Statik ve Dinamik Analizler ile Hesaplanan Risklere Dayalı Olarak Test Modellerinin İyileştirilmesi

Ceren Şahin Gebizli[1], Duygu Metin[1], Hasan Sözer[2]
[1] Vestel Elektronik, İzmir, Türkiye
{ceren.sahin, duygu.metin}@vestel.com.tr
[2] Özyeğin Üniversitesi, İstanbul, Türkiye
hasan.sozer@ozyegin.edu.tr

Özet. Model bazlı test teknikleri, sistem kullanım modelinden test senaryolarının otomatik olarak oluşturulmasını sağlayarak verimliliği artırmaktadır. Prensip olarak sonsuz sayıda test senaryosu oluşturmak mümkündür; ancak bu senaryoları sınamak için kaynaklar kısıtlıdır. Dolayısıyla, kullanılan modelin içeriği ve test senaryosu oluşturma teknikleri, etkin bir şekilde hataların tespit edilmesini sağlamalıdır. Bu çalışmamızda, model bazlı test için kullanılan model içeriği ve model parametrelerinin iyileştirilmesine yönelik özgün bir yaklaşım öneriyoruz. Yaklaşımımızda kullandığımız Markov zincirleri, istatistiksel verileri baz alarak, model parametrelerini, hata riski yüksek olan senaryolara ağırlık verecek şekilde güncellememize olanak vermektedir. Statik kod analiz teknikleri ve kullanım profili analizlerini değerlendirerek sık kullanılan ve hata ile karşılaşılma olasılığı yüksek olan işlevleri belirliyoruz. Model içeriğini bu işlevleri test etmek üzere oluşturuyoruz. Dinamik analiz sonuçlarına göre hata oluşumuna yatkın olan işlevlerin, oluşturulan test senaryolarına dâhil edilme olasılıklarını artıracak şekilde model parametrelerini güncelliyoruz. Bu yöntem ile gerçek bir Akıllı TV sistemi yazılımı için oluşturulan test senaryolarını kullandığımızda, hata tespit etkinliğinin arttığını gözlemledik.

Anahtar kelimeler: Tasarım Doğrulama, Yazılım Sınama ve Doğrulama, Test Otomasyonu, Kullanım Modeli Bazlı Test, Test Verimliliği, Kullanım Profili, Test Senaryosu Oluşturma, Kod Analizi, Yazılım Güvenilirliği

1 Giriş

Tüketici elektroniği alanında müşterilerin temel beklentisi, aldıkları ürünlerin son teknolojileri içermesi ve aynı zamanda ürünün işlevlerinin hatasız bir şekilde çalışması yönündedir. TV, uydu alıcı, beyaz eşya ve cep telefonları gibi sayılabilecek bu ürünler artık eski elektro-mekanik yapılarından uzaklaşıp karmaşık yazılım sistemleri haline dönüşmektedirler. Ürünlere son teknoloji özellikler eklendikçe test edilmesi gereken işlevlerin sayısı artmaktadır ve test süresinin kabul edilemez değerlere yükselmesine sebep olmaktadır. İşgücünün ve zamanın verimli kullanılması için test otomasyonu sağlanması yönünde çalışmalar yapılmıştır. Ancak test otomasyonu, test verimliliği için tek başına çözüm olmamaktadır. Ancak doğru

tasarlanmış, hata tespit oranı yüksek test senaryolarının otomatik olarak sistem üzerinde uygulanması, maliyetlerin azaltılmasını, ürün kalitesinin yükseltilmesini, kaynak ve zamanın daha etkin kullanılmasını sağlayabilmektedir.

Model Bazlı Test (MBT) [1], test senaryolarının otomatik olarak oluşturulmasını sağlayarak verimliliği arttıran test tekniklerinden biridir. Bu teknikte öncelikle sistem gereksinimleri analiz edilerek, test edilecek sistemin beklenen davranışları, kullanıcı aksiyonları ve girdileri modellenir. Bir MBT aracı ile tasarlanan model üzerinden otomatik olarak test senaryoları oluşturulur.

MBT ile prensip olarak sonsuz sayıda test senaryosu üretmek mümkündür, ancak bu senaryoları sınamak için kaynaklar kısıtlıdır. Verimin yüksek olması için otomatik oluşturulan test senaryolarının hata tespit etkinliklerinin artırılması gerekmektedir. Dolayısıyla, test senaryosu oluşturmak için kullanılan modelin içeriği ve kullanılan model parametreleri kritik öneme sahiptir.

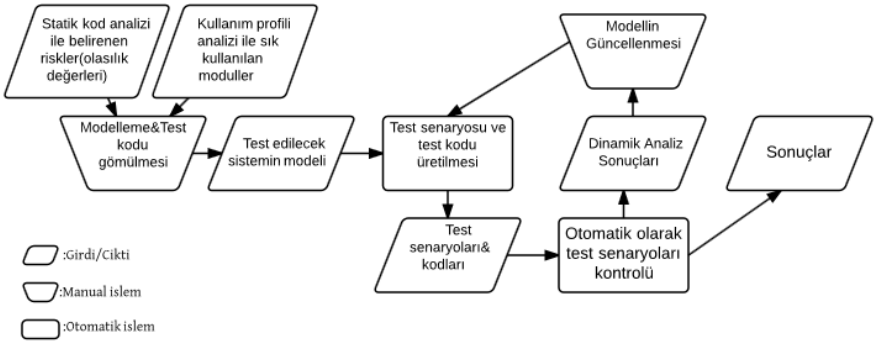
Bu çalışmada, hata tespit etkinliği yüksek test senaryolarının MBT ile oluşturulabilmesini sağlamak üzere, sistem modeli geliştirmek ve iyileştirmek için bir yöntem öneriyoruz. Yöntemimizde, sistem modelini Markov zincirleri [10][12] ile oluşturuyoruz. Bunun için öncelikle kullanım profili [6] ile statik kod analizlerinin sonuçlarını temel alarak, sık kullanılan ve hata oluşumuna yatkın olan işlevleri değerlendiriyoruz. Bu değerlendirmelere göre modelde temsil edilen durumlar ile bu durumlara geçiş olasılıklarını belirliyoruz. Geliştirilen model ile oluşturulan test senaryoları çalıştırılırken, sistem üzerinde dinamik analizler gerçekleştiriyoruz. Dinamik kod analizi sonuçlarına göre hata oluşumuna yatkın olan durumları tekrar belirliyoruz ve sistem modeli üzerinde durum geçiş olasılıklarını güncelliyoruz.

Önerdiğimiz yöntem ile gerçek bir Akıllı TV sistemi yazılımı için geliştirilen bir modele dayalı olarak test senaryoları oluşturduk. Bu test senaryolarını çalıştırırken gerçekleştirdiğimiz dinamik analizleri değerlendirerek modeli güncelledik. Güncellenen modele dayalı olarak tekrar test senaryoları oluşturduk ve bu süreci 3 kez tekrarladık. Her tekrarda, hataya daha yatkın olan yazılım işlevlerini kapsayan test senaryolarının sayısı artarken, toplam test senaryolarının sayısının azaldığını ve buna rağmen tespit edilen hata sayısının arttığını gözlemedik.

Bir sonraki bölümde yöntemimiz detayları ile açıklanmaktadır. 3. Bölümde ise yöntemimizi sınamak için gerçekleştirdiğimiz vaka analizi ve sonuçları sunulmuştur. Daha sonra literatürde yer alan ilgili çalışmalar 4. Bölümde açıklanmıştır ve son olarak 5. Bölümde bildirinin katkıları özetlenmiştir.

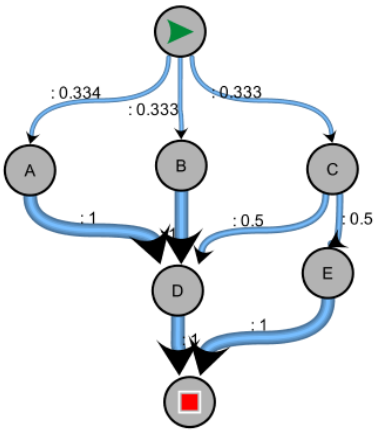
2 Yöntem

Yaklaşımımızda yer alan manuel ve otomatik işlemler ile bu işlemlerin girdileri ve çıktıları, Şekil 1'de bir akış şeması ile gösterilmektedir.



Şekil 1. MBT için model oluşturma ve iyileştirme yaklaşımı.

Şekilde gösterilen süreçte, ön çalışma ile elde edilen iki adet girdi bulunmaktadır: 1) Bir statik kod analiz aracı kullanılarak elde edilen, her bir yazılım modülü başına raporlanan uyarı/hata sayıları, 2) Her yazılım işlevinin ne sıklıkta kullanıldığı bilgisini içeren kullanım profilleri analizi [6] sonuçları. Bu girdiler sistem modelini Şekil 2'de örneklendiği gibi bir Markov zinciri [10][12] şeklinde oluşturmak için kullanılmaktadır. Markov zincirinde en sık kullanılan sistem işlevlerine ilişkin durumlara (örnek modeldeki A, B, C durumları gibi) yer verilmektedir. Bu durumlar arasındaki geçişler için kullanılan olasılık değerleri [11] ise (örnek modelde C durumundan D durumuna geçiş için belirlenmiş 0.5 olasılık değeri gibi), ilgili işlevlere tekabül eden modüller için statik kod analiz aracı tarafından raporlanan uyarı sayılarına göre belirlenmektedir. Fazla sayıda uyarı, yüksek risk taşıyan bir modüle işaret olarak görülmektedir. Eğer bir durumdan n farklı duruma ($n > 1$) geçiş söz konusu ise, her alternatif durum geçişi için, hedef durum i ($1 \leq i \leq n$) ile ilgili modüller için raporlanan uyarı sayısı, W_i olsun. Bu halde, i durumuna geçiş olasılığı, $P(i) = W_i \,/\, \sum_{k=1..n} W_k$ olarak hesaplanmaktadır. Alternatif durumlar için geçiş olasılıkları toplamı 1'e eşittir, yani $\sum_{k=1..n} P(k) = 1$.



Şekil 2. Markov zinciri olarak tasarlanmış örnek sistem modeli.

Oluşturulan model ile birlikte, test durumu ve geçişlerine ilişkin test adımlarının betikler (script) halinde tanımlanması gerekmektedir. Böylece, MBT aracı ile bir durum dizini halinde oluşturulan test senaryoları (örnek modelde mümkün olan {A, D}, {B, D}, {C, D} ve {C, E} dizinleri gibi), ilgili test adımları ile eşleştirilip otomatik olarak çalıştırılabilir. Uyguladığımız yöntemde test adımları modelin içine betikler olarak gömülmüştür. Bu betikler Vestel Elektronik firması içinde geliştirilmiş olan bir test otomasyon aracı[4] ile sistem üzerinde çalıştırılabilmektedir.

Sistem modeli oluşturulduktan sonra, bu model bir MBT aracına girdi olarak kullanılıp otomatik olarak test senaryolarının oluşturulması sağlanmaktadır. MBT aracı test senaryolarını oluştururken durumlar arası geçiş olasılıklarını dikkate almakta, böylece geçiş olasılığı yüksek olan durumların daha sık teste tabi tutulmasını sağlamaktadır. Oluşturulan test senaryolarında yer alan test adımları için betikler önceden tanımlanmış olduğundan, bu test senaryoları otomatik olarak çalıştırılabilecek durumdadırlar.

Oluşturulan test senaryoları sistem üzerinde çalıştırılırken dinamik analizler gerçekleştirilmektedir. Bu analizler farklı yazılım modülleri tarafından kullanılan bellek alanlarını kontrol etmektedir ve bellek açıklarını tespit etmektedir. Yazılım testi sırasında tespit edilen bellek açıklarına sebep olan yazılım modülleri, hataya sebep olma riski taşıyan modüllerdir[1]. Bu sebeple, modüllerin sebep oldukları bellek açıklarının miktarı, sistem modelindeki durum geçiş olasılıklarını güncellemek için kullanılmaktadır. Eğer bir durumdan n farklı duruma ($n > 1$) geçiş söz konusu ise, her alternatif durum geçişi için, hedef durum i ($1 \leq i \leq n$) ile ilgili modüllerin bir önceki test sırasında sebep olduğu bellek açığı miktarı, M_i olsun. Bu halde, i durumuna geçiş olasılığı, $P(i) = M_i / \sum_{k=1..n} M_k$ olarak güncellenmektedir. Alternatif durumlar için geçiş olasılıkları toplamı 1'e eşit olarak kalmaktadır, yani kısıt olarak $\sum_{k=1..n} P(k) = 1$ şartı sağlanmaya devam etmektedir.

[1] Bu çalışmada, sadece bellek açıkları ve yanlış bellek kullanımı sebebi ile oluşan hataların tespitine odaklanılmıştır.

Sistem modeli güncellendikten sonra tekrar MBT aracı için girdi olarak kullanılarak yeniden test senaryoları oluşturulmaktadır. Yeniden oluşturulan test senaryoları ile yazılım testi aynı yazılım versiyonu ile tekrar edilir. Dinamik analizler, model güncelleme, test senaryoları oluşturma ve yazılım test süreci kaynaklar elverdikçe bir döngü halinde devam etmektedir. Böylece her seferinde hataya yol açma olasılığı daha yüksek olan test senaryolarının sistem üzerinde daha çok sınanması sağlanmaktadır. Özyineleyen süreç kapsamında güncellenen ve test kapsamına dâhil olan senaryolar, giderek bellek açıklarına sebep olan işlevlere odaklanmakta ve bu açıklarla ilgili hataların ortaya çıkma ihtimalini artırmaktadır.

Bir sonraki bölümde, gerçek bir TV sistemi yazılımının test süreci kapsamındaki bir vaka analizi ile yöntemimizi değerlendiriyoruz.

3 Vaka Çalışması – Akıllı TV Sistemleri

Yöntemimizi değerlendirmek üzere, Vestel Elektronik firması tarafından geliştirilmekte olan Akıllı TV sistemlerinin test süreçleri kapsamında bir vaka analizi gerçekleştirdik. Gelişen yeni teknolojiler ve hızlı İnternet bağlantısı ile televizyon sistemleri akıllı televizyonlar haline dönüşmüştür. Bu sistemler, gelişmiş multimedya servisleri gibi geniş kapsamlı birçok interaktif servisler ve uygulamaları içermektedir (örnek olarak IPTV, interaktif TV, Hybrid Broadcast Broadband TV, Pay TV, Portal uygulamaları; Facebook, Twitter, Youtube ve daha birçok İnternet uygulamaları sayılabilir). Artık kullanıcılar TV sistemlerini bir kanal izlemek için kullanmanın yanı sıra, TV program takipleri, bir program izlerken diğer programları kaydedebilme, İnternet işlemleri, online video izleme, harici cihazlar bağlayarak TV'leri çok amaçlı kullanabilme gibi birçok ek işlem yapabilmekteler. Bu sebeple TV sistemlerinde kullanılan yazılım boyutu artmış ve karmaşık bir hal almıştır.

Vaka analizi çalışmamız kapsamında, önerdiğimiz yöntemi (Şekil 1) oluşturan işlemleri sırasıyla Akıllı TV sistemlerinin testlerini gerçekleştirmek üzere uyguladık ve bulgularımızı kaydettik. Öncelikle gerçek TV kullanıcıların hareketlerini kayıt altına alarak ve analiz ederek, kullanıcıların günlük kullanım hareketlerini yansıtan bir sistem modeli hazırladık. Tüm yazılım sistemini statik kod analizine tabi tuttuk ve hatalı olarak raporlanan modülleri, sistem modelinde yer alan durumlar ile eşleştirdik. Statik kod analizi sonucunda raporlanan hata sayılarına göre göre modeldeki durumlar arası geçiş olasılıklarını tayin ettik. Bir MBT aracı ile geliştirdiğimiz modeli kullanarak test senaryoları oluşturduk. Bu test senaryolarını otomatik olarak çalıştırırken dinamik kod analizleri gerçekleştirdik. Analiz sonuçlarına göre, modelde yer alan her durum için oluşan bellek açığı miktarlarını ölçtük ve modeldeki durumlar arası geçiş olasılıklarını ölçümlerimize göre güncelledik. MBT ile test senaryolarının oluşturulması, oluşturulan test senaryolarının sistem üzerinde çalıştırılması, dinamik analizlerin gerçekleştirilmesi ve modelin analiz sonuçlarına göre güncellenmesi işlemlerini 3 kez tekrar ettik. Sonuç olarak gerçekleştirdiğimiz testlerin hata tespit etkinliğini giderek geliştirdiğini gözlemledik. Aşağıda, vaka analizi sırasında

gerçekleştirdiğimiz işlemleri ve elde ettiğimiz sonuçları detaylı bir şekilde açıklıyoruz.

3.1 Kullanım Profili Analizi

Kullanım profili bilgilerini toplayabilmek için, 10 adet Akıllı TV sistemi, 10 farklı son kullanıcıya dağıtılmıştır. Kullanıcıların 10 gün boyunca bu sistemleri kullanmaları sağlanmıştır. Bu süre zarfında kullanıcıların sistem ile etkileşimi ve gerçekleştirdikleri işlemler kayıt altına alınmıştır. Kaydedilen bilgiler içinden "durum" ve "geçiş" bilgileri analiz edilmiştir. Bu durumlar kullanıcının girdiği herhangi bir menünün ya da kaynağın bilgisini içermektedir. Geçişler ise o menülere nasıl girildiğinin bilgisini vermektedir. 10 günün sonunda alınan veriler, durum-geçiş bilgileri izlenebilirlik matrisi oluşturularak analiz edilmiştir. Böylece kullanıcıların hangi işlevlerin kullanımına yoğunlaştıkları bilgisi ortaya çıkarılmıştır.

3.2 Statik Kod Analizi ve Test Modelinin Hazırlanması

Kullanıcılardan alınan bilgilere göre sık gerçekleştirilen işlemler, "kanal geçişleri, rehbere girme, rehber üzerinden kayıt kurma, USB üzerinden video izleme ya da başka içerikler görüntüleme, Teletext uygulamasına girme, kanal listesi operasyonları, İnternet uygulamalarına girme, Youtube üzerinden video izleme, başka kaynakları görüntüleme, kanallar arası navigasyon, kanal izleme, ses açma/kapatma" olarak belirlenmiştir. Bu işlemleri kapsayacak şekilde bir sistem modeli Şekil 3'te görüldüğü şekilde hazırlanmıştır. Modelleme aracı olarak MaTeLo[2] aracı [9] kullanılmıştır. Şekil 3'te gösterilen model en üst seviye modeldir. Bu modelde yer alan farklı her durum için ilgili durumda gerçekleştirilen işlemleri tanımlayan alt modeller bulunmaktadır. En üst seviye modelde, sık kullanılan temel işlevlere tekabül eden 11 adet duruma yer verilmiştir: HDMI (SSw), SCART (SSw), HBBTV, Portal (Pr), Youtube (Yt), Edit Channel List (ChO), Navigation Channels (ChL), EPG, Teletext (TXT), PVR, Media Browser (MB).

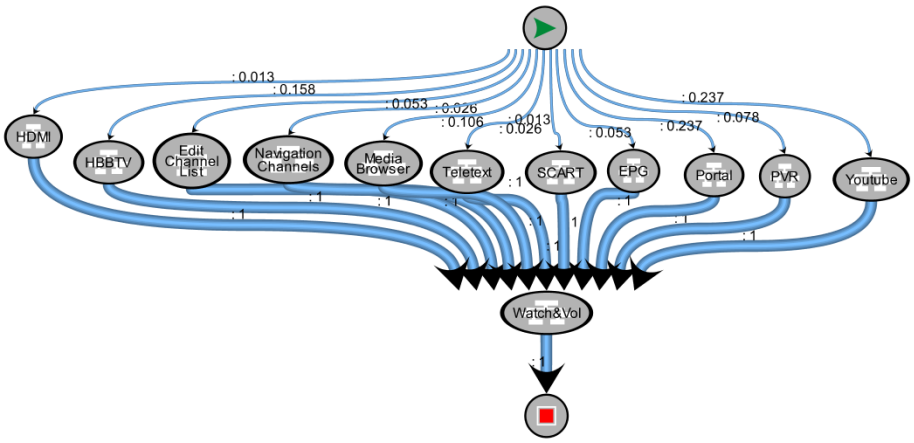
Modelde yer alacak durumları ve be durumlar arasındaki geçişleri belirledikten sonra, durum geçiş olasılıklarını belirlemek üzere statik kod analizleri gerçekleştirilmiştir. Klocwork[3] aracı kullanılarak, modelde yer alan işlevleri gerçekleyen modüller için raporlanan uyarı sayıları belirlenmiştir. Tablo-1'de, modelde yer alan durumlara ilişkin modüller için raporlanan hata sayıları ve hesaplanan durum geçiş olasılık değerleri listelenmiştir. Bazı işlevlerin gerçeklemesinde ortak yazılım modülleri kullanılmaktadır. Bu modüller için raporlanan hata sayıları, ilgili durum geçiş olasılığı hesaplamaları için eşit paylaştırılmıştır.

[2] http://www.all4tec.net
[3] http://www.klocwork.com

Tablo-1. Statik kod analizi sonuçları ve tayin edilen durum geçiş olasılıkları

Durum Geçişi	*Hata Sayısı*	*P*
Portal (Pr)	18	0,237
Youtube (Yt)	18	0,237
HBBTV	6	0,158
Media Browser (MB)	4	0,106
PVR	3	0,078
Navigation Channels (ChL)	2	0,053
EPG	2	0,053
Edit Channel List (ChO)	1	0,026
Teletext (TXT)	1	0,026
HDMI (SSw)	1	0,013
SCART (SSw)	1	0,013

Şekil 3'te görülen modele aynı zamanda test kodları da gömülmüştür. Modelde yer alan her durum geçişi bir test adımı olarak tanımlanmıştır ve her adım için Python dili ile test kodları geliştirilmiştir. Dolayısıyla, MBT aracı ile model üzerinden test senaryoları oluşturulduğunda, bu senaryolar bir durum geçişi dizisi olduğundan ve her geçiş için test kodu tanımlı olduğundan, otomatik olarak çalıştırılabilir senaryolar elde edilmiştir. İlk oluşturulan modelden toplam 650 adımlık bir test süit oluşturulmuştur. Oluşturulan testlerin çalıştırılması 3 saat sürmüştür ve testler sonucunda 1 adet hata bulunmuştur.



Şekil 3. Kullanım profili ve statik kod analizi sonuçları ile hazırlanan üst seviye model

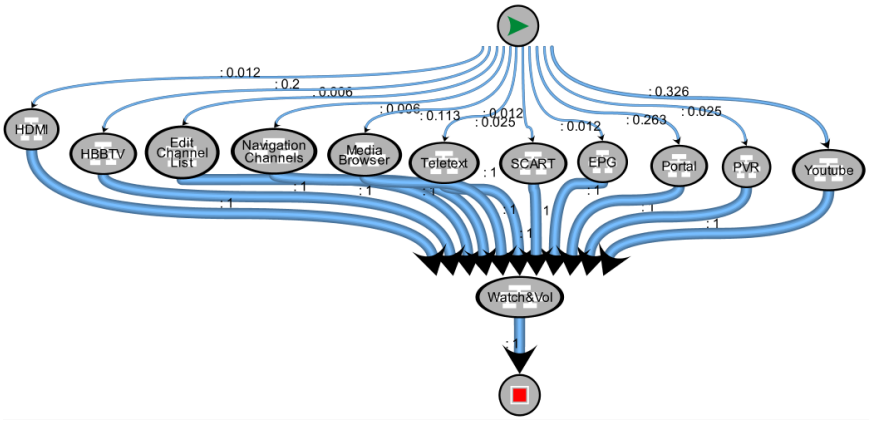
3.3 Dinamik Kod Analizi ve Test Modelinin Güncellenmesi

İlk hazırlanan model ile oluşturulan test senaryoları, Akıllı TV sistemi üzerinde otomatik olarak çalıştırılmıştır. Testlerin çalıştırılması sırasında, Teraterm[4] aracı ile sistemdeki yazılım modüllerinin bellek kullanımları kayıt altına alınmıştır. Kayıt altına alınan değerler ve bu değerlere göre hesaplanan yeni durum geçiş olasılıkları Tablo 2'de listelenmiştir. Statik kod analizi sonuçlarının değerlendirilmesinde de yapıldığı gibi, ortak kullanılan yazılım modüllerine ilişkin bellek kullanımı miktarlarının, bu modülleri kullanan işlevler tarafından eşit paylaşıldığı varsayılmıştır.

Tablo-2. Dinamik analiz sonuçları ve güncellenen durum geçiş olasılıkları

Durum Geçişleri	*Bellek Açığı (Mb)*	*P'*
Portal (Pr)	21	0,263
Youtube (Yt)	26	0,326
HBBTV	16	0,200
Media Browser (MB)	9	0,113
PVR	2	0,025
Navigation Channels (ChL)	1	0,006
EPG	1	0,012
Edit Channel List (ChO)	1	0,006
Teletext (TXT)	2	0,025
HDMI (SSw)	2	0,012
SCART (SSw)	2	0,012

Tablo 2'de görüldüğü üzere "Youtube", "Portal" ve "HBBTV" işlevleri bellek kullanımı açısından ön plana çıkmaktadır. Bu sonuçlar durum geçiş olasılıklarına yansıtılarak model güncellenmiştir (Bkz. Şekil 4).



Şekil 4. Dinamik analiz sonuçlarına göre güncellenmiş üst seviye model

[4] http://ttssh2.sourceforge.jp/

Modelde durum geçiş olasılıkları dışında herhangi bir değişiklik yapılmamıştır. Test adımları için önceden geliştirilmiş Pyhton test kodları tekrar kullanılmış ve MBT aracı ile model üzerinden otomatik çalıştırılabilir test senaryoları tekrar oluşturulmuştur.

Bu kez, dinamik analiz sonuçlarına göre riskli oldukları tespit edilen işlevlere daha fazla odaklanacak şekilde 540 adımlık bir test süit oluşturulmuştur. Oluşturulan testlerin çalıştırılması 2,5 saat sürmüştür ve testler sonucunda 2 adet hata bulunmuştur. Bu süreç iki kez daha tekrarlanmıştır. Her defasında MBT ile test senaryoları oluşturulmuş, oluşturulan test senaryoları sistem üzerinde çalıştırılmış, dinamik analizler gerçekleştirilmiş ve modeldeki durum geçiş olasılıkları analiz sonuçlarına göre güncellenmiştir. Sonuçlar Tablo 3'de listelenmiştir. İterasyon 0, ilk oluşturulan modele ilişkin paylaşılan sonuçları temsil etmektedir. Daha sonra tekrar eden her iterasyon kapsamında model güncellenmiştir. Model güncelleme işlemi sadece modeldeki olasılık parametrelerinin değiştirilmesinden ibaret olduğundan, bu işlem için geçen süre göz ardı edilmiştir.

Tablo-3. Tekrar eden adımlarda elde edilen test sonuçları

İterasyon	*Test Sayısı*	*Test Süresi (saat)*	*Hata Sayısı*
0	650	3	1
1	540	2,5	2
2	480	2	3
3	379	1,75	5

Her tekrarda, hataya daha yatkın olan yazılım işlevlerini kapsayan test senaryolarının sayısı artmıştır. Aslında, Tablo 3'de görüldüğü gibi toplam test senaryolarının sayısı ve test süresi azalmıştır. Bu duruma karşın, tespit edilen hata sayısı artmıştır. Bu hatalar statik kod analizi sonucunda raporlanmış olan kod seviyesindeki hatalar değil, kullanıcı tarafından gözlemlenen hatalardır.

4 İlgili Çalışmalar

Literatürde, MBT için birden fazla yaklaşım [1][5] ve modelleme teknikleri önerilmiştir. Sonlu durum makinaları [14] (Finite State Automata), Birleşik modelleme dili (UML), Markov zincirleri [8][12] bu yaklaşımlarda yaygın olarak model tipleridir. Raporlanan çalışmalarda daha çok modelleme yaklaşımları ve test senaryosu oluşturma metotlarına odaklanılmıştır. Modellerin geliştirilmesi ve iyileştirilmesi ise manuel olarak gerçekleştirilen ve ağırlıklı olarak alan bilgisi ile tecrübeye dayanan bir aktivitedir. Modellerin dinamik analizlere göre iteratif güncellenmesi konusu üzerine çok fazla çalışma yapılmamıştır.

Yakın zamanda modelleme ve capture-replay test tekniklerinin birlikte kullanılmasına yönelik bir yaklaşım önerilmiştir [15]. Bu yaklaşımdac capture-replay test araçları ile modelin güncellenmesine odaklanılmıştır. Biz çalışmamızda modeli

ilgili modullerin bellek aciklarina göre güncellemekteyiz. Ayrıca biz metodumuzu, yaptığımız deneysel çalışma verileri ve sonuçları ile desteklemekteyiz.

Yakın zamanda, modeli tecrübeye göre güncellemek konusunda bir yaklaşım sunulmuştur [2]. Bu çalışmanın amacı MBT ile tecrübeye dayalı testleri birleştirerek modelleri iyileştirmeye dayalıdır. Tecrübeli test mühendislerinin testler sırasında gerçekleştirdikleri işlemler kayıt altına alınarak ve analiz edilerek MBT için kullanılan modellerdeki eksiklerin bulunulması sağlanmıştır. Fakat kullanım profili, statik ve dinamik analizlere dayalı bir iyileştirme yapılmamıştır.

Daha önce de MBT sürecinin endüstride kullanımına ilişkin tecrübelerimiz paylaşılmıştır [3]. Bu tecrübeler kapsamında kullanım profilleri model geliştirme sürecinde değerlendirilmiştir. Ancak mevcut yaklaşımımızda dahil ettiğimiz statik kod analizine dayanan hesaplamalar ve dinamik kod analizi ile modeli her defasında iyileştirme yöntemi kullanılmamıştır.

Bir önceki çalışmamızda [16] dinamik analizlere dayalı risk hesaplayarak model güncelleme yaklaşımını değerlendirmiştir. Fakat bu yaklaşımı kapsamlı bir vaka analizi ile sınamamıştık. Ayrıca, bu çalışmada kullanım profili ve statik kod analizine dayalı geliştirilen ilk modelin hâlihazırda var olduğunu varsaymıştık.

5 Sonuç

Bu çalışmada, MBT için sistem modeli geliştirmek ve iyileştirmeye yönelik özgün bir yöntem önerdik. Yöntemimizde, kullanım profili ve statik kod analizlerinin sonuçlarını temel alarak, sık kullanılan ve hata oluşumuna yatkın olan işlevlere odaklanan bir model geliştirilmesini sağladık. Geliştirilen model ile oluşturulan test senaryoları çalıştırılırken, sistem üzerinde dinamik analizler gerçekleştirerek hata oluşumuna yatkın olan durumları tekrar tekrar gözden geçirerek sistem modelini iteratif olarak güncelledik.

Önerdiğimiz yöntemi sınamak adına, gerçek bir Akıllı TV sisteminin test süreçleri kapsamında bir vaka analizi gerçekleştirdik. İlk modelin geliştirilmesinin ardından model iyileştirme sürecini 3 kez tekrarladık. Her tekrarda, toplam test senaryolarının sayısının ve test süresinin azaldığını; fakat buna rağmen tespit edilen hata sayısının arttığını gözlemedik.

İleriki çalışmalarda, vaka analizi sayısını artırmayı ve bellek kullanımı dışındaki hata tiplerini de çalışma kapsamına almayı hedefliyoruz. Ayrıca, oluşturulan modelleri yazılım güvenilirliğine[7][13] ilişkin sayısal veriler elde etmek için de kullanmayı planlıyoruz.

Kaynaklar

[1] Neto, A.C.D., R.Subramanyan, M.Vieira, Travassos, G.H.: A survey on model-based testing approaches: A systematic review. In: Proceedings of the 1st ACM international workshop on Empirical assessment

[2] C.Ş. Gebizli, H. Sözer: Improving Models for Model Based Testing Based on Exploratory Testing. In: Proceedings of the 8th IEEE International Computer Software and Applications Conference Workshops, 2014

[3] C.Ş. Gebizli, D. Metin: Kullanım Modeli Bazlı Otomatik Test Tasarımı. In: Proceedings of the 7th National Software Engineering Symposium, 2013

[4] Marijan, V. Zlokolica, N. Teslic, V. Pekovic, T.: Automatic functional TV set failure detection system. IEEE Transactions on Software Engineering, vol. 56, no. 1, pp. 125–133, 2009

[5] J. Boberg: Early fault detection with model-based testing. In: Proceedings of the 7th ACM SIGPLAN workshop on ERLANG, 2008

[6] Wesslén, P. Runeson, B. Regnell. : Assessing the sensitivity to usage profile changes in test planning. In: Proceedings of the 11th International Symposium on Software Reliability Engineering ISSRE, 2000

[7] K. Weyns, P. Runeson: Sensitivity of software system reliability to usage profile changes. In: Proceedings of the 2007 {ACM} Symposium on Applied Computing (SAC), 2007

[8] S. J. Prowell: Using Markov Chain Usage Models to Test Complex Systems. In: Proceedings of the 38th Hawaii International Conference on System Sciences, 2005

[9] Guiotto, B. Acquaroli, A. Martelli : MaTeLo: Automated Testing Suite for Software Validation. In: Proceedings of DASIA, 2003

[10] W. Dulz, Z. Fenhua : MaTeLo – Statistical Testing Using Annotated Sequence Diagrams, Markov Chains and TTCN-3. In: Proceedings of the Third International Conference on Quality Software, 2003

[11] Feliachi, H. Le Guen : Generating transition probabilities to support model-based software testing. In: Proceedings of the Third International Conference on Software Testing, Verification and Validation (ICST), 2010

[12] Whittaker, M. Thomason : A markov chain model for statistical software testing. In: IEEE Transactions on Software Engineering, vol. 20, no. 10, pp. 812–824, 1994

[13] Musa, J. D: A theory of software reliability and its application In: IEEE Transactions on Software Engineering, vol. 1, no. 3, pp. 312-327,1975

[14] Chander, D. Dhurjati, S. Koushik, and Y. Dachuan. : Optimal test input sequence generation for finite state models and pushdown systems. In: Proceedings of the 4th IEEE International Conference on Software Testing, Verification and Validation, 2011

[15] V. Entin, M. Winder, B. Zhang, and S. Christmann, : Combining model-based and captue replay testing techniques of graphical user interfaces: An industrial approach, in 4th IEEE International Conference on Software Testing, Verifcation and Validation Workshops, 2011

[16] C.Ş. Gebizli, D. Metin, H. Sözer :Combining Model-Based and Risk-Based Testing for Effective Test Case Generation, In: Proceedings of the 10th IEEE International Conference on Software Testing, Verification and Validation Workshops, 2015