Yazılım Mimarisinin Kalite Gereksinimleri: Yazılım Güvenilirliği

Kaan Kurtel
İzmir Ekonomi Üniversitesi
kaan.kurtel@ieu.edu.tr

Şaban Eren
Yaşar Üniversitesi
saban.eren@yasar.edu.tr

Özetçe

Yazılım güvenilirliği pek çok açıdan yazılım geliştirmede önemli hale gelmektedir. Bu çalışma yazılımcıları yazılım güvenilirliği açısından temel konularda bilgilendirmeyi ve uygulama cesareti kazandırmayı hedeflemektedir.

Anahtar Sözcükler : Yazılım Güvenilirliği, Yazılım Kalitesi

Quality Requirements for Software Architecture: Software Reliability

Abstract

Quality Requirements for Software Architecture: Software Reliability. The aim of this paper is to give the reader a better understanding of the importance of software reliability. Furthermore it focuses on the basics of software reliability, prediction and modeling reliability, collection of data and also the relationship between the software quality and software reliability.

Keywords : Software Reliability, Software Quality

1 Giriş

Sayısal tabanlı bilgisayarların yarım yüzyıllık varlıklarının sonucunda, hemen herkes doğrudan veya dolaylı olarak bilgisayar sistemleri ile ilişki halindedir. Dünya üzerindeki pek çok faaliyetin başarı ile sürmesi, bilgisayarlara ve onun en önemli parçasından biri olan yazılımlara bağlıdır. Donanım ve yazılım ürünleri, başta finans, haberleşme ve ulaşım sistemleri, enerji, sağlık, askeri ve kamu alanları olmak üzere pek çok alanda hayatımıza girmiş ve belirleyici bir konum kazanmıştır. Bu durum sistemlere olan bağımlılığı artırmakta ve sürdürülmesini gerekli kılmaktadır.

21. yüzyılın iş ortamında ve sosyal hayatında yazılım ürünlerine olan talep artarken, yazılım ürünleri de gittikçe çeşitlenmekte ve karmaşıklaşmaktadır. Günümüzde yazılım geliştirme ortamı, heterojen iş ortamının beklentilerini karşılamak üzere; internet ortamına servis veren dağıtık sistemlerden, farklı ortamlarda yazılmış, farklı platformlarda çalışan ara programlardan ve birbirinden bağımsız çalışan çok sayıdaki yazılımdan oluşmaktadır.

Bu dinamikler nedeniyle, yazılım sistemlerinin kalite faktörüne dikkat edilerek üretilmesi, dünya üzerindeki faaliyetlerin sürekliliği ve ekonomikliği açısından kritik öneme sahiptir. Yazılım endüstrisinin karakteristik özelliği, hızla ve sürekli olarak gelişmesi ve değişmesinin yanı sıra, dünya ölçeğinde pek çok başarısız uyarlamaya sahne olmasıdır. Yazılımın beklenen kalite düzeyinde olmamasının ve dolayısıyla da müşteri beklentilerini kesin ve net bir şekilde karşılayamamasının neden olduğu maliyetler, ürüne ve ürünü üreten şirketin başarısına yönelik ciddi tehditleri de beraberinde getirmektedir. Bu durum daha kaliteli yazılım ürünlerine olan gereksinimi artırmaktadır.

Yazılım kalitesinin tanımlanmasında, yazılım güvenilirliği; işlevsellik, kullanılabilirlik, performans, kullanışlılık, yeterlilik, kurulabilirlik, bakılabilirlik ve dokümantasyonla benzer öneme sahip olmasına karşın diğer yazılım kalite kriterlerine göre daha az bilinmekte ve uygulanmaktadır. Bunun nedeni de ölçme ve değerlendirmenin sayısal büyüklüklerle yapılıyor olması, yazılım güvenilirliğinin ise yoğun olarak matematik ve istatistiksel değerlerle ifade edilmesidir.

Bu çalışmada yazılım mimarisinin kalite gereksinimleri, yazılım güvenilirliği açısından incelenmiştir. Yazılımın öneminin arttığı ve yaşamsal hale geldiği bu süreçte, bilgisayar ve

yazılım mühendislerine yazılım güvenilirliği konusunda yararlanabilecekleri bir çalışmayı ortaya çıkarmak ana hedef olarak amaçlanmaktadır. Böylece yazılım güvenilirliğinin; yazılım geliştiricileri tarafından farkındalığının sağlanması, daha güvenilir ve dolayısıyla daha kaliteli ve ekonomik ürünler üretmelerini sağlayarak rekabet avantajı elde etmeleri, ayrıca konu hakkındaki temel bilgileri edinerek uygulama cesareti kazanmaları sağlanmış olacaktır.

2 Temel Kavramlar

Yazılım güvenilirliği, sistem veya bileşenlerinin, belirli bir ortamda, belirli bir zaman dilimi içinde kendilerinden beklenilen işlevleri yerine getirebilme olasılığı olarak tanımlanmaktadır [1].

Yazılım güvenilirliği mühendisliği ise yazılım ürünlerinde müşteri memnuniyetini sağlamak üzere, güvenilirliğin kestirimi, ölçülmesi ve yönetilmesidir [2].

Yazılım güvenilirliği mühendisliği, müşteri açısından gerekli güvenilirlik ve kaliteye yönelik alttaki faaliyetleri içermektedir [3] [4]:

- Müşteri beklentilerini maksimize eden kalite faktörlerini belirlemek.
- Güvenilirlik hedeflerine karar vermek.
- Geliştirme sürecine ve hatanın tanımlanmasına, düzeltilmesine yönelik güvenilirlik modellerini kestirmek.
- Yüksek kalite güvenilirliği sağlamaya yönelik yazılım geliştirme sürecini desteklemek.
- Test stratejilerini destekleyen faaliyetleri üretmek.
- Güvenilirlik kestirimine yönelik ölçütleri tanımlamak ve bu ölçütlerin beklentileri karşılama konusundaki uygunluğunu sınamak.

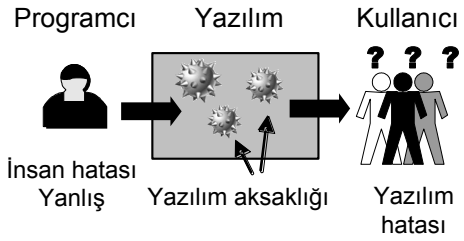
Güvenilir bir yazılımdan söz edildiğinde, ürünün planlama, üretim, test ve bakım aşamalarında arzu edilen güvenilirliği hedeflemesi anlaşılmalıdır. Bu da güvenilirlik çalışmalarında istatistik bilgisinin yanı sıra bir güvenilirlik hedefinin de bulunması gerekliliğini ortaya koymaktadır. Güvenilirlik hedefi müşteri beklentisi ile doğrudan ilgili olup, bu hedefe nasıl ulaşılacağı ile ilgili verilerin toplanması ve modellenmesini de gerektirmektedir.

Yazılım güvenilirliğine ait yukarıdaki her iki tanımda da güvenilirlik ile kalite arasındaki yakın ilişki vurgulanmaktadır. Diğer bir şekilde, güvenilirlik zamana bağlı olan kalite olarak tanımlanmaktadır. Bu da yazılım güvenilirliğinin, yazılım kalitesi için önemli bir ölçüt olarak değerlendirilmesini gerektirmektedir [5].

Yazılım güvenilirliğinin temel kavramları olan hata, aksaklık ve arızanın tanımı aşağıdaki şekilde yapılmaktadır [1] [2] [6]:

- Yanlış (*error*); hesaplanan, gözlemlenen veya ölçülen değerler arasında fark oluştuğundaki durumu anlatmakta kullanılmaktadır. Örneğin hesaplanma sonucunda 12 değeri bulunduğu halde, doğru sonuç 10 ise yanlışlıktan bahsedilebilir. Genellikle de programcının veya tasarımcının yanlış düşünceden kaynaklanan davranışları sonucunda ortaya çıkan doğru olmayan sonuçlar için kullanılmaktadır. Örnek olarak da programın doğru olmayan bir parçası veya işlemi verilebilir.
- Aksaklık (*fault*); yazılımda doğru olmayan bir adım, süreç veya veriyi tanımlamaktadır. Aksaklıklar yazılımın kodundaki yanlışların sonucunda ortaya çıkar. Örneğin, bir programlama dilinde *if* deyiminin kullanımı sırasında, dallanmada bir aksaklık oluşabilir ve deyim yanlış bir dala aktarılabilir. Böylelikle program istenilen sonucu üretemeyebilir ve bu durum da aksaklığa neden olabilmektedir.
- Hata (*failure*); (bozukluk olarak da kullanılmaktadır) bir sistemin kendisinden beklenilen işlevi yerine getirememe durumudur. Hata nedeni genellikle bir aksaklık olmaktadır.
- Kusur (*defect*); Üründeki anomalidir. Hata veya aksaklık için kullanılan genel bir terimdir ve genellikle de $\text{Kusur} = \{\text{Aksaklık}\} \cup \{\text{Hata}\}$ olarak tanımlanmaktadır.

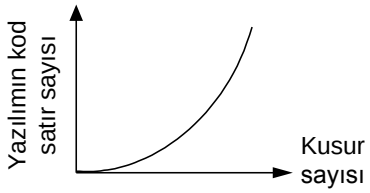
Şekil-1, yanlışların, yazılımdaki aksaklıkları ve sonuçta da hatayı nasıl tetiklediğini göstermektedir [7].



Şekil-1: Yanlış, aksaklık ve hata arasındaki ilişki

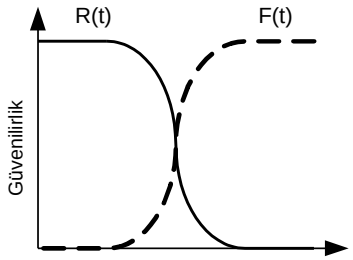
Araştırmalar programcıların her 1000 satırlık kod için ortalama altı yazılım kusuru üretiklerini göstermektedir. Bu da 350000 satırlık bir ticari yazılımın yaklaşık 2000 adet yazılım kusurunu barındırabileceği anlamına gelmektedir [8].

Yazılım büyüdükçe olası kusur sayısı da geometrik olarak artış göstermektedir (Şekil 2).



Şekil-2: Yazılım kusurlarının değişimi

Bir yazılım ürününün işler durumda olması olasılığı güvenilirlik olarak tanımlanırken, hata üretmesi veya çalışamaz durumda olma olasılığı ise güvenilmezlik olarak ifade edilebilir. Güvenilirlik fonksiyonu R(t) ile gösterilirken, güvenilmezlik fonksiyonu ise F(t) ile ifade edilmektedir. Güvenilmezlik azalırken, güvenilirlik artmaktadır (Şekil 3).



Şekil-3: Güvenilirlik ve güvenilmezlik fonksiyonları arasındaki ilişki

Böylece güvenilirlik ve güvenilmezlik fonksiyonları arasındaki ilişkinin ifadesi:

$$R(t) = 1 - F(t) \quad (1)$$

olarak ifade edilmektedir. Eğer *t* rastgele değişkeni *f(t)* olasılık yoğunluk fonksiyonuna (OYF) sahip ise güvenilirlik:

$$R(t) = 1 - F(t) = \int_t^{\infty} f(t)dt \quad (2)$$

olmaktadır. Bu durumda güvenilmezlik ise:

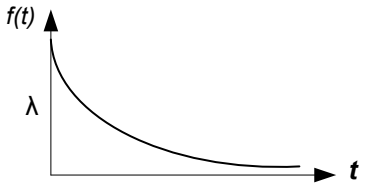
$$F(t) = 1 - R(t) = \int_{-\infty}^{t} f(t)\,dt \quad (3)$$

olarak ifade edilmektedir.

Üssel dağılış güvenilirliğin modellenmesinde kullanılan temel fonksiyonlardan biridir. Üssel dağılışta tüm hataların birbirinden bağımsız olduğu varsayılmakta ve t zamanındaki bir hatanın olasılık dağılımı

$$f(t) = \lambda e^{-\lambda t} \quad (4)$$

şeklinde ifade edilmektedir. Üssel OYF Şekil 4'de gösterilmiştir [7].



Şekil-4: OYF $f(x)=\lambda e^{-\lambda t}$

Böylece t zamanındaki güvenilirlik:

$$R(t) = \int_t^{\infty} \lambda e^{-\lambda t}\,dt = e^{-\lambda t} \quad (5)$$

formülü ile hesaplanmaktadır. Burada λ, hata yoğunluğunu ifade etmektedir.

Hata yoğunluğu

Hata yoğunluğu (*failure intensity, failure density* veya *failure rate*) olarak ifade edilmektedir ve belirli bir zaman diliminde karşılaşılan hataların sayısı olarak verilmektedir [1] [3].

$$HataYoğunluğu(\lambda) = \frac{Hatasayısı}{Toplamtestsüresi} \quad (6)$$

Örneğin; herbiri 10 saat süren 5 adet testin sonucunda, 3 testin 3, 7 ve 8. saatlerde hata vermesi durumunda hata yoğunluğu

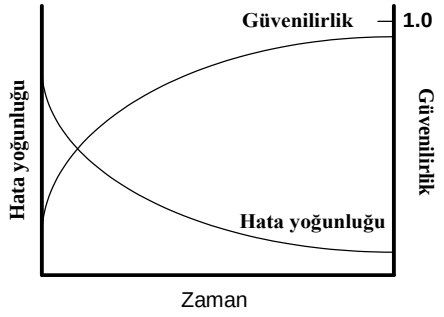
$$\lambda = \frac{3}{3+7+8+2(10)} = \frac{3}{38} = 0.0789$$

olarak hesaplanmaktadır.

Hata yoğunluğu ölçütü;

1. Ortaya çıkmamış hataların kestirilmesinde,
2. Test süresinin tamamlandığına ait kararların verilmesinde,
3. Standart hata yoğunluğunu saptamada kullanılmaktadır [6].

Hata yoğunluğu, güvenilirlik gibi ortama bağlı olarak tanımlanmaktadır. Şekil 5 hata yoğunluğu ile güvenilirlik arasındaki ilişkiyi, hatalar giderildikçe hata yoğunluğunun ve güvenilirliğin zaman ile nasıl değiştiğini göstermektedir [9].



Şekil-5: Güvenilirlik ve hata yoğunluğu

Güvenilirlik analizlerinde yukarıdaki kavramlara ek olarak aşağıdaki temel ifadeler de yaygın olarak kullanılmaktadır:

Ortalama hata süresi-OHS

OHS (*Mean Time To Failure-MTTF*), hataya kadar geçen ortalama çalışma süresidir. Hata zamanının beklenen değeri OHS'ni vermekte ve (7) nolu istatistik formülü yardımıyla elde edilmektedir [7].

$$OHS = \int_0^\infty t f(t)\, dt \qquad (7)$$

Buradan;

$$OHS = \int_0^\infty R(t)\, dt \qquad (8)$$

elde edilmektedir ve güvenilirlik fonksiyonunun bilindiği durumda OHS'nin matematiksel olarak hesaplanmasında kullanılmaktadır.

Hatalar arası ortalama süre-HAOS

HAOS (*Mean Time Between Failures-MTBF*), ürünün hatasız olarak çalışmasının ortalama süresi olarak tanımlanmaktadır. Ancak ürün bu süreden sonra da hata üretebileceği gibi, bu süreden önce de hata üretmiş olabilir. Dolayısı ile HAOS, güvenilirlik karşılaştırmaları amacıyla kullanılmaktadır [5]. HAOS;

$$HAOS(\mu) = \frac{\text{Toplam işletim süresi}}{\text{Hata sayısı}} \qquad (9)$$

olarak ifade edilmektedir. (6) ve (9) nolu eşitliklerden hata yoğunluğu;

$$\lambda = \frac{1}{HAOS} \qquad (10)$$

olarak hesaplanmaktadır.

Ortalama hata süresi-OHS

OHS (*Mean Time To Failure-MTTF*), hataya kadar geçen ortalama çalışma süresidir. Hata zamanının beklenen değeri OHS'ni vermekte ve (7) nolu istatistik formülü yardımıyla elde edilmektedir [7].

Örneğin; (9) nolu eşitliğe göre

$$HAOS = \frac{38}{3} = 12{,}66 \text{ saat}$$

Bu sonuç ürünün ortalama 12.66 saat hatasız olarak çalışabileceğini göstermektedir.

Örneğin; (10) nolu eşitliğe göre

$$HAOS = \frac{1}{0.0789} = 12{,}66 \text{ saat}$$

olarak hesaplanmaktadır.

Bozulma Oranı

Bozulma oranı (*Hazard Rate*) veya anlık hata yoğunluğu, birim zamanda bozulanların oranıdır ve

$$h(t) = \frac{f(t)}{R(t)} \qquad (11)$$

olarak hesaplanmaktadır [7] [9]. Üssel dağılım ile modellenen bozulma oranı (4) ve (5) nolu eşitliklere göre;

$$h(t) = \frac{\lambda e^{-\lambda t}}{e^{-\lambda t}} = \lambda \qquad (12)$$

olarak ifade edilmektedir.

3 Yazılım Güvenilirliğinin Kestirimi

Yazılım güvenilirliğinin kestirimi, yazılım güvenilirliği mühendisliğinin uygulama alanlarından biridir. Yazılım güvenilirliği mühendisliğin amacı; yazılım sistemlerinin güvenilirliğe ilişkin nicelikleri, kullanıcıların gereksinimlerini karşılayacak şekilde ortaya koymak ve güvenilirliğin hesaplanmasına yönelik verileri toplama, istatistiksel kestirim, ölçütlerin tespiti, yazılıma ait mimari özelliklerin belirlenmesi, tasarım, geliştirme ve bunlara yönelik çalışma ortamının belirlenmesi ve modellenmesini kapsamaktadır [10]. Yazılım güvenilirliği modellemesinde temel olarak rastgele özellik göstermeyen belirleyici (*deterministic*) modeller ile olasılıksal (*probabilistic*) modeller kullanılmaktadır [11].

Deterministik modeller; bir yazılımda birbirinden bağımsız işlemcilerin ve işlenenin yardımıyla, yazılımdaki bilgisayar komutlarının ve hata sayılarının incelenmesi esasına dayanmaktadır. Temel olarak da 1977'de Halstead tarafından geliştirilen karmaşıklık ölçütüne ve 1976'da McCabe tarafından geliştirilen döngüsel (*cyclomatic*) karmaşıklık ölçütüne dayanmaktadır. Bu çalışmada yazılım güvenilirliği modellemesi olasılıksal modeller açısından incelenmektedir.

Yazılım güvenilirliğinin modellenmesinde kullanılan ve olasılık bileşeni içeren modeller, belirli bir hata tarihçesine dayanarak, yazılımın hata oranının kestirilmesinde kullanılmakta ve kestirime yönelik parametreleri içermektedir. Temelde, hatalar arasındaki sürelerin ve testler arasındaki hata sayılarının kestirimine yönelik olup, genellikle de üssel bir fonksiyon ile gösterilmektedir. Modeller, özel bir veri kümesi için doğrulanmakta, böylelikle de kalan hataların veya daha sonra oluşacak olan hataların ne zaman ortaya çıkacağı kestirilmekte ve kestirmenin güven aralığının hesaplanması mümkün olmaktadır [12].

Yazılım güvenilirliği modelleri bu özelliklerinden dolayı yazılım geliştirmenin erken evrelerinde olan ve yazılım gereksinimlerinin eksik anlaşılması dolayısıyla meydana gelen yanlışlardan daha çok test sürecinde ortaya çıkan hatalar ile ilişkilidir [13].

Bundan dolayı yazılım güvenilirliğinin modellenmesinde, test sürecindeki hataların tanınması, ortadan kaldırılması ve çalışma ortamını dikkate alan temel faktörler önemlidir.

Hata tanıma, ürün ve ürünün geliştirme süreci ile ilgilidir. Temel ürün karakteristiği programın büyüklüğü olup, geliştirme süreci ise yazılım mühendisliği teknolojileri ve araçları, geliştiricilerin deneyimi ve gereksinimlerin değişebilirliği ile ilgilidir. Hata giderme, hataların ortaya çıkarılması ve düzeltilmesi süreçlerini içermekte olup, yapılan işin başarısı, elde edilen kalite sonuçları ile ilişkilidir. Tüm bunlar da çalışılan ortamın özellikleri ile ilgilidir. Güvenilirlik açısından hataların belirdiği ortam, olasılıksal, çalışma zamanındaki rastgele süreçler ile ifade edilmekte ve aşağıdaki ifadelerin kullanılmasını gerekli hale getirmektedir [3]:

1. Herhangi bir andaki ortalama hata sayısı
2. Zaman aralığındaki ortalama hata sayısı
3. Herhangi bir andaki hata yoğunluğu
4. Hata yoğunluğunun olasılık dağılımı

Hatalar arasındaki sürenin kestirimi, f(t) OYF'nu olarak ifade edilmekte ve fonksiyonun parametreleri daha önceden gözlemlenen hatalar arasındaki t_1, t_2, ..., t_{i-1} sürelerin değerlerine göre kestirilmektedir. Böylece OYF, bir sonraki hatanın zamanının kestiriminde kullanılmış olmaktadır. Teste başlanıldıktan sonra hatalar arasında i-1 adet sürenin gözlemlenmiş olduğu varsayıldığında ve i. hata ve i-1. hata arasındaki süre kestirilmeye çalışıldığında, bu t rastgele değişkeni ile gösterilir ve t rastgele değişkeninin beklenen değeri de (2) nolu eşitlikten [7]:

$$E(t_i) = \int_t^{\infty} t f_i(t) dt \qquad (13)$$

olarak elde edilmektedir. (13) nolu eşitlikteki fonksiyonun parametrelerinin hesaplanmasında en büyük olabilirlik (*maximum likelihood*) veya en küçük kareler yöntemi kullanılmaktadır.

İstatistiksel yazılım güvenilirlik modelleri hatanın meydana gelmesine ve aksaklıkların giderilmesine yönelik olasılıksal olaylar olarak sınıflandırılır ve başlıca güvenilirlik modelleri aşağıdaki gibi verilmektedir [11]:

- Yanlışı belirlemek
- Hata oranı
- Eğriye uydurmak
- Güvenilirlik geliştirmek
- Markov biçimlendirmesi
- Zaman serileri
- Homojen olmayan Poisson süreci

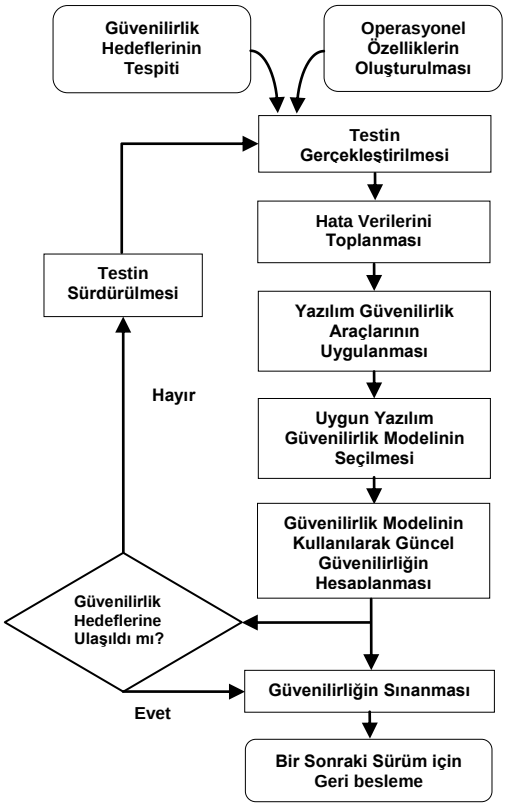
İyi bir yazılım güvenilirlik modelinin sahip olması gereken özellikler aşağıda verilmektedir [3][12]:

1. Gelecekte oluşacak hatalara ait iyi bir kestirime olanak vermesi
2. Yararlı nicelikleri hesaplaması
3. Basit olması
4. Geniş bir uygulama alanının bulunması
5. Kabul edilebilir varsayımları desteklemesi
6. Parametrelerinin kolay hesaplanması

Ebert ve diğerleri tarafından [4] yazılım güvenilirliğinin kestiriminin, modellenmesi ve en iyi modelin seçilmesine yönelik analiz ve karar verme süreci aşağıdaki şekilde tanımlanmaktadır:

1. Model seçme ve düzenlemeye yönelik faaliyetlerin temelinde uygun hedeflerin tespit edilmesi bulunmaktadır.
2. Hata ve aksaklıkların analiz edilmesi için uygun verilerin tanımlanması gerekmektedir. Örneğin, arıza veya hataları önemine göre sınıflandırmak, hatalar arası ortalama süreyi bulmak, hata nedenlerini araştırmak, hataları bulmaya yönelik test verilerine karar vermek.
3. Belirtilen hedeflere yönelik verileri modellemek üzere toplamak.
4. Geçmişe yönelik verilerin zaman bilgilerini de içerecek şekilde elde edilerek yazılım geliştirme sürecine dahil etmek.
5. Yazılım geliştirme sürecinin modellenmesi, hata ile karşılaşılıp, test sürecine başlamak ve model doğrulama işlemlerine gerçekleştirmek.
6. Güvenilirlik kestirim modelinin seçilmesini sağlamak.
7. Güvenilirlik modeli tarafından kullanılacak olan parametrelerini tespit etmek.
8. Verilen bir noktayı kullanarak gelecekteki olası hatalar hakkında kestirimde bulunmak.
9. Kestirilen hata ve arıza oranları ile gerçekleşen değerleri kıyaslamak.

Benzer şekilde Lyu tarafından yapılan çalışmada da [14], yazılım güvenilirlik mühendisliğinin süreçleri özetlenmektedir (Şekil 6).



Şekil-6: Güvenilirlik mühendisliği süreci

4 Veri Toplama

Kaliteye yönelik tüm çalışmalarda olduğu gibi yazılım kalitesinin ölçülmesi, güvenilirlik modellemesi ve kestirimi için her şeyden önce veriye gereksinim duyulmaktadır. Veri toplama, organizasyonun amacına yönelik çabayı gerektiren, veri toplamaya yönelik araç ve gereçlerinin ve tüm bunları gerçekleştiren çalışanların değişken bir ortamda varlığını sürdürdüğü, zor ve geniş kapsamlı temel bir süreçtir.

Veri toplama süreci, yazılımda bir problem gözlendiğinde, probleme neden olan durumların anlaşılması ve gerekli iyileştirmelerin yapılabilmesi için verilere ait hangi değerlerin toplanması ile de ilgilidir. Bu değerler aşağıdaki şekilde olabilir ve hataların raporlanmasında kullanır [7]:

- Konum: problem nerede meydana gelmiştir?
- Zaman: ne zaman olmuştur?
- Belirti: ne gözlemlenmiştir?
- Sonuç: hangi sonucu doğurmuştur?
- İşleyiş: nasıl meydana gelmiştir?
- Neden: neden meydana gelmiştir?
- Önem: kullanıcıyı ne kadar etkilemiştir?
- Maliyet: maliyeti nedir?

Yazılım güvenilirliğine ait veri toplama işlemine, meydana gelebilecek hataların amaca uygun şekilde sınıflandırılması ile başlanması uygundur. En temel şekliyle hatalar şiddetine göre sınıflandırılır [7, s.158]:

- büyük hatalar
- küçük hatalar
- önemsiz ve ihmal edilebilir hatalar
- belgeleme hataları
- bilinmeyenler

Şiddete göre sınıflandırma, temsil edilen değerlerin niteliksel olmasından dolayı soruna neden olabilmekte ve ayrıca da oluşan yazılım aksaklığının nasıl bir hata üretebileceğinin de kestirilmesi gerektirmektedir. Örneğin, belgeleme hataları, aslında büyük hatalara neden olabilir. Bu durumda sınıflandırma açısından tercih edilmesi gereken, yazılımın ait olduğu alana ait hangi hata sınıflandırmalarının (*taxonomy*) bulunduğunun ve bu sınıfların da olası hata üretme değerlerinin bilinmesidir.

IEEE Std 1044 ölçünü arıza tiplerini aşağıdaki şekilde sınıflandırmaktadır [15]:

- Mantıksal problemler
  - Unutulan durumlar ve adımlar
  - Tekrar eden mantıksal durumlar
  - Aşırı durumların ihmal edilmesi
  - Gereksiz fonksiyonlar
  - Yorumlama yanlışlıkları
  - Durum testlerinin atlanması
  - Yanlış değişkenlerin denetlenmesi
  - Döngülerin yanlış tekrarlanması
- Hesaplama problemleri
  - Eşitliklerin yetersiz veya doğru olmaması
  - Duyarlılık noksanlıkları
  - İşaret kural hataları
- Arabirim/zamanlama problemleri
  - İşin kesilmesinin doğru yönetilememesi
  - I/O zamanlama yanlışlıkları
  - Altprogram ve modül uyuşmazlığı
- Veri işleme problemleri
  - Başlangıçta verilerin düzgün olmaması
  - Veri erişimlerinde ve saklamada yanlışlıklar
  - Verilerin birim ve ölçeklendirme yanlışlıkları
  - Veri boyutlandırma hataları
  - Veri tipilerinde yapılan hatalar
- Veri problemleri
  - Veri hatalı veya düzgün değil
  - Veri yapılarındaki gömülü verilerde sorunlar
  - Dışarıdan gelen veriler yanlış veya atlanmış
  - Çıktı verileri yanlış veya atlanmış
  - Girdi verileri yanlış veya atlanmış
- Dokümantasyon problemleri
- Doküman kalite problemleri
- Geliştirme
  - Program gereksinimlerinin değişmesi
  - Açıklamaların geliştirilmesi
  - Kod verimliliğinin arttırılması
  - Yazınsal değişikliklerin gerçekleştirilmesi
  - Kullanışlığın arttırılması

Veri toplama, yazılım geliştirmenin zihinsel ve yüksek bilgi gerektiren şartlarının tersine gözlem ve raporlama esasına dayanmaktadır ve üç temel yöntemle gerçekleşmektedir [16]. Bunlar:

- Form aracılığıyla elle yapılan veri toplama
- Otomatik veri toplama
- Görüşme yolu ile veri toplama

Elle yapılan veri toplamada, yöneticilerin, sistem analistlerin, programcıların, testi gerçekleştirenlerin ve kullanıcıların verileri formlar aracılığı ile toplaması suretiyle gerçekleşmektedir. Bu formlar çeşitli şekillerde tasarlanmasına rağmen, formların hazırlanmasında basitlik ve yalınlık, hızla analiz edilebilirlik ve arızanın oluştuğu anda verinin toplanmasını sağlamaya yönelik beklentileri karşılaması gereklidir. Ayrıca, veriyi toplayan kişilerin verinin toplanmasının amaç ve yöntemlerini de iyi anlamış olması gereklidir. Bu işlem kasıtlı veya bilinçsizce yapılan önyargılı eylemlerden dolayı hataları, gecikmeleri ve ihmalleri içermektedir [7].

Otomatik veri toplama işlemi bazı veri grupları için kod çözümleyicisi yazılımlar veya derleyiciler aracılığı ile yapılmaktadır. Ticari yazılım uygulamalarında da özellikle programlama dilinin istisna işleme konusundaki özellikleri kullanılarak veriler elde edilmektedir. Özellikle gerçek zamanlı yazılımlarda, yazılımın çalışma sürecinde verilerin otomatik olarak elde edilmesi zorunludur.

Görüşme yolu ile veri elde etme yöntemi kişisel bakış açısını yansıtmasından dolayı tercih edilmemesine rağmen özellikle projeye ait verilerin toplanmasında kullanılmaktadır.

Verinin ne zaman toplanacağı, güvenilirliğin planlama süreci ile ilgilidir ve genellikle de yazılımın test politikası tarafından belirlenmektedir. Gerek üretilen yazılımların gerekse de yapılan değişikliklerin hemen test sürecine dahil edilmesi, benimsenen yazılım geliştirme metodolojisine bağlı olduğu gibi üretilen yazılımın doğuracağı sonuçların müşteri açısından önemine de bağlıdır. Yine de test politikası ve verilerin toplanması ile ilgili zamanlama ve işin sürdürülebilirliği büyük ölçüde proje bütçesi ile ilgilidir.

5 Yazılım Kalitesi ile Güvenilirlik İlişkisi

Yazılım güvenilirliği, yazılım kalite geliştirme çalışmalarındaki en etkili unsurlardan biridir. Yazılım geliştiricilerinin ürünlerinin güvenilirlik değerlerini bilmek istemelerinin başlıca nedeni, sıfır hata ile üretilen bir yazılımın en kaliteli yazılım olduğu yönündeki yaygın kanıdır. Yazılım kalitesi ile güvenilirlik arasındaki bu ilişki doğru olmakla beraber, özellikle ticari yazılım pazarındaki üreticilerin bir kaçı dışında ürünlerini sıfır hata veya tamamen güvenilir ürettiklerini söylemek zordur. Benzer şekilde kullanıcıların da fiyatlamanın doğru olması ve ürün özelliklerinin yeterli olması durumunda, bundan kaygılanmadıkları ve olası hata, kusur ve bozuklukları dikkate almadıkları kolaylıkla gözlemlenmektedir [17].

Yazılım geliştiren şirketler, özellikle de yaşamsal önemi olan yazılım sistemleri başta olmak üzere yazılım güvenilirliği ile ilgili çalışmaları ürünleri için yapmak zorundadırlar. Bunun başlıca nedeni, yazılım güvenilirliği ile ilgili çalışmalarının sonucunda sayısal bir değerin elde edilmesi ve böylece yazılım projelerinin geliştirilmesine yönelik kaynakların daha verimli kullanılmasını sağlamaktadır. Bunun diğer bir yararı da ürünlerin pazara sunulmadan önce olası hata oranı hakkında bilgi sahibi olunarak, ilerideki bakım, onarım gibi faaliyetlerin bütçelenebilmesidir.

Güvenilirlik konusu, kalite kontrol, test sürecinin düzenlenmesi ve sonlandırılmasına ilişkin çalışmalarda da yol göstericidir. Yazılım testi gibi kestiriminde güçlük bulunan bir konu da, test süresine ve içeriğine yönelik süreç ve işlerin tespit edilmesini ve de dolayısı ile kestiriminde, güvenilirlik ölçütlerinin önemi büyüktür. Böylelikle test süresinin sonlandırılarak, ürünün pazara sürülme zamanına karar verilmesi veya pazara sürülme zamanı esas alınarak, buna uygun test planlamasının yapılması, ürünün satış sonrasındaki çalışma zamanındaki en iyi ve kötü senaryolar hakkında ön bilgilerin elde edilmesi sonucunu doğurmaktadır ki bu açıdan elde edilecek bilgiler, yönetsel kararları önemli ölçüde desteklemektedir.

Yazılım kalitesinin sağlanmasına yönelik aşağıdaki model ve standartlarda da güvenilirlik mühendisliğinin prensipleri kullanılmaktadır.

- Yazılım Mühendisliği Enstitüsü (*Software Engineering Institute*) tarafından

geliştirilmiş olan Yetenek Olgunluk Modeli (*Capability Maturity Model, CMM*), yazılım süreçlerinin anahtar elemanlarını tanımlayan, süreçlerin olgunluğunun değerlendirmesinde kullanılan ve bunu olgun olmayan bir süreçten, olgun ve etkin bir sürece giden evrimsel bir yol izleyerek gerçekleştiren modeldir ve maliyetlerin denetlenmesinde, süreç iyileştirmelerinin ölçümlenmesinde ve değerlendirilmesinde güvenilirlik önemli bir rol oynamaktadır [18].

- ISO/IEC 9126 yazılım kalite ölçünlerinde, yazılım kalitesinin tanımlanması için ürün karakteristikleri ve bunlara bağlı alt karakteristikler tanımlanmaktadır [19]. Güvenilirlik, temel kalite standartları arasında yer almaktadır ve yazılım ürününün ölçülmesinde kullanılmaktadır.

- IEEE Std 1061 Yazılım Kalite Ölçütleri Metodolojisi, yazılımın arzu edilen kalite seviyesine ulaşmasını sağlamak üzere kullanılacak yazılım ölçütlerini tanımlamaktadır [20]. Aynı dokümanda kalite faktörü, "yazılımın sorumluluklarını yerine getirmesine yönelik fonksiyonlara katkıda bulunan nitelikler" olarak tanımlanmaktadır. Böylece IEEE 1061 ölçütününde önemli bir kalite ölçütü olarak sunulmaktadır.

6 Sonuç

Bu çalışmada yazılım güvenilirliği konusunun önemi vurgulanmış, konu hakkındaki temel eserler ve standartlar taranarak, yazılım güvenilirliğine ait önemli kavramlar açıklanmış, kolaylıkla anlaşılması için örneklenmiştir. Benzer şekilde yazılım güvenilirliğinin kestirilmesi ve kestirime ilişkin süreçler anlatılmış, veri toplama ve yazılım güvenilirliğinin yazılım kalitesi ile olan ilişkileri, yazılım güvenilirliği konusunda ilk kez çalışmayı düşünen geliştiriciler için temel bilgileri içerecek şekilde sunulmuştur.

Böylece konunun programcıları, yazılım proje yöneticilerini bilgilendirici, cesaretlendirici ve teşvik edici olması sağlanmış ve yazılım güvenilirliğine yönelik temel bilgilendirmeler yapılmıştır.

Kaynakça

[1] IEEE Std. 610.12-1990, Standard Glossary of Software Engineering Terminology, New York, IEEE Standards Board.

[2] Vouk, M.A., 2000. Software Reliability Engineering, 2000 Annual Reliability and Maintainability Sym.

[3] Musa, J.D., 1987. Software Reliabilty Engineering, McGraw-Hill.

[4] Ebert, C., Dumke, R., Bundschuh, M., Schmietendorf, A., 2005. Best Practices in Software Measurement, Berlin: Springer-Verlag.

[5] Summerville, N., Basic Reliability, An introduction to Reliability Engineering, Author House, 2004.

[6] IEEE Std 982.1-1988, Standard Dictionary of Measures to Produce Reliable Software, New York, IEEE Standards Board, 1988.

[7] Fenton, N.E., Pfleeger, S.L., 1997. Software Metrics: A Rigorous and Practical Approach, 2. ed., Boston: PWS Publishing.

[8] Pham, H., Zhang, X., 1999. A software cost model with warranty and risk cost, IEEE Transactions on Computers 48 (1), 71-75.

[9] Musa, J.D., Iannino, A., Okumoto, K., 1987. Software Reliability, Singapore: McGraw-Hill.

[10] Dalal, S.R., 2003. Software Reliability Models: A Selective Survey and New Directions, Handbook of reliability engineering, Hoang Pham (ed.), Springer-Verlag.

[11] Pham, H., 2006. System Software Reliability, Springer-Verlag.

[12] Wallace, D., Coleman, C., 2001. Hardware and Software Reliability (323-08), NASA Software Assurance Technology Center.

[13] Stutzke, M.A., Smidts, C.S., 2001. A stochastic model of fault introduction & removal during software development, IEEE Transactions on Reliability 50 (2), 184-193.

[14] Lyu, M.R., 2007. Software Reliability Engineering: A Roadmap, Future of Software Engineering.

[15] IEEE Std. 1044-1993, IEEE Standard Classification for Software Anomalies, New York, IEEE Standards Board.

[16] The Software Engineering Program, 2005. Software Measurement Guidebook, NASA National Aeronautics and Space Administration, NASA-GB-001-94.

[17] Hutcheson, M.L., 2003. Software Testing Fundamentals, Wiley.

[18] Ahern, D.M., Clouse, A., Turner, R., 2004. CMMI Distilled: A Practical Introduction to Integrated Process Improvement, 2. ed. Addison Wesley.

[19] Jung, H., Kim, S., Chung, C., 2004. Measuring Software Product Quality: A Survey of ISO/IEC 9126, IEEE Software, 88-92, September/October 2004.

[20] IEEE Std. 1061-1988, Standard for a Software Quality Metrics Methodology. Piscataway, NJ,: IEEE Standards Dept.