T.C. HALİÇ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ Y¦ KSEK LİSANS PROGRAMI

YÜKSEK LİSANS TEZİ

YAZILIM MÜHENDİSLİĞİ YÖNTEMLERİYLE YAZILIM TEST SÜRECİ

Danışman
Yrd. Doç. Dr. Ulviye Hacızade

Hazırlayan
Güneş Kuday

İstanbul, 2014

ÖNSÖZ

Bu bitirme projesi çalışması boyunca bilgi ve deneyimlerini benimle paylaşıp yapıcı eleştiriyle çalışmalarımı yönlendiren, bana her konuda destek olan değerli hocam, tez danışmanım Yrd. Doç. Dr. Ulviye Hacızade'ye teşekkür ederim.

Ayrıca Haliç Üniversitesi'nde aldığım yüksek lisans eğitimi boyunca bana emek veren, maddi ve manevi desteğini sonuna kadar sürdüren aileme sonsuz anlayışları için teşekkürlerimi sunarım.

Aralık, 2014 Güneş Kuday

İÇİNDEKİLER

TABLO SİTESİ

ŞEKİL LİSTESİ

GENEL BİLGİLER

Adı ve Soyadı: Güneş KUDAY

Anabilim Dalı: Bilgisayar Mühendisliği

Programı: Bilgisayar Mühendisliği

Tez Danışmanı: Yrd. Doç. Dr. Ulviye HACIYEVA

Tez Türü ve Tarihi: Yüksek Lisans - 2014

ÖZET

YAZILIM TEST MÜHENDİSLİĞİ YÖNTEMLERİYLE YAZILIM TEST SÜRECİ

Anahtar Kelimeler: Yazılım test mühendisliği, yazılım test süreci, test raporlama

Bu çalışmada yazılım mühendisliğinin yazılım test süreci detaylıca incelenip test süreci, test yöntemleri, test yönetimi, yazılım test düzeyleri ve teknikleri, test belgelendirilmesi, test yazılım araçları ve test otomasyonu açıklanmıştır.

Test süreci ve test yönetiminde test planlamasının yapılması, test ortamının hazırlanması, test gerçekleştirme, hata yönetimi ve raporlama gibi öğeler incelenip test süreci hakkında temel bilgiler açıklanmıştır.

Yazılım test düzeylerinde birim testler, tümleştirme testleri, sistem testleri, kabul testleri gibi test sürecinde uygulanan testler açıklanmış, bunların kara kutu, saydam kutu, gri kutu gibi test teknikleriyle nasıl yapıldığı gösterilmiştir.

Test belgelendirmesinde tamamlanan veya yenilenecek testlerin, yazılımın kalitesi açısından nasıl belgelendirilmesi gerektiği anlatılmıştır.

Test yazılım araçları ve test otomasyonunda ise bazı testlerin otomatik neden ve nasıl gerçekleştirildiği ve bazı yazılım test araçları açıklanmıştır.

GENERAL INFORMATION

Name and Surname: Güneş KUDAY

Field: Computer Engineering

Program: Computer Engineering

Supervisor: Assist. Prof. Dr. Ulviye HACIYEVA

Degree Awarded and Date: Master of Science - 2014

SUMMARY

SOFTWARE TEST PROCESS IN SOFTWARE ENGINEERING

Keywords: Software test engineering, software test process, test reporting

In this study, test process, test methods, test management, software test levels, test feedback and reporting, software test tools and test automation of software test engineering had been researched in detail.

In test process and test management, test planning, readying test environment, test running, error management and reporting issues are inspected and fundamental knowledge about test process are described.

In software test levels, knowledge about tests such as unit tests, integration tests, system tests, acceptance tests using with black box, white box, gray box test technics had been described.

In test reporting, how the reporting should be done for software's quality with the tests which are finished and will be reruned.

In software testing tools and test automation, why and how some tests must be automated and some testing tools are described.

1. GİRİŞ

Yazılım mühendisliğinin bir alt dalı olan ve gün geçtikçe büyüyen yazılım test mühendisliği, yazılım projesi müşterisine yazılımın kalitesi veya servisi hakkında bilgi veren bir araştırma yatırımıdır.

Yazılım testi, bir iş dalına kendi yazılımının piyasaya çıkmasından önce riskler hakkında bilgiler veren eylemlerdir. Yazılım testi, bir yazılım vey sistem parçasının gereksinimlerinin tasarım ve geliştirme aşamalarında karşılanıp karşılanmadığının, her türlü girdiye karşı gereken çıktıyı üretip üretmediğinin, geçerli bir süre içinde işlemleri gerçekleştirip gerçekleştirmediğinin, yeterince kullanışlı olup olmadığının, önceden tanımlanmış donanıma sorunsuzca yüklenip çalıştırıldığının, müşterinin isteklerini yerine getirip getirmediğinin kontrolünün yapılmasıdır.

Testler teorik olarak sonsuza kadar yapılabilir; ancak eas amaç, projenin bitiş süresi içinde testlerin en verimli şekilde yapılıp son ürünün olabildiğince sorunsuz çıkmasını sağlamaktır. Yazılım testleri, yazılım hakkında bağımsız fikir verip bir hata sonucunda ortaya çıkabilecek riskler hakkında senaryolar oluşturur.

Bu tezin genel bilgiler kapsamında yazılım felaketleri, yazılım testlerinin amacı, test stratejisi anlatılmıştır. Test süreci ve test yönetimi kapsamında yazılım test süreci detaylı olarak ve test yönetimi açıklanmıştır. Yazılım test düzeyleri bölümünde tek tek birim testler, entegrasyon testleri, sistem testleri, kabul testleri anlatılmıştır. Yazılım test teknikleri bölümünde kara kutu, beyaz kutu ve gri kutu test teknikleri anlatılmıştır. Yazılım test türlerinde statik testler, gözden geçirmeler, kod analizleri ve dinamik testler açıklanmıştır. Test belgelendirmesinde dökümantasyonların nasıl yapılabileceği anlatılmıştır. Yazılım test araçları ve test otomasyonu bölümünde kullanılan test araçları hakkında genel bilgiler verilmiştir. Son bölümde Plague Inc Evolved adlı oyunun kullanıcı arayüzü ve kabul testleri yapılarak test mühendisliğinde pratiklik kazanılmıştır.

2. YAZILIM MÜHENDİSLİĞİ NEDİR?

Yazılım mühendisliğinin ilk tanımı 1969 yılındaki konferansta Fritz Bauer tarafından şu şekilde tanımlanmıştır: *"Gerçek makineler üzerinde etkin ve güvenilir çalışan ekonomik yazılımlar geliştirilmesi amacıyla mühendislik ilkelerinin kullanılmasıdır"*. 1969'dan bu yana bu alanda bir çok çalışma gerçekleştirilmiş ve sonuçları yayınlanmıştır. Günümüz itibariyle yazılım mühendisliği, yazılım geliştirmenin her adımıyla ilgilenen bir mühendislik disiplinidir (sqaforums, 2009). Yazılım dünyasında hedeflenen zamanda, hedeflenen bütçede ve hedeflenen işlevleri yerine getiren yazılımların üretildiği projeler başarılı yazılım projeleri olarak kabul edilir. Yazılım mühendisliğinin amacı, başarılı yazılım projelerinin sayısını arttırmak ve bu projelerden elde edilen verilerle yazılım geliştirme alanında disiplinize olan yöntemler geliştirmektir. Bu amaçla yazılım mühendisliği daha kaliteli, daha güvenilir yazılımlar geliştirmek için bu alanda yapılan çalışmalara sistematik bir yaklaşım getirir. Yazılım geliştirmedeki eylemleri, bu eylemler ile ilgili ürünleri ve eylemlerin nasıl ve hangi sırada yerine getirileceğini tanımlar.

2.1 Yazılım Geliştirme Süreci

Yazılım geliştirme süreci bir yazılım ürününün geliştirilmesi ile ilgili bir grup eylemdir. Yazılım geliştirme süreci tanımlama (definition), ayrıntılandırma (elaboration), gerçekleştirme (construction), değerlendirme (evaluation), yayma (transition) ve yönetim (management) aşamalarından oluşmaktadır.

Tanımlama: Yazılım geliştirmenin ilk aşamasıdır. Bu aşamada amaçlanan yazılımın ne yapacağı, hangi probleme çözüm getireceği üzerinde durulur. Yazılımın hedeflenen işlevleri yerine getirirken nasıl bir ortamda çalışacağı ve ne gibi kısıtlamaları olacağı tanımlanır.

Ayrıntılandırma: Bu aşamada, problemin yazılımla nasıl çözüleceği üzerine odaklanılır. Tanımlama aşamasında problem tüm yönleriyle ortaya konulmuştur. Yazılımın hedefi belli olmuştur. Bu aşamada bu hedefe nasıl gidileceğine dair çalışmalar yapılır, farklı düzeylerde (genel ve ayrıntılı) modeller geliştirilir. Bu modeller ile geliştirilecek olan yazılımın hangi alt sistemlerden oluşacağı, bu sistemler arasındaki etkileşimlerin (bilgi alışverişi) nasıl olacağı tanımlanır.

Gerçekleştirme: Ayrıntılandırma aşamasında geliştirilen modeller programlama dilleri ile kodlanır. Her bir kod parçasının çalıştığını göstermek için kod parçalarının testleri gerçekleştirilir. Daha sonra çalışan her bir kod parçası bir araya getirilerek hedeflenen yazılım geliştirilir.

Değerlendirme: Bu aşama geliştirilen yazılımın hedeflenen soruna çözüm getirip getirmediğinin değerlendirilmesidir. Bu amaçla gözden geçirmeler ve testler kullanılarak geliştirilen yazılımın doğru çalıştığı ve kendinden beklenen işlevleri yerine getirdiği doğrulanır.

Yayma: Değerlendirilmesi tamamlanan yazılım kullanılmak üzere sahaya sürülür veya kullanıcılara teslim edilir. Bu aşamadan sonra sahadan ve kullanıcılardan gelen geri bildirimler ile yazılım üzerinde düzeltici, uyarlayıcı, iyileştirici ve önleyici bakımlar gerçekleştirilebilir.

Yönetim: Yazılım projeleri kapsamında tanımlama evresinden yayma evresine kadar gerçekleştirilen faaliyetlerin yönetimsel eylemleri bu aşamada gerçekleştirilir. Yazılım geliştirme süreci içerisinde yönetim aşaması ve değerlendirme aşaması diğer aşamalara paralel olarak gerçekleştirilir.

2.2 Yazılım Süreç Modelleri

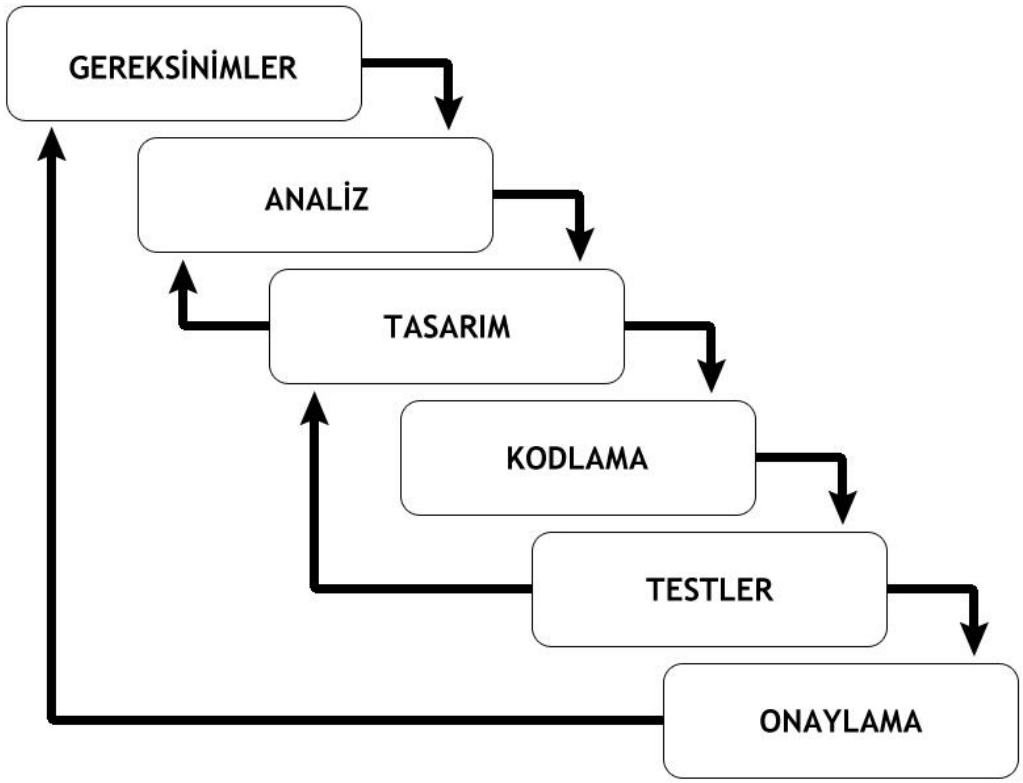
Yazılım modelleri, yazılım geliştirme sürecini özel bir bakış açısıyla ele alan soyut gösterim ve uygulama biçimleridir.

2.2.1 Şelale (Waterfall) Modeli

Şelale modelinde (klasik model), yazılım geliştirme aşamaları ardışık olarak ilerleyen adımlardan oluşmuştur.

Şelale modeli yazılım projelerinde kullanılan en yaygın süreç modelidir. Bu yapıda yazılımın geliştirilmesi doğrusal bir yol izler. Yani bir aşama tamamlandıktan sonra diğer aşama başlar. Bu nedenle yazılım geliştirmenin ilerideki aşamalarında bir sorunla karşılaşıldığında önceki aşamalara dönüş zordur. Bu tür dönüşler genellikle ek maliyet ve ek

zaman gerektirir. Şelale modelinin temsili , Şekil 2.1.'de gösterilmiştir.



Şekil 2.1. Şelale Modeli (yazgelistir, 2012)

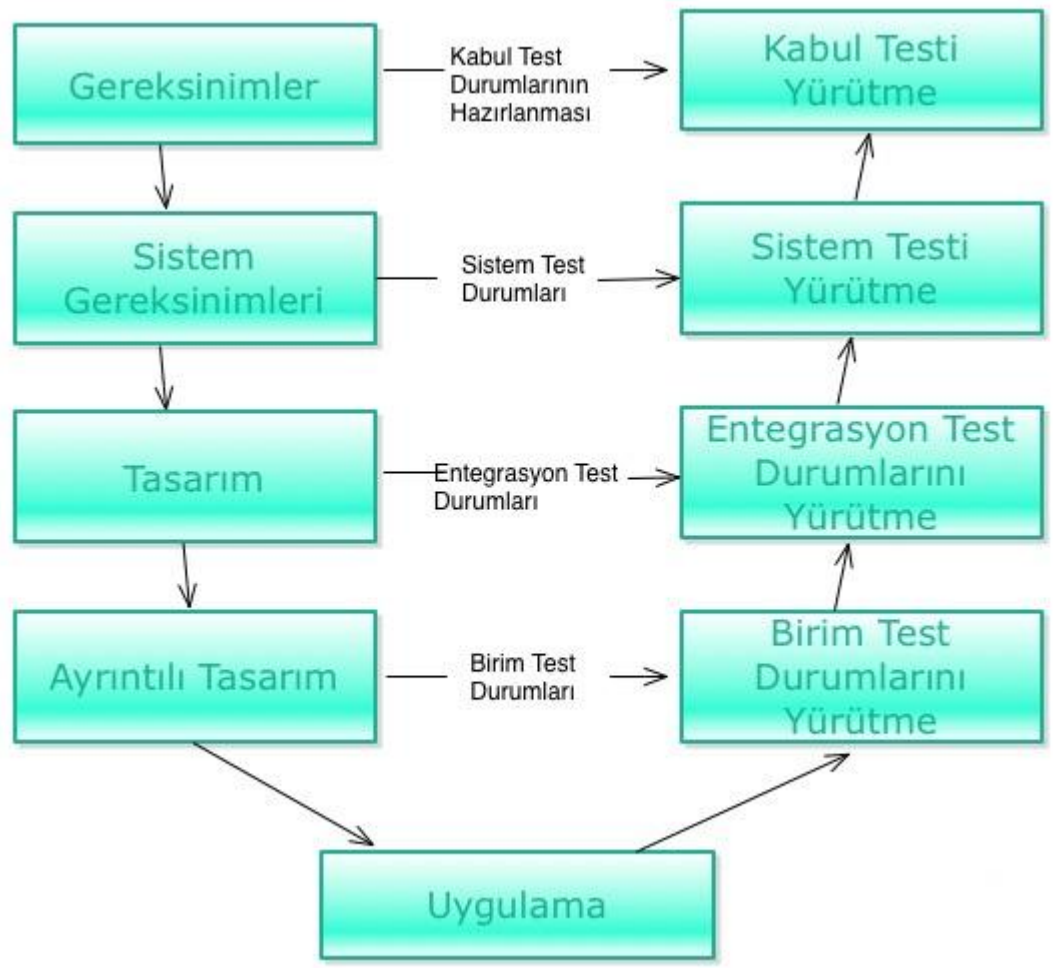
Yazılım projelerinde şimdiye kadar en çok kullanılan bir süreç modeli olmasına rağmen bu modelin uygulanmasında şu sorunlar vardır:

1. Bu modelde uygulama sırasında bir önceki aşamalara geri dönüş öngörülse bile geri dönüşlerin proje maaliyetine ve takvimine getirdiği yükler çoktur.
2. Geri dönüşler proje ekibi arasında sorunlara yol açabilmektedir.
3. Müşterinin tüm gereksinimlerinin işin en başında eksiksiz belirlenmesi güçtür. Bunun nedeni her zaman her projede olduğu gibi başlangıçta bazı belirsizlikler olabilmesidir. Bazı durumlarda kullanıcı isteklerini yazılım geliştirmenin erken bir evresinde tam olarak ifade edememektedir.
4. Bu modele göre çalışan bir program projelerin çok sonrasında ortaya çıkmaktadır. Bu da geliştirilen yazılımla ilgili müşterinin geribildirimi yapması için bazen çok geç olabilmekte, müşteri memnuniyetsizliği olabilmektedir.
5. Müşteri yazılımlar ortaya çıktıkça tepkisini vermeye başladığında yeni gereksinimler ortaya çıkabilmektedir. Bu da genellikle ciddi anlaşmazlıklara neden olmaktadır.

Şelale modeli en eski yazılım süreç modeli olarak yazılım geliştirmenin nasıl yapılacağı konusunda günümüzde tanımlı diğer modellere dayanak oluşturmuştur.

2.2.2 V-Modeli

V-modeli şelale modelinin genişletilmiş bir yaklaşımıdır. Bu yaklaşımda daha güvenilir ve doğru yazılımlar geliştirmek için her bir yazılım geliştirme süreç adımının karşılığı olan test eylemleri model içerisinde gösterilmiştir. V-modeli aynı zamanda doğrulama eylemlerini tanımlama ile birlikte başlattığından proje zamanında büyük kazanımlar getirir. Bu modelde V'nin sol ayağı gereksinim analizi, sistem tasarımı, yazılım tasarımı, gerçekleme (kodlama) adımlarını kapsamaktadır. V'nin sağ ayağı ise modelin doğrulama adımlarını içerir. Bunlar birim testi, tümleştirme testi, sistem testi ve kabul testlerini kapsar. V-modelinin temsili, Şekil 2.2.'de gösterilmiştir (salyangoz, 2013).



Şekil 2.2. V-modeli

2.2.3 Prototip Geliştirme Modeli

Yazılım projelerinin en iyi uygulamaları incelendiğinde görülmektedir ki, projenin başında kullanıcıların ancak ihtiyaçlarına ilişkin çok genel tanımlar verebildikleri gözlenmiştir. Geliştirilecek olan yazılımlar için gerekli girdileri, işlemleri ve çıktıları ayrıntılı bir şekilde tanımlayamazlar. Bu durumda kullanıcıların istediği yazılımların geliştirilmesi, gereksinimlerin belirsizliğinin ortadan kaldırılması için prototip geliştirme modeli kullanılabilir.

Bu modelin kullanılmasının karar verildiği durumlarda tespit edilebilen gereksinimler hızlıca toplanarak işe başlanılır. Bu iş için kullanıcılar ile birlikte geliştiricilerin de katıldığı toplantılarda geliştirilecek olan yazılımdan beklenen işlevler ile bu yazılımın girdileri ve çıktıları üzerinde hızlı bir fikir birliğine varılır. Daha sonra tespit edilen bilgiler ışığında hızlıca bir tasarım ile yazılımın kullanıcıya yansıyacak yönünü anlatan bir prototip oluşturulur. Bu yazılım kullanıcının kullanımına ve değerlendirilmesine sunulur. Bu değerlendirmelere bakılarak yeni gereksinimler tespit edilmeye çalışılır. Böylece kullanıcının istediği yazılımın ne olduğu anlaşılıncaya kadar bu çalışmaya devam edilir. Gereksinimler yeterince ayrıntılı hale getirildikten sonra istenilen sistem geliştirilir. Bu yaklaşımda kullanıcılar prototip yazılımı gerçek yazılım olarak algılayabilmektedir. Oysaki gereksinimleri tespit için geliştirilen örnek yazılım genelde yavaş, verimsiz, güvenirliliği az ve hataları fazladır. Prototip geliştirme modeli gereksinimlerin net olarak bilinmediği durumlarda başarılı bir şekilde uygulanabilir.

2.2.4 Çevik Modeller

Çevik modeller 1990'ların ortasında zor uygulanan, aşırı kuralcı klasik yazılım süreç modellerine tepki olarak ortaya çıkmıştır. Şelale ve V gibi klasik modeller doğalarında var olan aşırı belgelendirme faaliyetleri ve sıralı yaklaşımları ile daha yüksek maliyetle daha yavaş yazılım geliştirmeye sebep olarak görülmüştür. Yazılım geliştirme sürecini hızlandırmak amacıyla bu süreci daha etkin kullanmak ve gerektiğinde belgelendirme yapmak temeline dayalı çevik yazılım geliştirme yöntemleri ortaya çıkmıştır. Çevik geliştirme modelleri şu ilkelere dayanır:

i. Hızlı, devamlı ve kullanışlı yazılımlar üreterek müşteri memnuniyetini sağlamak amaçlanmalıdır.

ii. Çalışan yazılım gelişimin en önemli ölçüsüdür.

iii. Cevap verilemeyen veya geç cevap verilen talepler memnuniyeti azaltır.

iv. En iyi iletişim karşılıklı görüşmedir.

v. Basitlik önemlidir.

vi. Kendi kendini organize eden takım yapısı gereklidir.

Yazılım geliştirme amacıyla üretilen bu yöntem klasik yazılım geliştirme modellerine göre yazılım geliştirmeyi daha esnek hale getirir. Bu yöntem müşterinin isteklerinin net olmadığı, yazılım geliştirmenin her aşamasında müşteri isteklerinin değişebildiği, hedeflenen sistemin hemen görülmek istendiği durumlarda kullanılabilir.

Başlıca çevik modeller:

i. Uç (Sınırsal) Programlama (Extreme Programming)

ii. Çevik Birleştirilmiş Süreç (Agile Unified Process)

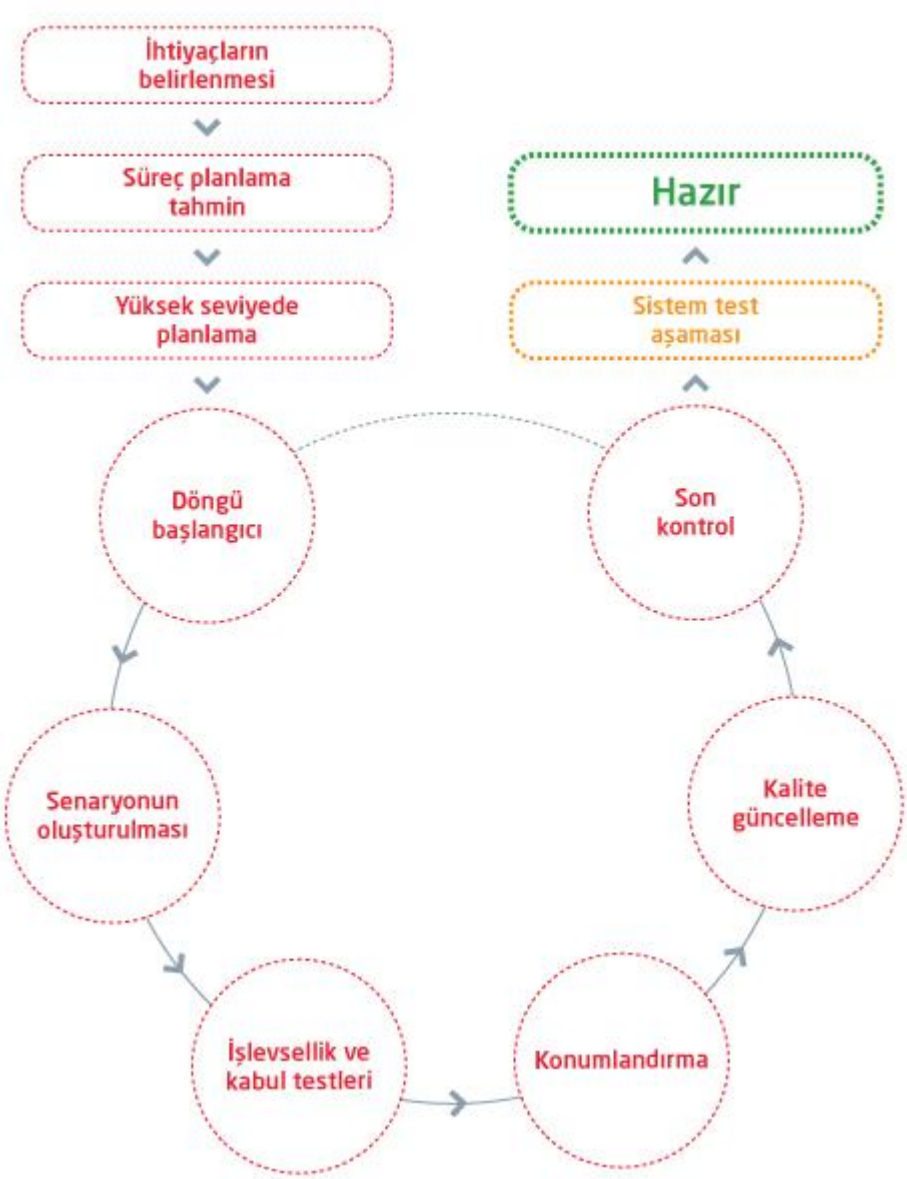
iii. Scrum

iv. Test Güdümlü Geliştirme (Test Driven Development)

v. Çevik Bilgi Yöntemi (Agile Data Method)

vi. Özellik Güdümlü Geliştirme (Feature Driven Programming)

Çevik yazılım geliştirme kısa vadeli planlar ve küçük gelişmeler halinde yazılımın geliştirilmesini öngörür. Kısa vadeli planlar yazılımın geliştirilmesinde yineleme getirir. Her yinelemede yazılım üzerine yeni özellikler eklenir. Çevik yazılım geliştirme değişikliklere uyum sağlamayı kolaylaştırır. Genellikle çevik geliştirmede tek bir yineleme ile ürüne hazır ürün çıkartılamaz. Fakat her yineleme sonunda en az sorunla çalışan hedeflenen yazılıma bir adım daha yaklaşmış bir sürüm elde edilir. Bir yazılımı istenen özellikte tamamlamak için birden çok yineleme gerekir. Çevik geliştirmede yazılımla ilgili belgeler gereken durumlarda üretilir. Bundaki amaç, belgelendirmenin en az insan gücü ile üretilmesi ve belgelendirme için harcanan çabanın işlevselliği ve verimliliği için harcanan çabanın önüne geçmemesidir. Çevik model Şekil 2.3.'de gösterilmiştir (innova, 2013).



Şekil 2.3. Çevik model yazılım geliştirme süreci

2.2.5 Evrimsel Geliştirme Modelleri

Evrimsel geliştirme modellerinde, ilk gerçekleştirimi hızlı yapıp onun üzerinde kullanıcının tepkilerini alıp iyileştirmeyi öngörerek yazılım geliştirilmesi hedeflenir.

Başlıca evrimsel geliştirme modelleri:

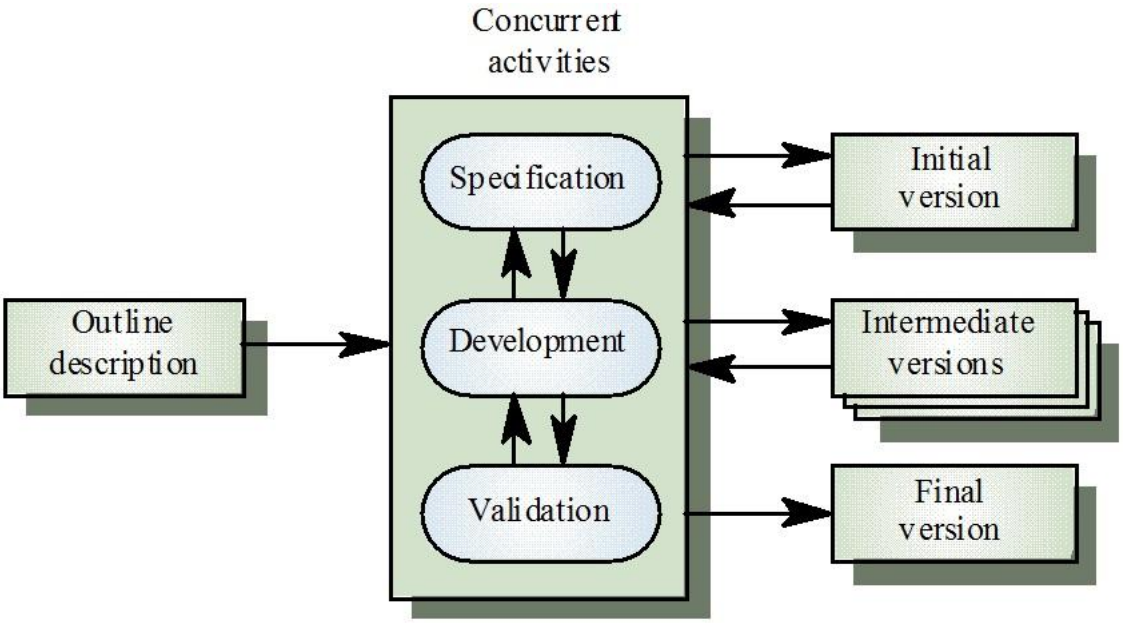
i. Araştırmacı (explatory) geliştirme

ii. Atılabilir (throwaway) prototipleme

Araştırmacı geliştirmede yazılım geliştirilmeye daha anlaşılır bir kesim ile başlanır; üstüne yeni özellikler eklenerek devam edilir. Bu yaklaşım yazılımın gereksinimlerinin daha iyi anlaşılmasını hedeflemektedir.

Atılabilir prototiplemede amaç yazılım gereksinimlerini keşfetmektir. Bu nedenle ilk anda tespit edilen gereksinimlerden bir prototip yazılım geliştirilerek müşteriye gösterilir. Buradan hareketle müşterinin kafasındaki hedef yazılım keşfedilmeye çalışılır. Gösterilen prototip daha sonra bir daha kullanılmadan atılabilir.

Evrimsel geliştirmenin başarısı büyük ilk evrimde ortaya çıkan ürüne bağlı olduğu bu modeli kullanan yazılım projelerinde görülmüştür. Kısa sürede yazılımın geliştirilmesinin istendiği küçük ve orta büyüklükteki projelerde kullanılabilir. Evrimsel geliştirme modeli Şekil 2.4.'tedir (ozgursaracoglu, 2011).

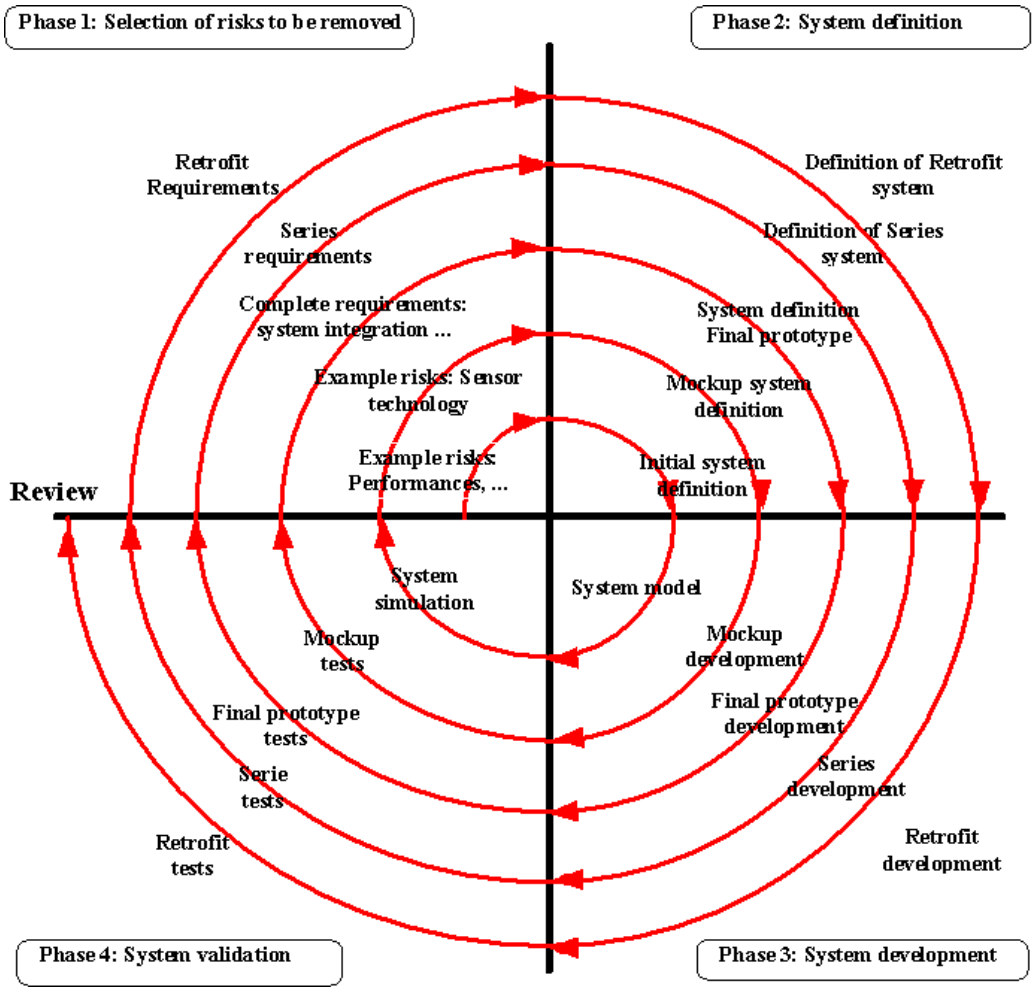


Şekil 2.4. Evrimsel geliştirme süreci

2.2.6 Spiral Model

Spiral modelde yazılım geliştirme süreci geriye dönüşü olan ardışık faaliyetler yerine spiral olarak genişleyen bir yapıda ifade edilir. Spiral model kullanıldığında her bir sarmalın sonunda yazılımın yeni bir sürümü ortaya çıkar. İlk halkanın sonunda artırımda ortaya çıkarılan yazılım uygulama ortamında kullanılanın gerçek bir prototipidir. Geliştirimi sırasında tüm süreç uygulanmıştır. Sonraki yinelemelerde üstüne yeni işlevler yine sürecin tüm adımları kullanarak eklenir. Kalite, verimlilik ve kapasite açısından geliştirilen yazılımın

uygun özellikleri taşıması her sarmal döngüde (her yeni sürümde) sağlanarak geliştirme süreci sürdürülür. Spiral modelin temsili Şekil 2.5.'tedir (espadon).



Şekil 2.5. Sarmal geliştirme süreci

Sarmalın her bir döngüsü sırayla tekrarlanan dört etkinlikten oluşur:

i. Planlama; yapılacakların eldeki kaynaklar ve zaman çizelgesi gibi konular göz önünde tutularak planlanması

ii. Risk çözümleme; teknik ve yönetsel risklerin çözümlenmesi

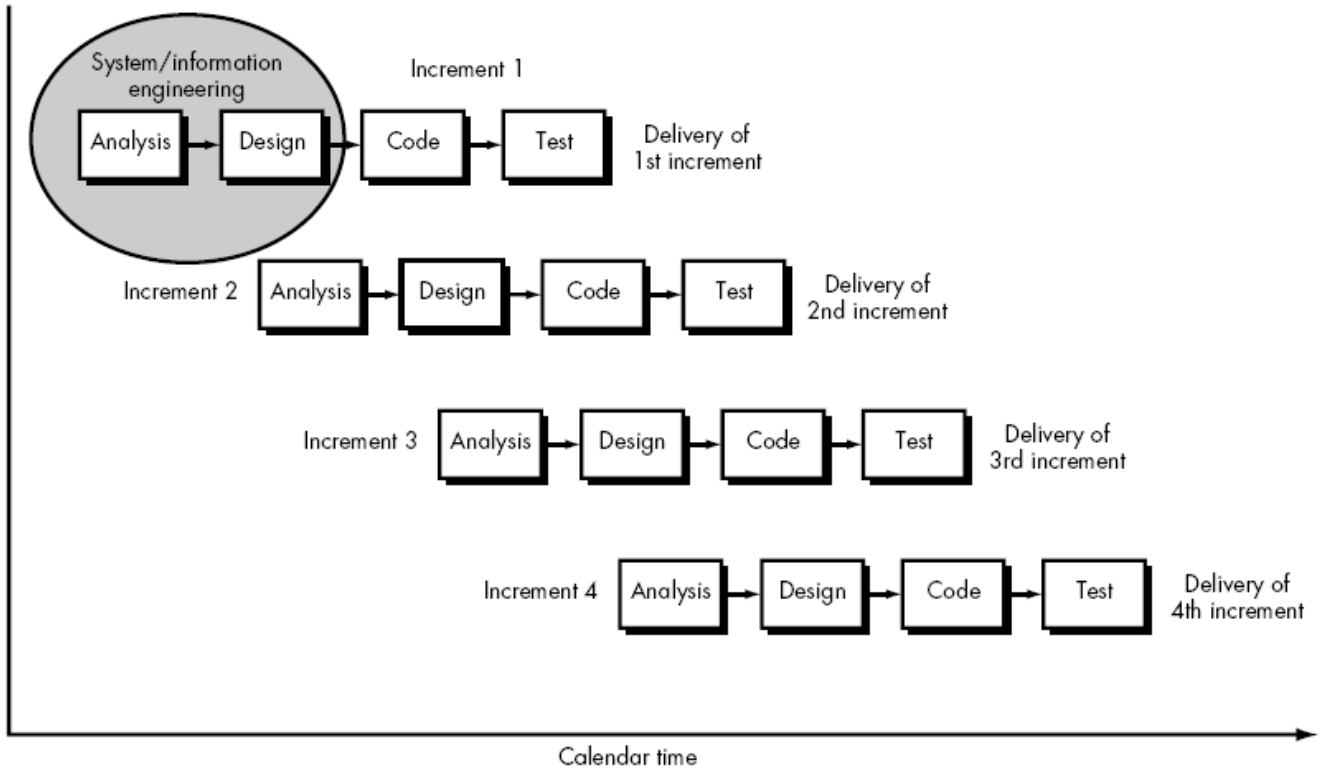
iii. Geliştirme; yazılımın gerçekleştirimi, çözümleme, tasarım, kodlama

iv. Değerlendirme; kullanıcı geribildiriminin alınması, yapılanın geçerliliğinin ve gerektiği gibi çalıştığının ortaya konması

Spiral model çok karmaşık büyük projelerde uygulanabilecek gerçekçi bir model olarak görülmektedir. Çünkü süreç ilerledikçe her bir evrimleşmede geliştiriciler ve müşteri birbirini daha iyi anlamaya başlar ve riskleri paylaşmayı öğrenir. Böylece hem müşteri hem de geliştirenler tarafından risklerin önceden öngörülmesi ile risklerin önlenmesi sağlanabilir.

2.2.7 Artımlı Model

Şelale modeline yinelemeli bir özellik katılarak artımlı model oluşturulmuştur. Bu model belirli bir takvime bağlı olarak yazılımı sürümler halinde geliştirip teslim etmeye dayanır. Her bir yeni sürüm, öncekinin üstüne bazı ek işlevlerin eklenmesini öngörür. Bu model gereksinimlerin tamamının belli olduğu durumlarda etkin bir şekilde kullanılabilir. Bu model evrimsel geliştirmeden farkı, çıkan her sürümün son ürünün sahip olduğu tüm işlevleri içermesidir. Artımlı geliştirme modeli Şekil 2.6.'da gösterilmiştir (1000sourcecodes, 2011).



Şekil 2.6. Artımlı geliştirme süreci

Bu model ile yazılım geliştirmede az sayıda çalışanla işin yapılmasını sağlama gibi bir üstünlüğü vardır. Ayrıca çalışanlar her yeni sürüme geçildiğinde uygulama alanına ilişkin daha çok deneyim kazanmış olurlar.

2.3 Yazılım Gereksinim Belirtimleri

Yazılım gereksinimi geliştirilen yazılımdan ne beklenildiğinin ifade edilmesidir. Amacına uygun hedeflenen yazılımların geliştirilmesi için yazılımdan beklenenlerin yani yazılım gereksinimlerinin çok iyi tespit edilmesi gereklidir. Bu amaçla yazılım geliştirmenin ilk adımında yazılım gereksinim analizi yapılır.

Bu süreç temel olarak şu adımları kapsar:

i. Olurluluk çalışması ve raporlaması

ii. Gereksinimlerin ortaya çıkarılması ve analizi

iii. Gereksinimlerin tanımlanması

iv. Gereksinimlerin doğrulanması

Bu adımların ilkinde öncelikle problemin anlaşılması, problem için geliştirlecek çözümün zaman ve para harcamaya değer olup olmadığına karar verilmesi için bir olurluluk analizi yapılır. Böylece geliştirilmesi düşünülen yazılımın çalışma ortamında problemin ilk bulgularına ulaşılır ve raporlanır. Daha sonra bu ilk bulgulardan hareketle yazılıma neden ihtiyaç duyulduğunun analizi yapılarak gereksinimler ortaya çıkartılmaya çalışılır.

Yazılım gereksinimlerinin ortaya çıkartılması başarılı yazılımlar geliştirilmesi açısından çok önemlidir. Çünkü doğru anlaşılmış bir ihtiyaç doğru yazılımın geliştirilmesi için atılan en önemli adımdır.

2.4 Yazılım Tasarımı ve Geliştirilmesi

Tasarım, herhangi bir mühendislik alanında geliştirilecek olan ürünün ilk modelinin ve gösteriminin ortaya çıkarılmasıdır. Bu açıdan tasarımların gerçekleştirilmesinde deneyim ve bilgi birikimi önemli yer tutar. Yazılım geliştirme sürecinde tasarım ve geliştirme tespit edilen yazılım isterlerinin çalışılabilir bir yapıya dönüştürülmesi evresidir. Yazılım tasarım sürecinde şu tasarımlar gerçekleştirilir:

i. Mimari tasarım

ii. Arayüz tasarımı

iii. Bileşen tasarımı

iv. Veri yapısı tasarımı

v. Algoritma tasarımı

Bu tasarımların ortaya çıkmasından sonra hedeflenen yazılımın kodlanmasına başlanır. Tasarım ile geliştirme faaliyetleri birbirine çok yakındır ve seçilen yazılım geliştirme süreç modeline göre iç içe gerçekleştirilebilen eylemlerdir.

Geliştirme tasarım aşamasında ortaya konulan yapıların çalıştırılabilir bilgisayar programlarına dönüştürülmesi eylemidir.

2.5 Yazılım Doğrulama ve Geçerlemesi

Yazılım doğrulama ve geçerleme geliştirilen yazılımın doğru çalıştığının, kendisinden beklenen davranışları yerine getirdiğinin ve müşterisinin beklentilerini karşıladığını göstermeyi amaçlar. Bu amaçla kullanılan iki araç vardır: Bunlar birisi yazılım gözden geçirmesi ve diğeri yazılım testleridir.

Yönetim Gözden Geçirmeleri: Proje gelişim sürecinin izlenmesi, plan ve proje takviminin durumunun saptanması, gereksinimlerin sistem bileşenleriyle ilişkilendirilmelerinin onaylanması, proje hedeflerine ulaşılabilmesi için uygulanan yönetimsel yaklaşımların verimliliğinin değerlendirilmesi amacıyla yönetim tarafından yapılan değerlendirmelerdir (softsmith, 2014).

Teknik Gözden Geçirme: Kalifiye bir ekip tarafından, bir iş ürünün hedeflenen amaca uygunluğunun saptanması, gözden geçirilen iş türünün kendi belirtimlerinden ve standartlardan sapmalarının tanımlanması amacıyla yapılan sistematik değerlendirmelerdir (softsmith, 2014).

İnceleme (Inspection): Bir iş ürünündeki hataları ve standartlardan sapmaları içeren anormalliklerin belirlenip tanımlanması için inceleme teknikleri konusunda eğitimli veya tecrübeli tarafsız kişilerin rehberliğinde gerçekleştirilen görsel sorgulamadır.

Denetleme (Audit): Bir iş ürününün veya bir sürecin, belirtimler, standartlar, sözleşme veya diğer kriterler bakımından uyumunun değerlendirilmesi için yapılan sistematik bağımsız değerlendirmedir.

Üzerinden Geçişler (Walkthrough): Bir yazılımın, geliştirici tarafından diğer geliştirme ekip üyelerine anlatılarak, yazılım ürününün iyileştirilmesine yönelik görüşlerin alınması ve projenin kodlama standardına uyulmadığı noktaların veya olası hataların tespit edilmesidir.

3. YAZILIM BAŞARISIZLIKLARI

Yazılım dünyasının gelişimine bakıldığında, yıllar geçtikçe geliştirilen yazılımların büyüklüğünün ve karmaşıklığının arttığı görülmektedir. Bu büyüme ve karmaşıklıkta, yazılımlarda hataların çok olması, güvenilirlik veya güvenliğin tam sağlanamaması gibi problemleri beraberinde getirmiştir. Aslında yazılım dünyasında ortaya çıkan problemler, ilk yazılımların geliştirilmesiyle başlamıştır. Bunun başlıca sebeplerinden biri o günlerde yeni gelişmekte olan yazılım dünyasının elinde daha önceki projelerden elde edilen tecrübe ve öğrenilen derslerden oluşturulan yeteri kadar iyi pratiklerinin olmayışı ve yazılım geliştirmeyi mühendislik disiplinleri açısından ele almayışlarıdır.

Bugün dünyadaki yazılım projelerine baktığımızda az miktarda projenin tamamen başarılı şekilde sonuçlandığı, çoğunun ise bitirilememiş veya gereksinimleri karşılanamamış şekilde bitirilen projeler olduğu görülmektedir. Bu rakamlar farklı bir şekilde ifade edilirse yazılım sektöründe yapılan gemilerin %15'inin daha suya indirilmeden battığı sonucuna ulaşılır. Ayrıca bu şekilde başarısız projeler nedeniyle yılda milyonlarca dolar zarar edilmektedir. Bu rakam yazılım hatalarının ne kadar önemli bir kayba neden olduğunun açık ve çarpıcı bir göstergesidir.

Başarısızlıklar ortaya çıktıkça bu başarısızlıklar üzerinde yapılan çalışmalar artmış ve bu çalışmalar, iyi tanımlanmış ve gelişmeye açık, disiplinli yazılım geliştirme yaklaşımının geliştirilmesine öncülük etmiştir. Başarısızlıkların incelenmesi amacıyla yapılan çalışmada öncelikle geliştirilen; fakat başarısız olan projelerden kayıtlar tutulmaya ve başarısızlık noktaları ile bu başarısızlıkların nedenleri tespit edilmeye çalışılmıştır. Yazılımların başarısızlıkla sonuçlanmasına neden olan hataların iletişim eksikliği, programlama hataları, ihtiyaç değişikliği, zaman baskısı, dökümantasyon eksikliği, geliştirme araçları eksikliği gibi etkenlerden kaynaklandığı belirlenmiştir. Bunun yanında başarıya ulaşmış olan projeler de incelenerek en iyi pratikler tespit edilmiştir. Tüm bunların sonucunda yazılım yaşam döngüsü ve bu döngü içerisindeki faaliyetler tanımlanmıştır.

3.1 Yazılım Felaketleri

Başarısızlıktan doğru derslerin çıkartıldığı durumlarda, başarısızlıklar daima başarıların anahtarı olmuştur. Başarısızlıklardan gerekli derslerin çıkartılabilmesi için dünya çapında başarısız yazılım projelerinden bazıları örnek olarak verilmiştir.

Denver Havaalanı Otomatik Bagaj Sistemi

ABD'nin Denver kentinde inşa edilen dünyanın ikinci büyük uluslararası havaalanının uçuşlardaki beklemeleri azaltmak, daha az çalışan maliyetiyle daha hızlı ve doğru bagaj hizmeti sunmak için geliştirilen otomatik bagaj sistemi 186 milyon dolarlık bir yazılımla yönetilerek 31 Ekim 1993'de açılması planlanıyordu. Ancak bagaj sisteminde ortaya çıkan yazılım hataları nedeniyle sistemin hizmete alınması gecikmeli olarak 28 Şubat 1995 tarihinde gerçekleşti. Bu gecikmenin günlük maliyetinin 1 milyon dolara yakın olduğu ve gecikme nedeniyle oluşan toplam zararın 340 milyon doları bulduğu hesaplanıyor. Sonuçta 70 milyon dolarlık yedek bir proje devreye sokuldu. O zamandan beri çeşitli sorunlarla çalıştırılan bu yazılımın da 2005 yılında artık iş göremeyeceği belirlenerek yenilenme kararı alındı (devtopics).

Arienne 5 Füzesi

Avrupa Uzay Ajansı ve Havacılık Dairesi 1996 yılında uydu taşıma amaçlı 7 milyar Euro'luk Arienne 5 isimli bir roket geliştirdi. 4 Temmuz 1996 günü fırlatıldıktan 37 saniye sonra roket kontrol yazılımındaki hatadan dolayı havada patladı. Arienne 5 füzesinde kullanılan bazı yazılımlar Arienne 4 füzesindeki bazı kodların yeniden kullanılmasıyla geliştirilmişti. Modül bazında testler gerçekleştirilmesine rağmen, her zaman olduğu gibi genel sistem testleri zaman sıkışıklığından dolayı tam olarak gerçekleştirilemedi. Arienne 5 füzesinin uçuş profili Arienne 4'ten farklı olduğundan kalkıştan kısa bir süre sonra 16 bit / 64 bit çevrimi sırasında bir değişkendeki taşma sonucu, kontrol sistemi devre dışı kalarak 500 milyon dolarlık kayba mal olan bu hata tarihteki en pahalı yazılım hataları arasında yerini almıştır (devtopics).

Therac-25 Tedavi Cihazı

Therac 25 radyoterapide hastalara belirli dozlarda radrasyon uygulayarak tümörlerin yok edilmesinde kullanılan bir cihazdır. Bu cihaz, 1985-1987 yılları arasında farklı Amerikan hastahanelerinde toplam altı kişinin ölümüne neden olmuştur. Cihaz düşük güç modu ve

yüksek güç modu olmak üzere iki temel modda çalışabilmekteydi. Cihazın temel seviyedeki elektron ışını hastaya direk olarak uygulanmakta; yüksek güç modunda ise otomatik olarak hastayla cihaz arasına metal bir plaka konulmaktaydı.Bu cihazın kullanımı sırasında operatörün cihaza önce yanlışlıkla yüksek güç moduna geç komutu vermesi ve ardından bunu iptal edip düşük güç moduna alması sonucu cihaz hem işlemin iptal edildiği yönde bir hata mesajı verir hem de iptal etmeyerek farkında olunmadan yüksek güçlü radrasyon hastalara metal koruma kullanılmaksızın uygulanmıştır. Hata mesajından dolayı operatörlerin işlemi bir kaç kere tekrar etmesiyle de hasta çok büyük oranda radrasyona maruz kalmıştır. Bu sebeple hastalara normalden 100 kat fazla doz verilmiştir. Bazı durumlarda yazılımsal bir hatadan dolayı 255. denemeden sonra tampon bellek taşması yaşanıyor ve ölümcül dozda ışının verilmesine sıfırdan başlanıyordu. Kontrol yazılımındaki bu hatadan dolayı Therac-25'in kontrol yazılımı ölümle sonuçlanan yazılım hataları arasında yerini almıştır (devtopics).

Sleipner A Sondaj Platformu

Slipner A adlı petrol platformunun temel yapısı, güverte inşaatı öncesi gerçekleştirilen kontrollü yük testi sırasında platformun dört beton sütunu kırılarak 200 metre derinliğe gömülmüştür. Yapılan araştırmada, olayın sebebinin simülasyon hesaplama programı Nastran'daki bir yazılım hatasının olduğu ortaya çıkartılmıştır. Program bu hatadan dolayı yerçekimini olması gerekenden %47 az hesapladığı için inşa sırasında 700 milyon dolarlık sondaj adasının beton duvarları gerekenden ince yapıldığı anlaşılmıştır.

Yazılım hatalarından kaynaklanan felaketlerin sayısını ve çeşidini arttırmak mümkündür. Konunun önemini anlatmak bakımından bu örnekler yeterlidir. Hayatın her alanına giren yazılımlar başarısız olmaları durumunda direk olarak can, mal ve para kaybına neden olmaktadırlar. Bu başarısızlıkların sebebi incelendiğinde öncelikle karşılaşılan problem, yazılım geliştirme sürecinde gerçekleştirilen test eylemleri ile yazılım içerisinde olabilecek olan hatalar daha erken safhada bulunabilir ve düzeltilebilir. Bu sayede daha yüksek kalitede, daha az hatalı yazılımlar üretilerek yazılım kaynaklı problemlerin ortaya çıkması sağlanabilir (devtopics).

3.2 Yazılım Testi Nedir?

Yazılım testleri, yazılım projelerinde daha az hatalı ürünler geliştirilmesi amacıyla kullanılan en önemli yazılım geliştirme süreç adımlarından biridir. Bu nedenle öncelikle

yazılım testinin ne olduğunun ve neyi amaçladığının öğrenilmesi gereklidir. Literatürde farklı yazılım test tanımlarına rastlanır. Bazıları şunlardır:

i. Yazılım testi, bir programın davranışını dinamik yöntemlerle, sonsuz bir küme içinden belirli sayıda seçilen test durumlarını kullanarak beklenen davranışa uymadığı durumları bulma işlemidir.

ii. Bir sistemin veya bileşenin belirli koşullar altında çalıştırılması, sonuçların gözlenmesi veya kaydedilmesi ve belirli özelliklerinin değerlendirilmesi sürecidir.

iii. Test, hata bulma amaçlı planlı bir şekilde gerçekleştirilen eylemler dizisi, bir doğrulama yöntemidir (tutorialspoint).

iv. Bir yazılım öğesinin mevcut ve olması gereken koşullar arasındaki farkın bulunarak analiz edilmesi ve yazılım öğesinin özelliklerini değerlendirilmesi sürecidir.

v. Test bir yazılım ürününün zayıf yönlerini veya makul hatalarını keşfetmek için gerçekleştirilen bir süreçtir.

3.3 Yazılım Testinin Amacı Nedir?

Yazılım testi, geliştirilen bir yazılımda hata olmadığını göstermek için değil, tam tersine yazılımın içerisinde hataların varlığını göstermek ve bu hataları bulmak amacıyla gerçekleştirilir. Bu nedenle ne kadar çok hata tespit edilmiş ve bunlar düzeltilerek yazılım daha olgun hale getirilmiş ise test eylemleri o derece başarılı olmuş demektir(softwaretestinghelp).

Yazılım testleri şu amaçlara ulaşmak için gerçekleştirilir:

i. Yazılım içerisindeki hataların varlığını göstermek ve tekrarlanan hataları önlemek.

ii. Yazılım içerisindeki hataları bularak nihai üründe meydana gelecek riskleri azaltmak.

iii. Yazılıma doğruluk, tamlık, güvenilirlik, hızlı ve verimli çalışabilirlik, taşınabilirlik, sürdürülebilirlik, kurulabilirlik, kurtarılabilirlik ve kullanılabilirlik gibi kalite kriterleri açısından bilgi vermek ve güven kazandırmak.

iv. Kullanıcı, sistem ve yazılım belirtimlerinde sapmaları belirlemek.

v. Test edilen yazılımın kalitesini arttırmak.

vi. İster belirleme, tasarım ve kodlama süreci boyunca meydana gelmiş ve gizli kalmış hataları ortaya çıkartmak.

vii. Müşteriye hatalardan arındırılmış ve müşteri isterlerini karşılar bir yazılım teslim etmek.

Testler yazılım içerisindeki hataların varlığına odaklanır. Ancak testler hatanın hangi kod parçalarından kaynaklandığını saptamayı amaçlamazlar. Hatanın kodun neresinden kaynaklandığının belirlenmesi işlemi hata ayıklama ile ilişkilidir ve izlenebilirlik ve hata ayıklama ile gerçekleştirilir (tutorialspoint).

4. TEST SÜRECİ VE YÖNETİMİ

Geliştirilen yazılımın en az derecede hata içermesi ve kendinden beklenenleri en üst seviyede karşılaması için yazılım test eylemleri, yazılım geliştirme sürecinde en erken safhada başlamalıdır. Testçiler, erken safhalarda başlayan test eylemleri ile muhtemel hataları yazılım geliştirme sürecinin en erken safhalarından itibaren bulmayı ve bulunan hataların düzeltilmesini amaçlarlar (softwaretestinghelp).

Yazılım dünyasındaki en iyi pratikler incelendiğinde görülecektir ki, testte belirlenen hataların sebepleri, yüksek oranda gereksinim analizi ile tasarım safhasına dayanmaktadırlar. Bunun temel sebepleri belirtimlerin açık, anlaşılır, tutarlı yazılmaması, yeterince ayrıntılandırılmaması ve yazılan belirtimlerin tasarıma tam olarak aktarılamaması ya da yazılan gereksinimden fazla tasarım modüllerinin oluşturulmasıdır. Bu gibi nedenlerden kaynaklanan hataların kodlama ya da yazılım testleri sırasında belirlenmesi düzeltici faaliyetleri gerektirir. Düzeltici faaliyetler de ekstra maliyet, zaman ve iş gücü demektir. Hatta hatanın ciddiyetine göre tasarıma ilişkin düzeltmelerin yapılması, dolayısıyla da yazılımın yeniden kodlanması anlamına gelir. Bu nedenle test eylemleri, gereksinim analizi safhasında başlatılır ve tasarım safhası ile devam ettirilirse bu tür hatalar daha erken safhalarda tespit edilebilir.

Yazılım yaşam döngüsü içerisinde ilgili safhalarda gerçekleştirilen test eylemleri:

Gereksinim Analizleri

i. Gereksinimlerin doğruluğuna karar vermek.

ii. Gereksinimlerin test edilebilirliğine karar vermek.

iii. Gereksinimlerin doğrulama metodolojisine karar vermek.

iv. Test stratejisini belirleyip test planlarını hazırlamak.

Tasarm

i. Kabul, sistem ve tümleştirme test prosedürlerini ve test durumlarını üretmek.

ii. Birim test stratejisini belirlemek.

iii. Testler için gerekli test verilerini üretmek ve doğrulamak.

iv. Hata bildirim yapısını tanımlamak ve işlerliğini denemek.

v. Testler için gerekli test ortamını tanımlamak.

Kodlama

i. Birim testlerin gerçekleştirilmesini izlemek ve sonuçları denetlemek.

ii. Kod gözden geçirmelerini izlemek ve sonuçları denetlemek.

iii. bütünleştirmek ve sistem testleri için test ortamını hazırlamak.

Test

i. Tümleştirme, sistem ve kabul testlerini gerçekleştirmek.

ii. Bulunan hataları bildirmek ve düzeltildiğini denetlemek.

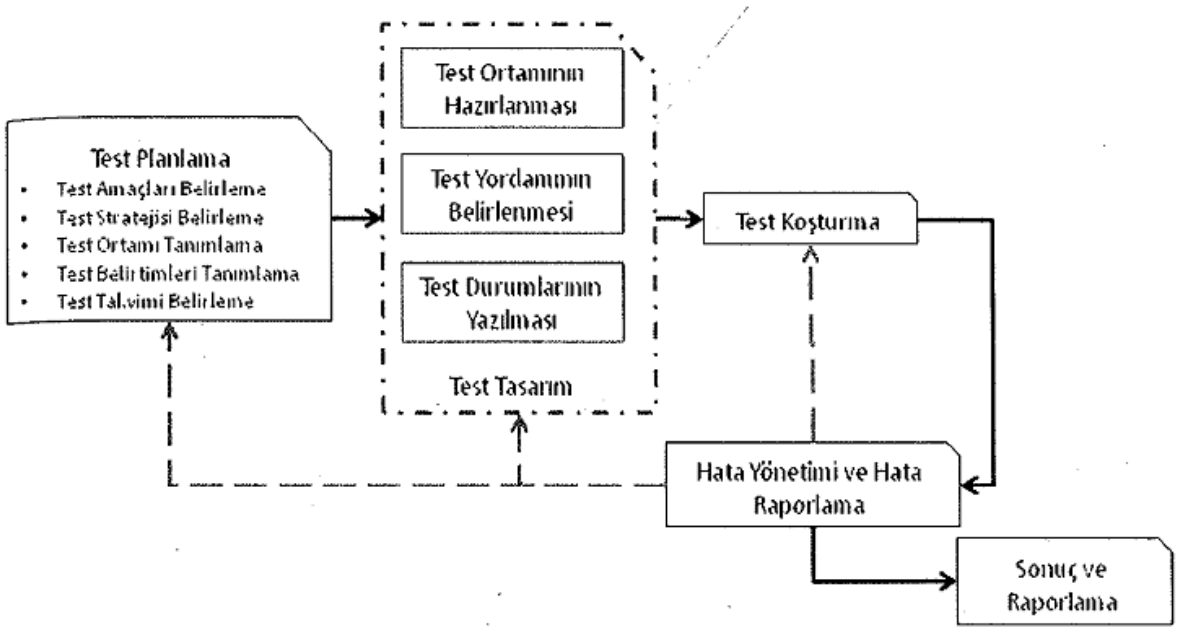
iii. Yineleme testlerini yapmak.

Bakım

i. Yineleme testlerini yapmak.

4.1 Yazılım Test Süreci

Yazılım test süreci önce planlanan, sonra icra edilip sonuçları kayıt altına alınarak belgelendirilen bir dizi eylemden oluşur. Bu süreç, geliştirilen yazılımdaki hataların varlığına odaklanır. Bir test süreci planlama, tasarım, koşturma, hata raporlama, sonlandırma ve belgelendirme adımlarından oluşur. Genel test süreci Şekil 4.1'de gösterilmiştir.



Şekil 4.1 Yazılım test süreci

4.1.1 Test Planlama

Yazılım projeleri kapsamında icra edilecek testler projenin başında planlanmalı ve belgelendirilmelidir. Bir test planı testin kapsamını, testin stratejisini, test ortamını, hangi yazılım parçalarının test edilip edilmeyeceğini, proje kapsamında amaçlanan test eylemlerini, kaynakları ve takvimi içeren bir dökümandır. Yazılım projeleri kapsamında geliştirilen yazılım doğasına uygun birim test planı, yazılım test planı, sistem test planı, kabul test planı gibi birden fazla test planı geliştirilebilir.

Test planlarının geliştirilmesindeki amaç proje kapsamındaki test stratejisinin tanımlanması, kaynak paylaşımının planlanması, sorumlulukların, risklerin ve önceliklerin açıklığa kavuşturulmasıdır.

Yazılım projelerinde test planlamasında iki farklı yaklaşım takip edilebilir:

Birinci yaklaşım: Bu yaklaşımda test planlama stratejisi her bir seviye test için ayrı test planı geliştirilmesine dayanır. Projenin genel test stratejisi Test ana planı adı verilen gene bir plan içerisinde belirtilebilir. Bu yaklaşıma göre kabul test planı, sistem test planı, tümleştirme test planı, birim test planı ayrı ayrı oluşturulur.

İkinci yaklaşım: İcra edilecek tüm testler için tek bir test planı geliştirilir. Genel olarak Kabul Test Planı veya Sistem Test planı diye adlandırılabilir. Geliştirilen plan birim, entegrasyon, sistem ve kabul testlerinin ve diğer testlerin planlamalarını kapsar.

Test planında olması gereken özellikler şöyledir (softwaretestinghelp):

Test amaçları tanımlanmalıdır: Test planlamasının ilk adımı testlerle neyin amaçlandığının belirlenmesidir. Test amaçlarının belirlenmesi aşamasında testten beklenenler, test için kritik başarı faktörleri, test faaliyetlerindeki karşılaşılacak kısıtlar, testin kapsamı, test sonrası sonuçların analiz ve değerlendirilmesinin nasıl yapılacağı gibi konular tanımlanır.

Test stratejisi (yaklaşımı) belirlenmelidir: Geliştirilmesi hedeflenen yazılımın doğasına uygun test yaklaşımı belirlenmelidir. Bundan kasıt hangi testlerin gerçekleştirileceği ve her bir testin nasıl icra edileceğidir. Kullanılacak olan test tekniklerinin, test başlama kriterlerinin, test sonlandırma kriterlerinin ne olduğu, test ile geliştirme eylemleri arasındaki koordinasyonun nasıl olacağı, test yönetim yaklaşımının (hata raporlama ve izleme yapısının, test süreci ve durumunun izlenmesinin, durum raporlamanın vb.) nasıl olacağı bu kapsamda düşünülmeli ve yöntemleri belirlenmelidir.

Test ortamı tanımlanmalıdır: Test ile ilgili gerekli donanım, yazılım, ağ alt yapısı, gerekli test araçları ve diğer raporlama ve izleme yazılımları belirlenmelidir. Eğer testler kapsamında kullanılacak olan bir test yazılımı geliştirilecek ise bu yazılım da test ortamı içerisinde tanımlanmalıdır.

Test belirtimlerinin nasıl geliştirileceği belirlenmelidir: Testlerin doğru planlanması ve test belirtimlerinin eksiksiz yazılabilmesi için test ekibi geliştirilecek olan yazılım hakkında bilgilendirilmelidir. Ekip arasında iş paylaşımı yapılmalı ve testlerle ilgili gerekli standartlar tespit edilmelidir.

Test takvimi belirlenmelidir: Test için ayrılan kaynaklar ve geliştirme takvimi göz önüne alınarak bir test takvimi belirlenmeli ve bu takvim proje teslim tarihleri ile senkronize edilmelidir.

Test planı gözden geçirilmeli ve onaylanmalıdır: Planın tamamlanmasından sonra test planı gözden geçirilmeli ve onaylanmalıdır.

Yazılım projeleri kapsamında geliştirilen test planları yaşayan dökümanlardır. Uygulamalar değiştikçe test planlarında güncellemeler gerekebilir. Bu nedenle test planlarının

hazırlanması yazılım yaşam döngüsünün ilk evreleriyle başlar ve test sınamalarının başlamasından önce son haline getirilir. Test planı bazen bir üründür, bazen proje kapsamında gerekli işlevlerin yerine getirilmesi için bir araçtır.

4.1.2 Test Tasarım

Test planlama süreci tamamlandıktan sonra test tasarım süreci başlar. Bu süreçte şu görevler icra edilir:

i. Test ortamının hazırlanması

ii. Entegrasyon, sistem ve kabul testleri için test durumlarının yazılması

iii. Test yordamlarının hazırlanması

4.1.2.1 Test Ortamının Hazırlanması

Testler, test mühendisleri tarafından tanımlanan yazılım ve donanımdan oluşan bir ortamda gerçekleştirilir. Test ortamı hazırlanırken testlerde kullanılacak olan yardımcı test yazılımları da geliştirilir veya hazır alınır. Yazılım testlerinde kullanılan bazı yardımcı yazılımlar:

Test Verisi Üreteçleri: Test durumlarının koşturulması için gerekli olan test girdi verilerini üreten yazılımlardır.

Hata Ayıklayıcılar (Debugger): Testler sırasında karşılaşılan hataların kaynak kod üzerinde bulunmasını sağlayan yazılımlardır.

Emülatör ve Simülatörler: Yazılımların ihtiyaç duyduğu gerçek donanımları taklit eden ve testler için gerekli olan donanım verilerini sağlayan yazılımlardır.

Koçan (Stub) ve Sürücüler: Testler için gerekli olan sistemin işlevsel olmayan bileşenlerinin yerini tutan ufak yazılım parçacıklarıdır.

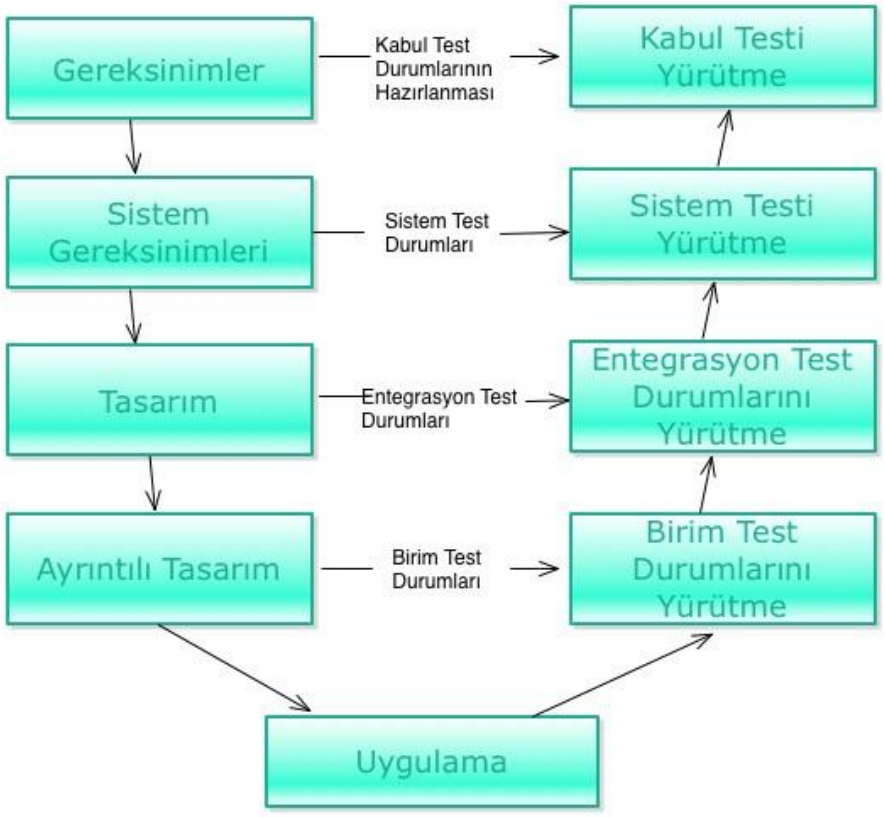
Birim veya birim birim tümleştirme testlerinde yardımcı yazılımlar genellikle koçanlar ve sürücülerdir. Ancak birim testlerin gerçekleştirilmesinde genellikle hazır test yazılımları kullanılır. Hazır test yazılımlarında birim testler için üretilen test durumları saklanabilir ve yineleme testlerinde tekrar bu test durumları kullanılabilir. Bazı test yazılımlarının özelliği olarak oluşturulması gereken test durumlarını otomatik olarak testi yapana sunmaktadır. Bu durum hazır test yazılımlarının tercih edilmesinin sebeplerinden biridir.

Yazılım tümleştirme, sistem veya kabul testleri için yardımcı yazılımları test verisi üreteci emülatör ve simülatörlerdir. Bu yazılımlar da genellikle hazır olarak alınıp kullanılır.

Testlerin başarılı bir şekilde gerçekleştirilmesi ve gereken durumlarda daha sonra tekrarlanması için, test edilecek yazılım ortamına verilirken, yazılımın hangi test ortamında test edileceği, kullanılan donanımların özellikleri, yazılımların sürümleri vb. mutlaka kayıt altına alınmalıdır. Yineleme testleri sırasında ya da yazılımdaki problemin tekrar görülmesi için test ortamının yeniden oluşturulması gerektiğinde bu kayıtlar işi kolaylaştıracaktır.

4.1.2.2 Test Durumlarının Yazılması

Bir yazılım projesinde gereksinim belirtimleri onaylandıktan sonra projenin tasarım safhası başlar. Sistem analistleri, sistem ve yazılım mühendisleri müşteri gereksinimleri, sistem gereksinimleri ve yazılım gereksinimlerini hazırlarken test ekibi de projenin doğasına uygun test stratejisinin belirlenmesi ve ilgili test planlarının güncellemsiyle uğraşır. Sistem ve yazılım geliştiriciler ön tasarım ve detaylı tasarımlar gerçekleştirirken test ekibi resmi halen gereksinimlerden test durumlarını oluşturmaya başlar. Gereksinim ve test durumları ilişkisi Şekil 4.2.'de gösterilmiştir.



Şekil 4.2. Gereksinim - test durumu ilişkisi

Bir gereksinim, bir test durumu ile doğrulanabildiği gibi birden fazla test durumuyla da doğrulanabilir. Bu amaçla, geliştirilen bir yazılımın kendi belirtimlerinin tümünü karşıladığını göstermek için her bir gereksinime en az bir tane test durumu yazılmalıdır. Ancak yazılan bu test durumları genellikle gereksinimin pozitif yönünü test edecek olan pozitif test durumları olur. Buna ek olarak yazılımın negatif durumlarda nasıl davranacağını gösteren test durumlarının da gerekli görülen gereksinimler için yazılması gereklidir. Geliştirilen yazılımın kendinden beklenen tüm davranışları gerçekleştirdiğinin göstergesi, yazılıma ait tüm test durumlarının testlerden geçmesidir (opensourcetesting)

Test durumu, belirli birp rogram parçasının çalıştığının veya bir gereksinimin doğrulandığının gösterilmesi için kullanılan girdiler, gerçekleştirilmesi gereken adımlar ve beklenen sonuçların belirtilmesidir. Test durumları testin en küçük parçasıdır.

Bir test durumunda olması gerekenler şöyle sıralanabilir (tutorialspoint):

1. Testin durumunun amacı ve gerçekleştirilme şartları
2. Test durumu ile ilgili test ortamının adım adım kurulması
3. Girdi verileri
4. Beklenen sonuç
5. Gerçekleşen sonuç
6. Yazılımın sürüm tanımı
7. Yazılımın çalışma ortamı
8. Test ID

Test edilen öğenin sadece bir özelliği test edilmelidir. Böylece test durumu çalıştırılıp başarısız olursa test edilen hangi özelliğin sorunlu olduğu kolayca görülür.

Test durumlarında bilien girdilere yer verilmelidir. Bu girdiler ile beklenen çıktılar test durumları içerisinde doğrulama noktası olarak verilir. Eğer beklenen girdiler ile beklenen sonuç yazılım tarafından verildiyse ilgili test durumu geçmiştir.

Test edilen öğeye ait test durumlarının tamamının veya belirlenen bir oranının testten geçmesiyle gerçekleştirilen testler başarılı sayılabilir.

Aşağıdaki örneklerde çeşitli test durumu form şablonları Şekil 4.3.'de gösterilmiştir (softwaretestinggenius, vietnamesetestingboard)

Test Name:	Describe the Name of the Test
Test Case Author:	Describe the Test Case Author Name
Tester Name:	Describe the Tester Name
Project ID / Name:	Describe the Name of the Project
Test Cycle ID:	Describe the Test Cycle ID
Date Tested:	Describe the Date on which Test Case Was Completed

Test Case ID	**Test Objective ID**	**Category**	**Condition**	**Expected Results**	**Actual Results**	**Requirement ID**
Describe the Test Case ID	Describe the ID from the Test Plan	Describe the Test Category e.g. (Edit, Numeric, Presentation etc.)	Describe the Specific Test Condition	Describe the Specific Results Expected on Executing the Condition	Record "Pass" or "Fail"	Describe the ID which Traces Back to the Specific Requirement

GENERAL INFORMATION			
Test Stage:	☐ Unit ☐ Functionality ☐ Integration ☐ System ☐ Interface ☐ Performance ☐ Regression ☐ Acceptance ☐ Pilot Specify the testing stage for this test case.		
Test Date:	mm/dd/yy	System Date, if applicable:	mm/dd/yy
Tester:	Specify the name(s) of who is testing this case scenario.	Test Case Number:	Specify a unique test number assigned to the test case.
Test Case Description:	Provide a brief description of what functionality the case will test.		
Results:	☐ Pass ☐ Fail	Incident Number, if applicable:	Specify the unique identifier assigned to the incident.
INTRODUCTION			
Requirement(s) to be tested:	Identify the requirements to be tested and include the requirement number and description from the Requirements Traceability Matrix.		
Roles and Responsibilities:	Describe each project team member and stakeholder involved in the test, and identify their associated responsibility for ensuring the test is executed appropriately.		
Set Up Procedures:	Describe the sequence of actions necessary to prepare for execution of the test.		
Stop Procedures:	Describe the sequence of actions necessary to terminate the test.		
ENVIRONMENTAL NEEDS			
Hardware:	Identify the qualities and configurations of the hardware required to execute the test case.		

Test Case ID: _____ Reviewed By: _____ Date of Review: _____ Version of Review: _____				Review Comments Incorporated By : _____ Date Incorporated: _____ New Version of Document: _____		
Sr.	Page No.	Step No.	Severity C – Critical, M – Major, S – Small	Comments	Action Taken	Close (Yes / No)
01						
02						
03						
04						
05						
06						
07						
08						
09						
10						

Şekil 4.3. Örnek test durum formları

Bu test durumlarının koşturulması ve sonuçlarının kaydedilmesi elle yapılabileceği gibi test betikleri (script) yazarak otomatik olarak da gerçekleştirilir. Test durumlarının koşturulması elle veya otomatik olsun, her şartta kayıt altına alınmalıdır.

4.1.2.1.1 Test Durumlarının Yapısı Nasıldır?

Yazılarak kayıt altına alınan test durumları üç bölümden oluşur. Her bölüm kendi içerisinde aşağıda verildiği gibi alt bölümlere ayrılır:

1. **Giriş/Genel Bakış:** Test durumu hakkında genel bilgiler içerir.

i. Tanımlayıcı: Her bir test durumuna ait olan eşsiz bir tanımlayıcıya sahiptir. Bu tanımlayıcı ile test durumu ilgili öğesi, test sonucu ve açılan hata bildirimleri ile ilişkilendirilir.

ii. Test Durumunu Yazan: Test durumunu kimin yazdığını belirtir.

iii. Sürüm: Test durumunun sürüm numarasıdır.

iv. Adı: Test durumunun ne ile ilgili olduğunu kolayca gösterecek bir ifadedir.

v. Gereksinim Tanımlayıcısı: Test durumunun hangi gereksinim için yazıldığının anlaşılması için gereksinime ait eşsiz tanımlayıcısının test durumu içerisinde verilmesi gereklidir.

vi. Amaç: Bu test durumuyla hangi işlevin test edildiğinin belirtilmesidir.

vii. Bağlılıklar: Bu test durumunun gerçekleştirilmesi için varsa kendinden önce gerçekleştirilecek test durumlarının eşsiz tanımlayıcılarının belirtilmesidir.

2. **Test Durumu Eylemleri**

i. Test Ortamı: Test durumunun koşturulabilmesi için gerekli olan yazılım, donanım ve çevresel koşulların belirtilmesidir.

ii. İlklendirme: Test durumu koşurulmasından önce varsa gerekli olan değişkenlerin başlangıç değerlerinin belirtilmesidir.

iii. Sonlandırma: Test durumları koşturulduktan sonra gerçekleştirilecek eylemler belirtilir. Örneğin eğer test durumu veritabanını siliyor ise veritabanının yeniden kayıtla doldurulması gibi.

iv. Eylemler: Testin tamamlanması için yapılacakların adım adım belirtilmesidir.

3. **Sonuçlar**

i. Beklenen Sonuç: Test durumunda belirtilen tüm adımların gerçekleştirilmesinden sonra test mühendisinin hangi sonuçla karşılaşacağının belirtilmesidir.

ii. Gerçek Sonuç: Test gerçekleştirildikten sonra ortaya çıkan sonucun belirtildiği alandır. Genellikle bu sonuçla beklenen sonuç karşılaştırılarak testin geçip geçmediği belirlenir.

4.1.2.3 Test Prosedürü Nedir?

Test prosedürü, her bir test durumunun test ortamının kurulması, koşturulması ve sonuçlarının değerlendirilmesi için ayrıntılı direktifler, açıklamalar listesi içeren ve test planı temel alınarak geliştirilen belgedir.

Her bir test durumu için ayrı test prosedürleri olabileceği gibi bir grup halinde olan test durumları için de bir test prosedürü hazırlanabilir. Bu durum, test planında tanımlanır. Örnek bir test prosedürü:

1. Hatasız derlenmiş yazılımı al.
2. Yazılımın teste hazır olduğunu ilgili kontrol listesini kullanarak doğrula.
3. Test ortamının hazır olduğunu kontrol listesini kullanarak doğrula.
4. Yazılımı çalıştır.
5. Yardımcı test yazılımlarını çalıştır.
6. Sıra ile test durumlarını koşturmaya başla.

4.1.3 Test Koşturma

Test koşturma, test süreci içerisinde birim testlerle başlayan evredir. Yazılımın kodlaması aşamasında gerçekleştirilen birim testlerin sorumluluğu kodculara aittir. Ancak testlerin denetlenmesi, birim ve modül testlerinde kullanılacak test durumlarının oluşturulma stratejisinin belirlenmesi testçilere aittir. Testçiler tarafından icra edilecek testler, tümleştirme testleriyle başlar. Tümleştirme testlerinin tamamlanmasıyla sistem gereksinimlerinin doğrulanması amacıyla sistem testleri gerçekleştirilir. Sistem testlerinde işlevsellik, performans ve sistem kullanılabilirliği gereksinimleri test edilir. Test koşturmada son adım

kullanıcı kabul testleridir. Bu testlerde sistemin kullanıcı gerekesinimlerini karşıladığı doğrulanır. Testler koşturulurken test mühendisleri tarafından genel olarak şu adımlar izlenir:

1. Yazılım ekibi tarafından test edilecek yazılım oluşturulur.
2. Testçiler gelen yazılıma uygulanacak test durumlarına karar verir.
3. Yazılımın test edilebilir olduğuna karar verilir.
4. Yazılım teste kabul edilirse testler başlar.
5. Testlerde bulunan hatalar raporlanarak yazılım ekibine bildirilir.
6. Bulunan hatalar yazılım ekibi tarafından düzeltilir ve yeni sürüm hazırlanır.
7. Testçiler yeni gelen yük ile yineleme testlerini yapar.
8. Bu adımlar, müşteriye yazılımın son kabul edilebilir yazılım verilinceye kadar devam eder.

4.1.3.1 Test Durumu Statüleri

Yazılım projelerinde test sürecindeki test eylemlerinin izlenmesi ve raporlanması amacıyla test durumlarına ve senaryolarına çeşitli statüler özellik olarak Tablo 4.1.'de gösterilmiştir.

Durum	**Açıklaması**
Testte	Testin devam ettiğini gösterir
Beklemede	Testin başladığını ancak başka bir sebepten dolayı beklediğini gösteren durum
İptal	Belli bir sebepten dolayı koşturulamayacak olan test durumunu belirtmek için kullanılır
Güncellenecek	Test durumları veya senaryolarında revizyon gerekiyorsa bunu gösterir

Başarılı	Koşturulan ve başarılı sonuçlanan test durumlarını veya senaryolarını belirtir
Başarısız	Koşturulan ve başarısızlıkla sonuçlanan test durumlarını veya senaryolarını belirtir
Başlamadı	Test koşturulması başalamayan test durumları veya senaryolarını gösterir

Tablo 4.1 Test durumu statüleri

4.1.4 Hata Yönetimi ve Hata Raporlama

Kullanıcının geliştirilen yazılımla yapmak istediklerinin yazılım tarafından yapılmaması veya eksik yapılmasına hata denir. Hatalar genellikle testler sırasında tespit edilir.

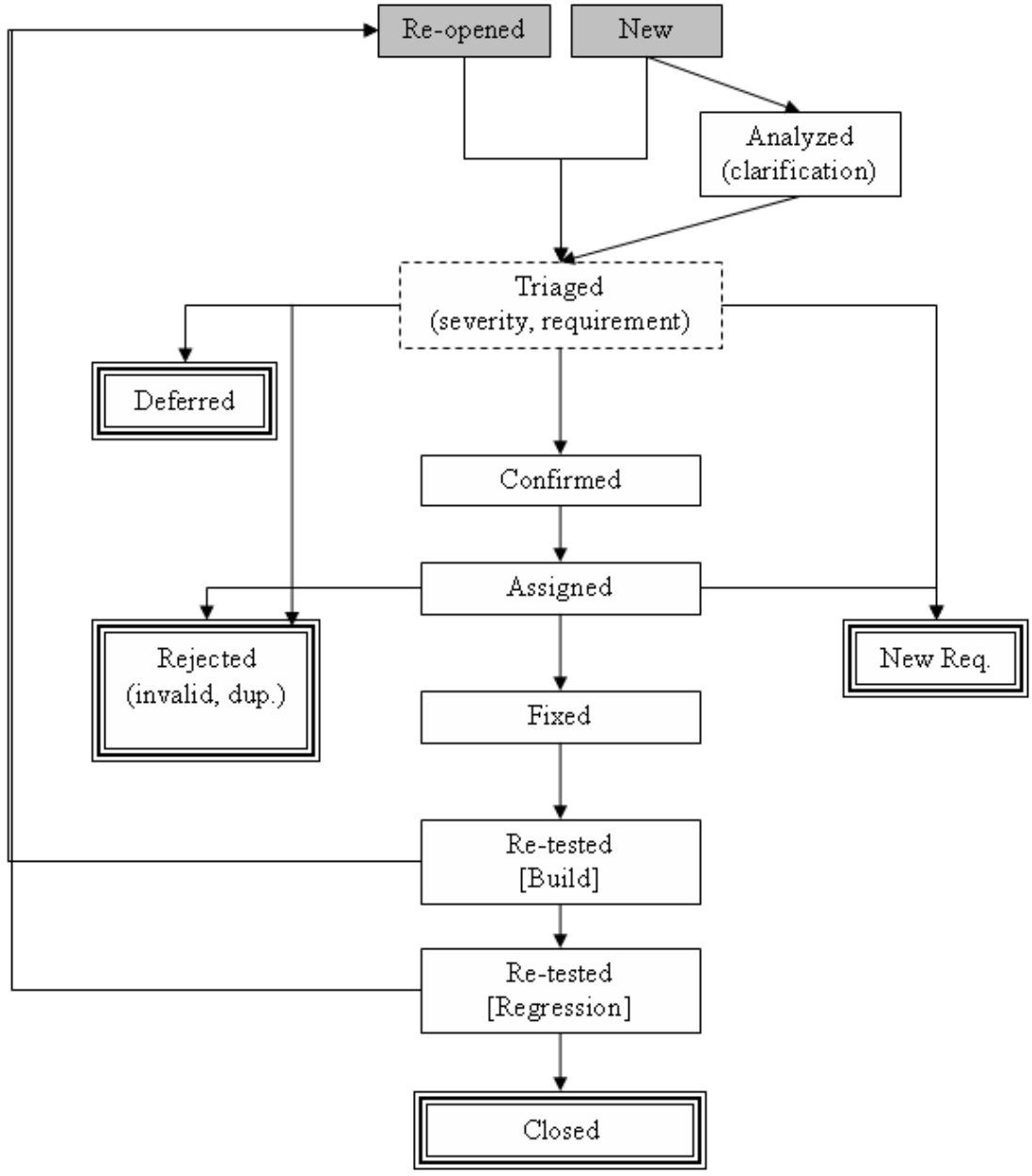
Testler sırasında belirlenen yazılım hatalarının düzeltilmesi iş gücü kaybına ve ek maliyete neden olur. Yani maliyeti yüksek sonuç ortaya çıkarır. Oysa testlerden önce gözden geçirme faaliyetleriyle bu hatalardan bazıları bulunabilir ve daha az maliyet ve zamanla düzeltilebilir. Bir yazılım hatalardan tamamen ayıklamak imkansızdır. Ancak bu hatalar testler ve gözden geçirmelerle en aza indirgenebilir.

Testler ve gözden geçirmeler sadece hataların varlığını gösterir. Hataların ortadan kaldırılması düzeltme faaliyetleriyle gerçekleştirilir. Bunun için yazılımın yaşam döngüsü vardır. Bu döngüyle daha sağlam, daha doğru ve hızlı yazılımlar üretilmesi hedeflenir. Aynı şekilde proje içerisinde hataların ve düzeltici faaliyetlerin kontrol altında tutulması için bir hata yönetimi yaşam döngüsüne ihtiyaç vardır.

4.1.4.1 Hata Yönetimi Yaşam Döngüsü

Bir yazılım projesinde hatalar, projenin doğasına uygun sistematik bir yaklaşımla ele alınmalıdır. Bunun için projede, tespit edilen hataların denetim altında tutulabileceği bir "hata yönetimi yaşam döngüsü" kurulmalıdır. Hata yönetimi yaşam döngüsü, proje kapsamına ve boyutuna göre farklı olabilir. Küçük projelerde Şekil 4.6'daki gibi basit bir hata yönetimi yaşam döngüsü kullanılabilir. Bu yaşam döngüsüne göre gözden geçirme veya test faaliyetleri devam ederken tespit edilen hata, yazılım geliştirme ekibine bildirilir. Bildirilen hata

incelenerek gerekli düzeltme faaliyetleri bu ekip tarafından gerçekleştirilerek yeni yazılım sürümü oluşturulur. Yeni yazılım sürüme üzerinde yineleme testleri yapılır. Yinleme testleri sonunda hata düzeltilmişse hata kapatılır. Küçük bir projede basit bir hata yönetimi süreci Şekil 4.6.'da gösterilmiştir.



Şekil 4.4. Basit bir hata yönetimi yaşam döngüsü

Bu yaşam döngüsü küçük çaplı yazılım projelerinde başarılı bir şekilde işleyebilir. Ancak projenin kapsamı genişlediğinde, karmaşıklığı arttığında projede çalışan ekip sayısı

fazlalaştığında hata yönetiminin başarılı şekilde gerçekleştirilmesi için daha karmaşık, farklı bir yaşam döngüsüne ihtiyaç vardır. Şekil 3.7'de bu kapsamda değerlendirilebilecek genel bir hata yönetimi yaşam döngüsü tanımlanmıştır. Bu yaşam döngüsü içerisinde Test Ekibi, Hata Değerlendirme Ekibi ve Yazılım Geliştirme Ekibi vardır. Test ekibi projede testleri yapmaktan, hataları açmaktan, açılan hatalar düzeltilmiş ise bu hataları kapatmaktan sorumludur. Hata Değerlendirme Ekibi ise proje boyunca kendisine gelen ve hata olduğu düşünülen bildirimleri değerlendirerek ilgili kişileri veya ekibi tespit edip görevlendirmekle sorumludur. Küçük çaplı projelerde bu görevi yazılım ekip lideri veya proje sistem mühendisi yapabilir. Yazılım geliştirme ekibiyse kendisine bildirilen hataları değerlendirerek çözümünü bulmaktan ve düzeltilmiş yeni yazılım sürümü oluşturmaktan sorumludur. Geliştirilen yazılım, tespit edilen hatalardan arındırılıncaya kadar veya hata sayısı test planında belirlenmiş bir orandan aşağıya düşünceye kadar bu yaşam döngüsü işletilir. Bu yaşam döngüsünün oluşturulması ve gereklerinin yerine getirilmesi yazılım test süreci içerisinde yer alan test tasarım safhasında gerçekleştirilir.

4.1.4.2 Hatalar Nasıl Raporlanır?

Hataların yazılım geliştiricilere bildirilmesi, bildirilen hataların düzeltildikten sonra hatanın kapatıldığının izlenmesi testler açısından önemlidir. En üst düzeyde hatalardan arındırılmış bir yazılımın geliştirilmesi isteniyorsa testler sırasında çıkan tüm hataların kapatıldığından emin olunmalıdır. Bulunan hatalar, hataların kaynakları, yapılan düzeltme faaliyetleri kontrol altına alınmalıdır. Bu amaç için oluşturulacak her bir hata raporu genel olarak şu maddeleri içerebilir (softwaretestinghelp):

Hata ID, Hata Tanımı, Yazan / Kodlayan, Sürüm / Yük No, Açılış Tarihi, Kapanış Tarihi, Problem Alanı, Hata / Geliştirme / İyileştirme, Test Ortamı, Hata Tipi, Hatayı Bulan, Hata Nasıl Tespit Edildi, Kime Atandı, Öncelik, Önem Derecesi, Durumu

Daha sonraki çalışmalarda bu kayıtlardan analizler yapılarak hataların kaynakları tespit edilmelidir. Tespit edilen bu bilgiler yazılım geliştiriciler ve testçiler ile paylaşılarak aynı hataları tekrarlamaları önlenmelidir.

4.1.4.3 Hata Önem Dereceleri

Test sürecinde tespit edilen hatalar raporlanırken şu hata önem dereceleri kullanılabilir:

Ölümcül: Testlerin devam etmesini engelleyecek hataları belirtir. Bu durumda testlerin devam etmesi imkansızdır.

Kritik: Testler devam edebilir. Ama bu hata derecesiyle yazılım teslim edilemez.

Büyük: Testler devam edebilir. Ürün bu hata ile teslim edilebilir. Ama yazılım kullanıldığında telafisi zor sonuçlar doğurabilir.

Orta: Testler devam edebilir. Ürün bu hata ile teslim edilebilir. Ama yazılım kullanıldığında telafisi mümkün sorunlar çıkartabilir.

Küçük: Testler devam edebilir. Ürün bu hata ile teslim edilebilir. Yazılımın işleyişine etkisi çok olmaz.

Kozmetik: Yazılım üzerindeki renk, font, büyüklük gibi görsel hatalardır. Olması durumunda ne testi durdurur ne de ürünün teslimini engeller.

Bu hata önem dereceleri hataların gruplandırılmasında ve testlerin başarılı olma kriterlerinde kullanılabilir. Örnek olarak büyük, kritik ve ölümcül hataların düzeltilmesi tamamlanana yazılım, testlerden başarılı şekilde geçti kabul edilebilir. Ancak bu karar verilirken test seviyesine göre farklılık gösterir. Bu tür kararlar kabul kriterleri olarak müşteriyle birlikte ve her seviye test için ayrı ayrı belirlenerek ilgili test planlarında yer almalıdır.

4.1.5 Test Sonuç Raporlama ve Değerlendirme

Testler yapıldıktan sonra elde edilen test verileri raporlanmalı, analiz edilmeli ve değerlendirilmelidir. Değerlendirmede test planlarında belirtilen test geçme/kalma kriterleri dikkate alınır. Test sonuçlarının raporlanması ve değerlendirilmesi için IEEE 829-1998 standardı "Test Özet Belgesi" formatı önerilebilir.

Test özet belgesi içerisinde test sonuçları şu bilgileri kapsayabilir (softsmith):

Modül/Birim: Hangi yazılım modülünün/biriminin test edildiği yazılır.

İhtiyaç Numarası: Test durumunun doğruladığı ihtiyacın numarası yazılır.

Test Durumu Numarası: Belirtilen modül üzerinde koşturulan test durumunun numarasıdır. Böylece hangi test durumunun sonucu olduğu belirtilir.

Test Sonucu Numarası: Test sonucunu diğerlerinden ayıracak bir tanımlayıcı (unique id) yazılır.

Test Tarihi: Testin yapıldığı tarihtir.

Testi Yapan: Test yapıp sonucu raporlayan kişinin adıdır.

Testi İzleyen: Müşteri tarafından testi izleyen varsa adı belirtilir.

Tekrar Sayısı: Test durumunun kaçıncı kez koşturulmuş olduğu yazılır. Böylece koşturulan her test durumunun tüm sonuçları kayıt altına alınarak hangi yazılım sürümlerinde başarısızlıkla sonuçlandığı gibi bilgiler ölçülebilir.

Hata Bildirim Numarası: Test durumunun başarısızlıkla sonuçlandığı durumlarda yazılan hata bildirim formu numarası yazılır.

Sürüm No: Testin yapıldığı yazılım sürüm numarasıdır.

Yardımcı Yazılımların Sürümü: Kullanılan yardımcı yazılım (emülatör, simülatör vb.) varsa buraya sürüm numaraları yazılır.

Donanım Sürümü: Test yapılırken donanımın sürüm numarası yazılır.

Notlar: Oluşan problemin kısa özeti, sorunun test prosedür adımı, sorunu düzeltmek için yapılan işlemler ve sonuçları, testin yenilenmesi için yedekleme ve geri dönüm noktaları gibi bilgiler verilerek testçinin test sırasında karşılaştığı, özellikle dikkat çekmek istediği konular, raporu netleştirmek için notlar buraya yazılır.

Test özet belgesinde test sonuçları analiz edilip değerlendirilirken kaç adet ihtiyacın, kaç adet test durumu ile kaç kere test edilerek doğrulandığı, ne tür hatalar çıktığı ve bu hataların hangi yazılım sürümünde düzeltilerek testlerden geçtiği raporlanır.

4.2 Test Yönetimi

Büyük ve karmaşık yazılımların gerektiği gibi test edilmesi oldukça zor ve zahmetli bir iştir. Testlerin etkin şekilde gerçekleştirilmesi farklı grupların iyi bir şekilde organize edilmesine, işlerin iyi planlanmasına, sorumlulukların ve ilişkilerin net tanımlanmasına bağlıdır.

4.2.1 Test Aktörleri

Test Yöneticisi: Test yöneticisinin temel görevi test stratejisini belirlemek ve bu stratejiye göre uygun test ekibini oluşturarak bu ekibe liderlik etmektir. Bu kapsamda ekip için gerekli olan eğitimleri tanımlar ve bu eğitimlerin alınmasını sağlar. Ekibin etkin bir şekilde çalışabilmesi için ekip içi sorumlulukları belirler. Test faaliyetlerini planlar ve bunları belgelendirir. Test ekibinin daha etkin ve verimli çalışmasını teşvik eder. Ekip içindeki elemanların performanslarını arttırıcı yönde performans değerlendirmeleri yapar.

Test yöneticisinin diğer bir görevi bir denetçi olarak gerçekleştirilen faaliyetleri gözlemlemesi, takip etmesi ve değerlendirmesidir. Bu amaçla kayıt altına alınarak gerçekleştirilen test faaliyetlerini değerlendirir. Hazırlanmış olan planlar ile sapmaları bulur ve elde edilen sonuca göre faaliyetler üzerinde yeni yönlendirmeler yapabilir.

Test yöneticisi deneyimiyle testlerde kullanılacak yazılımların seçiminde yardımcı olur. Test faaliyetlerinde ekip içerisinde kullanılabilecek yeni test yazılımlarının geliştirilmesini sağlar. Proje içerisinde test faaliyetlerinde genel kabul görmüş test standartlarının, metotlarının ve kriterlerinin uygulanmasını sağlar.

Test Mühendisi: Tüm test çalışmalarında görev alan mühendislerdir. Temel görevleri testleri planlandığı şekilde gerçekleştirmek ve yaptıkları faaliyetleri kayıt altına alarak belgelendirmektir. Test mühendisleri sorumluluklarını test planında belirtilen metodolojiye, test tekniklerine ve standartlara uygun olarak yerine getirir. İyi bir test mühendisinde şu özellikler olabilir:

i. Bireysel olarak kendi çalışmalarını organize edebilen

ii. Planlı bir şekilde hareket edebilen

iii. Test metotları, araçları ve kriterleri konusunda temel bilgi sahibi

iv. Test için verilen sistemi ve belirtim belgelerini anlayabilecek düzeyde teknik bilgiye sahip

v. Dökümanlardaki ayrıntıları yakalayabilen

vi. Çatışma ve çakışmaları ustalıkla yönetebilen

Bir test mühendisinin yapacağı görevleri:

i. Test planlarının hazırlanmasına ve test faaliyetlerinin organizasyonuna yardımcı olmak.

ii. Gereken durumlarda test yazılımlarının alınmasında görüş belirtmek ve gerekli incelemeleri yapmak.

iii. Test prosedürünü ve test verilerini geliştirmek.

iv. Test durumlarını tasarlamak ve belgelendirmek.

v. Test ortamının hazırlanmasına yardımcı olmak.

vi. Testleri yapmak.

vii. Test sonuçlarını ve tespit edilen hataları raporlamak.

viii. Test tasarımı ve test sonuç raporlarının gözden geçirilmesinde katkıda bulunmak.

ix. Gözden geçirme faaliyetleri sonucunda gereken durumlarda test tasarımındaki veya test sonuç raporlarındaki güncellemeleri yapmak.

x. Testlerin yinelenmesinde görev almaktır.

5. YAZILIM TEST DÜZEYLERİ

Yazılım projelerinde testler dört farklı aşamada gerçekleştirilir. Bu testlerin ayrıntı düzeyi arttıkça test edilen parçanın büyüklüğü azalır. Bu test aşamaları şunlardır:

1. Birim Testler
2. Yazılım Tümleştirme Testleri
3. Sistem Testleri
4. Kabul Testleri

5.1 Birim Testler

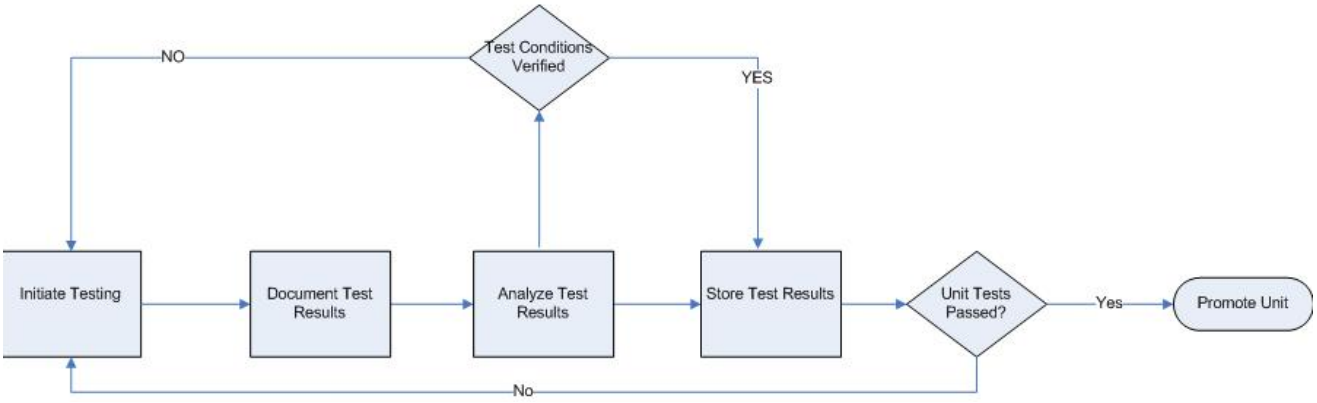
Metot, modül veya prosedür gibi bir kod parçasının kendisinden beklenen işlevselliği doğru olarak yerine getirdiğini ve hata içermediğini göstermek için gerçekleştirilen testlerdir (softwaretestinghelp). Birim testler bazen proje takviminden, bazen yeterli tecrübe olmayışından, bazen de gereken önemin verilmesinden dolayı acele ile geçiştirilirler veya ihmal edilirler. Aslında birim testler başarılı ve kaliteli yazılım ürünleri ortaya çıkartmak için kullanılabilecek olan önemli bir test aşamasıdır. Eğer bir yazılım projesinde birim testler başarılı bir şekilde gerçekleştirilirse, diğer test aşamalarında daha başarılı sonuçlar alınacaktır.

Birim testi yapılmasının en rahat yolu bir yazılım test aracı kullanılmasıdır. Eğer bir test yazılım aracı kullanılmayacaksa birim test kodları yazılım geliştirmeye paralel olarak geliştirilmelidir. Bu nedenle birim testler yazılım geliştirmenin bir parçası olarak düşünülmeli ve planlama buna uygun olarak gerçekleştirilmelidir. Gereksinimler veya kod güncellendikçe birim test kodları da güncellenmelidir.

Başarıyla sonuçlanan birim testler, yazılım tümleştirme ve sistem testlerinden önce geliştirilen yazılımın genel olarak istenen gereksinimleri karşıladığının bir göstergesidir. Ancak yazılımın hatasız olduğunu göstermez. Geliştirilen yazılımdan beklenenlerin tam olarak karşılandığını ortaya koymak için entegrasyon ve sistem testleri ile arayüzlerin testlerinin de gerçekleştirilmesi gerekir.

Birim Testler Nasıl Yapılır?

Birim testlerinin amacı, geliştirilen ve derlenebilen en küçük kod parçalarının (metot, prosedür, sınıf vb.) kendilerine ait görevleri doğru şekilde yerine getirdiğinin doğrulanmasıdır. Bu amaçla, birim testler yapılırken test edilecek birim, diğer birimlerden izole edilir. Test edilen birim için gerekli olan girdileri sağlayacak diğer birimlerin yerine o metotların koçanları (stub) kullanılır. Böylece diğer birimdeki olası bir hatanın test edilen birimi etkilemesi engellenir. Test edilecek birimin icra edeceği göreve göre test durumları oluşturulur. Bu durumlar kullanılarak test edilecek birim çalıştırılır ve elde edilen sonuçlar beklenen sonuçlarla karşılaştırılarak testlerin başarılı olup olmadığına karar verilir. Eğer test başarısız olmuşsa birimin kodu tekrar gözden geçirilir, gerekli düzeltmeler yapılır ve tekrar test edilir. Bu işlem, tüm birimler başarılı bir şekilde kendilerine ait birim testleri geçinceye kadar devam eder. Şekil 5.1.'de genel birim test süreci görülmektedir (efficientqa, 2011).



Şekil 5.1. Genel birim test süreci

Birim test süreci yazılım geliştirme metodolojilerine göre farklılık gösterebilir. Örneğin projede test güdümlü veya çevik yazılım geliştirme metodolojisi kullanılacaksa birimler yazılmadan önce test kodları yazılır.

Birim Testlere Bir Örnek:

pozitifMi (int k) isimli bir metot vardır. Bu metotta k integer değişkeni eğer negatifse metot true, k integer değişkeni pozitifse false değerini vermektedir. Bu metot aşağıdaki kodu içermektedir:

```
public bool pozitifMi (int k)
{
```

```
        if (k < 0 )
                return true;
        else
                return false;
    }
```

Bu metotun test edilmesi için k değerine önce 0'dan küçük, sonrada 0'dan büyük değerler verilerek test edildiği durumlar gösterilmektedir:

Test Durumu 1:

```
    public void testDurumu_1()
    {
        bool td1 = false;
        int m = 3;
        td1 = pozitifMi(m);
        if (td1 == true)
                System.out.println ("test durumu 1: geçti");
        else
                System.out.println ("test durumu 1: kaldı");
```

Test Durumu 2:

```
    public void testDurumu_2()
    {
        bool td2 = true;
```

```
int m = -4;

td2 = pozitifMi(m);

if (td2 == false)

        System.out.println("test durumu 2: geçti");

else

        System.out.println("test durumu 2: kaldı");
```

Birim Testlerde Kapsama Analizi: Birim testlerde önemli bir konu da kapsamadır. Kapsama bir yazılım birimi içerisindeki tüm yol ve karar noktalarının test edilmesidir. Bu durum özellikle emniyet ve görev kritik sistemler için daha büyük önem taşır. Yukarıdaki örnek incelendiğinde yazılan iki test durumu sırayla koşturulursa, ilk test durumunun koşturulmasında sonra bu metot üzerinde %50 bir kapsamaya ulaşılır. Yani bu kodun sadece yarısı çalıştırılmış olur. Eğer diğer test durumu da koşturulursa kodun geri kalan kısmı da çalıştırılmış olur. Bu iki test durumunun koşturulmasıyla bu metot %100 kapsamaya ulaşmış olur. Yani bu metodun içerisindeki tüm satırlar bu iki test durumuyla en az bir kez çalıştırılmış olur.

Birim Testlerde Sınır Değer Analizi: Birim testlerde önemli diğer bir nokta da sınır değerlerin kontrol edilmesidir. Bir metot içerisinde belirli bir değer aralığında işlem yapılıyorsa bu değer aralığının alt ve üst sınırının kontrol edilmesi gerekir. Yazılım testlerinde yazılımcıların en çok hataya düştükleri konuların başında sınır değerlerin yazılım içerisinde uygun bir şekilde kapalı veya açık aralık şeklinde kodlanmamasıdır. Yukarıdaki örnekte k = 0 durumunda ne gibi bir sonuçla karşılaşılır? Bu durumun kod içerisinde kontrol altına alınması gereklidir. Bu durum için üçüncü bir test durumu yazılmalı ve yazılımcıdan da bu durumu kodun nasıl davranacağı öğrenilerek elde edilen sonuçla karşılaştırılmalıdır.

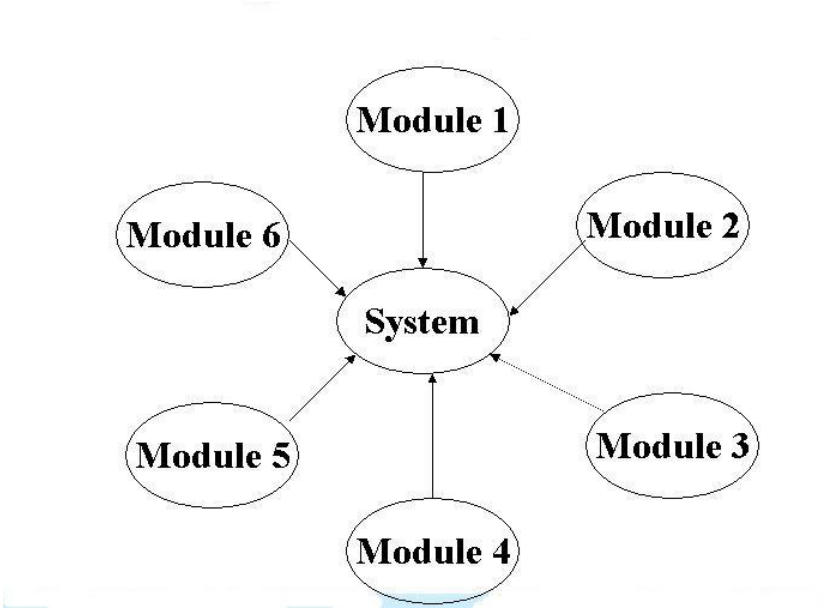
5.2 Tümleştirme Testleri

Yazılım projelerinde birim testlerin başarılı bir şekilde sonuçlandırılmasından sonra tümleştirme (entegrasyon) testleri başlar. Birim testlerde amaç modüllerin ayrı ayrı kendilerinden beklenen işlevleri yerine getirdiğinin doğrulanmasıyken tümleştirme testlerinin amacı bir araya gelerek entegre edilmiş olan bir yazılımın içerisindeki bileşenleri birbirleriyle uyum içerisinde doğru bir şekilde çalıştığının ve bileşenlerin kendilerine ait gereksinimleri

yerine getirdiğinin doğrulanmasıdır. Bu testlerin yapılması için kullanılacak test durumları, yazılım gereksinimleri ile arayüz gereksinimlerinden çıkartılır. Tümleştirme testlerinde kara kutu test yöntemi kullanılır. Testlerde hem pozitif hem negatif test durumları uygun parametreler kullanılarak koşturulur. Sonuçlar, beklenen değerlerle karşılaştırılarak sistemin kendisinden beklenen davranışları gerçekleştirdiği doğrulanır (softwaretestinghelp).

Tümleştirme testlerin yapılırken big bang, aşağıdan yukarıya veya yukarıdan aşağıya olmak üzere farklı stratejiler kullanılabilir.

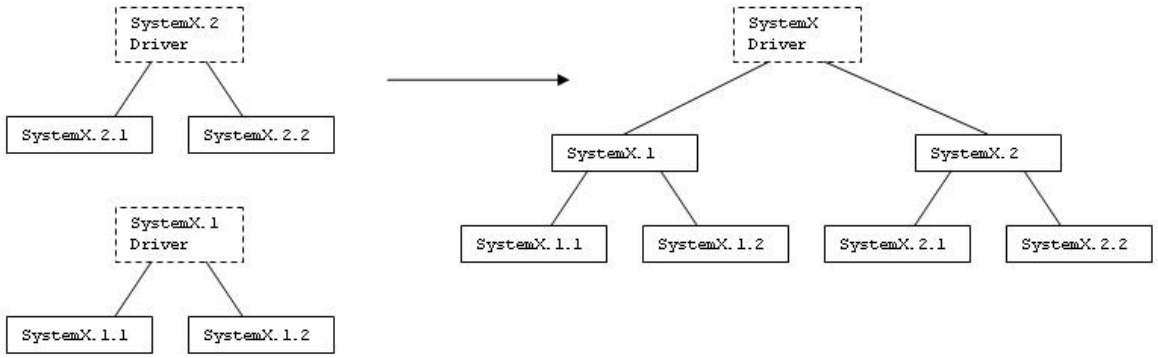
Big bang stratejisine göre geliştirilen tüm üst ve alt düzey modüller birbirleriyle entegre edildikten sonra testler başlar. Bu stratejide sistem bir bütün olarak ele alınarak testler yapıldığından testlerin gerçekleştirilmesi hızlıdır ve zamandan tasarruf edilir. Ama bir hata çıktığında bu hatanın hangi seviye katmanda oluştuğu veya hangi modülden kaynakladığının tespiti zordur. Big bang test yaklaşımı şeması şöyle gösterilebilir (istqbexamcertification, 2011).



Şekil 5.2. Big bang test yaklaşımı örneği

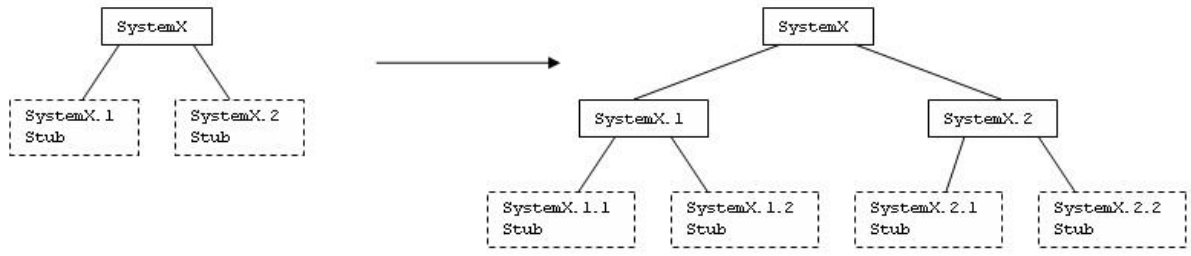
Aşağıdan-yukarıya tümlştirme test stratejisi öncelikle birim testlerin yapılmasıyla başlar. Birim testleri başarıyla tamamlanan alt düzey modüller birbirleriyle entegre edilir. Bu entegrasyondan sonra alt düzel modüller ile ilgili entegrasyon testleri yapılır. Bu testlerin başarılı şekilde sonlandırılmasından sonra bir üst katman modülleri sistemi entegre edilir ve

gerekli entegrasyon testleri yapılır. Bu entegrasyon tüm sistem inşa edilinceye ve entegrasyon testleri başarıyla sonuçlanıncaya kadar devam eder. Bu yaklaşımda temel şart, entegre edilecek modüllerin birim testlerinin başarıyla tamamlanmış olmasıdır. Bu yaklaşım ile hedeflenen sistemin yüzdelik olarak ne kadarının tamamlandığı ve test edildiği kolay bir şekilde belirlenebilir. Aşağıdan-yukarıya entegrasyon test stratejisi paralel programlama, döngüsel yazılım geliştirme metodolojileri için uygun bir test yaklaşımıdır (sce.uhcl.edu). Aşağıdan-yukarıya test yaklaşımı Şekil 5.3.'de gösterilmektedir (sce.uhcl.edu).



Şekil 5.3 Aşağıdan-yukarıya test yaklaşımı örneği

Yukarıdan-aşağıya tümleştirme test stratejisi, öncelikle sistem için en üst modülün (en dış modül, genellikle kullanıcı grafik arayüz modülü) test edilmesiyle başlar. Bu modülün ihtiyaç duyduğu alt modüllerin koçanları kullanılır. Bu yaklaşımda bir çok koçan yazılması gerekir. En üst modül başarılı şekilde test edildikten sonra bir alt düzey modüller sistemle bütünleştirilir (sce.uhcl.edu). Bu entegrasyondan sonra gerekli alt düzey tümleştirme testleri yapılır. Bu durum en alt düzey modüller entegre edilinceye ve entegrasyon testleri başarıyla snuçlanıncaya kadar devam eder. Bu yaklaşımda entegrasyon testleri kara kutu testleri olarak başlar ve gittikçe saydam kutu testlerine döner. Yukarıdan-aşağıya test yaklaşımı Şekil 5.4'de gösterilmektedir (sce.uhcl.edu).



Şekil 5.4 Yukarıdan-aşağıya test yaklaşımı örneği

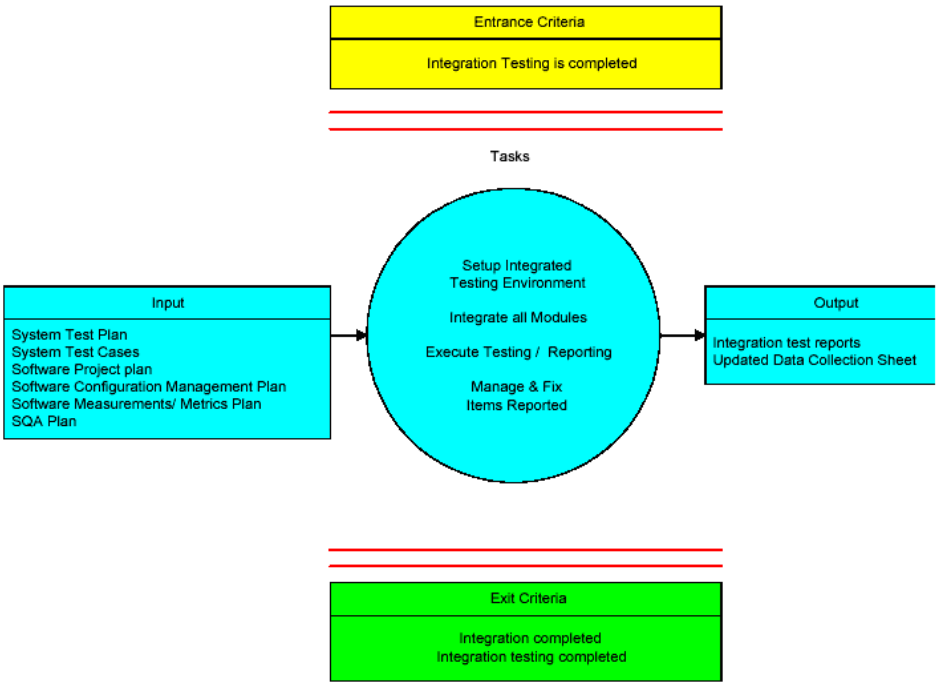
Entegrasyon testleri koşturulan test durumlarının geçme sayısı belirli bir orana ulaştığında veya tamamı geçtiğinde tamamlanmış olur. Bu aşamada bulunan hatalar yazılım geliştiricilere tanımlanmış olan bir hata bildirim mekanizmasına göre bildirilmelidir.

Bildirilen bu hatalar çözümlendikten sonra yeniden oluşturulan yazılım yeni bir sürüm olarak teste alınmalıdır. Yapılan değişikliklerin mevcut yazılıma etkilerinin görülmesi için de yineleme testlerinin yapılması ihmal edilmemelidir.

5.3 Sistem Testleri

Sistem testleri, geliştirilen yazılımın performans, güvenilirlik, işlevsellik gibi özelliklerini değerlendiren testlerdir. Birim ve entegrasyon testlerinde geliştirilen yazılımın tasarıma uygun olarak geliştirildiği doğrulanır. Sistem testleriyse müşterinin sistemden istediklerini doğrulamayı amaçlar. Bu nedenle sistem test durumlarında sistem gereksinimleri temel alınarak sistemin çalışacağı, gerçek ortamda karşılaşılabilecek olan senaryolar tanımlanır.

Sistem testleri, kullanıcı kabul testlerinden bir önceki adım olarak yazılım ve donanım entegrasyonundan sonra "sistem test planına" göre yapılır. Sistemin işlevsel, işlevsel olmayan gereksinimlerinin doğrulanması hedeflenir. Sistem testleri Şekil 5.5.'te gösterilmektedir (etestinghub, 2011).



Şekil 5.5. Sistem testinin adımları

Sistem testlerinin ilk adımı olarak işlevsel gereksinimlerin doğrulanması gerçekleştirilir. Bu doğrulama sonucunda sistemin işlevsel olarak çalıştığı gözlemlenir. İşlevselliği yönünden doğrulanan sistem üzerinde bir sonraki adımda, işlevsel olmayan gereksinimlerin karşılandığı gösterilmek amacıyla işlevsel olmayan testler yapılır. Bu testlerden bazıları:

Stres Testleri: Sisteme girdi oranı sistem tasarım oranını aştığı zaman sistemin davranışını gözlemlemek üzere gerçekleştirilen testlerdir.

Performans Testleri: Sistem çıktılarının belirlenen ve kabul edilebilecek olan zaman dilimi içerisinde üretebildiğinin değerlendirilmesinin yapılabilmesi için gerçekleştirilen testlerdir.

Konfigürasyon ve Uyumluluk Testleri: Geliştirilen sistemin farklı platformlarda ve donanımlarda nasıl davrandığının değerlendirilmesi için yapılan testlerdir.

Güvenlik Testleri: Sistemin izinsiz kullanım teşebbüslerindeki davranışlarının değerlendirilmesi için yapılan testlerdir.

Kullanılabilirlik Testleri: Kullanıcı - sistem etkileşimini ve ergonomisini değerlendirmek üzere yapılan testlerdir.

Geri Alma Testleri: Bir hata durumunda sistemin otomatik veya elle yeniden normal duruma dönmesini değerlendirmek için yapılan testlerdir.

Kullanıcı Arayüzü Testleri: Kullanıcının ve yazılımın grafik gösterimi olarak nasıl bir etkileşim içerisinde olacağını, kullanıcının klavye, ekran veya fare ile sisteme vereceği girdilerin sistem tarafından nasıl işleneceğini değerlendirmek için yapılan testlerdir.

İşlevsel olmayan testleri tamamlanan sistem artık doğrulanmış olarak kullanıcı kabul testlerine hazır demektir. Sistem testleri sırasında ortaya çıkan hatalar proje hata yönetimi sürecine göre raporlanır ve gerekli düzeltme işlemleri yapılır. Gerekli düzeltmelerden sonra, düzeltmelerden sistemin geri kalanının etkilenmediğinin değerlendirilmesi için sistem üzerinde yineleme testleri yapılır.

5.4 Kabul Testleri

Proje kabul testleri müşteri gereksinimlerinin doğrulanması ile gerçekleştirilir. Bunun için her bir kullanıcı gereksinimini doğrulayacak olan test durumları ve test senaryoları oluşturulur. Geliştirilen sistem, tanımlanan bu test durumları ve test senaryoları müşterinin de

katılımıyla “kabul test planına” uygun olarak koşturulur. Sistemin tamamının bu testlerden geçmesi ile sistem, müşteri tarafından kabul edilmiş olur.

Kabul testlerinin yapılmasıyla aşağıdaki durumlar gerçekleşir (softwaretestinghelp):

1. Müşteri gereksinimlerinin kapsandığı doğrulanarak, yazılımın bu gereksinimleri ne oranda karşıladığı ölçülür.

   Müşteri gereksinimleri bir proje için en önemli kayıttır. Eğer proje sonucunda bu gereksinimlerin tam olarak karşılandığı gösterilemezse projenin başarılı şekilde tamamlandığı söylenemez. Bu nedenle geliştirilen sistemin bu gereksinimleri tam olarak kapsadığı ve gereksinimlere uygun olarak çalıştığı doğrulanmalıdır.

2. Daha önceki test eylemlerinde tespit edilemeyen problemlerin tespiti yapılabilir.

   Birim testler sadece ilgili oldukları modül hakkında bilgi verir. Sistemin tamamı hakkında bir bilgi, hele de güven hiç vermez. Kabul testleriyle öncelikle geliştirilen yazılımın müşterinin istediği yazılım olduğu doğrulanır ve varsa daha önceki test aşamalarında tespit edilemeyen hataların bulunması da hedeflenir.

3. Geliştirilen sistemin müşteri gereksinimlerini nasıl karşıladığı gösterilir.

   Birim testler ile kod kapsamaya bakılırken, kabul testleriyle müşteri gereksinimlerinin kapsandığı doğrulanır. Bunu yaparken her bir kullanıcı gereksinimine karşılık en az bir test durumu yazılır. Gereksinimler için yazılan test durumları koşturulur. Kabul test planında tanımlanan kabul kriterlerine ulaşılınca kabul testleri tamamlanmış olur.

Kabul testleri elle yapılabileceği gibi, sistemin durumuna göre otomatik olarak da yapılabilir. Kabul testleri sistemin başarısını doğrudan gösterdiğinden, gerçekleştirilmesine ayrı bir önem verilmelidir. Kabul testlerinde karşılaşılan en önemli soru kabul test durumu ve senaryolarının kimin tarafından yazılacağıdır. Kabul testlerinin yazılması doğrudan müşteri tarafından olabileceği gibi, projenin test ekibi tarafından da yazılabilir. Eğer test durumları ve senaryoları test ekibi tarafından yazılmışlarsa, testler başlamadan mutlaka müşteri onayı alınmalıdır. Bazı projelerde ise ortak bir ekip kurularak birlikte geliştirme yapılır. Kabul test durumları ve senaryoları müşteri gereksinimlerinden çıkartıldıklarından dolayı, yazılım tamamıyla gerçekleştirilmeden yazılmış olmalıdırlar. Yazılan test durumları ve senaryolarıyla müşteri gereksinimlerinin tamamı için test eyleminin gerçekleştirileceği garanti altına alınmış olunur. Ancak bu, tüm gereksinimlerin başarıyla gerçeklendiği anlamına gelmez.

Kabul testleri için şu adımlar bir süreç olarak işletilir:

1. Müşteri gereksinimleri belirlenir ve belgelendirilir.
2. Kabul test planı hazırlanır.
3. Müşteri gereksinimlerini doğrulayacak test durumu ve senaryoları hazırlanır.
4. Test hazırlık gözden geçirme toplantısında hazırlanan test durumu ve senaryoları onaylatılır.
5. Kabul testleri için gerekli test ortamı hazırlanır.
6. Sistem testlerinin başarıyla sonuçlanmasından sonra kabul testleri yapılır.
7. Kabul test planında belirtilen kabul kriterlerine ulaşılıncaya kadar testlerin yapılmasına devam edilir.
8. Test sonuçları belgelendirilir ve müşteriye onaylatılır.

Kabul testleri, bir yazılım projesinin başarısında en kritik adımdır. İyi planlanmış ve belgelendirilmiş kabul testleri, projenin başarısına büyük katkı sağlar, müşteri memnuniyetini arttırır.

6. YAZILIM TEST TEKNİKLERİ

Yazılım testleri yapılırken test edilecek yazılıma bakış açısına göre üç test tekniği vardır. Bunlar kara kutu, saydam kutu ve gri kutu test teknikleridir.

6.1 Kara Kutu Testi

Test edilecek yazılımın iç işleyişi düşünülmeden yapılan testlere kara kutu testleri denilir. Bu yaklaşımda test edilecek yazılımın, sonucu bilinen bir davranışını doğrulamak için davranışın gerektirdiği girdi değerleriyle çalıştırılarak sınanır. Daha sonra yazılımın bu girdi karşısında elde edilen çıktısı, beklenen sonuçla karşılaştırılır. Bu yaklaşımın kara kutu olarak adlandırılmasının nedeni, testçinin testi uygularken yazılımın iç yapısıyla ilgilenmemesidir. Bu nedenle bu testler yazılım geliştiriclerden çok, test ekipleri tarafından uygulanır.

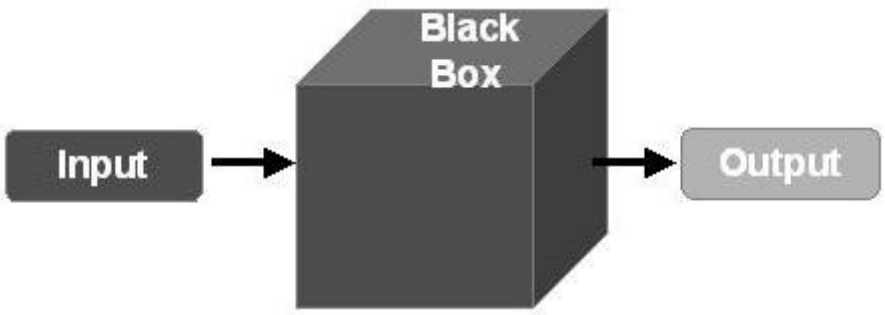
Kara kutu testinde bir yazılım parçası test ediliyorsa, test mühendisi bu yazılım parçasının girdisini ve buna karşılık sistem çıktısını bilir. Fakat bu çıktıya nasıl ulaşıldığıyla ilgilenmez. Çünkü kara kutu testlerinin amacı, verilen girdilerle istenilen çıktının elde edilmesidir. Bu nedenle yazılımın iç işleyişiyle ilgili diğer bilgilerle ilgilenilmez. Kara kutu test tekniğinde diğer önemli bir nokta da geliştirilen yazılımın tasarımından veya kodlanmasından kaynaklanabilecek hataların bulunmasıdır. Bu amaçla kara kutu testlerinin yapılması ve istenilen amaca ulaşılabilmesi için yazılım geliştiricilerle test ekibi birbirinden ayrı çalışmalıdır. Böylece test ekibi, tasarım ayrıntılarını bilerek yazılıma karşı ön yargı taşımaz.

Kara kutu test tekniğiyle geliştirilen yazılım içerisinde şu hata türleri tespit edilmeye çalışılır:

1. Doğru olmayan veya hiç olmayan işlevlerin tespiti
2. Arayüz hataları
3. Performans hataları
4. Veri tabanlarına ulaşma hataları veya veri yapılarındaki hatalar
5. İlklendirme veya sonlandırma hataları

6. Sınır değer hataları

Kara kutu test tekniği basit olarak Şekil 6.1.'de gösterilmektedir.



Şekil 6.1. Kara kutu test yaklaşımı

Kara kutu testinin avantajları/dezavantajları (msdn.microsoft):

Kara kutu testinde yazılım modüllerinin iç işleyişiyle ilgilenilmez. Yazılımın belirlenen girdi karşısında beklenen çıktıyı vermesi hedeflenir. Kara kutu testlerinin avantaj ve dezavantajları şunlardır:

Avantajlar:

i. Yazılımlarda hataların bulunması için etkin ve hızlı bir tekniktir.

ii. Test durumları yazılırken gereksinimlerden hareket edildiği için gereksinimlerdeki tutarsızlıkların ve belirsizliklerin belirlenmesinde önemlidir.

iii. Testçilerin yazılımın ayrıntılarını bilmesine gerek yoktur.

iv. Testçiler ve kodcular birbirinden bağımsız çalışabilirler.

v. Testçiler gereksinimleri doğrulamak ve gereken testleri gerçekleştirmek için yazılıma kullanıcı gözüyle bakarlar. Bu da kodcular tarafından fark edilemeyen pek çok olası hatanın ve eksiğin bulunmasına yardımcı olur.

vi. Testleri yapacak kişilerin sistem hakkında teknik ayrıntı bilmesine gerek yoktur.

Dezavantajlar:

i. Kara kutu testleri yazılımın belirli parçasını hedeflemez. Bu nedenle bir çok hata tespit edilmeden kalabilir, bunlar için başka testler gerekir.

ii. Sadece belirli sayıda girdi değeriyle testler yapılır. Tüm girdiler ile testlerin yapılması sonsuza kadar sürer

iii. Yazılım içerisinde bazı kod parçalarında birden fazla test yapılırken bazı kod parçaları hiç test edilmeden kalabilir

iv. Açık ve yalın olmayan gereksinimlerin test durumlarını tasarlamak ve testlerini yapmak kara kutu test tekniğinde zordur.

6.1.1 Kara Kutu Test Stratejisi

Kara kutu test tekniği tümleştirme, sistem ve kabul test aşamalarında kullanılır. Bu testlerde en verimli sonuçlara ulaşmak için şu kara kutu test stratejisi izlenebilir:

i. Kara kutu testleri rasgele belirlenmiş girdilerle gerçekleştirilmeli, testçiler sadece girdi aralığını belirtmelidir.

ii. Yazılımın sağlamlığının kontrolü için belirtilen aralığın dışındaki değerlerin de testi yapılmalıdır.

iii. Sınır değerler mutlaka test edilmeli, en yüksek ve en düşük değerlerin beklenen çıktıyı verdiği mutlaka doğrulanmalıdır.

iv. Değer artışlarında artış miktarı ayrıca test edilerek doğrulanmalıdır. Artış miktarı dışı artımlarda yazılımın tanımlı davranışına göre hareket ettiği doğrulanmalıdır.

v. Sayısal girişlerde sıfır değeri mutlaka girdi olarak sınanmalıdır.

vi. Özellikle gerçek zamanlı sistemlerde stres testi yapılmalıdır. Programın aşırı yüklenme altında nasıl çalıştığı test edilmelidir.

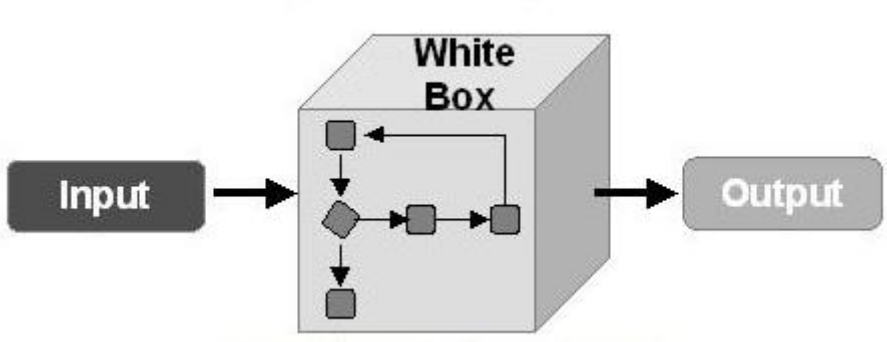
vii. Emniyet kritik, görev kritik gibi yazılımlar için yazılımın hatası durumunda nasıl davrandığı test edilmelidir. Bu amaç için kritik yazılımların kara kutu testlerinde test izleme yazılımlarından yararlanılabilir.

viii. Kara kutu testlerinde diğer bir amaç gereksinimlerin doğrulanması olduğundan her bir gereksinim için en az bir test durumu yazılmalı ve bu şekilde gereksinim kapsama gerçekleştirilmelidir.

6.2 Saydam Kutu Testi

Saydam kutu testleri yazılımın iç yapısı bilinerek tasarlanır. Bu nedenle saydam kutu testlerini gerçekleştirenler genellikle sistemin iç yapısını bilen kodculardır. Saydam kutu testiyle programın iç yapısındaki birimlerin içindeki hatalar araştırılır. Kaynak kod, saydam kutu testlerinin en önemli girdisi olduğundan, koda ulaşım olmadan saydam kutu testleri yapılamaz. Bunun yanında risk analizleri ve tasarım kısıtları da saydam kutu testlerinin planlanmasında, test stratejisinin belirlenmesinde, test araçlarının seçilmesinde ve test verilerinin oluşturulmasında kullanılır.

Saydam kutu testleri veri, kontrol ve bilgi akışlarının, kodlama standartlarının, hata yakalama ve ayıklama yapısının analizlerini içerir. Beyaz (saydam) kutu test tekniği basitçe Şekil 6.2'de gösterilmektedir.



Şekil 6.2 Saydam kutu testi şeması

Saydam kutu test yaklaşımı kullanılarak yapılan testler şunlardır:

Birim Testler: Saydam kutu testinin en iyi ve yaygın kullanımı birim testlerdir. Birim testler, kodcuların belirli bir kod parçasının görevini doğru şekilde yerine getirip getirmediğini anlamak için yapılan testlerdir.

Statik ve Dinamik Analizler: Statik analiz, kod içerisindeki muhtemel hataları bulmak için yapılan kod üzerindeki incelemeleri içerir. Dinamik analizlerse kodun çalıştırılmasını ve çıkan sonucun analiz edilmesini içerir. Bu nedenle saydam kutu testleri kaynak koda ulaşım hakkı getirir.

Deyim Kapsama (Statement Coverage): Bu tür testte kod çalıştırılarak kod içerisinde yer alan her deyimin en az bir kez çalıştırılması hedeflenir. Böylece her bir

deyimin bir yan etki göstermeden çalıştığı doğrulanır. Kod içerisinde çalıştırılmayan deyim olmadığı da doğrulanır.

Dal Kapsama (Branch Coverage): Hiçbir kod düz bir akışla yazılmaz. Kod içerisinde karar noktaları bulunur ve bu noktalardan kod yan dallara ayrılır. Dal kapsama ile program içerisinde yer alan tüm dalların kendilerinden beklenildiği şekilde çalıştığı doğrulanır.

Yol Kapsama (Path Coverage): Kod içerisindeki tüm yolların test edilmesidir.

Saydam kutu testiyle hata ve yanlışlar en erken safhada ve en hızlı şekilde bulunur. Böylece entegrasyon ve sistem testleri daha hızlı yapılabilir ve bulunan hata sayısında önemli bir azalma gözlemlenebilir.

Saydam kutu testinin avantajları/dezavantajları (msdn.microsoft):

Saydam kutu test tekniğiyle sınanan birimin veya modülün belirlenen girdiyle beklenen çıktıyı vermesi hedeflenir. Ancak bu çıktıyı nasıl verdiği ve kod içinde hangi yollardan geçildiği de incelenir.

Avantajlar:

i. Kod içerisinde gizli kalmış mantıksal hatalar bulunur.

ii. Saydam kutu testleriyle yazılan kodun optimizasyonuna katkıda bulunulur.

iii. Kod içindeki fazla satırlar ayıklanarak ölü kod parçaları bulunur.

iv. Kaynak kodun analiz edilmesi ve bu analize göre testlerin gerçekleştirilmesiyle yazılım içerisindeki hatalar daha erken aşamada ve daha hızlı bulunur.

v. Yazılımın geliştirilmesi için belirlenmiş olan kodlama rehberine uyumluluk, tasarım kararlarına kodlama içerisinde uyulup uyulmadığı saydam kutu testleriyle net olarak görülür.

vi. Saydam kutu testleriyle yazılımcıların kod geliştirme yetenekleri desteklenir ve güçlendirilir.

Dezavantajlar:

i. Eğer birim tümleştirme testleri test ekibi tarafından yapılacaksa, bu iş için kodun iç yapısının bilinmesi gerekir. Bu da maliyeti arttırır.

ii. Saydam kutu testleriyle sadece modül ve birimlerin iç işleyişleri test edildiğinden, tümleştirmeden sonra ortaya çıkabilecek olan hatalar tespit edilemez.

6.3 Gri Kutu Testleri

Gri kutu testleri, saydam kutu ve kara kutu test tekniklerinin birlikte kullanılmasıdır. Bu testlerde gereksinimleri doğrulayacak test durumları kodun iç yapısı esas alınarak yazılır. Böylece gereksinimlerin gerçeklendiği doğrulandığı gibi, yazılımın iç yapısı sınanmış olur.

Gri kutu testlerinde test ekibi yazılımın iç yapısını bildiğinden, yazılımın tasarımına ve kod yapısına karşı şartlanma olabileceğinden bazı hataları ortaya çıkartabilecek testler yapılmadan kalabilir.

7. YAZILIM TEST TÜRLERİ

Yazılımların büyüklüğü ve karmaşıklığı arttıkça, ortaya çıkabilecek hata ve eksiklerin de sayısı artar. Bu hata ve eksiklikler, yazılımların zamanında bitirilememesine, istenen yazılımın elde edilememesine, maliyetin artmasına ve sonuçta geliştirilen yazılımın başarısız olmasına neden olabilir. Bunun için hataların ve eksikliklerin, yazılım geliştirme yaşam döngüsünün erken safhalarından itibaren tespit edilmesi, gerekli düzeltici faaliyetlerin yapılarak bu faaliyetlere göre yazılımın geliştirilmesinin devam etmesi gereklidir. Yazılım yaşam döngüsü boyunca yapılan test faaliyetleri statik testler ve dinamik testlerdir.

7.1 Statik Testler

Statik test, yazılım veya yazılım parçasının çalıştırılmadan gerçekleştirilen test türüdür. Statik testlerde yazılımın doğruluğu kontrol edilir. Bu amaçla kodun çalıştırılması yerine, yazılan kod hata bulmak amacıyla okunur. Bu okumada kod gözden geçirilerek incelenir ve statik olarak analiz edilir. Bu testler kodun iç yapısı üzerinde yapıldığı için saydam kutu testlerinin bir safhası olarak projelerin doğrulama ve geçerleme süreci içerisinde yapılır.

Statik testlerin önemi, hataların yazılım yaşam döngüsü içerisinde erken safhalarda yakalanmasını sağlamasıdır. Böylece yazılım geliştirmenin gelecek adımlarında oluşabilecek hatalar daha erken safhalarda tespit edilmiş olmakla yazılım geliştirmenin uzamasına ve maliyetin artmasına engel olunur.

7.2 Yazılım Gözden Geçirmeleri

Yazılım gözden geçirmeleri, yazılım geliştirim yaşam döngüsü zamanının ve maliyetinin artmasını engelleyen ve yazılım ürününün kalitesini geliştiren bir süreçtir.

Yazılım gözden geçirmelerindeki amaç, geliştirilen ürünün standartlara ve belirtimlere/özelliklere uygun olarak geliştirildiğinden emin olmak, ürünün içerebileceği olası mantık, işlev veya uygulama hatalarını ve eksiklikleri en erken zamanda ortaya çıkarmaktır.

T

Tüm gözden geçirmeler genel olarak, yazılım ürünündeki hata ve eksiklikleri bulmak amacını paylaşır.

Gözden Geçirme: Bir yazılım ürününün proje elemanlarına, yöneticilere, kullanıcılara veya diğer ilgili kişilere, yorum yapmaları, eksiklik ve hataları ortaya koymaları veya onaylamaları için sunulması süreci veya bu amaçla yapılan toplantıdır.

Yönetimsel Gözden Geçirme: Proje gelişim sürecinin izlenmesi, plan ve proje takviminin durumunun saptanması, gereksinimlerin sistem bileşenleriyle ilişkilendirilmelerinin onaylanması, proje hedeflerine ulaşılabilmesi için uygulanan yönetimsel yaklaşımların verimliliğinin değerlendirilmesi amacıyla yönetim tarafından ya da yönetim adına atanan kişilerce yapılan sistematik değerlendirmelerdir.

Yönetimsel gözden geçirme, üst yönetim katılımıyla yapılır ve diğer gözden geçirme tiplerinde ele alınan alt düzey teknik ayrıntılara girilmez.

Teknik Gözden Geçirme: Kalifiye bir ekip tarafından, bir yazılım ürününün hedeflenen amaca uygunluğunun saptanması, gözden geçirilen ürünün kendi belirtimlerinden ve standartlarından sapmalarının tanımlanması amacıyla yapılan sistematik değerlendirmelerdir.

Denetleme: Bir yazılım ürününün, bir yazılım sürecinin veya bir dizi yazılım süreç faaliyetlerinin belirtimler, standartlar, sözleşme veya diğer unsurlar bakımından uyumunun değerlendirilmesi için yapılan sistematik bağımsız değerlendirmedir.

İnceleme: Bir yazılım ürünündeki hatalar ve standartlardan sapmalara neden olan anormalliklerin belirlenip tanımlanması için inceleme teknikleri konusunda eğitimli, tarafsız kişilerin rehberliğinde yapılan sorgulamadır.

Kod Geçişleri: Bir yazılımın geliştiricisi tarafından diğer geliştirme ekip üyelerine anlatılarak, yazılım ürününün iyileştirilmesine yönelik görüşlerin alınması ve standartların ihlali veya olası hataların belirlenmesidir.

7.2.1 Gözden Geçirmelerde Amaç

Gözden geçirmelerde amaç, geliştirilen ürünün standartlara ve kendi belirtimlerine uygun geliştirildiğinden emin olmak, ürünün içerebileceği olası mantık, işlev veya uygulama hatalarını ve eksiklerini en erken zamanda ortaya çıkarmaktır. Gözden geçirmelerin bazı amaçarı şunlardır:

i. Gözden geçirilen ürünlerde hata sayısını azaltmak.

ii. Daha erken hata tespiti, daha az düzeltici faaliyetle proje takviminin uzamasına engel olmak.

iii. Problemleri keşfetmek ve yönetimin dikkatini bu problemler üzerine çekmek.

iv. Ürünlerin kendi belirtimlerine ve standartlara uygun olduğunu doğrulamak.

v. Alternatif çözümleri keşfetmek ve değerlendirmek.

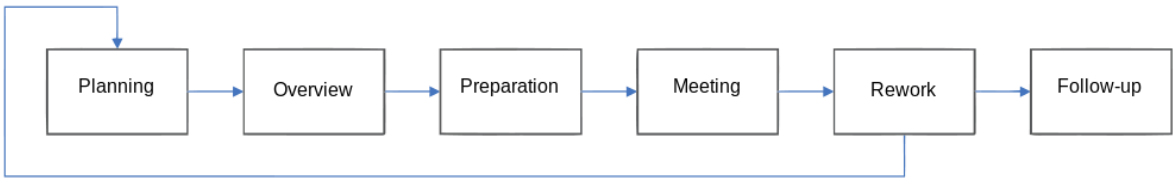
vi. Projenin gelişiminin planlandığı gibi gerçekleştiğini paydaşlara, müşteriye ve üst yönetime göstermek.

vii. Proje elemanlarına, elde edilen tecrübelerle ilgili eğitim vermek.

7.2.2 Gözden Geçirme Süreci

Yazılım gözden geçirmeleri 1972 yılında IBM Kingston'da çalışan Michael Fagan tarafından "yazılım kalitesini geliştirmek ve yazılım geliştiricilerin verimliliğini arttırmak amacıyla gerçekleştirilen toplantılar" şeklinde duyurulmuştur.

Projelerde gözden geçirmeler resmi ve resmi olmayan şeklinde uygulanır. Resmi gözden geçirmeler, belirli kurallarla tanımlı bir sürece bağlı kalınarak gerçekleştirilen toplantılardır. Sürece bağlı olmadan, günlük çalışmalar esnasında yapılan teknik değerlendirmelerse resmi olmayan gözden geçirmelerdir. Fagan'ın 6 adımlı gözden geçirme tekniği şöyle gösterilebilir (csis.pace.edu).



Şekil 7.1 Fagan'ın 6 adımlı gözden geçirme süreci

Planlama: Gözden geçirme sürecinin hazırlanması ve organize edilmesidir. Bu kapsamda gözden geçirme materyalleri, prosedürleri, toplantının takvimi, gözden geçirmeye katılacak kişiler ve rolleri hakkında hazırlıklar yapılır.

Bilgilendirme: Bu aşamanın amacı, gözden geçirmeye katılacak olanların gözden geçirme hakkında ve gözden geçirilecek olan ürün hakkında eğitilmesi yani tüm ekibin temel bilgi düzeyine ulaşmasını sağlamaktır.

Bireysel Hazırlık: Ürün hakkında gerekli bilgileri öğrenen ekip elemanları daha sonra kendilerine ait rollerle gözden geçirilecek olan ürünü inceler ve ilgili gözden geçirme kayıtlarını doldururlar. Bununla, gözden geçirme toplantısından önce ürün üzerindeki hataların keşfedilmesi ve toplantı sırasında bunların dile getirilmesi amaçlanır.

Grup Toplantısı: Bu aşamada, bireysel olarak bir önceki adımda tespit edilen hatalar bir araya getirilir. Gözden geçirme toplantısı genelde, gözden geçirilecek ürün sorumlusunun ürünü kısaca tanıtımıyla başlar. Daha sonra bireysel gözden geçirmelerde tespit edilen hatalar teker teker gündeme getirilir. Bunlar hakkında tüm katılımcıların görüşleri alınır. Gerekli düzeltici faaliyetler planlanarak ilgili kişilere göre ataması yapılır.

Tekrar Çalışma: Bu süreç, hata düzeltme süreci olarak da adlandırılır. Grup toplantısında karar verilen düzeltici faaliyetlerin kişilerce gerçekleştirildiği süreçtir.

İzleme: Bu aşamada, belirlenen tüm eylem maddelerinin yerine getirilip getirilmediği gözden geçirme sorumlusu tarafından izlenir ve kontrol altında tutulur. Gereken durumlarda ürün için yeni bir gözden geçirme toplantısı da planlanabilir.

7.3 Statik Kod Analizleri

Yazılım çalıştırılmadan yazılım içerisindeki hataları ortaya çıkarmak amacıyla yapılan statik kod analizleri, statik test test tekniklerinden biridir. Yazılımın koşturulması sırasında fark edilemeyecek veya geç fark edilecek bazı hatalar, statik kod analizleriyle bulunabilir. Statik kod analizleri yazılımın kalitesi, çalışma performansı, işlevselliği, güvenilirliği açılarından önemlidir.

Statik kod analizlerinin temel mantığı, kod üzerinde sorgulamalar yapmaktır. Örneğin tanımlanan bir k değişkenine ilk değer verilmiş mi? İlk değer verilmeden kullanılan bir değişken var mı? Değer atanmadan önce okunan bir değişken var mı? k değişkenine atanan değer, alabileceği en küçük ve en büyük değer aralığında mı? Gibi sorgulamalar statik kod analizleri kapsamındadır.

Statik kod analizlerinde kullanılan bazı yöntemler şunlardır:

Bağlam Duyarlı Fonksiyonlar Arası Analiz (Context Sensitive Interprocedural Analysis) (codeproject) : Bu analizle fonksiyon/prosedürler arası ilişkiler ve veri akışları sorgulamaları yapılır. Aşağıdaki örnek kod içerisinde buggy() fonksiyonu p ve 2 parametresiyle my_free() fonksiyonunu çağırmaktadır. my_free() içerisinde ikinci parametre olarak 2 gönderildiğinden ve bu değer sıfırdan büyük olduğundan dolayı p serbest bırakılmaktadır; ancak serbest bırakılmasına rağmen daha sonra buggy() fonksiyonu içerisinde p'nin gösterdiği sanılan yere boşluk ataması yapılmaya çalışılmaktadır. Bu tür hatalar bu analiz yöntemiyle testlerden önce belirlenerek düzeltilebilir. Bu analiz yöntemi örneği Şekil 7.2.'de gösterilmektedir.

```
100   void buggy(char *p) {
101         my_free(p, 2);
102         *p = '\0';
103   }
104
105   static int total_alloc;
106
107   void my_free(void *p, int sz) {
108         if(sz > 0)
109               free(p);
110         total_alloc -= sz;
111   }
```

Şekil 7.2 Bağlam Duyarlı Fonksiyonlar Arası Analiz Örneği

Veri Akış Analizi (Data Flow Analysis) (codeproject): Yazılım içerisindeki değişkenlerin tanımlanması, değikenlerin kullanılması, metot dışından gelen değerlerin değişkene atanması veya değişkenin metot dışına değer göndermesi, değişkenlerin kullanıldıktan sonraki durumunun analiz edilmesidir. Bu analiz sonucunda elde edilen bulgular kontrol akış grafikleri veya çağrı grafikleri kullanılarak gösterilir.

Veri akış analizleri ileri yönde ve geri yönde olmak üzere iki kapsamda gerçekleştirilir. İleri yönde gerçekleştirilen veri akış analizinde, akış kontrolü yapılan değişkenin ileriye doğru değer aktarması incelenrek kontrol akış grafiği çizilir. Geri yönde veri akış analizlerindeyse akış kontrolü yapılacak olan değişkenin en son noktada aldığı değerden başlanarak kod içerisinde ilk değer atanmasının yapıldığı yere kadar incelenerek kontrol akış grafiği çizilir. Bu analiz sonucunda değişkenlerin tanımlı olup olmadığı, tanımlı olduğu halde hiç referans edilmediği, tanımlı bir değişkenin ilk değer verilmeden kullanılmaya başlandığı, değer aktarımlarında değişkenin alabileceği değer aralığı dışında bir değerin aktarıldığı gibi hatalar veri akış analizi kapsamında bulunabilecek olası hatalardır.

Kodlama Standardı Analizi (Coding Standard Analysis) (codeproject): Yazılan kodun belirlenmiş kodlama standartları çerçevesinde analiz edilmesidir. Bu yöntemde kullanılacak standartlar uluslar arası kabul görmüş standartlar olabileceği gibi, projenin kendisinin belirlediği kodlama standartları da yazılımın analiz edilmesinde kullanılabilir.

İşaretçi Analizi (Pointer Analysis) (codeproject): Bu analiz yönteminde yazılımın çalıştırılması sırasında işaretçilerin hangi değişken veya yordamı göstereceği, bu işaretçilerin kullanımında hataların olup olmadığı üzerine analizler yapılır.

Yanlış Yol Budama Analizi (False Path Pruning) (codeproject): Geliştirilen yazılım içerisinde yazılımı amacına ulaştıracak bir çok yol ve yan yollar bulunur. Yazılım çalıştırılması sırasında o an yapacağı göreve göre bu yolları takip ederek kendisinden beklenen sonuca ulaşmaya çalışır. Bu yollar ve sayısı, programcıya göre değişiklik gösterir. Kullanılan algoritmik yaklaşıma göre bu yolların sayısında değişiklik olacaktır. Çok sayıda yolun bulunduğu durumlarda bazı yollar hiçbir şekilde çalıştırılmadan kalabilir. Yanlış yol budama analizi, yazılım içerisinde çalışması mümkün olmayan yolları bulmayı amaçlar. Böylece hem geliştirilen yazılımın gereksiz satırlar içermesi engellenir hem de yazılımdaki toplam satır sayısının gereksiz yere artması önlenir. Bu durum özellikle sınırlı bellek alanının bulunduğu gömülü sistem yazılımlarında önemlidir.

Fonksiyon Değer Analizi (Function Value Analysis) (codeproject): Yazıılmdaki fonksiyonların dönüş değerlerinin analizinin yapıldığı yöntemdir.

Statik analiz yöntemleriyle farklı programlama dillerine özgü hatalar, güvenlik açıklıklarından kaynaklanan hatalar, yazılım içindeki olası ölü kod parçaları bulunabilir. Statik kod analizleri manuel olarak gerçekleştirildiğinde zor, maliyetli ve hataya açıktır. Bu nedenle statik kod analizleri yapılırken yazılım test araçları kullanılır.

7.4 Dinamik Testler

Geliştirilen yazılımın dinamik davranışının gözlemlenmesi için yapılan testlerdir. Dinamik testler kapsamında sistemin değişen veriler karşısında nasıl tepki verdiği gözlemlenir. Dinamik testlerin yapılması için yazılan kod derlenir ve koşturulur. Yazılımın üzerinde çalıştırılacağı platforma ve yazılımın tipine göre dinamik testler farklılık gösterir. Bazı dinamik testler şunlardır:

Performans Testleri: Geliştirilen yazılımın hedeflenen platform üzerindeki veya birlilkte çalışacağı diğer uygulamalarla birlikte çalışırkenki performansını ölçmek veya belirlenen performans hedefine ulaşılıp ulaşılmadığını göstermek amacıyla yapılan testlerdir. Bu testlerle uygulamanın aşırı yüklenmeler altında nasıl davrandığı gözlemlenir. Performans testleri genellikle yazılım test araçlarıyla otomatik olarak yapılır. Bu tür test yazılımları aşırı yüklenmeleri, istisnai durumlardaki girdileri ve normal girdilerin benzetimini gerçekleştirerek uygulamanın her tür koşul altında test edilebilmesini sağlar.

Uyumluluk Testleri: Bir yazılımın farklı tarayıcılarda, işletim sistemlerinde veya donanım üstünde beklenen şekilde çalışıp çalışmadığını göstermek üzere yapılan testlerdir. Bu testler otomatik veya manuel olarak yapılabilir.

Yükleme Testleri: Yazılıma çökene kadar yüklenilerek yazılımın sınırlarının ölçüldüğü veya maksimum yük seviyelerinin belirlendiği testlerdir.

Stres Testleri: Yazılımın belirtilen limit değer ve limit değer aşımlarında nasıl davrandığını ve başarısızlığa gittiğini değerlendirmek amacıyla yazılıma yüklenilerek gerçekleştirilen testlerdir. Bu testler altında sistemin ölümcül hatalara sebebiyet vermeyen sonuçlar vermesi beklenir. Bazen stres testleri performans testleriyle aynı süreç içerisinde yapılabilir.

Uygunluk Testleri: Yazılımın kendi belirtimlerine uygun geliştirildiğini doğrulamak üzere yapılan testlerdir. Geliştirilen yazılımın taşınabilirlik, birlikte çalışabilirdik gibi kendisiyle ilgili gereksinimleri sağladığının gösterilmesi bu testlerin amacıdır.

İşlevsel Testler: Bir yazılımın kendi belirtimlerine uygun olduğu ve kendinden beklenen işlevleri yerine getirdiğini göstermek amacıyla gerçekleştirilen testlerdir. Bu doğrulamanın gerçekleştirilmesi için yazılımın tüm özellikleri doğru ve yanlış veriler kullanılarak test edilir. Bu testler kullanıcı grafik arayüz testlerini, veritabanı yönetim testlerini, güvenlik, enstalasyon, ağ bağlantısı vb testlerinin manuel veya otomatik olarak yapılmasını kapsar.

Keşif Testleri: Keşif Testleri aynı anda öğrenmenin, test tasarımının ve test koşturulmasının yapılmasıdır. Yapısal bir yaklaşım sergilenmez. O anki duruma göre testin bir sonraki adımı gerçekleştirilir. Her şey anlık ve geçici olarak gerçekleştirilir.

Yineleme Testleri: Yazılım üzerinde yeni bir işlevsel birim geliştirildikten veya bir düzeltici faaliyet gerçekleştirildikten sonra yapılan testlerdir. Amaç, yapılan geliştirmenin veya

düzeltme faaliyetinin yazılımın diğer bölümlerini olumsuz yönde etkilemediğinin gösterilmesidir.

Duman Testleri: Büyük bir planlama yapmadan hızlı bir şekilde yapılan testlerdir. Yazılımın ince ayrıntılarına bakılmadan genel işlevleri yerine getirip getirmediğine bakılır. Bu testle gelecek yazlımın test edilebilir durumda olup olmadığı anlaşılabilir. Bu amaçla yazılımın temel akışlarından birkaç senaryo seçilip koşturulur. Senaryoların başarı oranına göre yazılımı testine başlanabilir veya yazılım geliştirmesi tamamlanmadığı belirtilerek reddedilebilir.

Sürüm Doğrulama Testleri: Yazılım yeni sürümü çıktığında test için test grubuna teslim edilmesinden hemen önce sürümün test edilebildiğinin doğrulanmasıdır. Bu amaçla seçilen test durumları, sürümün o anki durumu için geçerli olan ana işlevleri kapsar. Bu süreç, geliştirilen yazılımın çalışacağı platforma göre otomatikleştirilebilir. Bu testten kalan yazılım sürümü test amacıyla testçilere teslim edilmez ve düzeltilmesi için geliştiricilere geri gönderilir. Sürüm doğrulama testleriyle hatalı bir üretim varsa testlerden önce keşfedilerek zaman ve maliyetten kazanılmış olunur.

Geri Alma Testi: Sistemdeki çökmeler sonrasında yazılımın davranışlarını inceleyip zararın en aza indirgenmesi için uygulanan testlerdir. Özellikle hareket (transaction) bazında işlemler içeren uygulamalarda testin olmazsa olmaz bir parçasıdır.

Güvenlik Testi: Yazılımın, iç ve dış kaynaklı yetkisiz erişimlere, kötü amaçlı kullanımlara karşı korunması ve güvenliğinin yeterliliğini değerlendirmek amacıyla yazılımdaki açıkları tanımlamak ve gidermek için gerçekleştirilen testlerdir. Bu testten geçebilmek için yazılımın tasarım ve kodlama sürecinde risk noktaları belirlenmeli ve bu noktalara karşı yazılımsal önemler alınmalıdır.

8 TEST BELGELENDİRMESİ

Test ekibinin aşması gereken zorlukların başında test çalışmaları kapsamında standart belgelerin ve şablonların üretilmesi zorunluluğudur.

8.1 Belgelendirme Tanımları

1. **Test Belirtimleri (Spesifikasyon)**

   - **Test planı:** Testlerin zaman, maliyet ve gerçekleştirilmesi yönünden planlamasını içeren belgedir.
   - **Test tasarım belirtimleri:** Yazılımların test edilebilmesi için gerekli test gereksinimlerinin tanımlandığı belgedir.
   - **Test durumu belirtimleri:** Test durumlarını içerir.
   - **Test yordamı belirtimleri:** Testlerin ne şekilde koşturulacağını tanımlar.
   - **Test öğeleri dağıtım raporu:** Test için üretilecek olan öğeleri tanımlar.

2. **Test Koşturma**

   - **Test logları:** Gerçekleştirilen testlerin zaman damgalı ayrıntılı kayıtlarını içerir.
   - **Test olay raporu:** Testlerin gerçekleştirilmesi sonucunda ortaya çıkan, araştırılması gereken beklenmedik olay ve davranışların detaylı kayıtlarını içerir.

3. **Test Raporlama**

   - **Test sonuç raporu:** Testlerin değerlendirildiği ve sonuçlarının kayıt altına alındığı belgedir.

8.1.1 Test Planı

Yazılım projelerinde proje başarısını doğrudan etkileyen eylemlerin başında iyi bir test planlaması gelmektedir. Bu amaçla yazılım projelerinde farklı aşama test planları hazırlanır.

Bu planlarda test edilecek yazılım parçaları, özellikler (işlevsellik, performans, güvenlik, kullanılırlık vb.), yapılacak görevler, çıktılar, gerekli kaynaklar, sorumluluklar, takvim ve gerekli onaylar tanımlanır. Test planı şablonuna göre geliştirildiğinde şu başlıkları kapsar (softwaretestinghelp):

1. Test plan tanımlayıcısı
2. Giriş
3. Test öğeleri
4. Test edilecek özellikler
5. Test edilemeyecek özellikler
6. Yaklaşım (Strateji)
7. Geçme/Kalma kriterleri
8. Testi durdurma ve teste yeniden devam etme kriterleri
9. Test çıktıları
10. Test görevleri
11. Çevresel ihtiyaçlar
12. Sorumluluklar
13. Personel ve eğitim ihtiyaçları
14. Takvim
15. Riskler ve öngörülmeyenler
16. Onaylar

Test plan tanımlayıcı: Konfigürasyon yönetimindeki belgeye sürüm kontrol için verilen bir sayıdır. Proje kapsamında geliştirilen test planlarının, projenin çeşitli safhalarında gelinen nokta itibariyle güncellenmesi gerekebilir. Bu nedenle konfigürasyon yönetimi altında test planları, test plan tanımlayıcısıyla belgenin sürüm kontrolü sağlanmış olur.

Giriş: Projenin kapsamının belirtildiği, bu kapsam içerisinde gerçekleştirilecek olan testlerin neleri kapsayacağının açıklandığı ve geliştirilen planın amaçlarına ilişkin bilgilerin verildiği bölümdür. Bu bölüm, şu bilgileri içerecek şekilde ayrı başlıklarda yazılabilir:

i. **Proje Kapsamı:** Projenin tarihsel gelişimi, özellikleri, neyi hedeflediği gibi bilgiler verilebilir.

ii. **Test Planı Kapsamı:** Test planının hangi testleri kapsadığı ve hangilerini kapsamadığı bilgileri verilir.
iii. **Test Planı Amacı:** Test planının neyi amaçladığı bilgisi verilir.

Test Öğeleri/İşlevleri: Hazırlanan test planı kapsamında test edilmesi amaçlanan nesnelerdir. Temel olarak, test planının seviyesine göre yazılıma ait hangi öğelerin (işlevlerin) test edileceğine dair oluşturulan bir listedir.

Test Edilecek Özellikler: Bu bölümde nelerin test edileceği sıralanır.

Test Edilmeyecek Özellikler: Bu bölüm, her ne sebeple olursa olsun yazılımın test edilemeyecek olan özelliklerinin belirtildiği bölüdür. Bazı öğelerin test edilmemesinin şu gibi çeşitli sebepleri olabilir:

i. Yazılımın çıkacak sürümü bu öğeleri içermeyecektir.

ii. Yazılımın düşük önem seviyesine sahip bölümüdür. Daha önceden kullanılmış ya da test edilmiş olabilir.

iii. Yazılım içerisinde bulunacaktır. Ancak yazılımın bu sürümü içerisinde olduğu belgelendirilmeyecek ve test edilmeyecektir.

Eğer emniyet kritik bir yazılım geliştiriliyorsa geliştirilen yazılımın tüm özelliklerinin test edilmesi hedeflenir.

Yaklaşım (Strateji): Bu bölüm, test planının en önemli parçasıdır. Bu bölümde testin nasıl gerçekleştirileceği tanımlanır. Planın yapıldığı test seviyesine (ana, kabul ve sistem gibi) uygun açıklamalar bu bölüm içerisinde yer alır. Eğer yazılan plan ana test planıysa bu bölümde tüm projenin test yaklaşımıyla gereksinimlerin kapsanmasıyla ilgili tanımlar da yapılmalıdır. Gerçekleştirilecek testlerle ilgili şu gibi sorular cevaplanmaya çalışılır:

i. Test için kullanılacak özel araçlar var mı? Varsa bunlar nelerdir?

ii. Kullanılacak araçlar özel bir eğitim gerektiriyor mu?

iii. Hangi metrikler toplanacak?

iv. Test ortamı, donanım, yazılım gereksinimleri neler?

v. Hangi seviyede gerileme gerçekleştirilecek ve her bir test seviyesinde ne kadar gerileme testi yapılacak?

vi. İnsan, para ve kaynak kullanımı nasıl gerçekleştirilecek?

Örnek olarak sadece tamamlanmış bileşenler test edilecek veya belirlenmiş bir grup özellik bir arada test edilecek gibi özel durumlar varsa bu yöndeki strateji kararları bu bölümde verilmelidir.

Metodoloji, kaynaklar, test kapsamları, konfigürasyon yönetimi, otomasyon ve kullanılacak araçlarla ilgili kararlar da bu bölüm içerisinde belirtilir.

Şu soruların cevapları da bu bölüm içerisinde verilir:

i. Testçiler projeye ne zaman dahil olacak?

ii. Test çalıştırması ne zaman başlayacak?

iii. (Varsa) Ne kadar alfa öncesi, alfa veya beta testçisi kullanılacak?

iv. Hangi test teknikleri (gözden geçirme, inceleme vb.) kullanılacak?

v. Planlama, tasarım ve koşturma için kaç tane testçi gerekiyor?

vi. Hangi seviyelerde testler yapılacak?

Strateji bölümü içerisinde belirtilmesi gereken diğer bir konu da kontrol ve gözden geçirmelerdir. Kodun, tasarımın ve gereksinimlerin gözden geçirilmesi doğrulama teknikleri olup yazılım kalitesi açısından önemlidir. Bunlar test eylemleri olmamasına rağmen test eylemleri için tamamlayıcı özellik taşırlar. Bu nedenle test planı içerisindeki strateji bölümünde bunlardan bahsedilmesi gereklidir.

Geçme/Kalma Kriterleri: Test edilen öğelerin geçme kalma kriterleri bu bölümde listelenir. Örneğin test edilen öğenin test sonuçlarında küçük ve görsel kusurların dışında büyük kusur bulunmaması, test edilen öğelerin testten geçme kriteri olarak tanımlanabilir.

Testi Durdurma ve Teste Yeniden Devam Etme Kriterleri: Testler devam ederken testin durdurulması gerekebilir. Bu durdurmanın hangi koşullarda gerçekleştirileceği bu bölümde listelenir. Örneğin testler gerçekleştirilirken test edilen yazılımda bulunan hata yoğunluğunun

belli bir yüzdeye ulaşması, testleri durdurma sebebi olabilir. Durdurulan testlerin hangi koşullar gerçekleştiğinde yeniden devam edeceği de bu başlık altında belirtilir.

Test Çıktıları: Test süreci çıktıları olarak hangi belgelerin üretileceği bu bölümde verilir.

i. Test planları

ii. Test tasarım belgeleri

iii. Emülatör ve simülatörler

iv. Hata logları

v. Problem raporları vb

Proje kapsamında geliştirilen ve test edilen yazılım bir test süreci çıktısı değildir.

Test Görevleri: Test ortamının hazırlanması ve testlerin yapılması için gereken görevlerin tanımlandığı bölümdür.

Çevresel Gereksinimler: Testlerin yapılması için gerekli test ortamının tanımlandığı bölümdür. Bu bölüm içerisinde test ortamının fiziksel özellikleri, gerekli yazılım (simülatörler, emülatörler vb) ve donanım alt yapısı ve iletişim ortamı tanımlanır.

Sorumluluklar: Test yönetimi, test tasarımı, test ortamı hazırlığı, testlerin koşturulması gibi test eylemlerinden kimlerin veya hangi grupların sorumlu olduğu burada verilir.

Eleman ve Eğitim Gereksinimleri: Testlerin başarılı bir şekilde yapılması için gerekli eleman profili ve bu profil için gerekli eğitimler burada verilir.

Takvim: Test sürecini ve test dönüm noktalarını net şekilde gösteren bir proje takvimi bu bölümde verilir.

Planlanan Riskler ve Olasılıkları: Riskler testçilere nelerin test edileceğini öncelik vermeye yardımcı olur. Bu önceliklendirmeyle testçilerin dikkatlerinin bu alanlar üzerine yoğunlaşması amaçlanır. Eğer geliştirilen emniyet kritik bir yazılımsa bu bölüm, gerçekleştirilmesi gereken emniyet ve tehlike analizlerini de kapsar.

Bu toplantı sonucunda hangi yazılımların test edileceği ve bunların kritik alanlarının neler olduğu tanımlanır. Yazılım projelerinde rastlanan risklerden bazıları şunlardır:

i. Hazır alınan ürünlerin zamanında teslim edilmemesi

ii. Arayüz yazılımlarının yeni sürümleri

iii. Yeni paket ve yazılımların anlaşılması ve kullanılma kabiliyeti

iv. Karmaşık işlevler

v. Önceki hatalara göre bileşenlerin üzerindeki modifikasyonlar

vi. Yetersiz belgelendirme ve yetersiz değişiklik istekleri

Yazılım risklerinin diğer bir yönü de orijinal gereksinimlerin yanlış anlaşılmasıdır. Bu yanlış anlaşılma yönetim, kullanıcı veya geliştirici seviyesinde olabilir. Burada görev testçilere düşer. Yanlış anlaşılmaya sebebiyet verebilecek gereksinimlerin tespiti, bunların gerekirse yalın ve açık halde yeniden düzenlenmesi sağlanmalıdır. Bu kapsamda gereksinim gözden geçirmelerinde ve test durumları yazılırken test edilemeyecek gereksinimler belirlenmeli ve gerekli düzeltme veya iptal edilme işlemleri yapılmalıdır.

Onaylar: Geliştirilen planın kim veya kimler tarafından onaylanacağı ve onaylama sürecinin nasıl olacağı bu bölümde verilir.

8.1.2 Test Tasarım Belirtimleri

Test tasarım belirtimleri belgesi, gereksinim ve tasarım belgelerine göre testlerin yapılabilmesi için gerekli test koşullarını tanımlar. Tanımlanan bu test koşullarına test gereksinimleri denir. Bu belgeyle test yaklaşımı, yapılan tasarıma göre test edilecek öğeler ve bu öğelerle ilişkili testler ayrıntılı duruma getirilir. Test tasarım belirtimleri örnek olarak belgesi şu başlıklara göre yazılabilir:

i. Test tasarım belirtimleri tanımlayıcısı

ii. Test edilecek özellikler: Test belirtimleri belgesinin içerdiği test edilecek öğeler listelenir. Bu öğelerin özellikleri verilir. Burada verilen her bir özellik en az bir gereksinimle ilişkilidir.

iii. Stratejinin ayrıntılandırılması: Test planında proje genelinde bahsedilen test stratejisi testlerde kullanılacak test teknikleri, test sonuçlarının analiz metotları gibi bilgiler bu başlıkta verilir.

iv. Test tanımlaması: Tasarımla alakalı her bir test durumunun test tanımı ve tanımlayıcı numarası bu başlık altında verilir.

v. Geçti/Kaldı kriterleri özellikleri: Test edilen özelliğin testlerden geçip kalması ile ilgili kriterler bu başlık altında listelenir.

8.1.3 Test Durumu Belirtimleri

Test durumu, bir gereksinimin karşılanıp karşılanmadığına karar vermek için testçi tarafından yazılan değişkenler ve şartlar grubudur. Test durumları testin en küçük parçasıdır. Test durumları belirtim belgesi, tüm test durumlarının belirli bir düzen içerisinde yazıldığı bir dökümandır.

8.1.4 Test Yordamı Belirtimleri

Bu belge, testçinin testlerin fiziksel olarak nasıl çalıştıracağını, gerekli fiziksel ortamın nasıl hazırlanacağını ve bunlar için takip edilmesi gereken prosedürün anlatıldığı dökümandır.

8.1.5 Test Öğesi Dağıyım/Yayım Raporu

Bir önceki test evresinde ayrıntıların ve bu adım için yayınlanacak test öğelerinin bilgilerini içeren ve her bir öğe için sorumlu kişi, öğenin fiziksel yeri ve durumunu gösteren belgedir.

8.1.6 Test Logları

Koşturulan test durumlarının koşturulma sırasını, ara ve son çıktılarını içeren kayıt dökümanıdır. Test işletimini izlemek, yani ne zaman ne yapıldı, ne oldu sorularına yanıt bulmak için kullanılır. Eğer testlerden beklenen sonuç elde edilmişse “Test Hata Raporu” yazılır. Test kayıtları hataya neyin sebep olduğunun bulunması açısından önemlidir.

8.1.7 Test Hata Raporu

Test süreci sırasında ortaya çıkan beklenmedik olayların raporlanmasıdır. Bu belge, hataları ve bu hataların hangi test durumları koşturulurken, hangi adımlarda ortaya çıktığını içerir. Hangi test loglarıyla ilgili olduğu da belirtilir.

8.1.8 Test Özeti

Yapılan test eylemlerinin ve test sonuçlarının değerlendirildiği belgedir. Genel olarak bu bölüm, yapılan tüm testlere genel bir bakışı ve yazılımın kalitesiyle ilgili ifadeleri kapsar. Test Özeti Belgesi örnek olarak şu başlıklarda yazılabilir:

i. Test Özet Rapor Tanımlayıcısı

ii. Özet: Test edilen öğeler, sürüm numarası ve test ortamı hakkında bilgi verilir.

iii. Farklılıklar: Test planından, test prosedüründen veya tasarımdan sapmalar, farklılıklar burada anlatılır.

iv. Kapsamlı Değerlendirme: Tüm test çalışmalarının genel değerlendirilmesi bu başlıkta verilir.

v. Sonuçların Özetlenmesi: Test çalıştırılması sonucunda elde edilen test sonuçları (geçti/kaldı) burada verilir.

vi. Değerlendirme: Test sonuçlarının test planındaki geçti/kaldı kriterlerine göre değerlendirilmesinin yapıldığı bölümdür.

vii. Eylemlerin Özetlenmesi: Önemli test eylemleri, gerçekleşen olaylar, harcanan toplam adam/ay, toplam test süresi vb bilgiler özetlenir.

viii. Onaylar: Bu raporu onaylayanların isimleri, unvanları belirtilerek imza ve tarih alanı bu bölümde verilir.

Testler yapıldıktan sonra elde edilen test verileri raporlanmalı, analiz edilmeli ve değerlendirilmelidir. Değerlendirmede test planlarında belirtilen test geçti/kaldı kriterleri dikkate alınır. Test Özet Belgesi içerisinde test sonuçları şu bilgileri kapsayacak şekilde özetlenebilir:

i. **Modül/Birim:** Yazılımın hangi modülünün/biriminin test edildiği yazılır.

ii. **İhtiyaç Numarası:** Test durumunun doğruladığı gereksinim numarası yazılır.

iii. **Test Durumu Numarası:** Belirtilen modül üzerinde koşturulan test durumunun numarası yazılır. Böylece test sonucunun hangi test durumunun sonucu olduğu belirtilmiş olur.

iv. **Test Sonucu Numarası:** Test sonucunu diğerlerinden ayıracak tanımlayıcı kimlik yazılır.

v. **Test Sonucu:** Her bir testin sonucu geçti/kaldı olarak verilir.

vi. **Test Tarihi:** Testin yapıldığı tarih yazılır.

vii. **Testi Yapan:** Testi yapıp sonucu raporlayan kişinin adı yazılır.

viii. **Testi İzleyen:** Müşteri tarafından testi izleyen varsa onun adı yazılır.

ix. **Tekrar Sayısı:** Test durumunun kaçıncı kez koşturulmuş olduğu yazılır. Böylece koşturulan her test durumunun tüm sonuçları kayıt altına alınarak hangi sürümlerde testlerin başarısızlıkla sonuçlandığı gibi bilgiler ölçülebilir.

x. **Hata Bildirimi Numarası:** Test durumunun başarısızlıkla sonuçlandığı durumlarda yazılan hata bildirim formu numarası yazılır.

xi. **Yazılım Sürümü:** Testin yapıldığı yazılım sürüm numarası yazılır.

xii. **Yardımcı Yazılımların Sürümü:** Kullanılan yardımcı yazılım (emülatör, simülatör vb) varsa buraya sürüm numaraları yazılır.

xiii. **Donanım Sürümü:** Testin yapıldığı kullanılan donanım sürümü numarası yazılır.

Test Özet Belgesi içerisinde test sonuçları analiz edilip değerlendirilirken kaç adet ihtiyacın, kaç adet test durumuyla kaç kere test edilip doğrulandığı verilir. Ortaya çıkan hatalar belirtilerek bu hatanın hangi ihtiyaçla ilgili olduğu, neden gerçeklenemediği, bu hatayla ilgili çözüm önerileri verilir.

9 TEST YAZILIM ARAÇLARI VE TEST OTOMASYONU

İyi organize edilmiş bir test süreci, doğru test yazılımlarıyla desteklendiğinde daha da başarılı sonuçlar verecektir. Test yazılımları test planlama, kontrol, tasarım, test verisi üretimi, test koşturumu veya değerlendirilmesi gibi test eylemlerini desteklemek amacıyla geliştirilmiş olan yardımcı programlardır. Test süreci içerisindeki her test adımı için farklı yazılımlar üretilmiştir. Bu kapsamda test yazılımlarının daha iyi değerlendirilebilmesi için yazılımlar şöyle sınıflandırılabilir (softsmith):

Test Planlama, Kontrol ve Raporlama Yazılımları

i. Test Yönetim Araçları

ii. Hata Yönetim Araçları

iii. Ayarlama Yönetim Araçları

Test Hazırlık Yazılımları

i. Karmaşıklık Analiz Araçları

ii. Durum Analiz Araçları

iii. Test Durumu Üreteçleri

Test Çalıştırma Yazılımları

i. Test Verisi Üreteçleri

ii. Kayıt ve Yeniden Oynatım Araçları

iii. Yükleme ve Stres Yazılımları

iv. Simülatör ve Emülatörler

v. Koçan ve Sürücüler

vi. Hata Ayıklayıcılar ve Hata Yakalayıcılar

vii. Statik Kod Analiz Araçları

viii. Kod Kapsama Analiz Araçları

9.1 Test Planlama, Kontrol ve Raporlama Yazılımları

Test planlamasında kullanılan yazılımlar genellikle proje yönetim yazılımları içerisinde bu yazılımların ihtiyaçları temel alınarak geliştirilmiştir. Ancak test eylemlerinin yazılım projelerindeki önemi anlaşıldıkça bu konuya özel, yazılım testleri planlama, kontrol ve raporlama araçları da piyasaya sürülmüştür. Bu kapsamdaki yazılım araçları üç sınıf altında incelenebilir (softsmith):

Test Yönetim Araçları: Diğer tüm yazılım geliştirme süreçlerine paralel bir yapıda işleyen ve bağımsız ekipler tarafından gerçekleştirildiğinde etkinliği artan test eylemlerinin, kendi içerisinde daha detaylı takip edilmesi, test sürecinin projeye katkısını arttıracak ve istenen hedefe varılmasına yardımcı olacaktır. Bu amaçla geliştirilen test yönetim araçlarında test sürecinin planlaması yapılır. Eldeki kaynakların test süreci içerisinde nasıl ve ne zaman kullanılacağı, takvime göre ne zaman hangi test eylemlerinin gerçekleştirilip hangi çıktıların üretileceği bu araçlar yardımıyla planlanır ve gerçekleştirilen eylemler bu araçlardan elde edilen raporlarla belgelenir.

Hata Yönetim Araçları: Testler sırasında bulunan hataların raporlanması ve bu hataların düzeltilip yeniden test edilerek kapatılması süresince hatanın yaşam çevriminin izlenebileceği yardımcı yazılımlardır. Küçük çaptaki yazılım projelerinde hata yönetimi daha basit şekilde bir elektronik tablo dosyası üzerinden takip edilebilir. Ancak projenin büyüklüğü, geliştirilen yazılımın karmaşıklığı ve yazılımın kritikliği arttıkça, projede ortaya çıkan hataların takip edilmesinin de önemi artar. Bu nedenle projelerde bu hataların basit bir dosya üzerinden takibinden daha profesyonelce takip edileceği yapıların kurulmasına ihtiyaç doğar. Bu amaçla farklı yazılım firmaları, farklı ölçeklerde hata yönetim araçları geliştirmiştir. Bu araçlar aynı zamanda test yönetim ve proje yönetim araçlarıyla da entegre şekilde çalışır hale getirilerek etkinlikleri arttırılmıştır.

Konfigürasyon/Ayarlama Yönetim Araçları: Tüm test eylemleri belirlenen bir şekilde kayıt altına alınmalıdır. Test planları, test tanımlama belgeleri, test sonuç raporları, test ortamı tanımlama belgeleri gibi test süreci çıktıları proje ekibince belirlenen bir yapıda kalite kontrolleri yapıldıktan sonra resmi hale getirilmeli ve kayıt altında tutulmalıdır. Bunun için test yöneticisi tarafından test süreci içerisinde üretilecek ayarlama öğelerini belirlemeli ve

ayarlama kütüphanesinde kaydetmelidir. Test faaliyetleri için test planları, test durumları tanımlama belgesi, test sonuç raporları, test ortamı tanımlama belgeleri, kullanılan test yardımcı yazılımları, testlerde kullanılmak üzere geliştirilen koçanlar, sürücüler, simülatörler ve emülatörler test ayarlama öğeleri olarak belirlenerek ve ayarlama kütüphanesi içerisinde yerleştirilerek değişiklikleri kontrol altında tutulur. Bu yaklaşımdan hareketle test planlaması ve yönetiminde ayarlama yönetim araçları da test yardımcı araçları arasında yerini alır.

Test yönetim araçları test yöneticilerinin işlerini kolaylaştırmak, test yönetim sürecinin en etkin şekilde işletilmesini sağlamak, aksaklıkları zamanında tespit ederek gerekli düzeltme ve önlemleri yapmak için geliştirilen yardımcı yazılımlardır.

9.2 Test Hazırlık Yazılımları

Bu kapsamdaki yazılımlar testlerin hazırlık, tasarım aşamasında test yöneticilerine ve test mühendislerine yardımcı olurlar (softsmith).

Karmaşıklık Analiz Araçları: Bu kapsamda değerlendirilecek olan yazılımlar geliştirilecek yazılımın karmaşıklığı hakkında bilgi vermektedir. Buradan elde edilen bilgilerle yazılım için yazılması gereken test durumu sayısıyla yazılımdan beklenen toplam hata sayısı hakkında bilgi elde edilebilir. Bu bilgiler testlerin tamanlanma kriterlerinde kullanılabilir. Bu tür araçlar, test sürecinin etkinliğini arttırır.

Test Tasarım Araçları: Test durumlarının yazılması emek isteyen yoğun bir iştir. Özellikle birim testlerde ve modül testlerinde çok fazla sayıda test durumuna ve bunların tüm girdi ve çıktılarının ayrı ayrı tanımlanılmasına ihtiyaç vardır. Bu amaçla test süreci içerisinde yardımcı yazılım kullanılması gerekir. Test tasarım araçları genellikle matematik veya model tabanlı bir dil yaklaşımıyla test durumlarının üretilmesini sağlar. Örneğin bu amaç için geliştirilmiş "Z" dili vardır. Model tabanlı test tasarım araçları UML'deki kullanım durumlarından (use case) test durumlarının üretir.

Test tasarım araçlarının bir diğer şekli de test durumu yönetim araçlarıdır. Bu araçlar test durumlarının organize edilmesinde ve çalıştırılabilir gruplar haline getirilmesinde test mühendislerine yardımcı olur.

9.3 Test Koşturma Yazılımları

Test koşturma safhası kapsamında şu yazılımlar değerlendirilebilir:

Birim Test Araçları: Birim testlerinin yapılmasında kullanılan yazılımlardır. Bu yazılımlar geliştirme ortamlarıyla entegre çalışan yapılarda geliştirilmişlerdir.

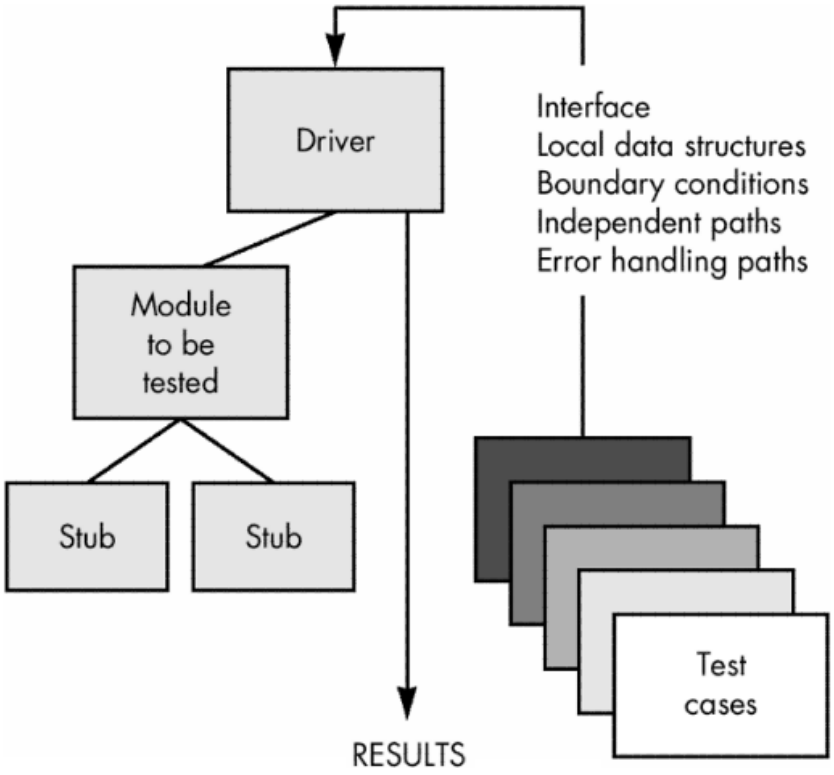
İşlevsel Test Araçları: Geliştirilen yazılımların kendilerinden beklenen işlevleri yerine getirdiğini test etmek amacıyla kullanılan yazılımlardır. Bu testler manuel olarak yapılabileceği gibi test araçlarıyla da yapılabilir.

Simülatör ve Emülatörler: Gerçek şartların oluşturulamadığı veya testler sırasında mal ve can kaybına neden olabilecek emniyet ve görev kritik yazılımların testlerinde simülatör veya emülatörler kullanılır. Simülatörler geliştirilen yazılımlara, gerçek şartlardaki olayları benzeterek verilmesi gereken girdileri gerçek zamanlı olarak sağlar. böylece yazılımın bu şartlar altında nasıl davrandığı test edilmiş olur.

Emülatörler bir cihazın çalışmasını taklit eden programlardır. Özellikle bir cihazı kontrol edecek bir yazılımı test etmek için doğrudan cihaza bağlamak yerine cihazın emülatörüne bağlayarak yazılımın çalışmasının test edilmesi için geliştirilir. Buradaki en temel amaç, yazılımdaki herhangi bir hatadan dolayı cihaza zarar gelmesini engellemektir. Emülatörlerin diğer bir kullanım amacı da test ortamına getirelemeyen cihazların sanki test ortamındaymiş gibi gerekli testlerin yapılmasıdır.

Koçan (Stub) ve Sürücüler: Özellikle birim testler sırasında iki yazılım parçası arasındaki arayüz ancak her iki parçada da hazırsa test edilebilir. Ancak parçalardan birinin hazır olmadığı durumlarda aksaklığa uğramadan testlere devam edilmesinin yolu testlerde diğer yazılım parçasının arayüz davranışını sergileyecek olan koçanların ve bu parçaları çağıracak sürücülerin kullanılmasıdır.

Sürücü ve koçan kullanımı, Şekil 9.1.'de gösterilmektedir (csis.pace.edu).



Şekil 9.1 Test ortamında sürücü ve koçan kullanımı

Sürücüler ayrıca görselliği olamayan yazılımların işlevsel testlerinde de kullanılır.

Hata Ayıklayıcılar (Debugger) ve Hata Yakalayacılar: Hata ayıklayıcılar, yazılımlar koşturulmadan önce program üzerinde sözdizimsel hataların olup olmadığını tespit eden ve geliştiriciyi uyaran yardımcı programlardır. Aynı zamanda bu programlarla yazılımlar gerçek çalışma ortamlarında adım adım çalıştırılarak yazılım içerisindeki değişkenlerin durumu izlenebilir. Değişkenlere müdahele ederek hatalı değer almaları sağlanabilir. Hatalı değer sonucu yazılımın nasıl davranacağı görülebilir.

Hata yakalayıcılar ise sadece yazılımların gerçek çalışma ortamlarındaki çalışma zamanlarında faal olup yazılımın üzerinde meydana gelen bellek taşması, karşılaştırma hatası, yanlış boyutta değişken kullanılması gibi hataları tespit ve teşhis eder. Yani sadece hatayı değil, hatanın oluşmasına neden olan gerçek sebebin ne olabileceğine dair bilgi verirler. Bu da hatanın nereden oluştuğunun teşhisine dair geliştiricilere ipucu verir.

Statik Kod Analiz Araçları: Yazılım çalıştırılmadan önce yazılım içerisindeki hataları ortaya çıkarmak için statik kod analizlerini gerçekleştirmek amacıyla kullanılan yazılımlardır. Statik kod analiz araçlarıyla yazılımın test ortamında testleri gerçekleştirilmeden bazı hataların statik kod analizleriyle bulunması sağlanır.

Kod Kapsama Analiz Araçları: Bu araçlar, koşturulan test durumlarıyla yazılım içerisindeki kod parçacıklarının ne kadarının çalıştırıldığını izler ve raporlar. Böylece testlerde yazılan tüm kodların ne kadarının koşturulduğu metrik olarak elde edilir. Koşturulmayan kod parçacıklarının niçin koşturulamadığı tespit edilerek ya yazılımın içerisinden bu kod parçaları çıkartılır ya da bu kod parçacıklarının da test edilmesi için gerekli test durumları oluşturulur. Özellikle birim testlerde kod kapsamanın %100 olması önemlidir.

Test koşturma yazılımları kapsamında kullanılabilecek yardımcı yazılımlar bunlarla sınırlı değildir. Bu kapsamda veri tabanı test yazılımları, performans analiz yazılımları, thread ve olay analiz yazılımları, thread hata yakalama yazılımları gibi pek çok yardımcı, yazılım testlerinin etkinliğini arttırmak için kullanılır.

9.4 Test Otomasyonu

Test otomasyonu, testlerde harcanan ve sürekli yinelenen işleri en aza indirmek için kullanılan bir stratejidir. Otomasyon zaman, para ve kalite açısından yararlıdır. Aynı zamanda yoğun ve tekrarlanan test eylemlerinin testçiler üzerindeki psikolojik baskıyı da ortadan kaldırır. Teorik olarak her test otomatikleştirilebilir. Ama pratiğe bakıldığında bazı testlerin otomatikleştirilebildiği, bazılarınınsa yapılamadığı görülür. Bunun sebebi otomatikleştirmenin beraberinde getirmiş olduğu maliyet, zaman ve bakım gereksinimleridir *[Broekman - 2003]*. Test otomasyonu şu durumlarda kullanılabilir:

i. Testlerin birden fazla yinelenmesi durumunda

ii. Basit bir testin fazla sayıda aynı tipte farklı değerde girdiyle test edilmesi gereken durumlarda

iii. Girdi ve çıktı veri kümelerinin net belli olduğu durumlarda

Testlerin otomatikleştirildiği durumların sayısı arttırılabilir. Ama tüm testlerin otomatikleştirilip otomatikleştirilmeyeceği belirsizdir; çünkü bazen otomatikleştirme için ayrılan zaman ve para, testlerin yeniden tekrarlanarak yapılmasında gereken zaman ve paradan daha fazla olabilmektedir. Otomatikleştirilmiş testlerle bir sürüm üzerinde bir kaç kez

aynı test durumları koşturulabilir. Sürümlerin sayısı arttıkça otomatik olarak koşturulmak üzere tasarlanmış test durumlarının güncellenmesi ve yeni sürüme göre çalışır hale getirilmesi gerekecektir. Test yönetimi, stratejik açıdan otomatik yapıların bakım ve sürdürülebilirliğiyle testleri manuel olarak yapmanın zamanı ve maliyeti açısından bir değerlendirme yaparak test stratejisini belirlemelidir.

Yazılım projelerinde test otomasyonu ve bu otomasyonda kullanılabilecek test araçlarına karar verileceği durumlarda şu soruların cevapları net olarak ortaya konulmalı ve buna göre test otomasyonu karara bağlanmalıdır:

i. Test sürecinin otomatikleştirilmesiyle ne elde edilmek isteniliyor? Test zamanının kısaltılması mı yoksa maliyet açısından bir azalma mı bekleniyor?

ii. Karar vermekte en önemli kriterler ne olacak?

iii. Yazılımın geliştirildiği alanda kullanılabilecek hangi test otomasyonu yazılımları var?

iv. Amaç ve kriterleri hangi test otomasyon araçları karşılamakta?

v. Seçilen aracın ve bu araç üzerinde geliştirilecek test otomasyonunun maliyeti ve kazancı ne kadar olacak?

Test otomasyonunda otomasyon kararının verilmesinin yanında diğer bir önemli konu da bu alanda kullanılacak yazılımların seçilmesidir. Test otomasyon alanı içerisinde geliştirilen yazılımlar, kullanıcılarına genel olarak test betiği (script) oluşturma, bunları koşturma ve bakım alanlarında hizmet vermektedir.

Test betiklerinin oluşturulması test otomasyonunda önemli bir adımdır. Bu özellik, kullanılan yazılıma göre farklılık göstermesine rağmen betikler genellikle manuel kodlamayla, var olan betiklerin değiştirilmesiyle veya test edilecek yazılımın kullanılıp kaydedilmesiyle yapılabilir. Test otomasyonu kapsamında kullanılan araçlar genel olarak şöyle sınıflandırılabilir:

Test Durumu Üreticileri: Kaynak kod analiz edilerek test edilecek modül için olası girdiler ve çıktılar otomatik olarak tespit edilmekte ve test otomasyonunda kullanılacak test betiğinin oluşturulması için kullanıcıdan sadece girdi ve beklenen çıktının parametre olarak verilmesi beklenmektedir. Bazı durumlarda girdi kümesi de otomatik üretilmekte, kullanıcıdan sadece çıktı değerler kümesi beklenmektedir. Daha sonra üretilen bu test durumları bir kütüphane

içerisinde saklanmakta ve yazılımın yeni sürümü çıktığında oluşturulan bu test durumu betik kütüphanesi, yazılım üzerinde otomatik koşturularak sonuçları metin ve görsel olarak raporlanır.

Kayıt ve Yeniden Oynatım Araçları: Yineleme testlerinde oldukça yararlı olan yardımcı test yazılımlarıdır. Bu yazılımlarla testler yapılırken yazılımın kullanım şekli ve bu testlerden beklenen sonuçlar kaydedilir. Daha sonra yazılım geliştirildiğinde ya da değiştirildiğinde yazılım üzerinde yapılması gereken yineleme testleri için bu yazılımlar, geliştirilen yazılımı daha önceki kayıtlara uygun olarak test ederler. Tabii bu testler yazılımın değiştirilmeyen ya da eskiden var olan mdülleri üzerinde gerçekleştirilir. Böylece test mühendisleri yazılımın sadece değişen veya yeni geliştirilen bölümüne odaklanarak testlerde tekrar gerektiren iş yükünden kurtulurlar.

Test Verisi Üreteçleri: Otomatik test çalıştırması sırasında gerekli olan girdi, veri havuzlarının üretilmesi amacıyla geliştirilen ve kullanılan yazılımlardır. Bu yazılımlar genellikle alan bağımlı çalışırlar.

Yükleme ve Stres Yazılımları: Geliştirilen yazılımların, özellikle web uygulamalarının aşırı yük altında davranışlarını gözlemlemek, güvenilirliklerine karar vermek amaçlı kullanılan yazılımlardır. Böylece örneğin bir e-ticaret sitesine aynı anda binlerce müşteri bağlanmış gibi yüklenilerek test edilebilir ve yazılımın performansı ve davranışı aşırı yük altında sınanabilir.

10. PLAGUE INC EVOLVED OYUNUNUN TEST SÜRECİ

Burada Plague Inc Evolved adlı oyunun son versiyonunun (Evrim 11) kullanıcı arayüzü testi yapılmıştır.

En başta IOS işletim sistemine geliştirilmiş olan Plague Inc adlı oyun, gösterilen yoğun ilgiden sonra Android işletim sistemine de çıkmış, en son olarak da Windows işletim sistemli bilgisayarlara, grafik ve oyun motoru çok daha geliştirilmiş olarak Plague Inc Evolved adıyla yayınlanmıştır.

Bu oyunda özet olarak bakteri, virüs, mantar, parazit, prion, nano virüs, biyolojik silah gibi gerçek hayatta da olan hastalıklardan veya nöraks solucanı, nekroa virüsü, maymun gribi gibi bilimkurgusal hastalıklardan biri seçilerek başlanır. Seçilen bu hastalık türüne, oyunu bitrerek açılan genlerden seçilenler eklenir ve oyuna girilir. Daha sonra hastalığın başlayacağı ülke seçildikten sonra oyun sırasında çeşitli şekillerde kazanılan DNA puanları sayesinde hastalık çeşitli şekillerde geliştirilerek tüm dünyaya yayılıp bütün insanlığı yoketmeye çalışılır. Oyun defalarca bitirildikçe her seferinde açılan yeni hastalık, gen, senaryo gibi özelliklerle oyun sürekliliği sağlanır.

Açıklaması bu şekilde olan Plague Inc Evolved adlı oyunun tam sürümü çıkmadan önceki varolan en son sürümünün kullanıcı arayüzü testleri ve test durumları, hata, düzeltmeler ve her sürümde eklenen yeni özellikler aşağıda anlatılmıştır. Daha sonra oyunun çıkan sürümlerinde bazı özellikler değişiklik gösterebilir.

10.1 Test Ortamı

Testlerin yapıldığı bilgisayarın sistem özellikleri şöyledir:

i. İşletim Sistemi: Windows 7 Ultimate 64bit, Service Pack 1

ii. Directx Versiyonu: 11

iii. CPU: HexaCore AMD Phenom II X6 Black Edition 1090T, 3431 MHz

iv. Anakart: AMD 770

v. RAM: 2 x Kingston 9905403-400.A00LF 4 GB DDR3-1333 DDR3 SDRAM

vi. Ekran Kartı: Nvidia GeForce GTX 560

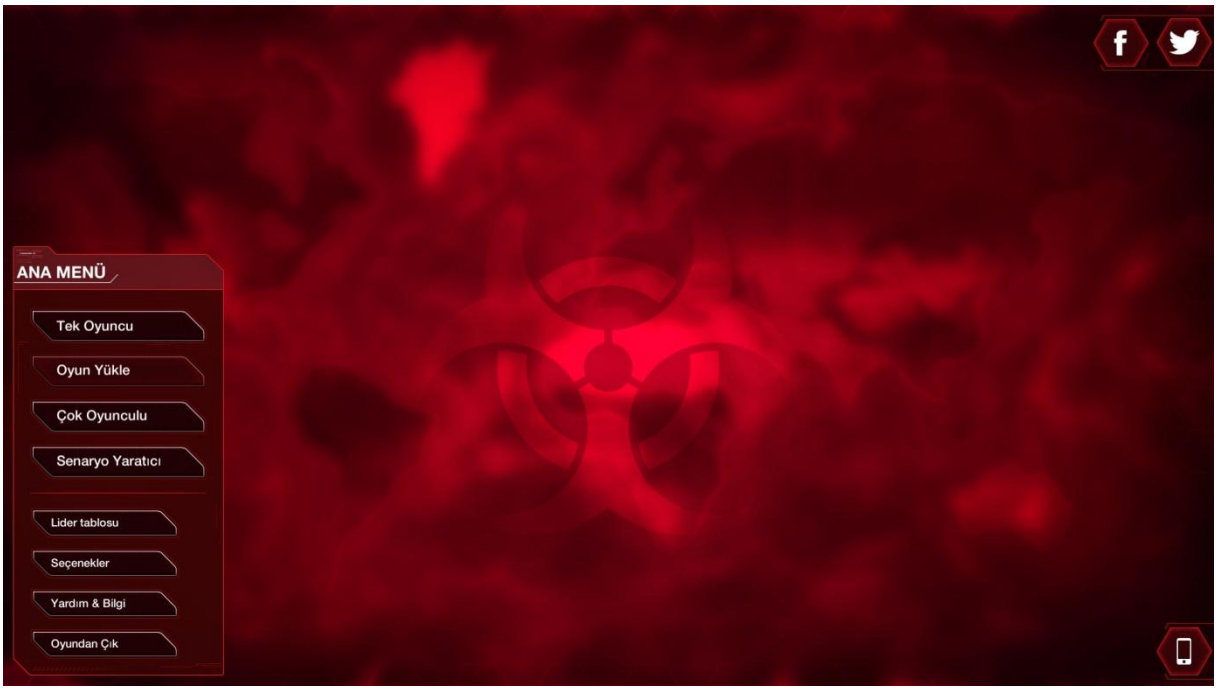
vii. Ses Bağdaştırıcısı: VIA VT1818S @ ATI SB800 - High Definition Audio Controller

viii. Hard Disk: ST500DM002-1BD142 ATA Device (465 GB, IDE) ve Hitachi HDT725025VLA380 ATA Device (250 GB, 7200 RPM, SATA-II)

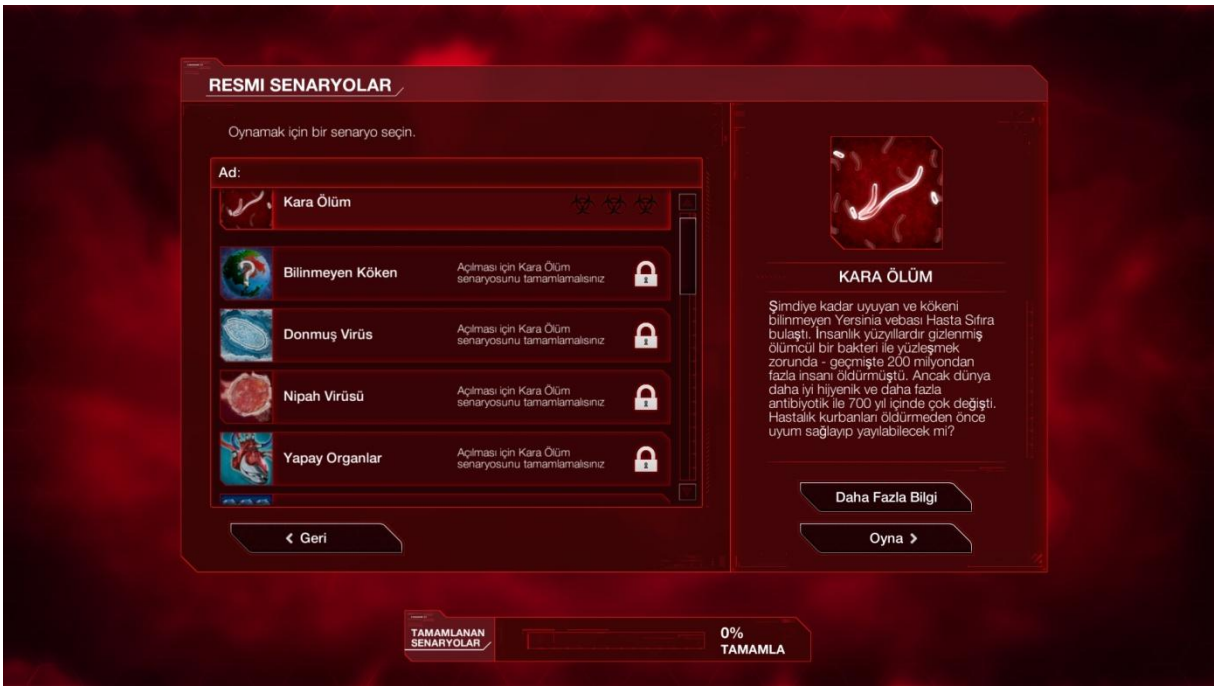
ix. Klavye ve Mouse: Standard PS/2 Keyboard, HID-compliant mouse

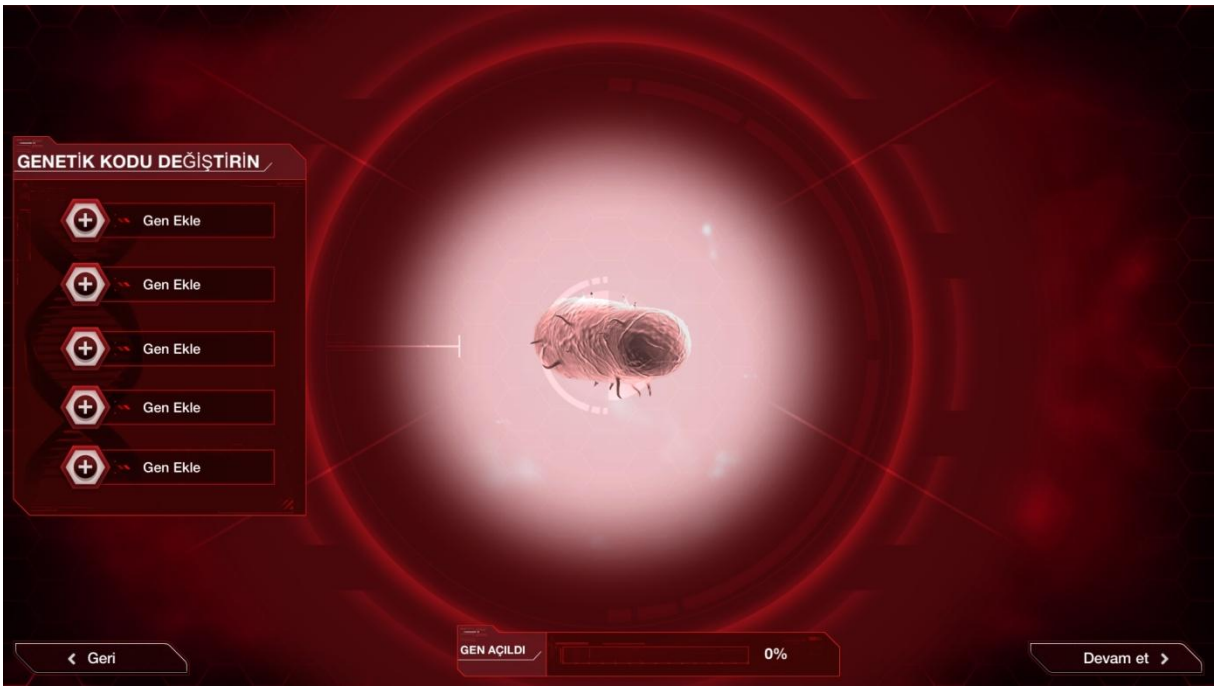
10.2 Test Durumları

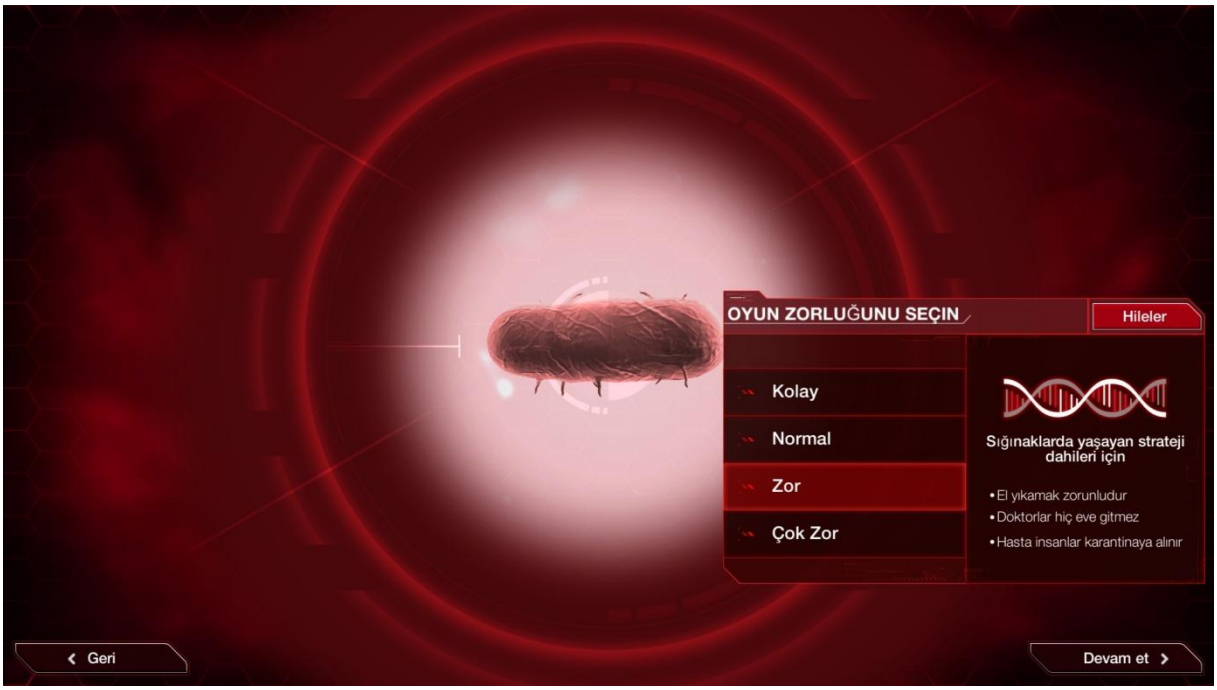
Plague Inc Evolved oyununun kullanıcı arayüzü testleri için test durumları aşağıda verilmiştir. Test durumlarının çokluğu, projenin süresini de etkileyeceği için aşırı sayıda test durumu yazılmasından kaçınılmıştır. Bu yüzden test durumları belli yerlerde kısa kesilmiştir. Oyunun kullanıcı arayüzü testlerinin yapıldığı bazı ekranlar Şekil 10.1'de gösterilmektedir.


ANA MENÜ
Tek Oyuncu
Oyun Yükle
Çok Oyunculu
Senaryo Yaratıcı
Lider tablosu
Seçenekler
Yardım & Bilgi
Oyundan Çık


TEK OYUNCU
Ana Oyun
Resmi Senaryolar
Özel Senaryolar
Süre Yarışı
Başlangıç Rehberi
Geri
ANA OYUN
Plague Inc. oyunu - Dünyaya bir salgın yayabilir misin?
Oyna


RESMI SENARYOLAR
Oynamak için bir senaryo seçin.
Ad:
Kara Ölüm
Bilinmeyen Köken
Açılması için Kara Ölüm senaryosunu tamamlamalısınız
Donmuş Virüs
Açılması için Kara Ölüm senaryosunu tamamlamalısınız
Nipah Virüsü
Açılması için Kara Ölüm senaryosunu tamamlamalısınız
Yapay Organlar
Açılması için Kara Ölüm senaryosunu tamamlamalısınız
KARA ÖLÜM
Şimdiye kadar uyuyan ve kökeni bilinmeyen Yersinia vebası Hasta Sıfıra bulaştı. İnsanlık yüzyıllardır gizlenmiş ölümcül bir bakteri ile yüzleşmek zorunda - geçmişte 200 milyondan fazla insanı öldürmüştü. Ancak dünya daha iyi hijyenik ve daha fazla antibiyotik ile 700 yıl içinde çok değişti. Hastalık kurbanları öldürmeden önce uyum sağlayıp yayılabilecek mi?
Daha Fazla Bilgi
Oyna
Geri
TAMAMLANAN SENARYOLAR
0%
TAMAMLA


GENETİK KODU DEĞİŞTİRİN
Gen Ekle
Gen Ekle
Gen Ekle
Gen Ekle
Gen Ekle
Geri
GEN AÇILDI
0%
Devam et



Şekil 10.1 Kullanıcı arayüzü testi yapılan ekran örnekleri

Test Durumu 1

Testin Görevi: Oyuna girişteki intro filminin düzgün olduğunu test et.

Test Açıklaması: Oyuna girişte başlangıçta oynatılan intro filmindeki seslerin, görüntülerin düzgün çalıştığına, filmin bitişinde oyun ama menüsüne girdiğine ve herhangi bir tuşa basıldığında intro'nun geçildiğine emin olmak.

Beklenen Sonuç: Oyuna girişte hiç bir görsel ve işitsel sorunun olmaması, herhangi bir tuşa basıldığında gecikme olmadan ana menüye geçmesi.

Test Durumu 2

Testin Görevi: Oyun ana menüsünde arka planda çalan müziğin doğruluğunu test et.

Test Açıklaması: Oyun ana menüsünde arka plandaki müziğin doğru müzik olduğunu, "Seçenekler" bölümündeki ayarlara göre doğru çalıştığının ve herhangi bir ses bozukluğunun olup olmadığının kontrolünün yapılması.

Beklenen Sonuç: Müziğin ana menü müziği olması, ses bozukluğunun olmaması ve "Seçenekler" bölümündeki ses ayarlarına göre doğru çalması.

Test Durumu 3

Testin Görevi: Ana menüdeki "Tek Oyuncu" butonunun test edilmesi.

Test Açıklaması: Ana menüdeki "Tek Oyuncu" butonunun üstüne gelinir ve herhangi bir doku ve animasyon sorununun olup olmadığı kontrol edilir. Daha sonra butona basılarak doğru pencereye geçiş yapıp yapmadığı kontrol edilir.

Beklenen Sonuç: "Tek Oyuncu" butonunda herhangi bir doku ve animasyon sorununun çıkmaması, butona basıldığında doğru pencereye geçmesi.

Test Durumu 4

Testin Görevi: "Tek Oyuncu" butonuna basıldıktan sonra çıkan pencerede "Resmi Senaryolar" butonunun test edilmesi.

Test Açıklaması: Ana menüdeki "Resmi Senaryolar" butonunun üstüne gelinir ve herhangi bir doku ve animasyon sorununun olup olmadığı kontrol edilir. Daha sonra butona basılarak doğru pencereye geçiş yapıp yapmadığı kontrol edilir.

Beklenen Sonuç: "Resmi Senaryolar" butonunda herhangi bir doku ve animasyon sorununun çıkmaması, butona basıldığında doğru pencereye geçmesi.

Test Durumu 5

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Kara Ölüm" senaryosunun test edilmesi.

Test Açıklaması: "Resmi Senaryolar" kısmındaki seçilebilerek oyun senaryolarından "Kara Ölüm" senaryosunun açık olduğunun kontol edilmesi, sağdaki açıklamasının doğru olup olmadığının kontrol edilmesi ve "Daha Fazla Bilgi" butonuna basıldığında doğru internet sayfasına gidip gitmediği kontrol edilir.

Beklenen Sonuç: "Kara Ölüm" senaryosunun açık olması, açıklamasının doğru olması ve "Daha Fazla Bilgi" butonuna basıldığında doğru internet sayfasına gitmesi.

Test Durumu 6

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Bilinmeyen Köken" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Bilinmeyen Köken" senaryosunun açık olup olmadığı, sağdaki açıklamasının doğru olup olmadığı kontrol edilir.

Beklenen Sonuç: "Bilinmeyen Köken" senaryosunun açık olması ve açıklamasının doğru olması.

Test Durumu 7

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Donmuş Virüs" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Donmuş Virüs" senaryosunun açık olup olmadığı, sağdaki açıklamasının doğru olup olmadığı ve "Daha Fazla Bilgi" butonuna basıldığında doğru internet sayfasına gidip gitmediği kontrol edilir.

Beklenen Sonuç: "Donmuş Virüs" senaryosunun açık olması, açıklamasının doğru olması ve "Daha Fazla Bilgi" butonuna basıldığında doğru internet sayfasına gitmesi.

Test Durumu 8

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Nipah Virüsü" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Nipah Virüsü" senaryosunun açık olup olmadığı, sağdaki açıklamasının doğru olup olmadığı ve "Daha Fazla Bilgi" butonuna basıldığında doğru internet sayfasına gidip gitmediği kontrol edilir.

Beklenen Sonuç: "Nipah Virüsü" senaryosunun açık olması, açıklamasının doğru olması ve "Daha Fazla Bilgi" butonuna basıldığında doğru internet sayfasına gitmesi.

Test Durumu 9

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Yapay Organlar" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Yapay Organlar" senaryosunun açık olup olmadığı, sağdaki açıklamasının doğru olup olmadığı kontrol edilir.

Beklenen Sonuç: "Yapay Organlar" senaryosunun açık olması ve açıklamasının doğru olması.

Test Durumu 10

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Altın Çağ" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Altın Çağ" senaryosunun açık olup olmadığı, sağdaki açıklamasının doğru olup olmadığı kontrol edilir.

Beklenen Sonuç: "Altın Çağ" senaryosunun açık olması ve açıklamasının doğru olması.

Test Durumu 11

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Buz Çağı" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Buz Çağı" senaryosunun açık olup olmadığı, sağdaki açıklamasının doğru olup olmadığı kontrol edilir.

Beklenen Sonuç: "Buz Çağı" senaryosunun açık olması ve açıklamasının doğru olması.

Test Durumu 12

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Çiçek Hastalığı" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Çiçek Hastalığı" senaryosunun açık olup olmadığı, sağdaki açıklamasının doğru olup olmadığı ve "Daha Fazla Bilgi" butonuna basıldığında doğru internet sayfasına gidip gitmediği kontrol edilir.

Beklenen Sonuç: "Çiçek Hastalığı" senaryosunun açık olması, açıklamasının doğru olması ve "Daha Fazla Bilgi" butonuna basıldığında doğru internet sayfasına gitmesi.

Test Durumu 13

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Domuz Gribi" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Domuz Gribi" senaryosunun açık olup olmadığı, sağdaki açıklamasının doğru olup olmadığı ve "Daha Fazla Bilgi" butonuna basıldığında doğru internet sayfasına gidip gitmediği kontrol edilir.

Beklenen Sonuç: "Domuz Gribi" senaryosunun açık olması, açıklamasının doğru olması ve "Daha Fazla Bilgi" butonuna basıldığında doğru internet sayfasına gitmesi.

Test Durumu 14

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Eşit Yaratılma" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Eşit Yaratılma" senaryosunun açık olup olmadığı ve sağdaki açıklamasının doğru olup olmadığı kontrol edilir.

Beklenen Sonuç: "Eşit Yaratılma" senaryosunun açık olması ve açıklamasının doğru olması.

Test Durumu 15

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Her Şeyi Kapat" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Her Şeyi Kapat" senaryosunun açık olup olmadığı ve sağdaki açıklamasının doğru olup olmadığı kontrol edilir.

Beklenen Sonuç: "Her Şeyi Kapat" senaryosunun açık olması ve açıklamasının doğru olması.

Test Durumu 16

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Kimin Umrunda" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Kimin Umrunda" senaryosunun açık olup olmadığı ve sağdaki açıklamasının doğru olup olmadığı kontrol edilir.

Beklenen Sonuç: "Kimin Umrunda" senaryosunun açık olması ve açıklamasının doğru olması.

Test Durumu 17

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Korsan Salgını" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Korsan Salgını" senaryosunun açık olup olmadığı ve sağdaki açıklamasının doğru olup olmadığı kontrol edilir.

Beklenen Sonuç: "Korsan Salgını" senaryosunun açık olması ve açıklamasının doğru olması.

Test Durumu 18

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Küresel Isınma" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Küresel Isınma" senaryosunun açık olup olmadığı ve sağdaki açıklamasının doğru olup olmadığı kontrol edilir.

Beklenen Sonuç: "Küresel Isınma" senaryosunun açık olması ve açıklamasının doğru olması.

Test Durumu 19

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Ters Dünya" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Ters Dünya" senaryosunun açık olup olmadığı ve sağdaki açıklamasının doğru olup olmadığı kontrol edilir.

Beklenen Sonuç: "Ters Dünya" senaryosunun açık olması ve açıklamasının doğru olması.

Test Durumu 20

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Varsayılan Egemen" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Varsayılan Egemen" senaryosunun açık olup olmadığı ve sağdaki açıklamasının doğru olup olmadığı kontrol edilir.

Beklenen Sonuç: "Varsayılan Egemen" senaryosunun açık olması ve açıklamasının doğru olması.

Test Durumu 21

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Volkanik Kül" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Volkanik Kül" senaryosunun açık olup olmadığı ve sağdaki açıklamasının doğru olup olmadığı kontrol edilir.

Beklenen Sonuç: "Volkanik Kül" senaryosunun açık olması ve açıklamasının doğru olması.

Test Durumu 22

Testin Görevi: "Resmi Senaryolar" butonu seçilip daha sonra "Oyna" butonuna basıldığında çıkan pencerede "Yabancı Karşıtlığı" senaryosunun test edilmesi.

Test Açıklaması: Oyun, Kara Ölüm senaryosuyla bir kere bitirildikten sonra tekrar aynı pencereye gelinir ve "Yabancı Karşıtlığı" senaryosunun açık olup olmadığı ve sağdaki açıklamasının doğru olup olmadığı kontrol edilir.

Beklenen Sonuç: "Yabancı Karşıtlığı" senaryosunun açık olması ve açıklamasının doğru olması.

Test Durumu 23

Testin Görevi: "Özel Senaryolar" butonu seçildikten sonra ekrana gelen pencerenin test edilmesi.

Test Açıklaması: "Özel Senaryolar" kısmına girildikten sonra oyunu oynayan diğer kullanıcıların Senaryo Yaratıcısı tarafından yarattıkları özel senaryoların oyun sunucusu vasıtasıyla getirilip getirilmediği ve bunlara sorunsuzca girilip girilmediği kontrol edilir.

Beklenen Sonuç: "Özel Senaryolar"kısmındaki diğer kullanıcıların yarattıkları kendi senaryolarının oyun sunucusu tarafından getirilmesi ve bunlara sorunsuz girilmesi.

Test Durumu 24

Testin Görevi: "Tek Oyuncu" butonuna bastıktan sonra çıkan ekrandaki "Süre Yarışı" butonunun ve sonrasında "Bakteri" hastalığının testi.

Test Açıklaması: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Bakteri" hastalık çeşidi seçilip oyuna girilip girilmediği test edilir.

Beklenen Sonuç: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Bakteri" hastalık çeşidinin seçilip oyuna sorunsuzca girilmesi.

Test Durumu 25

Testin Görevi: "Tek Oyuncu" butonuna bastıktan sonra çıkan ekrandaki "Süre Yarışı" butonunun ve sonrasında "Virüs" hastalığının testi.

Test Açıklaması: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Virüs" hastalık çeşidi seçilip oyuna girilip girilmediği test edilir.

Beklenen Sonuç: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Virüs" hastalık çeşidinin seçilip oyuna sorunsuzca girilmesi.

Test Durumu 26

Testin Görevi: "Tek Oyuncu" butonuna bastıktan sonra çıkan ekrandaki "Süre Yarışı" butonunun ve sonrasında "Mantar" hastalığının testi.

Test Açıklaması: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Mantar" hastalık çeşidi seçilip oyuna girilip girilmediği test edilir.

Beklenen Sonuç: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Mantar" hastalık çeşidinin seçilip oyuna sorunsuzca girilmesi.

Test Durumu 27

Testin Görevi: "Tek Oyuncu" butonuna bastıktan sonra çıkan ekrandaki "Süre Yarışı" butonunun ve sonrasında "Nöraks Solucanı" hastalığının testi.

Test Açıklaması: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Nöraks Solucanı" hastalık çeşidi seçilip oyuna girilip girilmediği test edilir.

Beklenen Sonuç: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Nöraks Solucanı" hastalık çeşidinin seçilip oyuna sorunsuzca girilmesi.

Test Durumu 28

Testin Görevi: "Tek Oyuncu" butonuna bastıktan sonra çıkan ekrandaki "Süre Yarışı" butonunun ve sonrasında "Parazit" hastalığının testi.

Test Açıklaması: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Parazit" hastalık çeşidi seçilip oyuna girilip girilmediği test edilir.

Beklenen Sonuç: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Parazit" hastalık çeşidinin seçilip oyuna sorunsuzca girilmesi.

Test Durumu 29

Testin Görevi: "Tek Oyuncu" butonuna bastıktan sonra çıkan ekrandaki "Süre Yarışı" butonunun ve sonrasında "Nekroa Virüsü" hastalığının testi.

Test Açıklaması: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Nekroa Virüsü" hastalık çeşidi seçilip oyuna girilip girilmediği test edilir.

Beklenen Sonuç: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Nekroa Virüsü" hastalık çeşidinin seçilip oyuna sorunsuzca girilmesi.

Test Durumu 30

Testin Görevi: "Tek Oyuncu" butonuna bastıktan sonra çıkan ekrandaki "Süre Yarışı" butonunun ve sonrasında "Prion" hastalığının testi.

Test Açıklaması: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Prion" hastalık çeşidi seçilip oyuna girilip girilmediği test edilir.

Beklenen Sonuç: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Prion" hastalık çeşidinin seçilip oyuna sorunsuzca girilmesi.

Test Durumu 31

Testin Görevi: "Tek Oyuncu" butonuna bastıktan sonra çıkan ekrandaki "Süre Yarışı" butonunun ve sonrasında "Nano Virüs" hastalığının testi.

Test Açıklaması: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Nano Virüs" hastalık çeşidi seçilip oyuna girilip girilmediği test edilir.

Beklenen Sonuç: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Nano Virüs" hastalık çeşidinin seçilip oyuna sorunsuzca girilmesi.

Test Durumu 32

Testin Görevi: "Tek Oyuncu" butonuna bastıktan sonra çıkan ekrandaki "Süre Yarışı" butonunun ve sonrasında "Biyolojik Silah" hastalığının testi.

Test Açıklaması: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Biyolojik Silah" hastalık çeşidi seçilip oyuna girilip girilmediği test edilir.

Beklenen Sonuç: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Biyolojik Silah" hastalık çeşidinin seçilip oyuna sorunsuzca girilmesi.

Test Durumu 33

Testin Görevi: "Tek Oyuncu" butonuna bastıktan sonra çıkan ekrandaki "Süre Yarışı" butonunun ve sonrasında "Maymun Gribi" hastalığının testi.

Test Açıklaması: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Maymun Gribi" hastalık çeşidi seçilip oyuna girilip girilmediği test edilir.

Beklenen Sonuç: "Süre Yarışı" butonu seçilip "Oyna" butonuna basıldıktan sonra çıkan ekranda "Maymun Gribi" hastalık çeşidinin seçilip oyuna sorunsuzca girilmesi.

Test Durumu 34

Testin Görevi: Ana menüdeki "Senaryo Yaratıcı" butonunun testi

Test Açıklaması: Ana menüde bulunan "Senaryo Yaratıcı" butonuna basılır ve ayrı bir program olarak Senaryo Yaratıcı'nın açılıp açılmadığı kontrol edilir.

Beklenen Sonuç: "Senaryo Yaratıcı"nun sorunsuzca açılması.

Test Durumu 35

Testin Görevi: Ana menüdeki "Lider Tablosu" butonunun testi.

Test Açıklaması: Ana menüde bulunan "Lider Tablosu" butonuna basılıp açılan pencerede sıralamaların çıkıp çıkmadığı kontrol edilir.

Beklenen Sonuç: "Lider Tablosu" ekranında oyunu oynayanlar arasındaki sıralamaların çıkması.

Test Durumu 36

Testin Görevi: Ana menüdeki "Seçenekler" butonu ve içindeki "Genel" bölümündeki ayarların testi.

Test Açıklaması: "Seçenekler" içindeki "Genel" sekmesinde bulunan Ara Yüz, Fare üzerinde Yardım, Oto Oyun Durdur, İlave Baloncuk, Arka Planda Duraklatma ve Fragman ayarlarının tek tek kontrolünün yapılması.

Beklenen Sonuç: "Genel" sekmesi içinde bulunan tüm ayarların eksiksik şekilde çalışması.

Test Durumu 37

Testin Görevi: Ana menüdeki "Seçenekler" butonu ve içindeki "Ses" bölümündeki ayarların testi.

Test Açıklaması: "Seçenekler" içindeki "Ses" sekmesinde bulunan Ana Ses, Müzik Sesi, Efekt Sesi ayarlarının tek tek kontrolünün yapılması.

Beklenen Sonuç: "Ses" sekmesi içinde bulunan tüm ses ayarlarının düzgün çalışması.

Test Durumu 38

Testin Görevi: Ana menüdeki "Seçenekler" butonu ve içindeki "Görüntü" bölümündeki ayarların testi.

Test Açıklaması: "Seçenekler" içindeki "Görüntü" sekmesinde bulunan Çözünürlük, Kenar Yumuşatma, Hedef FPS, VSync, Eşyönsüz, Pencere Modu, Dinamik Efektler, Parçacık Efektleri ayarlarının tek tek kontrolü ve Değişiklikleri Uygula butonunu testi yapılır.

Beklenen Sonuç: "Görüntü" sekmesinde bulunan tüm görüntü ayarlarının, Değişiklikleri Uygula butonuna basıldığında düzgün çalışması.

Test Durumu 39

Testin Görevi: Ana menüdeki "Seçenekler" butonu ve içindeki "Kontroller" bölümündeki ayarların testi.

Test Açıklaması: "Seçenekler" içindeki "Kontroller" sekmesinde bulunan klavye tuş değişimlerinin tek tek kontrolü yapılır.

Beklenen Sonuç: "Kontroller" sekmesinde bulunan tüm tuşların değiştirilmesi sonucunda düzgün çalışmaları.

Test Durumu 40

Testin Görevi: Ana menüdeki "Seçenekler" butonu ve içindeki "Erişilebilirlik" bölümündeki ayarların testi.

Test Açıklaması: "Seçenekler" içindeki "Erişilebilirlik" sekmesinde bulunan Dil ve Yazı Tipi ayarlarının testi yapılır.

Beklenen Sonuç: "Erişilebilirlik" sekmesinde ayarlar değişitirildiğinde bu ayarların düzgün çalışması.

Test Durumu 41

Testin Görevi: Ana menüdeki "Yardım ve Bilgi" butonunun ve içindeki "Bize Ulaşın" bölümünün testi.

Test Açıklaması: "Yardım ve Bilgi" içindeki "Bize Ulaşın" kısmına girildiğinde gönderilecek e-posta'nın gönderilip gönderilmediğinin testi yapılır.

Beklenen Sonuç: "Bize Ulaşın" kısmındaki e-posta gönderme işleminin sorunsuz gerçekleşmesi.

Test Durumu 42

Testin Görevi: Ana menüdeki "Yardım ve Bilgi" butonunun ve içindeki "Nasıl Oynanır" bölümünün testi.

Test Açıklaması: "Yardım ve Bilgi" içindeki "Nasıl Oynanır" kısmına girildiğinde çıkan pencerede oyunun nasıl oynanacağını gösteren rehberin bölümlerinin tek tek testi yapılırak açıklamaların doğru olup olmadığına bakılır.

Beklenen Sonuç: "Nasıl Oynanır" kısmındaki oyun rehberindeki açıklamaların hepsinin doğru olması.

Test Durumu 43

Testin Görevi: Ana menüdeki "Yardım ve Bilgi" butonunun ve içindeki "Katkıda Bulunanalar" bölümünün testi.

Test Açıklaması: "Yardım ve Bilgi" içindeki "Katkıda Bulunanlar" kısmına girildiğinde çıkan kayan yazıda oyuna katkıda bulunan herkesin isminin çıkıp çıkmadığı ve doğru yazılıp yazılmadığı kontrol edilir.

Beklenen Sonuç: "Katkıda Bulunanlar" kısmındaki yazıda oyuna katkıda bulunan tüm herkesin isminin çıkması ve isimlerin doğru yazılmış olması.

Test Durumu 44

Testin Görevi: Ana menüdeki "Yardım ve Bilgi" butonunun ve içindeki "Gizlilik Sözleşmesi" bölümünün testi.

Test Açıklaması: "Yardım ve Bilgi" içindeki "Gizlilik Sözleşmesi" kısmına girildiğinde çıkan sözleşmenin doğruluğunun kontrolü yapılır.

Beklenen Sonuç: "Gizlilik Sözleşmesi" kısmındaki sözleşmenin doğru yazılmış olması.

Test Durumu 45

Testin Görevi: Ana menüdeki "Yardım ve Bilgi" butonunun ve içindeki "Lisanslar" bölümünün testi.

Test Açıklaması: "Yardım ve Bilgi" içindeki "Lisanslar" kısmına girildiğinde oyunda kullanılan lisansların eksiksiz yazılıp yazılmadığı kontrol edilir.

Beklenen Sonuç: "Lisanslar" kısmındaki lisansların eksiksiz açıklanmış olması.

Oyunun testinin de yapıldığı bu son sürümünde, yazılan tüm bu test durumları geçmiş ve oyunun daha kaliteli bir seviyede oynanılması sağlanmıştır.

10.3 Oyun Versiyonuna Göre Hatalar, Eklemeler ve Düzeltmeler

Küçük Evrim 0.5.1 (forum.ndemiccreations)

i. Steam Trading Cards özelliği eklendi

ii. Başarımların düzgün açılması ve Steam ile eşleşmesi sağlandı

iii. Oyun, Steam kayıt dosyalarıyla düzgün çalışır hale getirildi

iv. Bazı ülkelerin açıklamalarındaki XML hataları giderildi

v. Kontrol oklarına bağlı olarak yazım açıklamaları düzeltildi

Küçük Evrim 0.5.1.2 (forum.ndemiccreations)

i. "Steam'e bağlanılamıyor" hatası düzeltildi

Evrim 2 (forum.ndemiccreations)

i. Yüksek performanslı bilgisayarların aşırı ısınmasına neden olmamak için FPS kilitleme özelliği eklendi

ii. Oyun açıkken tab tuşuyla windows'a dönülüp tekrar oyuna girildiğinde oluşan ekran kartı sorunu giderildi

iii. Doku yönetimi geliştirildi

iv. Oyunun normal veya daha yüksek zorlukta bitirildiğinde sonraki hastalık tipinin açılması ve tüm hastalık tipleriyle en zorda bitirildiğinde hilelerin açılması eklendi

v. Oyun sırasındaki bir kaç olay mesajı için yazım hataları düzeltildi

vi. Genel performans gelişimi yapıldı

vii. Nöraks Solucanı hastalık tipi eklendi

Küçük Düzeltme 0.5.6 (forum.ndemiccreations)

i. Hastalık tiplerinin yanına, zorluğa bağlı olarak derecelendirme sistemi madalyaları düzeltildi

ii. Kilitli genlere iki kere tıklanıldığında açılma hatası giderildi

iii. Oyun zor/çok zorda bitirildiğinde hilelerin düzgün açılması düzeltildi

iv. Euro bölgesi yazım hataları düzeltildi

v. Necroa Virüsü'nün açılma mesajı düzeltildi

vi. CDC haberlerindeki XML hatası düzeltildi

Evrim 3 (forum.ndemiccreations)

i. Süre Yarışı modu eklendi

ii. Yetenekler/Belirtiler için açıklamalar düzeltildi

iii. Hastalık yaratmada yanlışlıkla seçim hatasını ortadan kaldırmak için çift tıklama özelliği kaldırıldı

iv. Oyun içinde hata belirtimi yaparken çift e-posta sorununun olmaması için onay ekranı eklendi

v. Tekrar oynatma sırasında baloncukların olmamasına rağmen baloncuk sesinin çıkması düzeltildi

Evrim 4 (forum.ndemiccreations)

i. Nekroa Virüsü hastalık tipi eklendi

ii. Sadece hastalık bulaşmış ülkelerin o şekilde gösterilmesi özelliği değiştirildi

iii. Menü ayarlarında ESC tuşu sorunu düzeltildi

iv. Dünya ekranındaki araştırma bölümünün % olarak gösterilmesi değiştirildi

v. Oyun bitirildiğindeki ekran renkleri geliştirildi

vi. Yeni Zelanda'daki manyetik değişim etkisi azaltıldı

vii. Veri yönetiminin daha verimli yapılabilmesi için oyun kodu gözden geçirildi

viii. Nöraks Solucanı için beyin etkisi özelliği geliştirildi

ix. Oyun sonundaki tekrar oynatma bölümü için tekrar oynatma hızı arttırıldı

Evrim 5 (forum.ndemiccreations)

i. Fransızca, Almanca, Rusça dil destekleri eklendi

ii. VSync, FPS ve Anisotropic görüntü ayarları eklendi

iii. Ndemic logosuyla açılış ekranı eklendi

iv. Oyuncu oyun içerisinde değiştirmediği sürece oyunun dili, Steam platformunun dili olarak otomatik seçilme özelliği getirildi

v. Gen menü kutucuklarının seçilen dile göre dinamik olarak tekrar şekillenme özelliği getirildi

vi. Tercümeler ve görüntü ayarları için ipuçları güncellendi

Evrim 6 (forum.ndemiccreations)

i. Resmi Senaryolar bölümü eklendi

ii. Oyun içindeki hastalık ekranındaki hastalığın 3D modellemesi ve zaman içindeki değişim modellemesi geliştirildi

iii. 1'den 4'e kadar tuşlarla hastalık ekranında hızlı gezinti özelliği eklendi

Evrim 7 (forum.ndemiccreations)

i. Yeni başarımlarla birlikte 4 yeni senaryo eklendi

ii. Yeni hileler ve hile fonksiyonları eklendi

iii. Yeni oyun sonu ekranları eklendi

iv. Portekizce, İspanyolca ve İtalyanca dilleri eklendi

v. Oyun motoruna önemli güncellemeler getirildi

Evrim 8 (forum.ndemiccreations)

i. Dawn of the Planet of the Apes filmindeki Maymun Gribi hastalığı eklendi

ii. Oyun fragmanı eklendi

iii. Geleneksel ve basitleştirilmiş Çince'ye lokalizasyon yapıldı

Evrim 9 (forum.ndemiccreations)

i. Senaryo Yaratma uygulaması ve Özel Senaryolar eklendi

Evrim 10 (forum.ndemiccreations)

i. Nekroa Virüsü ve Prion hastalıklarının açılma sırası değiştirildi

ii. Basit hilelerin tüm hastalıklarla normal zorlukta bitirildiğinde açılma özelliği getirildi

iii. Oyun içindeki ipuçlarına güncellemeler getirildi

iv. Oyun yapay zekası geliştirildi

v. Özel Senaryolar ekranında Yeni sekmesi eklenerek yeni yaratılmış senaryolara görülme kolaylığı sağlandı

vi. Özel Senaryolar için indirme hızı arttırıldı

vii. Ülke nüfusları güncellendi

viii. Norveçce, Türkçe, Polonyaca ve Japonca dil destekleri eklendi

Evrim 11 (forum.ndemiccreations)

i. Nüfus grafikleri bölge grafikleri olarak değiştirildi

ii. Ana oyun nüfus ipuçları güncellendi

iii. Özel Senaryolar için dil filtresi eklendi

Plague Inc Evolved oyunu halen geliştirilmekte olduğundan daha sonra da bir çok güncelleme gelecektir. Şimdiye kadar yapılan güncellemeler burada yazılmıştır. Tüm güncellemeleri okumak için oyunun Steam platformu bölümüne veya internet sitesi forumlarına bakılabilir

11. SONUÇ

Yazılım ve donanım teknolojisinin sürekli gelişmesiyle birlikte paralel olarak yazılım projeleri de büyümektedir. Böylece yazılım testlerinin önemi her gün artmaktadır. Büyük bir problem oluşturmayacak düzeyde geç çıkan; ama sorunlarından arınmış bir yazılım, erken; ama testleri tam yapılmadığı için çok fazla hatası olan bir yazılıma tercih edilir. Çok büyük projelerde ise yazılım testlerinin önemi ekstra derecede önemlidir. Piyasaya sürülmüş bir ürünün hatalar nedeniyle sürekli sorun çıkarıp işlevlerini düzgün yapamaz hale gelmesi çok büyük para ve zaman kaybına yolaçar.

Banka yazılımları, askeri yazılımlar, kamu yazılımları, oyunlar gibi yazılım teknolojisinde önemli yerleri olan projeler için teorik bilgisi de iyi düzeyde olan test mühendisleri çok önemlidir. Gün geçtikçe iyi kodcular dışında iyi testçilere de ihtiyaç artmaktadır. Bu tezle birlikte gerekli teorik bilgiler öğrenilip yazılım test mühendisliği alanında önemli bir noktaya gelinmesi hedeflenmiştir.

KAYNAKLAR

Sqaforums, Erişim Tarihi: 4 Eylül 2014, http://www.sqaforums.com/forums/forum.php

Yazgelistir, Erişim Tarihi: 4 Eylül 2014, http://www.yazgelistir.com/makale/proje-yonetim-surecleri

Softsmith, Erişim Tarihi: 6 Eylül 2014
http://www.softsmith.com/course_materials.html#testing

Softwaretestinghelp, Erişim Tarihi: 6 Eylül 2014, 7-8-9-12-16-24-26-30 Ekim 2014
http://www.softwaretestinghelp.com/

Devtopics, Erişim Tarihi: 7 Eylül 2014, http://www.devtopics.com/20-famous-software-disasters/

Tutorialspoint, Erişim Tarihi: 7 Eylül 2014,
http://www.tutorialspoint.com/software_testing/testing_types.htm

Opensourcetesting, Erişim Tarihi: 10 Eylül 2014, http://opensourcetesting.org/functional.php

Pushtotest, Erişim Tarihi: 11 Ekim 2014, http://www.pushtotest.com/web-testing-with-selenium-pushtotest

MSDN, Erişim Tarihi: 13 Ekim 2014, http://msdn.microsoft.com/en-us/library/ee417692(v=vs.85).aspx ,

Yazgelistir, Erişim Tarihi: 8 Ekim 2014, http://www.yazgelistir.com/makale/proje-yonetim-surecleri

Salyangoz, (2013), Erişim Tarihi: 7 Ekim 2014,
http://salyangoz.com.tr/blog/2013/11/02/digerleri/yazilim-gelistirme-surec-modelleri-nelerdir/

Innova, Erişim Tarihi: 8 Ekim 2014, http://www.innova.com.tr/uygulama-gelistirme-yasam-dongusu.asp

Ozgursaracoglu, (2011), Erişim Tarihi: 9 Ekim 2014 ,
http://ozgursaracoglu.blogspot.com.tr/2011/02/yazlm-muhendisligi-yazlm-muhendisligi.html

Espadon, Erişim Tarihi: 10 Ekim 2014,
http://www.espadon.org/espadon/tech_content/methodoloy/methodo.htm

1000sourcecodes, (2012), Erişim Tarihi: 11 Ekim 2014,
http://www.1000sourcecodes.com/2012/05/software-engineering-incremental-model.html

Softwaretestinggenius, Erişim Tarihi: 13 Ekim 2014,
http://www.softwaretestinggenius.com/test-case-and-its-sample-template

Vietnamesetestingboard, Erişim Tarihi: 14 Ekim 2014,
http://www.vietnamesetestingboard.org/zbxe/?document_srl=114820

Cs.bgu, Erişim Tarihi: 16 Ekim 2014, http://www.cs.bgu.ac.il/~bpmn/Bug_Lifecycle

Efficientqa, Erişim Tarihi: 16 Ekim 2014, http://www.efficientqa.com/Process/pm.html

Istqbexamcertification, Erişim Tarihi: 17 Ekim 2014, http://istqbexamcertification.com/what-is-integration-testing/

Sce.uhlc, Erişim Tarihi: 17 Ekim 2014,
http://sce.uhcl.edu/whiteta/sdp/subSystemIntegrationTesting.html

Etestinghub, Erişim Tarihi: 18 Ekim 2014,
http://www.etestinghub.com/stages_of_life_cycle1.php

Csis.pace, Erişim Tarihi: 19 Ekim 2014,
http://csis.pace.edu/~marchese/cs615sp/L7New/Lec7new.html

Codeproject, Erişim Tarihi: 20 Ekim 2014,
http://www.codeproject.com/Articles/5083/Advanced-Unit-Testing-Part-III-Testing-Processes

Steam, (2014), Erişim Tarihi: 6 Kasım 2014, http://store.steampowered.com/app/246620/

Steam, (2014), Erişim Tarihi: 6 Kasım 2014,
http://store.steampowered.com/news/?appids=246620

Ndemic, (2014), Erişim Tarihi: 11 Kasım 2014, http://forum.ndemiccreations.com/

Ndemic, (2014), Erişim Tarihi: 11 Kasım 2014, http://forum.ndemiccreations.com/showthread.php?tid=10

Ndemic, (2014), Erişim Tarihi: 11 Kasım 2014, http://forum.ndemiccreations.com/showthread.php?tid=44

Ndemic, (2014), Erişim Tarihi: 11 Kasım 2014, http://forum.ndemiccreations.com/showthread.php?tid=1393

Ndemic, (2014), Erişim Tarihi: 11 Kasım 2014, http://forum.ndemiccreations.com/showthread.php?tid=1702

Ndemic, (2014), Erişim Tarihi: 11 Kasım 2014, http://forum.ndemiccreations.com/showthread.php?tid=1799

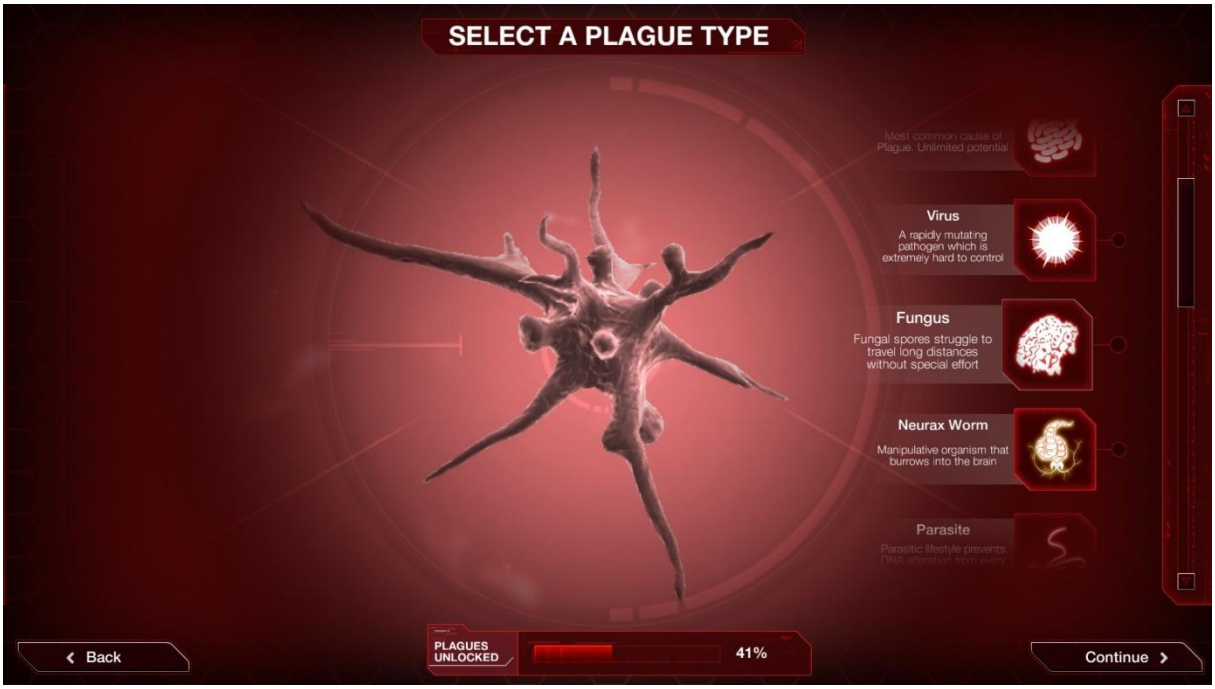
Ndemic, (2014), Erişim Tarihi: 11 Kasım 2014, http://forum.ndemiccreations.com/showthread.php?tid=1863

Ndemic, (2014), Erişim Tarihi: 11 Kasım 2014, http://forum.ndemiccreations.com/showthread.php?tid=1863

EKLER

Burada oyunun tanıtılması amacıyla çeşitli kısımlarından ekran görüntüleri alınmıştır.

Plague Inc Evolved Oyununun Ekran Görüntüleri


SELECT A PLAGUE TYPE
Most common cause of Plague. Unlimited potential
Virus
A rapidly mutating pathogen which is extremely hard to control
Fungus
Fungal spores struggle to travel long distances without special effort
Neurax Worm
Manipulative organism that burrows into the brain
Parasite
Parasitic lifestyle prevents DNA alteration from even
Back
PLAGUES UNLOCKED
41%
Continue


NEWS
starts cure research
9-4-2015
World
INFECTED
//4,005,270,166
DEAD
//114,463
DISEASE
39
WORLD
1%

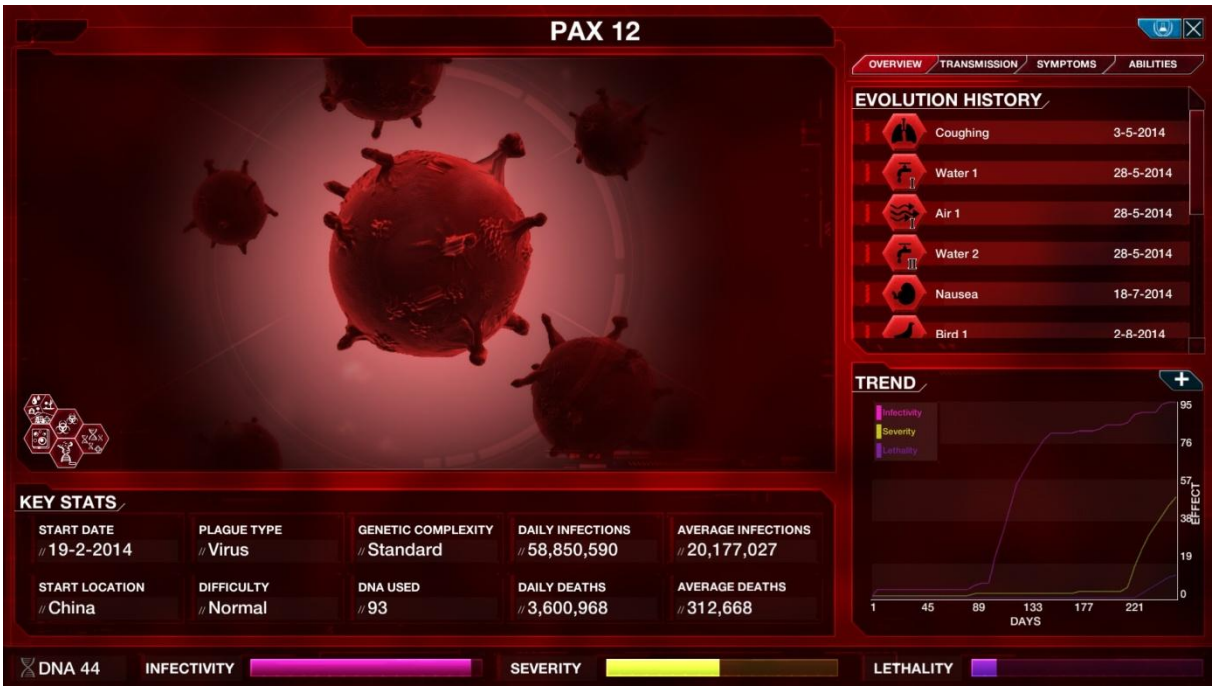

NEWS
23-4-2015
NEWSFLASH - APE ESCAPE IN USA
A group of apes have broken out of a primate shelter in USA and destroyed a Gen-Sys lab - causing citywide chaos. Authorities were unable to contain them and the apes escaped
OK
APE
CANADA
INFECTED
//102,240
DEAD
//0
DISEASE
23
WORLD
3%


PAX-12
OVERVIEW
TRANSMISSION
SYMPTOMS
ABILITIES
DEVOTION
INFORMATION
// Surges of Dopamine and Serotonin compel the host to constantly think about the worm and triggers intense rushes of pleasure
COST
// 27 DNA Points
EVOLVE
DNA 32
INFECTIVITY
SEVERITY
LETHALITY


PAX 12
OVERVIEW
TRANSMISSION
SYMPTOMS
ABILITIES
TOTAL ORGAN FAILURE
INFORMATION
// Catastrophic cell death of multiple tissue types causes body-wide organ failure and rapid death
COST
// 34 DNA Points
EVOLVE
DNA 37
INFECTIVITY
SEVERITY
LETHALITY


WORLD
Business as usual
20
38
12
19
7
13
10%
15%
TOTAL
6,811,136,495
74%
CURE
STATUS - Researching
// 23%
// 173 Days until completion
// Cure Date ETA: 11-8-2015
KEY RESEARCH CONTRIBUTORS:
// Bolivia
// Zimbabwe
// Angola
RESEARCH PRIORITY:
// High
POTENTIAL
ACTIVE
DESTROYED
0
50
150
200
250
300
350
COUNTRY STATE
Iceland
Sweden
USA
Canada
Greenland
Mexico
Brazil
Argentina
Caribbean
C. America
Colombia
STATS
HEALTHY
// 1,041,832,302
INFECTED
// 5,067,261,107
DEAD
// 702,043,086


ALZ-113
OVERVIEW
TRANSMISSION
SYMPTOMS
ABILITIES
SIMIAN NEURO-GENESIS
INFORMATION
// Modifies simian genes to significantly increase ape intelligence. Irises turn green and no immune response triggered. It's time for apes to rise!
COST
//12 DNA Points
DEVOLVE
DNA 19
INFECTIVITY
SEVERITY
LETHALITY


PAX 12
OVERVIEW
TRANSMISSION
SYMPTOMS
ABILITIES
EVOLUTION HISTORY
Coughing 3-5-2014
Water 1 28-5-2014
Air 1 28-5-2014
Water 2 28-5-2014
Nausea 18-7-2014
Bird 1 2-8-2014
TREND
Infectivity
Severity
Lethality
95
76
57
38
19
0
EFFECT
1
45
89
133
177
221
DAYS
KEY STATS
START DATE
//19-2-2014
PLAGUE TYPE
//Virus
GENETIC COMPLEXITY
//Standard
DAILY INFECTIONS
//58,850,590
AVERAGE INFECTIONS
//20,177,027
START LOCATION
//China
DIFFICULTY
//Normal
DNA USED
//93
DAILY DEATHS
//3,600,968
AVERAGE DEATHS
//312,668
DNA 44
INFECTIVITY
SEVERITY
LETHALITY

ÖZGEÇMİŞ

Güneş Kuday, 19 Mart 1985'de doğmuştur. İlkokulu İstanbul'da Nurettin Teksan İlköğretim okulunda okumuş, ortaokul ve liseyi Antalya'da Akdeniz Kolejinde bitirmiştir.

Üniversite için İstanbul'a gelmiş ve Haliç Üniversitesi Bilgisayar Mühendisliğini bitirmiştir. Yüksek Lisansını da Haliç Üniversitesi Bilgisayar Mühendisliğinde yapmıştır.

Şu anda Ankara'da Başarı Mobile'da Yazılım Mühendisi olarak çalışmaktadır.