BÖLÜM

8

12. Yazılım Testi

Yazılım geliştirme sürecinin çeşitli aşamalarında, insan ya da başka nedenlerden dolayı çeşitli hataların veya tersliklerin oluşması kaçınılmazdır. Önemli olan, bu gibi olumsuz durumların zamanında belirlenmesi ve giderilmesidir. İnsanların iletişim ve üretim konularında mükemmel olamamalarından dolayı insan kaynaklı hatalar, geliştirme sürecinin en başında yer alan tanımlama aşamasında ortaya çıkmaya başlar.

Hataların oluşmaması, oluşanların da ortadan kaldırılmasını sağlamak üzere yazılım geliştirme işleri nitelik güvence etkinlikleriyle beraber yürütülür. *Yazılım testi* ya da diğer adıyla *sınama*, yazılım geliştirme sürecinin en önemli aşamalarından biri olarak çözümleme, tasarım ve kodlama aşamalarının son bir değerlendirmesi yerine geçer. Sınaması yapılmamış bir yazılım aslında "çalışamaz" demektir. Belirli bir yöntemin uygulanmadığı geliştirme işlerinde sınamanın ne zaman ve nasıl yapılacağı önemli bir sorun oluşturur. Sınamanın olası en erken zamanda ve yapılabildiğince sık yapılması doğru ve sağlam bir ürün geliştirilmesine yardımcı olur. Fakat testin yani sınamanın nasıl yapılacağı mutlaka çözümleme aşamasında tanımlanmalıdır. Yazılımın "ne" yapacağının belirlenmemesi onun doğru olarak çalıştığının sınanmasını da engeller. Bu nedenle, en gevşek geliştirme etkinliğinde dahi, çözümleme, tasarım ve test aşamalarından geçilmesi gereklidir. Daha önce sınanmamış mimari ve tasarımın testleri için gerçekleştirimin sonuna kadar bekleyerek ciddi bir test işleminden kaçınmak büyük sorunlar ortaya çıkarabilir ve zaman kaybına yol açabilir. Test işlemi, gerçekleştirimi izleyen bir düzeltme süreci değildir. "Sonra yaparız" düşüncesiyle ertelenen bir işlem değil, planlaması ve tasarımı yapılan bir etkinliktir. Test aşamasının sonunda yazılım kullanıma hazırlanarak müşteriye teslim edilir.

Yazılım testinin önemi o kadar büyüktür ki, yazılım projeleri için gerekli özkaynakların genellikle üçte birinden daha fazlasını test aşaması için ayırmak yaygın hale gelmiştir. Hele görev kritik ve askeri yazılım projeleri için bu test maliyetleri daha da yüksek olabilmektedir.

Bu bölümde testin önemi üzerinde durulmaktadır; nasıl ve ne tür testler yapılabileceğine değinilmiştir.

12.1. Yazılım Testi/Sınamasının Temelleri (12.1.)

Şekil-8.1 genel süreç içinde yazılım testinin ve onunla beraber olan teslim işleminin yerini göstermektedir:

Şekil-8.1. Yazılım testinin yeri.

Yazılım testi ilk ortaya çıktığı sıralarda yalnızca hata ayıklama amacıyla yapılmaktaydı. Sonra testler yazılımın doğru çalıştığını göstermek amacıyla yapılmaya başlandı. Daha sonraları testlerin yapıcı olmaktan çok, yıkıcı bir şekilde yapılmasının daha iyi sonuçlar verdiği görüldü. 1980'li yıllardan sonra daha kurallı geliştirme teknikleri kullanılmaya başlandığından tüm geliştirme sürecini içeren aşamalı testler kullanıldı. Günümüzde de bu yöntem yanında hataları önlemeye yönelik testler yapılmaktadır.

Yazılım, tasarlanıp kodlandıktan sonra belirli bir mantıkla test edilmeli, istenen sonuçları gerçekten verip vermediği sınanmalıdır. Aslında, test aşamasında yapıcılıktan çok yıkıcılık amaçlanır, dersek pek yanlış olmaz. Yazılım geliştiriciler üretici bir doğaya sahip iken testle görevli kişiler yıkıcı bir tutum içinde olmak zorundadırlar. Testin temel ilkesi, sorunsuz ve hatasız çalışan, doğru sonuçlar üreten, sağlam bir sistem ortaya çıkmasını sağlamaktır. Bunun için de çok acımasız testler yapılması normaldir. İyice sınanmadan müşteriye teslim edilen bir ürün, yerine koyması mümkün olmayan yanlışlıklara neden olabileceği gibi, zaman, emek, para, onur, mal hatta can kaybına neden olabilir. Teslim süresini kısaltmak amacıyla yeterli nitelikte test uygulanmayan yazılım ürünleri için bu son kaçınılmazdır. Geniş bir sistem için yalnızca yazılım testi değil, donanım ve tümleştirmeyi de içeren testler dikkate alınmalıdır. Bunun için de teslim amaçlı olarak sistem kabul testi yapılmalıdır.

Testin en iyi şekilde yapılabilmesi için test mühendisleri tarafından test senaryoları tasarlanmalıdır. Senaryolar, sistemin isterleri karşılayıp karşılamadığını göstermek üzere gerçek koşullar düşünülerek hazırlanmalıdır. Bazı yazılım geliştirme yöntemlerinde, test aşamasını birden fazla alt aşamaya bölmek de mümkündür. Her aşamada belirli bir ayrıntıda test yapılır.

12.1.1. Testin Amaçları

En genel şekliyle bir yazılım ürününün testi, bir yazılım sistemini veya bir birimini kullanıcıya teslim etmeden önce kusur bulabilmek amacıyla çalıştırmaktır. Bir yazılım içinde bulunması olası olan *yazılım hatası* (error) kodlayıcı tarafından yapılmış kodlama yanlışlığı olup bunun sonucunda *hatalı* (defective) bir yazılım ürünü ortaya çıkar. Bu hatalar nedeniyle de yazılım yürütme anında sorun çıkarabilir, yani tıpkı bir otomobil parçasının aracı durdurması gibi *aksama* (failure) durumuna düşürür. Bu sorunlardan kaçınabilmenin tek yolu yazılımı iyice test etmek, yani sınamaktır. Sınamanın da *deneme* ve *kabul* olarak iki amacı olduğunu özellikle vurgulamak gereklidir.

12.1.1.1. Deneme Testleri

Yazılımın kaynak kodu yazılıp derlendikten sonra doğru çalışıp çalışmadığı sınanmalıdır. Bu amaçla geliştiricinin rahat çalışabildiği bir test ortamında, yazılım birimini denemesi, önce birim sonra da tümleştirme testlerini uygulaması gereklidir. Hata bulmak niyetinden çok önemli işlevlerin yerine getirilip getirilmediğini anlayabilmek amacıyla yapılan bu ön testlerin amaçlarını şöyle sıralayabiliriz:

- Kodlanıp derlenen, sonra da bağlanıp yürütülebilir hale getirilen yazılım biriminin çalışıp çalışmadığını denemek;
- Birimin girdilere doğru işlem yapıp doğru çıktı ürettiğini denemek;
- Birimin tüm isterleri karşıladığını denemek;
- Birimin yanlış girdiler alması halinde hataya dayanıklılığını denemek;
- Birimin ilk çalıştırma sırasında ve sonlandırma sonrasında sistem özkaynaklarını nasıl kullandığını ve serbest bıraktığını denemek ve
- Bulunan her hatadan sonra başka bir hata olup olmadığını araştırmak.

Bu denemeleri yapabilmek için küçük ölçekli yazılımlarda özel test senaryolarının tasarlanmasına gerek olmayabilir. Ancak, büyük çaplı yazılımlarda, bu test senaryolarının önceden tasarlanması, belgelere geçirilmesi ve uygulanması gereklidir. Belgelendirme, daha sonra aynı yazılım birimi üzerinde değişiklik yapılması durumunda da kullanılır.

12.1.1.2. Kabul Testleri

Yazılım ürününün tamamının istendiği gibi doğru çalıştığını kullanıcıya veya onun temsilcisine ya da proje örgütü içindeki yetkili bir makama göstermek amacıyla, uygulama alanına bağlı olarak, önce üretim yerinde sonra da kullanım yerinde testler yapılır. Bu testler için genellikle ayrı test ekipleri bulunur. Bu testlerin amaçlarını şöyle sıralayabiliriz:

- Kullanıcıya, tanımlanan isterlere göre sistemin doğru çalıştığını göstermek, yani doğrulamasını (verification) yapmak;
- Sistemin kullanıcının amaçladığı şekilde kullanımı için yeterli olduğunu, doğru sistemin geliştirildiğini, yani geçerliliğini (validation) göstermek;

- Kullanıcının isterlerini çalışan sistem üzerinde bir kez daha gözden geçirip amaçlarına tam uyacak şekilde ince ayarlar yapabilmesini sağlamak;
- Uygulama alanının özelliğine göre, ya gerçek durumlarda ya da ona en yakın benzetim olanaklarıyla sistemi kullanım yerinde denemek,
- Sistemi olabildiğince ağır yükte ve en kötü koşullarda çalıştırarak güvenilirliğini kanıtlamak.

Küçük ya da büyük her türlü yazılımın kabulü için test senaryoları tasarlanmalı, belgelerde tanımlanmalı, ürünü kabul edecek müşteri ya da makam tarafından onaylanmalıdır. Test belgesinin onaylanması, test kıstaslarının taraflarca kabul edilmesi anlamına gelir.

12.1.2. Testin Yapılışı

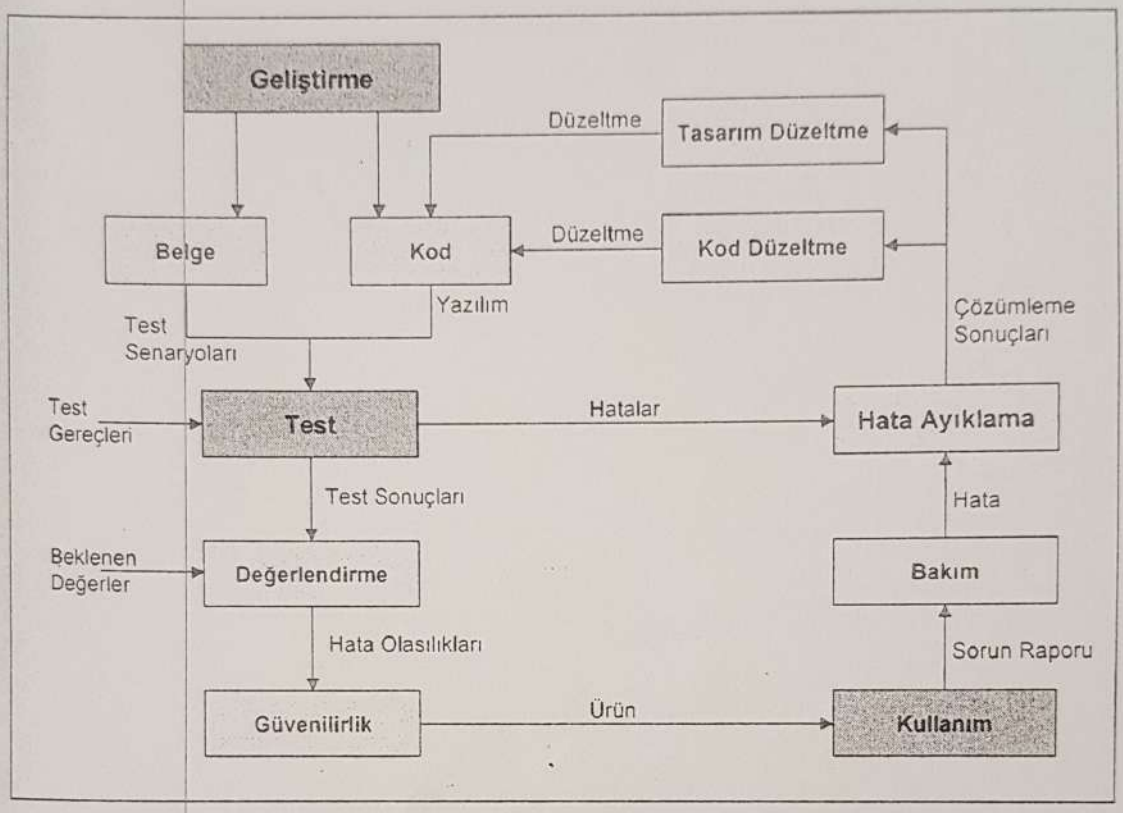
Yazılım sistemlerinin testinin yapılabilmesi için aşağıdaki girdilerin bir düzenleşim sisteminde bulunması gereklidir:

- Yazılım isterleri belirtimi (Nelerin test edilmesinin istendiğini belirtir.)
- Tasarım belirtimi (Nelerin test edilmesinin gerektiğini belirtir.)
- Kaynak kod (En son üretim ve hata bulma amacıyla kullanılır.)
- Test ortamı (Hedef sistem donanımının bir benzeri veya kendisidir.)
- Test planlaması (Hangi testin ne zaman ve nasıl yapılacağını belirtir.)
- Test tanımlaması (Test senaryolarını ayrıntılı olarak anlatır.)
- Test yardımcı gereçleri (Ölçüm ya da veri giriş/çıkışı için kullanılır.)
- Benzetim araçları (Gerçek ortamdaki davranışları denetim altında yaratabilmek üzere kullanılır.)

Bütün bunlar sağlandıktan sonra, düzenleşim sisteminden çekilerek üretilen ve test sistemine yüklenen yazılıma test personeli tarafından test senaryoları tek tek uygulanır, sonuçları daha sonra değerlendirilmek üzere kaydedilir, kaydedilen sonuçlar olması gereken değerlerle karşılaştırılır; farklılık bulunması halinde nedenleri değerlendirilerek hataların bulunmasına çalışılır. Test değerlendirmesi sonunda, ya tatmin edici bir sonuç elde edilerek yazılımın yeterli niteliğe sahip olduğuna kanaat getirilir; ya da yazılım testi geçemeyerek hataları düzeltilmek üzere tekrar üreticiye geri verilir. Testlerin olumlu geçmesi durumunda hatasız veya düşük hata olasılığına sahip ürün kullanıcıya teslim edilir.

Olumsuz test sonuçları kaynak kodun düzeltilmesiyle giderilebileceği gibi tasarımın hatta isterlerin dahi değiştirilmesine neden olabilir. Bu değişme ve düzeltmelerden sonra kaynak kodlar ve etkilenerek değiştirilen belgeler tekrar gözden geçirilip onaylanarak düzenleşim sistemine konur ve test tekrar edilir.

Test süreci boyunca gerçekleşen aşamalar Şekil-8.2 de özetlenmektedir:



Şekil-8.2. Test aşamaları.

Örneğin, bir veritabanı yönetim sisteminde saniyede 10000 erişim yapılması bir ister olarak ortaya konmuş olabilir. Yapılan testlerde bu değere ulaşmak mümkün olmayıp 2000 düzeylerinde kalınabilir. Kaynak kod incelemesinde herhangi bir yanlış kodlama veya verimsizlik bulunamadığı takdirde, tasarımda belirlenen veri yapısı ve erişim algoritmasında bir eksiklik aranır. Burada da bir hata görülmemesi üzerine müşteri ile anlaşılarak isterde belirtilen erişim düzeyini mevcut donanım, işletim sistemi ve yazılım mimarisi ile karşılanabilecek bir değere çekmek gerekebilir. Böyle sonuçlar, çözümlemenin iyi yapılmadığı ya da var olan teknolojik kısıtlamaların tam olarak dikkate alınmadığı durumlarda ortaya çıkabilir. Bazı durumlarda önemli maddi kayıplar ortaya çıkabilir.

12.1.3. Test Yöntemleri

Her türlü üretimde olduğu gibi, yazılım ürünü testinin iki ana amacı vardır: Bunlardan birincisi olan *deneme*, bir ürünün tüm iç yapısının doğru çalıştığından emin olmak üzere yapılır. İkincisi olan *kabul testleri* ise ürünün yerine getirmesi gerekli olan tüm işlevleri başarıyla gerçekleştirdiğinin gösterilmesidir.

Birinci test yaklaşımına *saydam kutu testi* (white-box test), ikinci test yaklaşımına da *kara kutu testi* (black-box test) adı verilmektedir [5].

Şimdi bunları ayrıntılarıyla inceleyelim.

12.1.3.1. Saydam Kutu Testi

Yazılımın bu test için saydam bir kutuya benzetilmesi onun tüm iç yapısının (yordamların, veri yapılarının, değişkenlerinin, iletişim biçimlerinin) ortaya konmasından kaynaklanmaktadır. Veri akışları, bunların yazılım içinde izlediği mantıksal ve fiziksel yollar, çeşitli test durumları ile test edilirler. Saydam kutu testini de iki aşamaya ayırmakta yarar vardır:

- **Tasarım tabanlı test**
  İsterler çözümlemesi sonunda ortaya çıkan sistem gereksinimleri, tasarıma esas olacak sistemi tanımlar. Ancak, çoğu zaman, bu tanımlamalar kodlama aşaması için fazla üst düzeyde kalır. Bu nedenle isterler daha ayrıntılı hale getirilerek yazılım birimleri olarak kabul edilen modüllere paylaştırılır. Testler sırasında bu modüller birer kara kutu olarak kabul edilerek, arayüzleri ile tanımlanan girdi-çıktı verilerine göre test durumları tasarlanır ve o şekilde test edilirler. Yazılım daha küçük parçalara bölündüğü için bu tür testler daha alt düzeyde yapılırlar. Örneğin, bir ileti alındığında, iletinin herbir alanının tipine ve sınırlarına göre doğru değerlendirme yapıldığının, sonuçta beklenen değer aralıklarında veriler elde edilmekte olduğunun testi yapılır. Bu tür testler genellikle kodlayıcının kendisi tarafından en basit test ortamında yapılır.
- **Kod tabanlı test**
  Bazı durumlarda yalnızca kara kutu testi yapmak o yazılım biriminin güvenilir olduğunu kanıtlamayabilir. Örneğin, belirli bir isteri doğru olarak gerçekleştirdiğini gösteren bir kara kutu testi sırasında, modüllerden birinin içinde o test sırasında kullanılmamış bir program akış yolu (flow path) bulunabilir. Yani, `if` ya da `switch` deyimlerinden bazılarına hiç girilmemiş olabilir. Girilmemiş bu deyimlerde önemli hatalar bulunabilir ve testi geçtiği sanılan yazılım, kullanım sırasında bu mantıksal akış yollarından birinin gerçekleşmesi halinde çöküşe neden olabilir. Kod içinde hata üretebilen başlıca noktalar arasında şunları sayabiliriz:
  - *Mantık hataları:* Bir işi bilgisayara yaptırabilmek için kullanılan kod parçalarının veya deyimlerin yanlış yerleşimi, bir değişkene ilk değer verilmemesi, bir isterin yanlış yorumlanması gibi nedenlerden dolayı mantıksal hatalar yapılır ve çalışan fakat yanlış sonuç verebilen yazılımlar ortaya çıkar.
  - *Kodlama hataları:* Yeterli deneyimin bulunmadığı kodlama işlemleri sırasında kodlayıcı dilin esnekliği nedeniyle yanılgıya düşüp önemli hataların oluşmasına neden olabilir. Örneğin, dinamik veri yapıları üzerinde öbek kopyalama sırasında dizi boyunun aşılması, işaretçilerin yanlış yerleri göstermesi gibi hatalar yüzünden bazı testlerden geçtiği halde güvenilirliği son derece zayıf olan yazılımlar üretilebilir. Bir örnek vermek gerekirse, veritabanına kayıt ekleme ve çıkarma işlemi başarılı olabilir; fakat tasarımda dikkate alınmayan bir şekilde, aynı kayıt tekrar eklenirse yazılımın davranışının ne olacağı belirsiz hale gelir.

- *Akış yolu varsayımı:* Yazılım içinde denetim akışının düzenli olarak hep aynı sırayı izlediğini varsaymak bir hataya yol açabilecek tehlikedir. Hiç öngörülmeyen bir anda, farklı bir akış yolunun izlenmesi yazılımın hata üretmesine neden olur. Örneğin, bir girdinin alabileceği değerlerin 1 ile 10 arasında olması gerektiği varsayımına göre tasarlanan ve kodlanan bir yazılım birimine -1 ya da 150 gibi bir değer gelmesi halinde, ya değer reddedilmeli ya da farklı bir dallanma izlenmeli ve hataya düşülmemelidir.
- *Yazım hataları:* Günümüzde kullanılan birçok derleyici, kod yazım hatalarını yakalayıp hata vermekte ve kod düzelinceye kadar derleme yapmamaktadır.

Belirtilen bu nedenlerden dolayı, *yürütülebilir kod* olarak kabul edilen bir yazılım birimini oluşturan önemli deyim satırlarının da bir şekilde test edilmesi gereklidir. Zira, kara kutu testi bu hataları yakalayamaz. Hiç değilse önemli iş yapan kod parçaları ayrı bir yazılım içinde yerel olarak denenmelidir. Örneğin, bir aygıttan okunan sekizli dizisi şeklindeki veriyi tarayıp çözümleyen kısım tüm yazılımla beraber değil de ayrı bir şekilde denenmelidir. Aksi takdirde, diğer işlevlerin de yanlış çalışmasına neden olarak testlerde zaman ve emek kaybına neden olabilir.

12.1.3.2. Kara Kutu Testi

Yazılım testi ele alındığında, kara kutu testi yazılımın arayüzü düzeyinde yapılır. Her ne kadar test hata bulmak için yapılsa da asıl amaç yazılımın işlevlerini yerine getirdiğini göstermektir. Bu amaçla, yazılımın tamamına ya da test edilmekte olan birimine arayüzde belirtilen girdi sağlanır ve doğru çıktıyı vermesi beklenir. Girdi ya da çıktılar çeşitli ortamlarda sağlanan veri ya da bilgi olabileceği gibi bir donanımdan veri alınması (ısı algılayıcısında veri okuma) veya bir donanıma kumanda edilmesi (bir kapıyı açma) de olabilir.

Bu testlerde isterler çözümlemesi sonunda belirlenen sistem gereksinimleri esas alınarak kullanıcıya yönelik test durumları yaratılır, sonuçları gözlemlenir. Kullanıcı, bu testler sırasında yazılımın iç yapısıyla ilgilenmez. Testlerin düzeyi kullanım alanına göre değişiklik gösterebilir. Örneğin, ürünün ilk kez test sisteminde denenmesi başka olurken (bir uçuş sisteminin laboratuvar koşullarında denenmesi), gerçek ortamda denenmesi (bir uçak üzerinde havada denenmesi) başka olur.

10. Hft
1. Grp
Sa