kalite güvencesinde kullanılabileceği saptanmıştır.

Referanslar

[1] O. Kalıpsız, Yazılım Mühendisliği, İ.Ü Yayını, 1992.

[2] O. Kalıpsız, Yazılım Geliştirme Sürecinde Kalitenin Sağlanması, T.B.D. 9. Ulusal Bilişim Kurultayı, Eylül 1992, s. 27-34.

[3] M. E. Sarıdoğan, Yazılım Mühendisliği, Papatya Yayıncılık, 2004.

[4] C. Yilmaz, A. Memon, A. Porter, A. S. Krishna, D. C. Schmidt, A. Gokhale, ve B. Natarajan, Preserving Distributed System's Critical Properties--A Model-Driven Approach , IEEE Software Special Issue on the Persistent Software Attributes, Kasım 2004.

[5] A. S. Krishna, D. C. Schmidt, A. Memon, A. Porter ve C. Yilmaz, A Distributed Continuous Quality Assurance Process to Manage Variability in Performance-intensive Software, OOPSLA '04 Workshop on Component And Middleware Performance.

[6] C. Yilmaz, M. Cohen, ve A. Porter, Covering Arrays for Efficient Fault Characterization in Complex Configuration Spaces, Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), Boston, Massachusetts, 2004.

[7] Ulusal Dağıtım Projesi (ULUDAĞ), www.uludag.org.tr/sss.html

[8] A. Memon, A. Porter, C. Yilmaz, A. Nagarajan, D. C. Schmidt, ve B. Natarajan, Skoll: Distributed Continuous Quality Assurance, Proceedings of the 26th IEEE/ACM International Conference on Software Engineering (ICSE), Edinburgh, Scotland, Mayıs 2004.

[9] A. Krishna, D. C. Schmidt, A. Porter, A. Memon ve D. Sevilla-Ruiz, Improving the Quality of Performance-intensive Software via Model-integrated Distributed Continuous Quality Assurance, The 8th International Conference on Software Reuse, ACM/IEEE, Madrid, Spain, Temmuz 2004.

[10] C. Yilmaz, A. S. Krishna, A. Memon, A. Porter, D. C. Schmidt, A. Gokhale, ve B. Natarajan, A Model-based Distributed Continuous Quality Assurance Process to Enhance the Quality of Service of Evolving Performance-intensive Software Systems, Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS), Edinburgh, Scotland, UK, Mayıs 2004.

[11] C. Yilmaz and A. Porter ve D. C. Schmidt, "Distributed Continuous Quality Assurance: The Skoll Project," in Workshop on Remote Analysis and Measurement of Software Systems (RAMSS), (Portland, Oregon), IEEE/ACM, Mayıs 2003.

[12] C. Yilmaz, A. Krishna, A. Memon, Adam Porter, D. C. Schmidt, A. Gokhale, ve B. Natarajan, Main Effects Screening: A Distributed Continuous Quality Assurance Process for Monitoring Performance Degradation in Evolving Software Systems, 27th International Conference on Software Engineering, St. Louis, MO, Mayıs 15-21, 2005.