Ölçümlemeli Test Platformu

Tuncay Şentürk, İsmail Aydemir, Barış Eker

Merkezi Kayıt Kuruluşu, İstanbul

tuncay.senturk@mkk.com.tr, ismail.aydemir@mkk.com.tr,
baris.eker@mkk.com.tr

Özet. Kaliteli yazılım, belirlenmiş olan teknik ve işlevsel gereksinimler ve yazılım geliştirme standartları doğrultusunda geliştirilen yazılımdır. Bilişim dünyasındaki hızlı gelişmelerden dolayı ortaya çıkan rekabette öne çıkmak isteyenler hizmet kalitesini artırmak, ürünlerini daha az hatayla müşteriye sunmak ve bu hatalardan dolayı ortaya çıkabilecek maddi ve/veya prestij kayıplarının önüne geçebilmek için teste verdikleri önemi artırmalıdır. Yazılım projelerinde yapılan temel hatalardan birisi, kullanıcı kabul testleri esnasında bütün senaryoların test edildikten sonra, yapılan yazılım güncellemelerinde sadece değiştiği ve etkilendiği düşünülen kısımların test edilmesidir. Ancak çoğu durumda yapılan değişiklik tahmin edilenden fazla çıkmakta ve test edilmeyen kısımlarda gerek işlevsel gerek performans olarak sorunlar olduğu görülmektedir. Bu tarz sorunların önüne geçilmesi için kullanıcı tarafından yapılan bu testler otomatize edilmeli ve sürüm öncesinde sorunlar tespit edilebilmelidir. MKK'da yapılan Test Çatısı Projesi ile, sistemin işlevsel olarak çalışır olmasına ek olarak, tanımlanmış kaynaklara göre beklenen metrikler doğrultusunda, performans olarak da beklenen düzeyde olup olmadığı anlaşılabilmektedir. Yapılan her değişiklik sonrasında, gerçek ortamdan alınan verilerle entegrasyon testleri yeniden çalıştırılmakta ve ardından alınan sonuçlar, testlerin başarı durumu, ne kadar sürdüğü XML tabanlı olarak sürekli entegrasyon sunucusu üzerinde detaylı ve görsel olarak geçmişteki sonuçlarıyla birlikte raporlanmaktadır.

Anahtar kelimeler. Test, Çatı, Performans, Yazılım, Mimari, Entegrasyon, Değişiklik Yönetimi, KPI

1 Giriş

Kalite, bir ürün ya da hizmetin belirlenen ya da olabilecek ihtiyaçları karşılama kabiliyetine dayanan özelliklerin toplamıdır[1]. Yazılımda kalite ise kişiye veya kuruma göre değişebilen ve bir çok boyutu olan bir kavramdır. Kimine göre yazılımda hataların çok az olması hayati öneme sahip olabilirken bazı durumlarda yazılıma yeni özelliklerin kolayca eklenebiliyor olması ya da ürünün kullanımının kolay olması daha kritik bir kalite ölçüsü olabilmektedir. Bu kaliteyi ölçümleyebilmek için önceden

belirtilmiş olan isteklere uygunluk, standartlara uygunluk ve bunların yanısıra belirtilmeyen ama varsayılan isteklere uygunluk (Örneğin bakım kolaylığı) gibi kriterlere önceden belirlenen metriklerle bakmak gereklidir.

Yazılım bir kişi ya da farklı görevlerde kişilerin bulunduğu yazılım ekibinin ürettiği bir üründür. Yazılım projelerinde yer alan analistlerin, tasarımcı ve yazılımcıların deneyimleri, bilgi seviyeleri ve yetenekleri çeşitlilik göstermektedir. Bu yüzden aynı proje içerisinde çalışan yazılımcıların hepsinden aynı derecede standartlara uygunluk ve yazılım kalitesi beklenemez. Dolayısıyla, geliştirilen yazılımlarda kaynağı ve sebebi değişebilen çeşitli hataların olması kaçınılmazdır. Uygulamanın alanına ve hatanın seviyesine göre bu hataların sonuçları da oldukça değişkendir. Örneğin bir finans uygulamasında yapılan küçük bir hata bile çok büyük miktarlarda para kaybına neden olabilirken başka kuruluşlara hizmet veren kritik bir kamu kurumunun uygulamasının hizmet verememesi ciddi prestij kaybına neden olur. Ya da askeri bir uygulamada yapılan bir hata maddi kaybın yanında can kayıplarına da neden olabilir. Bu nedenle, yazılımların içindeki hataların bulunması ve düzeltilmesi, yazılımın geliştirilmesi kadar önem arz etmektedir. Bu kapsamdaki çalışmalar projenin ilk gereksinimleri belirlenirken başlamalı ve ürün müşteriye teslim edilene kadar devam etmelidir.

Kalite, yazılım tamamlandıktan sonra düşünülmesi gereken bir konu olmaktansa, bütün geliştirme evrelerine yönelik etkinlikler ile sağlanmalıdır. ISO/IEC 9126 standardı [2] yazılım kalite kriterlerini tanımlarken "Test Edilebilirlik" kriterini önemli bir kalite kriteri olarak sunmaktadır. Bunun sağlanabilmesi için, daha proje başlangıcında yazılım gereksinimleri tanımlanırken bu niteliği sağlayacak kalite metriklerinin belirlenmesi, geliştirme aşamasında da bu metriklerin sağlanması için çalışmaların yürütülmesi gerekmektedir.

Bu makalede yazılım kalitesinin geliştirilmesi kapsamında entegrasyon testlerinin otomatize edilmesi amacıyla MKK'da yapılan Test Çatısı Projesi anlatılmaktadır. Makalenin geri kalan kısmı şu şekilde düzenlenmiştir. Bölüm 2'de yazılım test süreçleriyle ilgili genel bilgi verilmiştir. Bölüm 3'de MKK Test Çatısı Projesi hakkında detaylı bilgi verilmekte ve Bölüm 4'de bulunan Sonuç bölümüyle makale tamamlanmaktadır.

2 Yazılım Projelerinde Test Sürecinin Yeri

Yazılımda test, ürünün beklenen seviyede olup olmadığını belirlemek, değilse de istenen ölçüye gelmesini sağlamak için kullanılan bir süreçtir. Test, müşteriye sunulmadan önce ürünün kalitesinden emin olmak, yeniden çalışma (düzeltme) ve geliştirme masraflarını azaltmak, geliştirme işleminin erken safhalarında yanlışları saptayarak ileri aşamalara yayılmasını önlemek, böylece zaman ve maliyetten tasarruf sağlamak, hatalardan dolayı ortaya çıkabilecek prestij kaybını önlemek, müşteri memnuniyetini artırarak sonraki siparişler için zemin hazırlamak, ürünü en kesintisiz ve performansı en yüksek hale getirmek için yapılır.

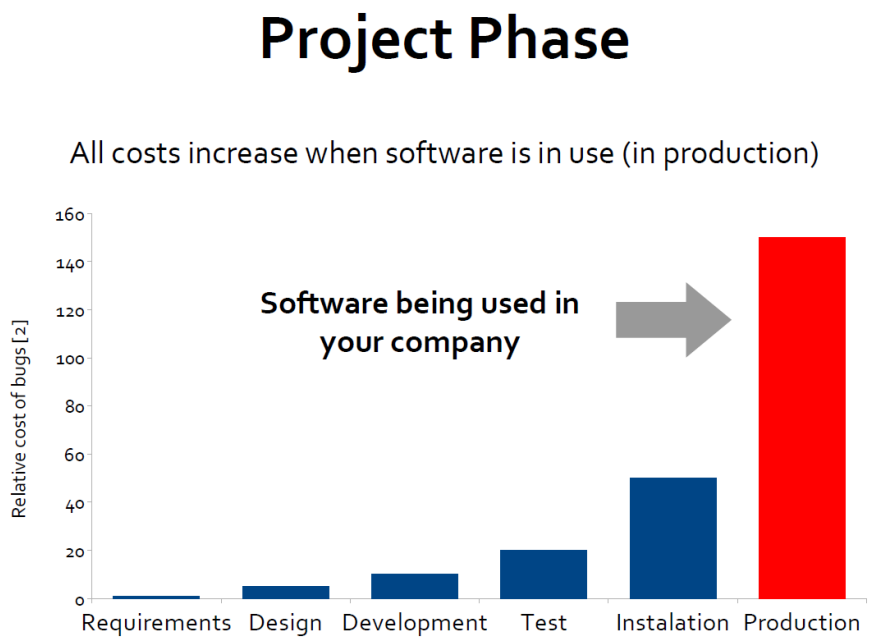
Yazılım test sürecinin proje döngüsüne katılmasıyla, sonuçta ortaya çıkacak olan üründeki hatalar olabildiğince aza indirilebilir. Burada unutulmaması gereken şey ise

mükemmel yazılımın olmadığı ve hiçbir yazılımın %100 test edilemeyeceğidir. Bir yazılımı %100 test etmeye çalışmak hem maddi gerekçelerle hem de pratik olarak mümkün olmadığından, yazılımın beklendiği gibi çalıştığını gösterebilecek sınırlı sayıda ancak özenle seçilmiş bir test kümesinin belirlenmesi ve bu test kümesi üzerinden test yapılması daha mantıklıdır[3]. Bu kapsamda dikkat edilmesi gereken noktalar şunlardır:

- Yazılım mutlaka dinamik olarak çalıştırılarak test edilmelidir.
- Yazılımın neredeyse sonsuz sayıda olabilecek çalışma alanlarının tümünün testi imkansız olacağından; kritiklik düzeylerine göre sıralanıp, yeterli görülen sayıda, en kritikleri test edilmelidir.
- Test edilecek davranışın doğasına uygun ve muhtemel riskleri göz önünde bulunduran uygun şekilde seçilmiş testler gerçekleştirilmelidir.
- Test edilecek yazılımın, kullanıcı beklentilerine, gereksinimlerine ve akla uygun, mantıklı beklentilere cevap verebildiği test edilmelidir.

Hiçbir profesyonel yazılım projesi, canlı ortama çıktığı haliyle kalmamakta ve sürekli bir değişim ve gelişim göstermektedir. Yazılımda ortaya çıkan hataların düzeltilmesi, varolan ürüne yeni özelliklerin eklenmesi ihtiyacı ve ürünü isteyen kurumların kendilerine özgü değişiklikler talep etmelerinden dolayı güncelleme yapmak gerekmektedir. Bu yapılan güncellemelerle varolan sistem olumsuz şekilde etkilenebilmektedir. Burada gözden kaçan önemli bir nokta ise sadece değişiklik yapılan kısımlarda testler yapılırken sistemin bütününde olabilecek etkilerin hesaba katılmamasıdır. Bunun sonucu olarak başlangıçta müşteriye teslim edilen ürün oldukça düzgün ve istenen metriklerle çalışırken yapılan güncellemelerle sistemde başka işlevsel ya da performans sorunu gibi istenmeyen durumlar ortaya çıkabilmektedir.

Yapılan çalışmalar doğrultusunda yazılım projelerinde, hataların giderilmesi için geliştirme işgücü maliyetinin en az üçte biri kadar işgücü planlanması önerilmektedir. Bu işgücü maliyetinin düşürülmesi ve yapılan değişikliklerden erken geri besleme alınabilmesi için testlerin düzenli olarak yapılması gerekmektedir. Yazılım geliştirme sürecinin başlangıcında farkedilen bir hatanın maliyetiyle müşteriye teslim edilmiş bir üründeki hatanın maliyeti arasında Fig. 1 de görüldüğü gibi çok büyük fark vardır[4].



Şekil 1. Yazılım hata maliyetinin zamana göre gösterimi *(Barry Boehm, "Equity Keynote Address" March 19, 2007 alıntılanmıştır)*

Tüm sistemin, işlevsel ve performans anlamında düzgün çalıştığının görülmesi için insan gücüyle yapılan manuel testler oldukça maliyetlidir ve her değişiklikte tekrarlanması gerçek dünyaya pek de uygun değildir. Bunun yerine yazılımın kod kalitesini korumak ve daha ileriye götürebilmek için test kodları yazmak ve yardımcı test araçlarını kullanmak daha etkin bir yöntem olacaktır. Sistem üzerinde yapılan küçük-büyük her değişiklik yan etkilere sebep olabileceği için, sistemin her zaman çalışır olduğu ve değişikliğin başka yerleri etkileyip etkilemediği ancak bu şekilde otomatik testlerle ölçümlenebilir.

3 MKK Ölçümlemeli Test Platformu Projesi

Yazılım testleri, tüm proje boyunca devam eden ve sadece yazlımdaki hataları bulup düzelten değil, bunları daha oluşmadan önlemeyi de hedef alan bir süreçtir.

MKK Test Çatısı projesi ile, sistemin çalışır durumda olup olmadığının yanında, beklenen metrikler doğrultusunda, yani sadece işlevsel olarak çalışmasına değil, aynı zamanda performans olarak da beklenen düzeyde olup olmadığına bakılmaktadır.

3.1 Birim Testi, Entegrasyon Testi ve Performans Testi

Birim, bir yazılımın test edilebilen ve en alt seviyede yeralan küçük parçasıdır. Birim testi ise yazılımın kaynak kodunun belli bölümünün istenen şekilde çalışıp çalışmadığını kontrol etmek için yazılan testlerdir. Birim testlerini yapmayı desteklemek ve kolaylaştırmak üzere hazırlanmış birçok programlama dili için yardımcı test çatısı mevcuttur.

Entegrasyon testi, birim testlerinin bir sonraki aşamasıdır. Birimler arası ya da sistem içerisindeki farklı modül ve bileşenler arası yapılan testlerdir.

Performans testi ise sistemin kendisinden beklenen işlemleri gerçekleştirme hızını, isteklere cevap verebilirliğini, bunları gerçekleştirirken ihtiyaç duyduğu kaynak miktarını ölçmek ve daha iyi hale getirmek için yapılan testlerdir. Bu testler yazılımın daha önce tanımlanmış kaynaklar (donanım) ile çalıştırıldığında beklenen sonuçlarıyla test sonrası elde edilen sonuçların kıyaslanması prensibine dayanır.

Birim testleri sadece yazılım ekibi tarafından gereksinimlere göre projede yazılım geliştirilirken karar verilip yazılırken, MKK iş süreçleri kapsamında diğer testlerin neler olması gerektiğine Analiz, Yazılım ve İş grupları birlikte karar verirler. Belirlenen testler geliştirilir, izlenir, raporlanır, düzenli olarak gözden geçirilir ve uygulamada yapılan değişikliklere göre güncellemeleri yapılır.

3.2 Yazılım Testlerine Hazırlık

Yazılımın herşeyini test etmek mümkün olmadığı için, eldeki kaynakları verimli kullanmak adına hangi testlerin öncelikli olarak yazılacağına karar vermek önem taşımaktadır. MKK'da hangi testlerin yazılacağına risk analizleri ve önceliklendirmeyle karar verilir. Belirlenen dokümante edilmiş kullanım durumu ve test senaryoları düzenli olarak gözden geçirilir, güncellenir, yazılımın ve sistemin farklı bölümlerinde bulunabilecek muhtemel riskler için yenileri yazılır. Buna ek olarak zaman içerisinde gerçek ortamda yaşanan problemler için gerekli düzeltmeler yapıldıktan sonra, benzer durumların yaşanmaması için testlerin yazılması da MKK'da dikkat edilen bir durumdur.

Entegrasyon testlerinin performasının da ölçümlenebilmesi için öncelikle risk analizleri yapılarak ve önceliklerine göre sıralanarak yazılımın anahtar performans göstergeleri (Kep Performance Indicators - KPI) değerleri tespit edilmelidir. Testler sırasında elde edilen sonuçları değerlendirmek için temel ölçüt bu değerler olacağından bu konu oldukça önemlidir. Bu değerler belirlenirken genelde üretim ortamı referans alınır, ancak aynı değerleri başka bir ortamda da beklemek anlamlı olmayacağı için göstergelerin (KPI) testin yapılacağı ortama göre belirlenmesi gerekmektedir. Öte yandan farklı konfigürasyondaki ortamlar için bu tarz bir belirleme yapmak olduça zor olduğu için, MKK'da nihai testlerin yapıldığı ve konfigürasyonu üretim ortamına yakın olan üretim öncesi test ortamları bulunmaktadır. Kullanıcılar nihai kabul testlerini bu ortamda yaptığı gibi, entegrasyon ve performans testleri de otomatik olarak bu ortamda yapılır. Herhangi bir hata olması veya belirlenen göstergelerden sapma olması durumunda ilgili birimler düzeltici aksiyonlar alırlar.

3.3 Yazılım Ölçümleme Çalışmaları

Otomatik testlerle, yazılım ölçümlemesi için yapılan temel çalışmalar şunlardır:

Test Veritabanının Ayarlanması.

Sürekli olarak aynı test verileriyle çalışmak çok faydalı olmayacak ve gerçek dünyaya gözlerini kapamak anlamına gelecektir. En iyi test verilerinin, gerçek ortam verileri olduğu tartışılmaz bir gerçektir. Bu yüzden kullanılacak veritabanındaki verilerin sürekli olarak gerçek ortamdaki verilerden alınması daha yararlıdır. Ancak, bu verilerin kullanılması ayrı bir titizlik gerektirir. Veri gizliliği esas olduğundan, canlı ortamdaki veriler test ortamına maskelenerek taşınmalıdır. Belirlenecek taşıma algoritması doğrultusunda kritik veriler maskelenmeli ve test veritabanına otomatik olarak bu şekilde aktarılmalıdır. Burada veri tutarlığı ise seçilen örnek veriler üzerinden dinamik olarak sağlanır. Örneğin, test için seçilen örnek bir müşterinin bakiyesi alınır, sonrasında hesap üzerinde gerçekleştirilen belli işlemler sonucunda bu müşterinin yeni bakiyesinin oluşması beklenir.

MKK Test Çatısında, tüm süreçlerde olduğu gibi test veritabanı da otomatik bir şekilde maskelenerek oluşturulmaktadır. Değişiklik olduğunda ve testlerin çalışma zamanı geldiğinde öncelikle canlı ortamdaki veritabanı verileri maskelenerek test veritabanı oluşturulur. Geliştirme ortamı veritabanı ile canlı ortamdaki veritabanı yapısı arasında farklılıklar olabileceği için test veritabanının, geliştirme ortamı versiyonuna yükseltilmesi gerekmektedir. Bunun için de tüm veritabanı değişiklikleri (tablo ekleme, kolon silme, vb gibi DDL işlemleri) otomatik olarak veritabanı sistemine yansıtılır.

Sürekli Entegrasyon.

Kalite için sürekli entegrasyon ve sürekli takip şarttır. Entegrasyon testlerinin çalıştırılması ve versiyon kontrol sistemine atılan (commit) her değişiklikte tetiklenmesi sürekli entegrasyon sunucusu (Continuos Integration Server) olan Jenkins üzerinde yapılmaktadır. Yaptığı iş kısaca; kodları versiyon kontrol sisteminden almak, derlemek, testleri çalıştırmak ve istenen sunucuya yüklemektir.

Entegrasyon test uygulama sunucusu üzerine sürüm değişiklikleri otomatik olarak yansıtılır. Örneğin test uygulama sunucusu üzerindeki sürüm ile yazılım geliştirme ortamı üzerindeki sürüm değişiklikleri (kod, xsd, veritabanı değişiklikleri gibi) otomatik olarak aktarılır. Güncel hale gelen uygulama sunucusu üzerindeki yazılım ile veritabanında maskelenen veriler kullanılarak entegrasyon testleri çalıştırılır.

Sürekli entegrasyon ile uzun ve zahmetli entegrasyon süreleri kısalır, yazılan kodun çalıştığı çok daha hızlı görülür, entegrasyon problemleri azaltılıp daha hızlı gerçek ortama geçilir.

Testler Arası Bağımlılık.

Entegrasyon testleri arasında sonucu birbirine bağımlı testler olabileceği düşünülerek geliştirme yapılmıştır. Bu kapsamda, ilk test adımında hesap yaratıp, ikinci testte aynı hesabı yaratmaya çalışırken “Hesap zaten tanımlıdır” hatasının

alınması doğrulanacak ise, ilk testin öncelikli olarak çalışması ve ancak başarılı olma durumunda ikinci teste geçilmesi uygun olacaktır.

Testler arası bağımlılık, bir testin çalışması için sadece bir testin başarılı çalışması olarak düşünülmemelidir. Bir testin çalışması için bağımlı olduğu testler, Şekil 2'de olduğu gibi dizi şeklinde tanımlanarak, bağımlı tüm testlerin başarı kriteri beklenebilir. MKK Test Çatısı'nda testin çalışmaya başlayabilmesi için bağımlı olduğu öncelikli test(ler)in, başarılı ve belirlenen kriterlerde (performans metrikleri gibi) başarılı olarak tamamlanması gerekmektedir. Yani, bir testin başarıyla tamamlanması, işlevini başarıyla yapması anlamına gelmemektedir. Aynı zamanda belirlenen performans kriterlerini de başarıyla geçtiğini göstermektedir.

```
@Test(dependsOnMethods = {"shouldCreateAccount", "shouldAssertThatAccountHasBeenCreated"})
public void shouldDeleteAccount() throws Exception {
    SP sp = new SP();
    sp.setServiceName("ACCOUNT DELETE ACC");
```

Şekil 2. Testler arası bağımlılık

Asenkron Testler.

Birçok kurumsal ve büyük projede gereksinimler gereği kuyruklar üzerinden veya başka yollarla paralel gerçekleşen işlemler mevcuttur. Ancak entegrasyon testlerinin doğası senkron çalışma üzerine kurulduğu için arka planda asenkron çalışan sistem testi yazmak daha karmaşık ve zordur. Mesela, toplu bir hesap açma işlemi yapmak için teste başlamadan önceki ve tamamlandıktan sonraki aktif hesap sayıları karşılaştırılıp zaman ölçümü yapılabilir. Hazırlanan çatı ile, asenkron ve/veya paralel işlemler, girilen başarı kriteri sağlanıncaya kadar aktif olarak bekletme yöntemi ile hem işlevsel hem de performans açısından test edilebilmektedir. Sonsuz döngüye girilmemesi için, bir sonraki bölümde anlatılan zamana bağlı gösterge değeri kullanılmaktadır. Belirlenen performans eşik değerinin üzerine çıkılmadıkça, tüm işlevler tamamlanıncaya kadar aktif bekleme devam eder. Test metoduna, senkron veya asenkron servis testi yapacağı bilgisinin verilmesi, arka planda bu karmaşıklığı test geliştiren kişiden soyutlamaktadır.

Zamana Bağlı Testler.

Bazı testlerin çalıştırılması beklenildiğinden uzun sürebilir. Her bir test metodu için tanımlanabilen zaman aşımı değeri (metodun başına girilerek) aşıldığında, işlem başarılı tamamlansa bile eşik değer aşıldığı için test hatalı sonuçlanacak şekilde çalışma yapılmıştır. Zaman aşımı değeri Şekil 3'te görüldüğü üzere test metodunun başına girilen "timeOut" değeri ile belirlenmektedir. Bu zaman aşımı değeri, senkron olsun asenkron olsun tüm testler için tanımlanabilmektedir.

```
@Test(dependsOnMethods = {"shouldCreateBulkAccountsUsingCsvFile"}, timeOut=20000)
public void shouldDeleteBulkAccountUsingCsvFile() throws Exception {
    logger.info("shouldDeleteBulkAccountUsingCsvFile started ...");
    SP spMessage = createMessageForMkkUser("TH", "UPDATE_ACCOUNT_CSV");

    SP sp = new SP();
    sp.setServiceName("ACCOUNT_UPDATE_ACCOUNT_CSV");
```

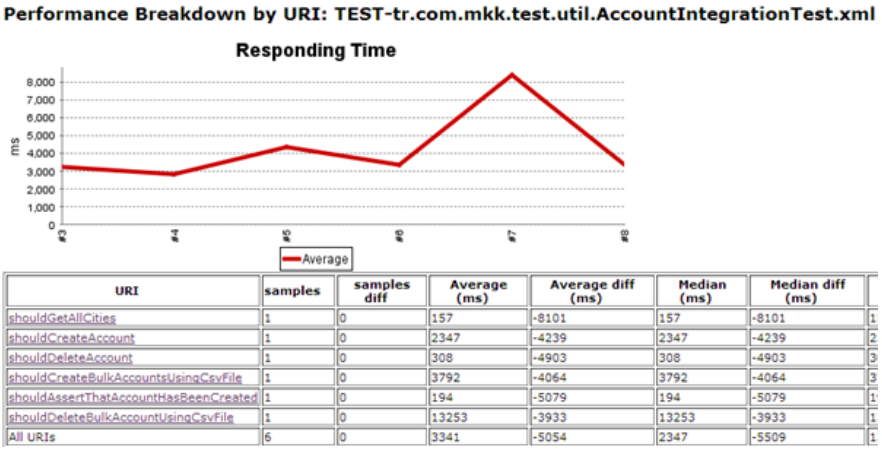
Şekil 3. Testler için zaman aşımı değeri milisaniye cinsinden verilebilmektedir

Testlerin Tamamlanması.

Testlerin tamamlanmasının ardından bir sonraki testte de aynı testlerin başarılı şekilde çalışabilmesi için hazırlanan veritabanının tekrar eski haline gelmesi gerekmektedir. Örneğin, finans projelerinde hesap yaratma testi yaptığınızda, ikinci testte aynı hesap yaratılmaya çalışıldığında "Hesap zaten tanımlıdır" hatasının alınmaması için, testler süresince yapılan tüm işlemlerin geri alınması (rollback) gerekmektedir. Bu yüzden veritabanı katmanında, testlerin çalışmasından sonlanmasına kadar geçen süre içinde yapılan tüm işlemlerin fiziksel commit olmadan, sadece test süresince commitlenmiş gibi davranabilmesi için çalışma yapılmıştır.

Test Sonuçlarının Raporlanması.

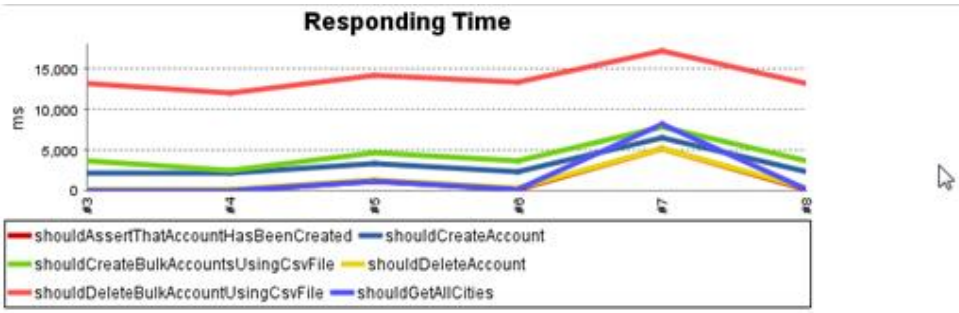
Testlerin çalışmasının ardından, testlerin başarı durumu, ne kadar sürdüğü XML tabanlı olarak sürekli entegrasyon sunucusu üzerinde raporlanır. Oluşturulan XML dosyalarını işleyerek analiz yapan Jenkins eklentisi tarafımızdan geliştirilmiştir. Bu raporlarda, grafiksel ve matris şeklinde tüm test adımları geçmiş verilerle birlikte görüntülenebilir. Böylece hatalı sonuçlanan veya beklenen performans eşiğinden fazla süren testler için detaylı raporlar (hangi aşamalarda ne kadar zaman geçirdiği detayı ile birlikte) raporlanabilmektedir. Örneğin Fig. 4'te, sürekli entegrasyon sunucusu olan Jenkins tarafından tetiklenen ve hesap ile ilgili tüm testlerin toplam aldığı süre gösterilmiştir. Ayrıca hesap entegrasyon testlerinin her birine ait son performans değerleri matris olarak eklenmiştir.



URI	samples	samples diff	Average (ms)	Average diff (ms)	Median (ms)	Median diff (ms)	
shouldGetAllCities	1	0	157	-8101	157	-8101	1
shouldCreateAccount	1	0	2347	-4239	2347	-4239	2
shouldDeleteAccount	1	0	308	-4903	308	-4903	3
shouldCreateBulkAccountsUsingCsvFile	1	0	3792	-4064	3792	-4064	3
shouldAssertThatAccountHasBeenCreated	1	0	194	-5079	194	-5079	1
shouldDeleteBulkAccountUsingCsvFile	1	0	13253	-3933	13253	-3933	1
All URIs	6	0	3341	-5054	2347	-5509	1

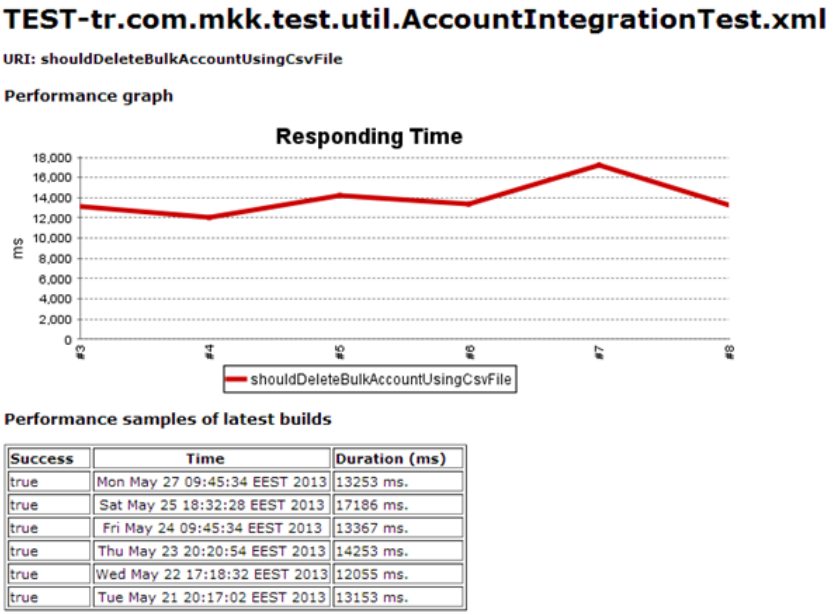
Şekil 4. Entegrasyon test sınıfı bazında performans değerleri

Hesap entegrasyon testleri sınıfında yer alan her bir metod için geçen süre, MKK tarafından geliştirilen Jenkins eklentisi üzerinde Fig. 5'te görüleceği üzere rapor-lanmaktadır.



Şekil 5. Tüm test metodlarının performans grafikleri

Ayrıca, Fig. 4'te yer alan her bir test metodu için ilgili link tıklandığı takdirde Fig. 6'da yer alan, test performans grafiği ve önceki tüm testler için performans matrisi listelenmektedir.



Şekil 6. Hesap silme test metodunun performans grafiği

Her bir işlev testi özet olarak aşağıdaki formül ışığında yaşamaktadır.

$$test\ senaryosu + zaman\ gostergesi + basari\ kriteri + (asenkron / senkron) + bagimli\ \ oldugu\ \ testler = test\ sonuc + performans\ analizi$$

4 Sonuçlar

MKK da yazılım ve sistem geliştirme sürecinde kaliteyi ön plana çıkaracak şekilde Yazılım, Analiz ve Sistem ekipleri organize olmuştur. Yazılımda kalite, oluşturulan Test grubunun sorumluluğunda test edilir, ölçümlenir ve raporlanır. Tüm bu işlemler ihtiyaçlara göre tasarlanmış, gerekli kaynakları hazırlanmış, başlangıçtan itibaren planlanmış ve tamamen otomatize edilmiş ortamlarda gerçekleştirilir. Bu kapsamda MKK Test Çatısı Projesi geliştirilmiştir. Böylece;

- Yazılım süreçleri daha izlenebilir ve denetlenebilir hale gelmiştir.
- Yazılım hatalarının olabildiğince erken safhada tesbit edilmesiyle her türlü kaybın önüne geçilmiştir.
- Yapılan her değişiklik tüm sisteme çok hızlı bir şekilde yansıtılıp sistemin tümüne olan etkisi çok kısa sürede farkedilebilmektedir.
- Testlerden elde edilen sonuçların analiz edilmesiyle ortaya çıkan görsel verilerle aksiyon alınabilmektedir.

- Zor olan ve farklı durumları kolayca test edilebildiği bir ortam oluşturulmuştur.
- Test sonuçları ve testler sırasında oluşan herhangi bir hata detaylarıyla birlikte istenilen gruba otomatik olarak bildirilmektedir.

Kaynaklar

1. Mina Özveren, "Toplam Kalite Yönetimi ve Toplam Kaliteye Ulaşmada Önemli Bir Araç-ISO 9000 Kalite Güvence Sistemleri", 1996.
2. ISO/IEC 9126 Software Engineering - Product quality
3. SWEBOK, Guide to the Software Engineering Body of Knowledge, 2004 Version
4. Barry Boehm: EQUITY Keynote Address, March 19th, 2007