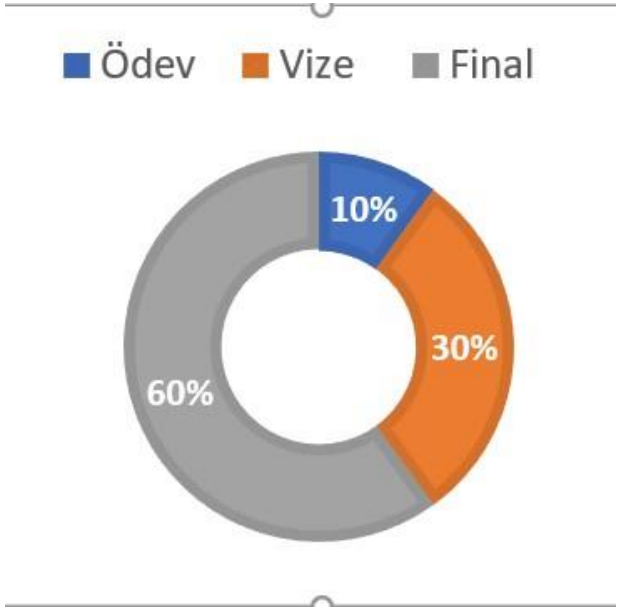
Yazılım Mühendisliği
1906003082015

Dr. Öğr. Üy. Önder EYECİOĞLU
Bilgisayar Mühendisliği

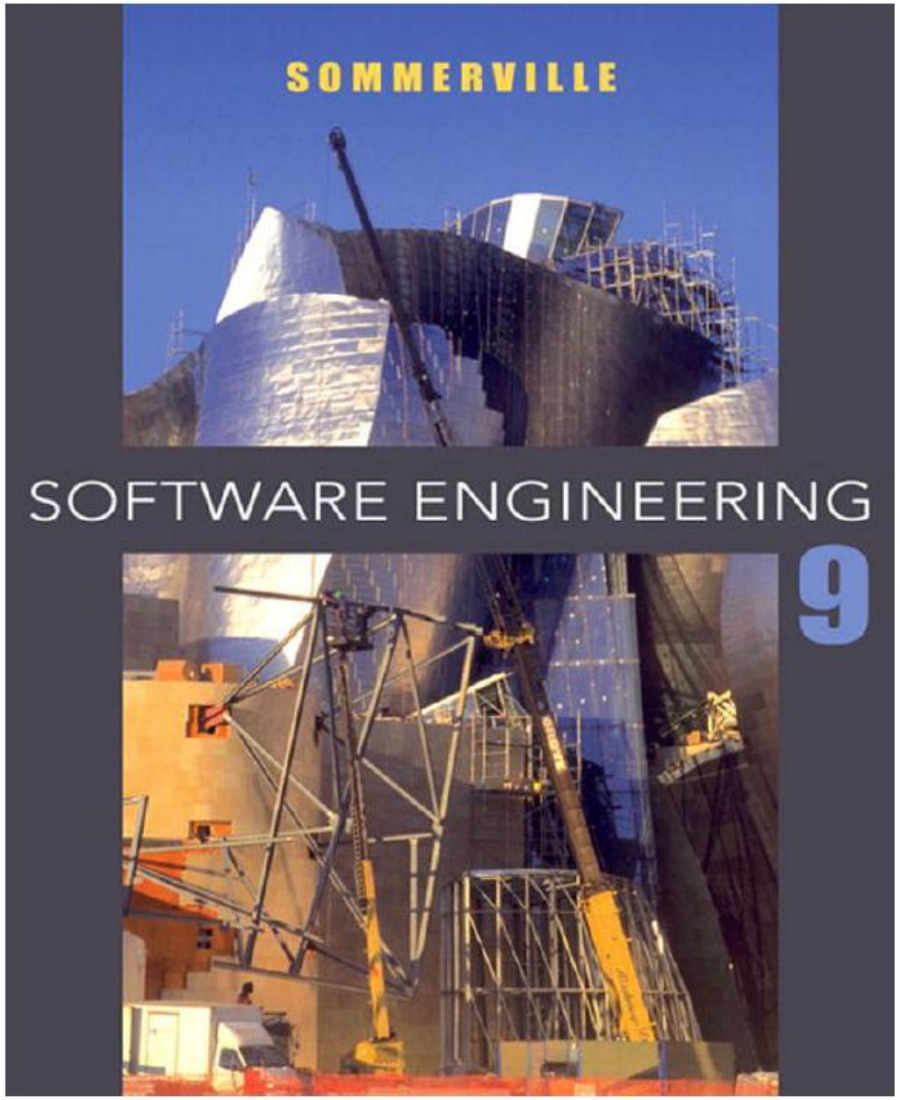
Giriş

Ders Günü ve Saati:
Salı: 09:15-13:00
Devam zorunluluğu %70



HAFTA	KONULAR
Hafta 1	Yazılım Mühendisliğine Giriş
Hafta 2	Yazılım Geliştirme Süreç Modelleri
Hafta 3	Yazılım Gereksinim Mühendisliği
Hafta 4	Yazılım Mimarisi
Hafta 5	Nesneye Yönelik Analiz ve Tasarım
Hafta 6	Laboratuar Çalışması: UML Modelleme Araçları
Hafta 7	Yazılım Test Teknikleri
Hafta 8	Ara Sınav
Hafta 9	Yazılım Kalite Yönetimi
Hafta 10	Yazılım Bakımı - Yeniden Kullanımı ve Konfigürasyon Yönetimi
Hafta 11	Yazılım Proje Yönetimi (Yazılım Ölçümü ve Yazılım Proje Maliyet Tahmin Yöntemleri)
Hafta 12	Yazılım Proje Yönetimi (Yazılım Risk Yönetimi)
Hafta 13	Çevik Yazılım Geliştirme Süreç Modelleri
Hafta 14	Yazılım Süreci İyileştirme, Yeterlilik Modeli (CMM)

Kaynaklar



10.
Yazılım Güvenirliliği
Yazılım Test Planlama Ve Yönetim

Giriş

BÖLÜM HEDEFLERİ

- Yazılım kalite ölçütlerinin incelenmesi
- Yazılım ölçütlerinin test adımlarında kullanımının anlaşılması
- Kalite Güvenirliğinin kestirimini ve modellerini tanima

9. HAFTA İÇERİĞİ

- Yazılım Güvenirliği
- Yazılım Güvenirliliği Modelleri
- Yazılım Güvenirlik Modeli Değerlendirme
- Yazılım Güvenirliği Sağlama Adımları

Yazılım Güvenirliği

- Yazılım güvenilirliği, yazılım ve yazılım bileşenlerinin belli ortamda, belirli zaman içe-risinde kendisinden beklenen özellikleri yerine getirebilme olasılığıdır. Başka bir deyişle bir yazılıma ne kadar güvenilebileceğinin ölçümüdür.
- Yazılımda ortaya çıkan hatalar yazılıma olan güveni azaltırken yazılımın kullanılabilirliğini de engellemekte-dir. Uzun süre hata vermeden, kendisinden beklenen özellikleri yerine getiren bir yazılım ise ona olan güveni arttırmaktadır. [1]

Yazılım Güvenirliği

- Tüm yazılımların aynı kalitede ve aynı bileşenlerle aynı sonucu vermesi beklenen bir durum değildir. Ayrıca, bir yazılımı tümüyle test etmek de mümkün değildir. 1000 kodluk bir ticari programda en az 1 kod satırı hatalı çıkmakta olduğu bilinmektedir.
- Kalite güvenirliği; geçmiş bilgilere ve sınamaya dayalı olarak,

Başarı oranı = Başarılı süre / Toplam işletim süresi

formülü ile kestirilmektedir.

Yazılım Güvenirliği

- Kalitenin etkenliği; gider kalemlerinin, formülü ile kestirilmektedir.

$$C3 > C1 + C2$$

eşitsizliği ile belirlenmektedir.

C3 = kalite yetersizliği nedeni ile hata düzeltme gideri

C1 = kaliteyi yükseltme gideri

C2 = yine de bulunabilecek hataları düzeltme gideri

Yazılım Güvenirliği

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

Yazılım Güvenirliği

- Yazılım güvenilirliği, sistem veya bileşenlerinin, belirli bir ortamda, belirli bir zaman dilimi içinde kendilerinden beklenilen işlevleri yerine getirebilme olasılığı olarak tanımlanmaktadır.
- Yazılım güvenilirliği mühendisliği ise yazılım ürünlerinde müşteri memnuniyetini sağlamak üzere, güvenilirliğin kestirimi, ölçülmesi ve yönetilmesidir

Yazılım Güvenirliği

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

Yazılım Güvenirliği

- Yazılım güvenilirliğine ait yukarıdaki her iki tanımda da güvenilirlik ile kalite arasındaki yakın ilişki vurgulanmaktadır. Diğer bir şekilde, güvenilirlik zamana bağlı olan kalite olarak tanımlanmaktadır. Bu da yazılım güvenilirliğinin, yazılım kalitesi için önemli bir ölçüt olarak değerlendirilmesini gerektirmektedir.
- Yazılıma ne kadar güvenebileceğimiz yazılımda ortaya çıkan hataların sayısına, kategorisine ve yoğunluğuna bağlıdır. Günümüzde yapılan çalışmalarda, hatalar ortaya çıkmadan bir yazılımda bulunan hata miktarı tahmin edilerek yazılım güvenilirliği ölçülmeye çalışılmaktadır. [2]

Yazılım Güvenirliği

- Yazılım güvenilirliğinin temel kavramları;
- Yanlış (Error)
- Aksaklık (Fault)
- Hata (Failure)
- Kusur (Defect)

Yazılım Güvenirliği

Yanlış (Error)

- Hesaplanan, gözlemlenen veya ölçülen değerler arasında fark oluştuğundaki durumu anlatmakta kullanılmaktadır. Örneğin hesaplanma sonucunda 12 değeri bulunduğu halde, doğru sonuç 10 ise yanlışlıktan bahsedilebilir. Genellikle de programcının veya tasarımcının yanlış düşünceden kaynaklanan davranışları sonucunda ortaya çıkan doğru olmayan sonuçlar için kullanılmaktadır. Örnek olarak da programın doğru olmayan bir parçası veya işlemi verilebilir.

Yazılım Güvenirliği

Aksaklık (Fault)

- Yazılımda doğru olmayan bir adım, süreç veya veriyi tanımlamaktadır. Aksaklıklar yazılımın kodundaki yanlışların sonucunda ortaya çıkar. Örneğin, bir programlama dilinde *if* deyiminin kullanımı sırasında, dallanmada bir aksaklık oluşabilir ve deyim yanlış bir dala aktarılabilir. Böylelikle program istenilen sonucu üretemeyebilir ve bu durum da aksaklığa neden olabilmektedir.

Yazılım Güvenirliği

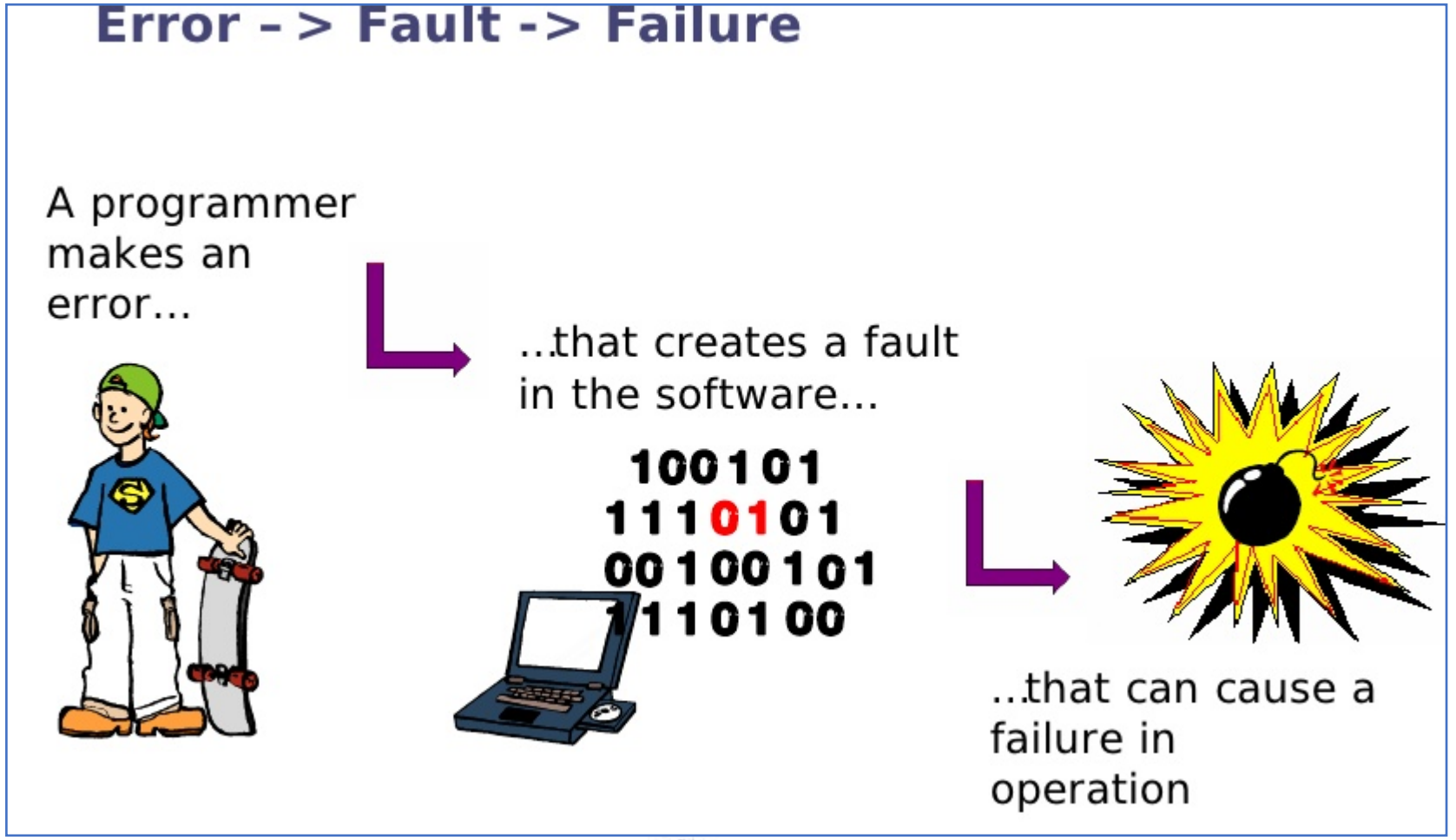
Hata (failure)

- (bozukluk olarak da kullanılmaktadır) bir sistemin kendisinden beklenilen işlevi yerine getirememe durumudur. Hata nedeni genellikle bir aksaklık olmaktadır.

Kusur (Defect)

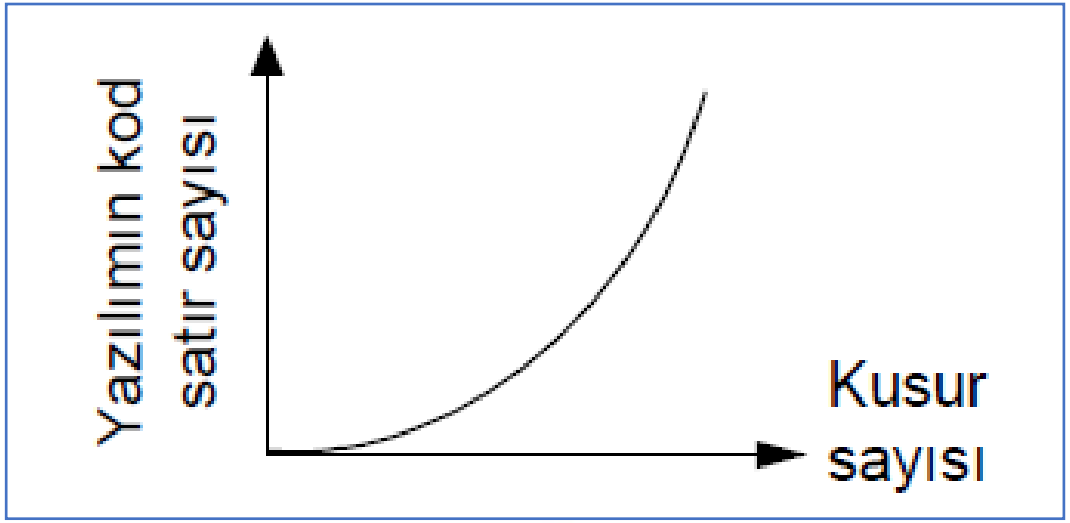
- Üründeki anomalidir. Hata veya aksaklık için kullanılan genel bir terimdir ve genellikle de
- Kusur= {Aksaklık}U{Hata}
- olarak tanımlanmaktadır. [2]

Yazılım Güvenirliği



Yazılım Güvenirliği

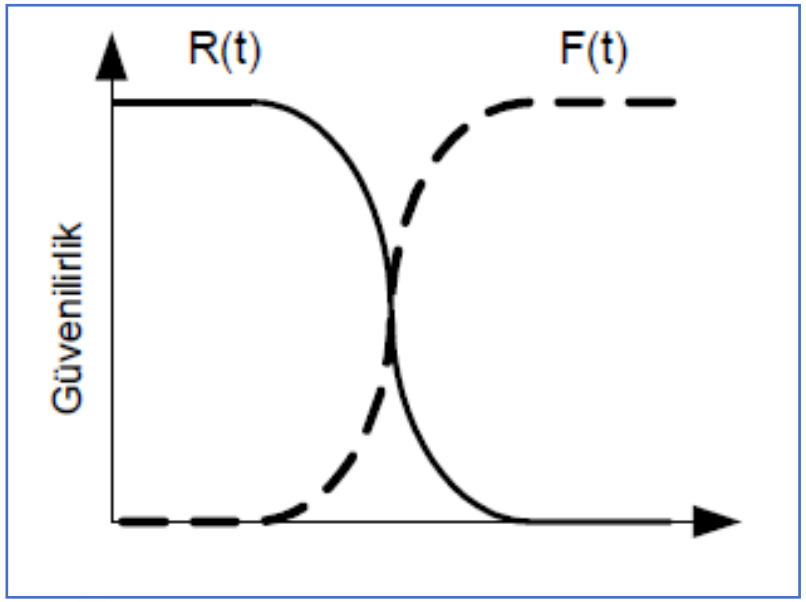
- Araştırmalar programcıların her 1000 satırlık kod için ortalama altı yazılım kusuru üretiklerini göstermektedir. Bu da 350000 satırlık bir ticari yazılımın yaklaşık 2000 adet yazılım kusurunu barındırabileceği anlamına gelmektedir.
- Yazılım büyüdükçe olası kusur sayısı da geometrik olarak artış göstermektedir [2]



Yazılım Güvenirliği

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

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



Yazılım Güvenirliği

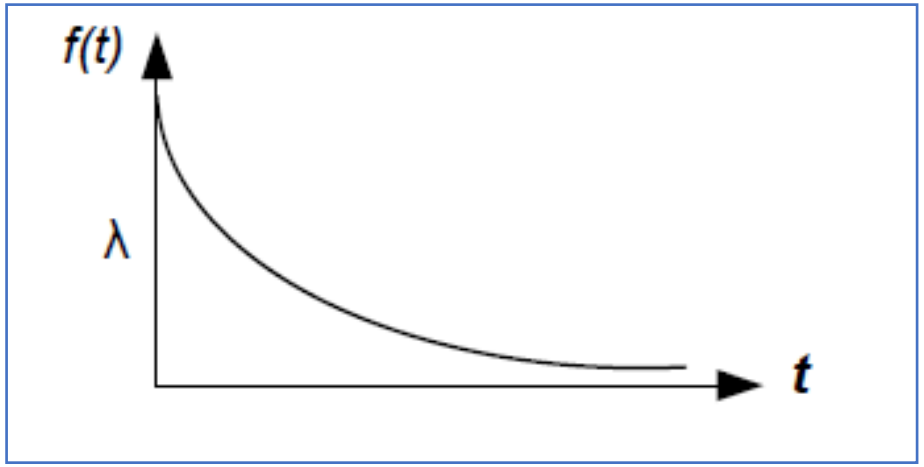
- f(t): Bir test case yürütmek için gerekli zamana (t), bağlı olasılık yoğunluk fonksiyonu olmak üzere ve λ hata yoğunluğu olmak üzere;

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

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

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

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

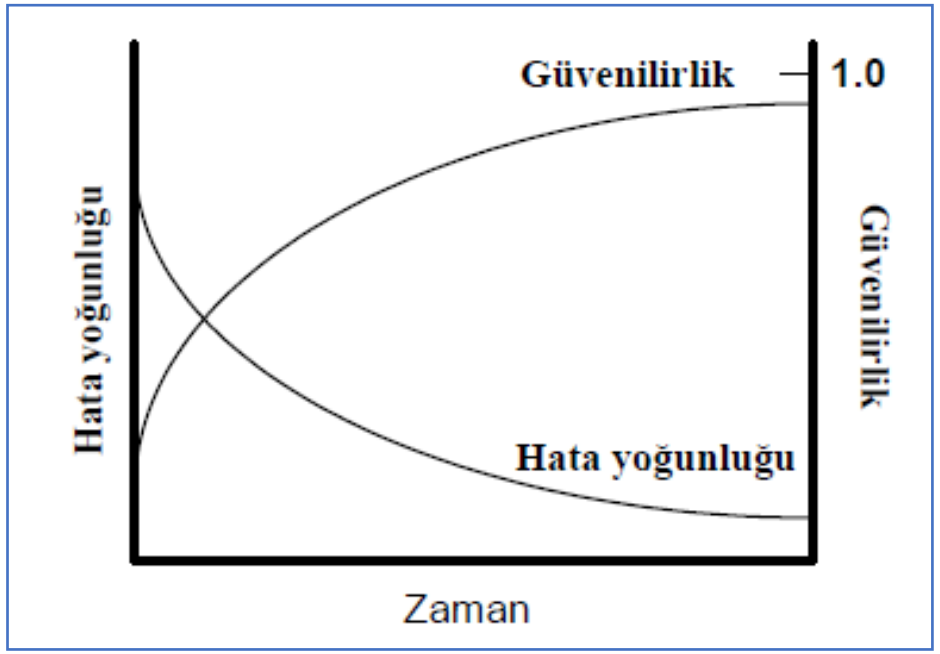


Yazılım Güvenirliği

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

1. Hata yoğunluğu ölçütü;
2. Ortaya çıkmamış hataların kestirilmesinde,
3. Test süresinin tamamlandığına ait kararların verilmesinde,
4. Standart hata yoğunluğunu saptamada kullanılmaktadır

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



Yazılım Güvenirliği

Ortalama Hata Süresi (OHS)

- OHS (*Mean Time To Failure-MTTF*), hataya kadar geçen ortalama çalışma süresidir. Hata zamanının beklenen değeri OHS'ni vermektedir.

$$OHS = \int_0^{\infty} R(t)\,dt$$

Yazılım Güvenirliği

Ortalama Hata Süresi (OHS)

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

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

$$\lambda = \frac{1}{\text{HAOS}}$$

Yazılım Güvenirliliği Modelleri

- Yazılım güvenilirlik modelleri, yazılımda önceden ortaya çıkan ha-taların davranışlarını inceleyerek ilerde oluşabilecek hata durumlarının matematiksel olarak ifade edilmiş halleridir.
- Yazılım güvenilirlik modelleri, özellikle hataların olu-şum zamanları veya belirli aralıklarda ortaya çıkan toplam hata sayısı gibi verileri kul-lanarak yazılımda gelecekte birim zamanda ne kadar hata oluşabileceğini tahmin ederler.
- Bu şekilde farklı varsayımlar öngörülerek oluşturulmuş farklı yazılım güveni-lirlik modelleri ortaya çıkmıştır. Ayrıca yapılan çalışmalarda önceden geliştirilmiş bir-çok yazılım güvenilirlik modeli farklı projeler için test edilmiş, birbirleri ile karşılaştı-rılmış ve kullanışlılıkları ölçülmüştür.

Yazılım Güvenirliliği Modelleri

- Yazılım güvenirliliğinin ve geçerliliğinin ölçülmesi için 1980'lerde 100'e yakın model geliştirilmiştir.
- Musa & Okumoto modelleri aşağıdaki kriterlre göre sınıflandırır.

1. Zaman etki alanı: yürütme süresine göre duvar saati
2. Kategori: Sonsuz zamanında toplam hata sayısı: sonlu, sonsuz
3. Tip: t zamanına kadar yaşanan arıza sayısının dağılım fonksiyonu türü: Poisson, binomal, bayes
4. Sınıf (yalnızca nite hatası modelleri): Zamanın bir fonksiyonu olarak hata yoğunluğunun işlevsel şekli
5. Aile (sadece sonlubaşarısızlık modellerinde): Arıza şiddetinin fonksiyonel biçimi, beklenen başarısızlık sayısına göre fonksiyonudur.

Yazılım Güvenirliliği Modelleri

- En yaygın olan dört model:

A. Jelinski/Moranda Model

B. Schneidewind Model

C. Musa Basic - Musa/Okumoto Log. Poison Execution-time Model

D. Littlewood-Verrall Model

Yazılım Güvenirliliği Modelleri

- Güvenirlik modelleri, istatistiki verilere dayanarak yazılımın güvenirliliğini saptamaktadır. Bu modellerin çalıştırılması sonucunda,

1. Toplam hata sayısı
2. Güvenirlilik düzeyi
3. Sistemde yapılan revizyon sonucunda kalan hata sayısı vs.

Gibi değerler elde edilmektedir.

Jelinski-Moranda Model

- Binom modelidir.
- Arızalar arasındaki süre, kalan hataların sayısıyla orantılı bir parametre ile üstel olarak dağıtılır.
- Eğer N hatası ile başlarsak ve i -1'ini çıkarırsak, hatalar arasındaki ortalama süre

$$1/\phi(N-(i-1))$$

Varsayımlar

1. Hata algılama oranı, yazılımdaki # sayısıyla orantılıdır
2. Hata algılama aralığı, arızalar arasındaki aralıkta sabittir
3. Arızalar yeni hatalar olmadan anında düzeltilir
4. Her hata, bir şiddet sınıfında karşılaşılma şansına sahiptir
5. Arızalar bağımsızdır.

Varsayımlar 4-6, güvenilirlik modellemesi için standart varsayımlardır.

Schneidewind Model

- Gerçekten veri yaşlanmasına hitap eden bir meta modeldir
- Model 1: Tüm hata sayılarını n dönemlerinden kullanın.
- Model 2: Hata sayımlarını sadece son s periyotlarından kullanın
- Model 3: Son s periyotlardan, bir zaman periyodu $(t_{n-s} - t_0)$ boyunca tek bir $\sum_{i=1}^{n-s} fi$ sayısından önce gelen hata sayımlarını ekleyin.

Musa's Basic Execution Time Model

- En çok kullanılan
- Yürütme süresini kullanır
- Arıza yoğunluğu işlevini bireysel hataların özelliklerinden ayırır.

Varsayımlar:

- Standart varsayımlara ek olarak;
- Sonlu bir hata modelidir.
- Arızalar arasındaki yürütme süreleri üstel olarak dağıtılır.
- Test için mevcut kaynaklar toplam gözlem süresi boyunca sabittir

Littlewood-Verrall Model

- Littlewood-Verrall modeli, n ve n + 1 arızaları arasındaki zaman aralığını tahmin etmek için bir sistemin ilk n başarısızlıkları arasındaki zaman aralıklarını kullanır.
- Hataların sayısı genellikle yüksek güvenilirlik sistemleri için düşüktür, dolayısıyla Sistem kullanımının başlangıcı sırasında yapılan tahminler, simülasyonlarımız tarafından onaylandığı üzere düşüktür.
- Modeli daha iyi uyarlayan ve tahminlerin doğruluğunu artıran bir model geliştirmeyi öneriyoruz. İyileştirmenin özü, zaman aralığının, zaman aralığının öngörülen medyanının gerçek değerinden daha az olduğu her durumda, bir yalancı başarısızlık getirerek, zaman aralıklarının ara arızalar arasında daha iyi bir şekilde bölünmesini içerir. Bu prosedür aynı zamanda, modelin parçalara ayırıcı hata oranlarıyla sistemleri tanımlamak için daha uygun hale getirilmesi avantajına da sahiptir.

Littlewood-Verrall Model

Varsayımlar.

- Sistemin hata fonksiyonu parçalı sabit fonksiyondur.
- Başarısızlık, başarısızlık oranının süreksizliğidir.
- Arıza oranının sabit değerleri üzerindeki belirsizlik, bir tek ortak şekil parametresi olan bir Gama öncesi dağılım ailesi tarafından tarif edilir.
- Herhangi bir onarım, sistemin güvenilirliğini geliştirmeye kıyasla daha fazladır. Bu nedenle, ölçek parametreleri, aralık numarasının bilinen bir fonksiyonudur.

n	number of failures cumulated up to the prediction time.
i	subscript used to denote the time-interval.
k	subscript used to denote a certain simulated value of a random variable.
x_i	time interval between failures or pseudo-failures.
β_i	scale parameter of the Gamma-prior assigned to a constant value of the failure rate.
α	shape parameter of the priors.
e_i	normalized error of a prediction.

$$x_{n+1;R} = \beta_{n+1}[\theta^{R^{-1/(n+1)}-1} - 1];$$

$$\theta \equiv \prod_{i=1}^{n} (1 + x_i/\beta_i)$$

Yazılım Güvenirlik Modeli Değerlendirme

- Bir yazılım birden fazla modele göre test edilmeli ve sonuçlar karşılaştırılmalıdır. Model değerlendirilirken şu sorulara cevap aranmalıdır;

1. Modele göre yazılım ne kadar geçerlidir?
2. Modeli kullanarak yazılımın güvenirliliğinin ölçümü ne kadar kolaydır?
3. Modeli kullanmak için yazılımın ne kadarına ihtiyaç vardır?
4. Modeli yazılım ölçümünde kullanmak kolay ve anlaşılır mı?
5. Model dış etkenlerden etkileniyor mu?

Yazılım Güvenirliği Sağlama Adımları

Bell laboratuarlarında yapılan bir çalışma ile yazılım güvenirliğini sağlamak için en iyi yöntemlere odaklanılmış ve yazılımın gelişiminden müşteriye teslimine kadar 25 aktivite belirtilmiştir;

- Olabilirlilik&Gereksinimler
  - İşlevsel Profili Belirlemek
  - Hataları tanımlamak ve sınıflandırmak
  - Müşteri güvenirlilik ihtiyaçlarını tanımlamak
  - Organizasyonu yönetmek

Yazılım Güvenirliği Sağlama Adımları

- Tanımlamak & Uygulamak
  - Bileşenler arasından güvenirliliği belirlemek
  - İşlevsel kaynaklara odaklanmak
  - Hatalı kodlamaları görmek
  - Yazılım güvenirliliğini ölçmek

Yazılım Güvenirliği Sağlama Adımları

- Sistem Testi
  - Testleri Yönetmek
  - Testleri Gözlemlemek
  - İhtiyaç Duyulan ek testleri belirlemek
  - Test edilen objelerin güvenirliliğini geçerlemek

Yazılım Güvenirliği Sağlama Adımları

- Teslim&Bakım
  - Yazılımı kullanıma sunmak
  - Yazılımın güvenirliliğini kontrol altında gözlemlemek
  - Müşteri memnuniyetini izlemek
  - Ürüne rehberlik etmek ve ilerleme kaydetmek

GİRİŞ

BÖLÜM HEDEFLERİ

- Test spesifikasyonu belgesi içeriğini inceleme
- Test Planı içeriğini öğrenme
- Test senaryolarını hazırlayabilmek
- Yazılım bakımı (software maintenance) kavramını anlama,
- Yazılım mühendisliğinde konfigürasyon yönetiminin yeri ve önemini anlama,
- Yazılım değişim kontrolü ve versiyon kontrolü yollarını inceleme

12. HAFTA İÇERİĞİ

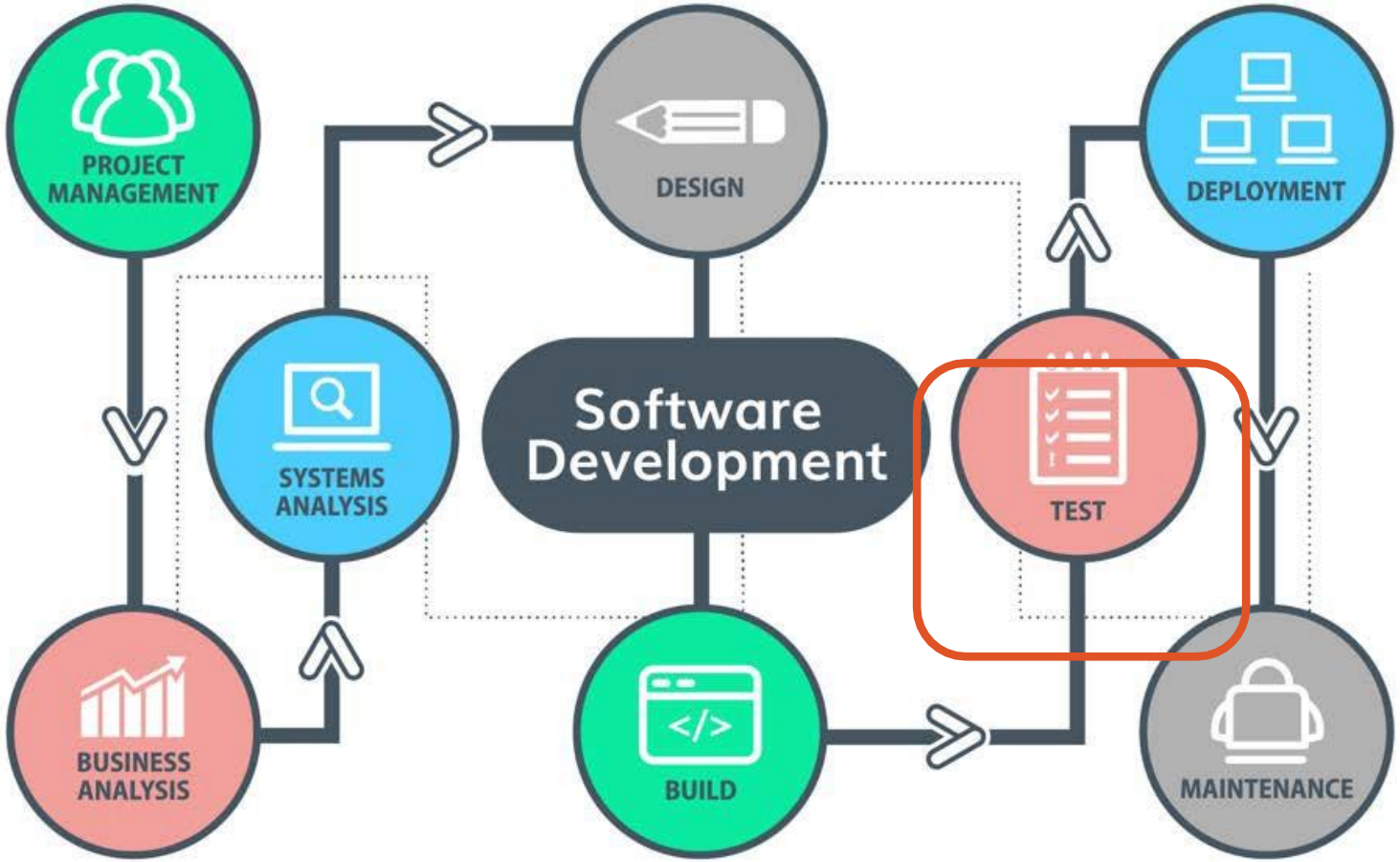
- Test Yönetimi
  - Test Planı
  - Test senaryoları
  - Test Çalışma Ekibi Yapısı

YAZILIM TESTİ

- Yazılım geliştirmenin bir parçası olarak yazılım testi çok çeşitli bir konudur - hatta bir sanat olarak da görülür[1] . Bir yazılım geliştirirken, yazılımın en az derecede hata içermesi ve kendinden beklenenleri enüst 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 amaçlar.

[1] Myers, G.J., Sandler, C.: The Art of Software Testing, 2nd edn. Wiley, Chichester (2004)

YAZILIM TESTİ



YAZILIM TESTİ

- Yazılım testi, bir Programın davranışını statik ve Dinamik yöntemlerle, Sonsuz bir küme içinden belirli Sayıda seçilen Test durumlarını kullanarak, beklenen davranışa uymadığı durumları bulma işlemidir.
- Test, hata bulma amaçlı planlı bir şekilde gerçekleşJrilen eylemler dizisi, bir doğrulama metodudur [ANSI Std –1991].
- Yazılım testi, yazılımdaki hataları bulmak, riskleri tespit etmek ve mevcut uygulamayı tanımlanan en yüksek kalite seviyesine ulaştırmak için yapılan testler bütünüdür

YAZILIM TESTİ

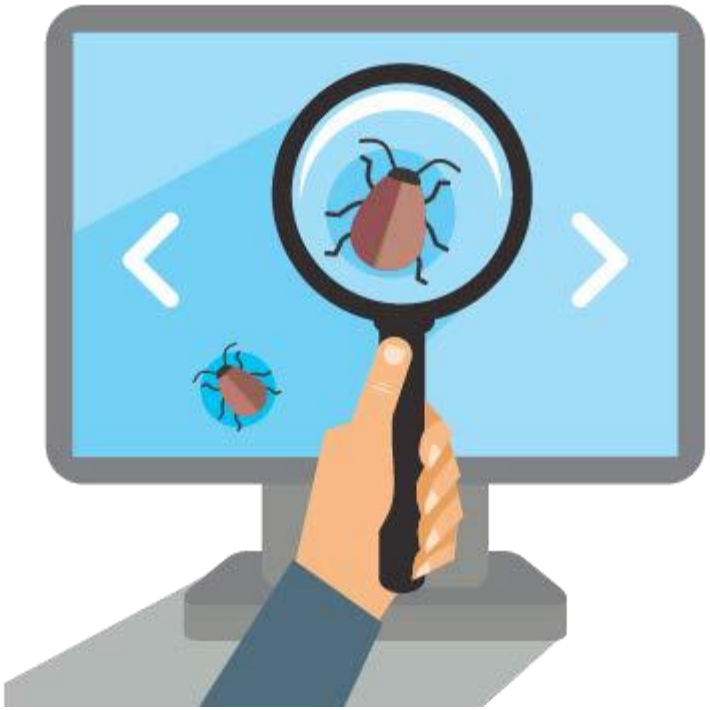
Yazılım testi;

- Yazılım içerisindeki hataların varlığını göstermeyi ve tekrarlanan hataları önlemeyi,
- Yazılım içerisindeki hataları bularak nihai üründe meydana gelecek riskleri azaltmayı,
- Kullanıcı, sistem ve yazılım belirtimlerinde sapmaları belirlemeyi,
- Yazılımı doğruluk, tamlık, güvenirlik, hızlı ve verimli çalışabilirlik, taşınabilirlik, sürdürülebilirlik, kurula bilirlik, kurtarıla bilirlik, kullanılabilirlik gibi kalite kriterleri açısından değerlendirmeyi ve yazılımı güvenilir kılmayı,

YAZILIM TESTİ

Amaçlar.

- Test edilen yazılımın kalitesini arttırmayı,
- Gereksinim belirleme, tasarım ve kodlama süreci boyunca meydana gelmiş ve gizli kalmış hataları ortaya çıkartmayı,
- Müşteriye hatalardan arındırılmış ve müşteri gereksinimlerini karşılayan bir yazılım teslim etmeyi amaçlar.



TEST SÜRECİ VE YÖNETİMİ

- Test eylemleri yazılım yaşam döngüsünün tüm aşamalarında gerçekleştirilir. Test hazırlıkları yaşam döngüsü sürecinde devam etmektedir, sadece test adımları ile sınırlı değildir

Gereksinim Analizi:

- Gereksinimlerin doğruluğuna karar vermek
- Gereksinimlerin test edilebilirliğine karar vermek
- Gereksinimin doğrulama metodolojisine karar vermek
- Test stratejisini belirleyip test planı hazırlamak

TEST SÜRECİ VE YÖNETİMİ

Tasarım

- Kabul, sistem ve tümleştirme test prosedürlerini ve test durumlarını üretmek
- Birim test stratejisini belirlemek
- Testler için gerekli test verilerini üretmek ve doğrulamak
- Hata bildirim yapısını tanımlamak ve işlerliğini denemek
- Testler için gerekli test ortamını tanımlamak

TEST SÜRECİ VE YÖNETİMİ

Kodlama

- Birim testlerin gerçekleştirilmesini izlemek ve sonuçlarını denetlemek
- Kod gözden geçirmelerini izlemek ve sonuçları denetlemek
- Bütünleştirmek ve sistem testleri için test ortamını hazırlamak

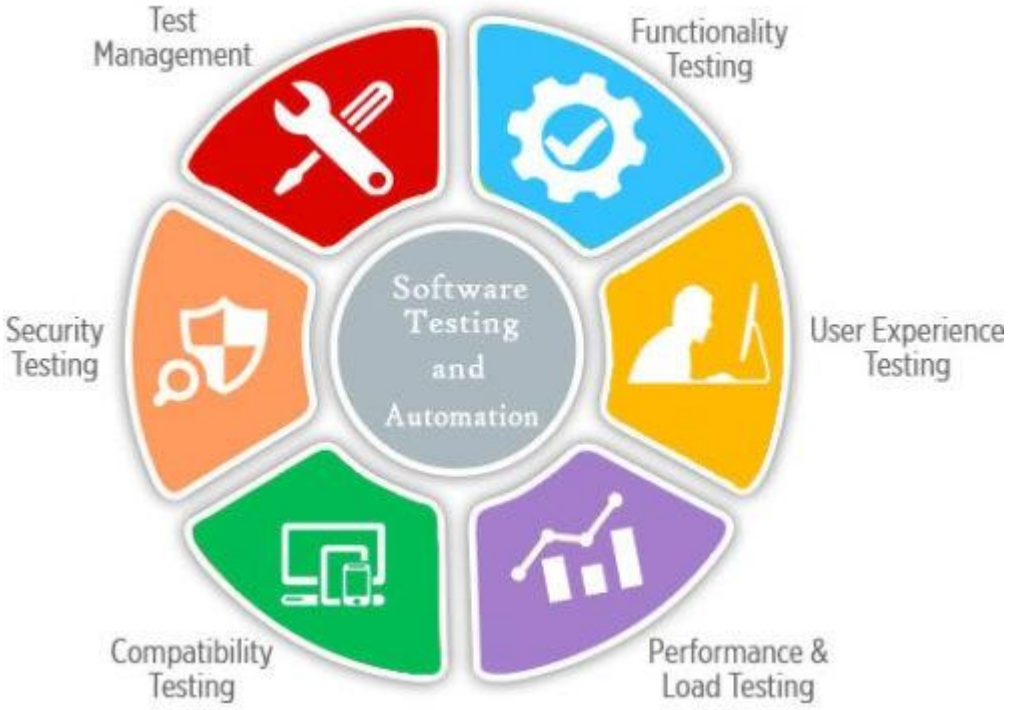
TEST SÜRECİ VE YÖNETİMİ

Test

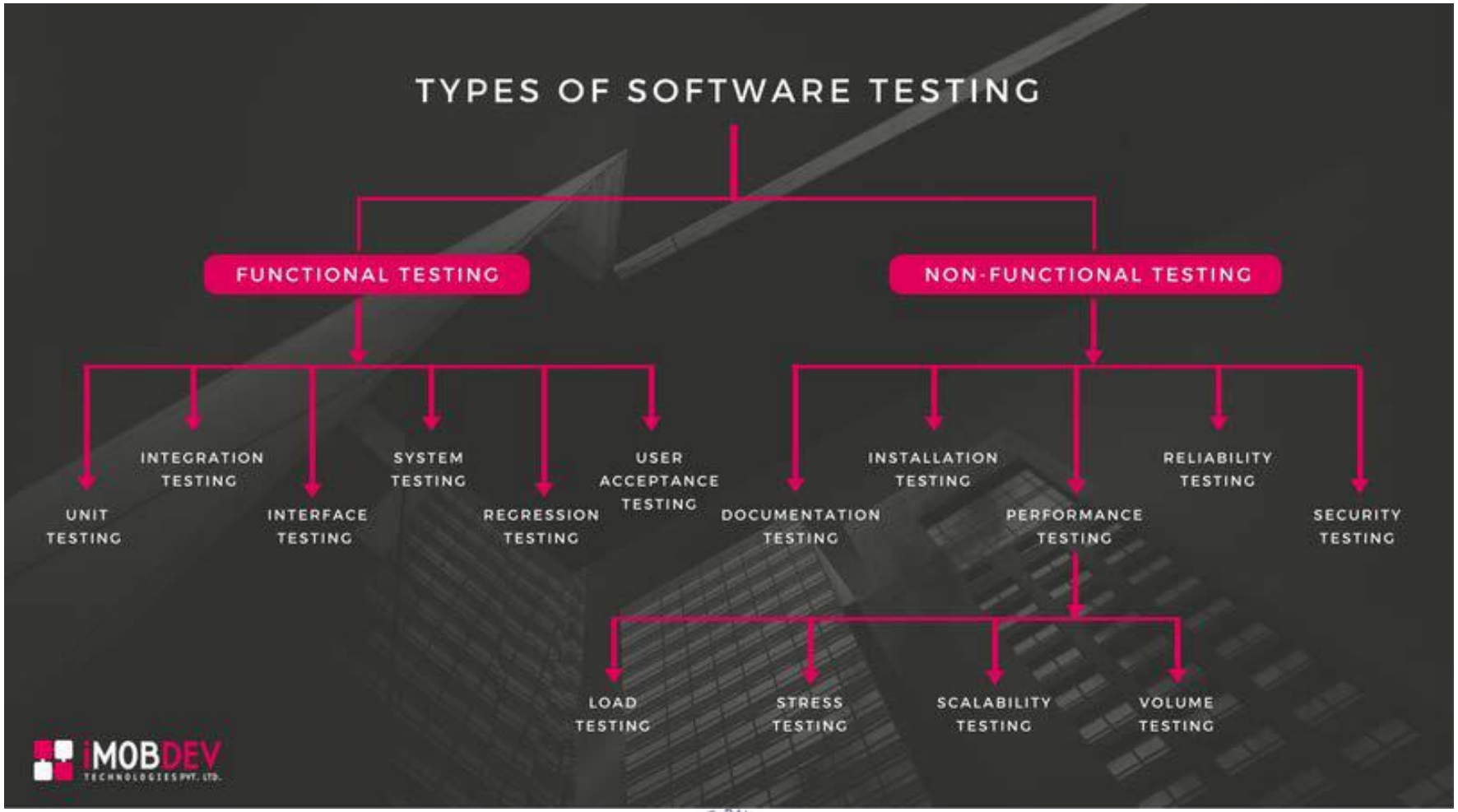
- Tümleştirme, sistem ve kabul testlerini gerçekleştirmek
- Bulunan hataları bildirmek ve düzeltildiğini denetlemek
- Yineleme testlerini gerçekleştirmek

Bakım

- Yineleme testlerini gerçekleştirmek

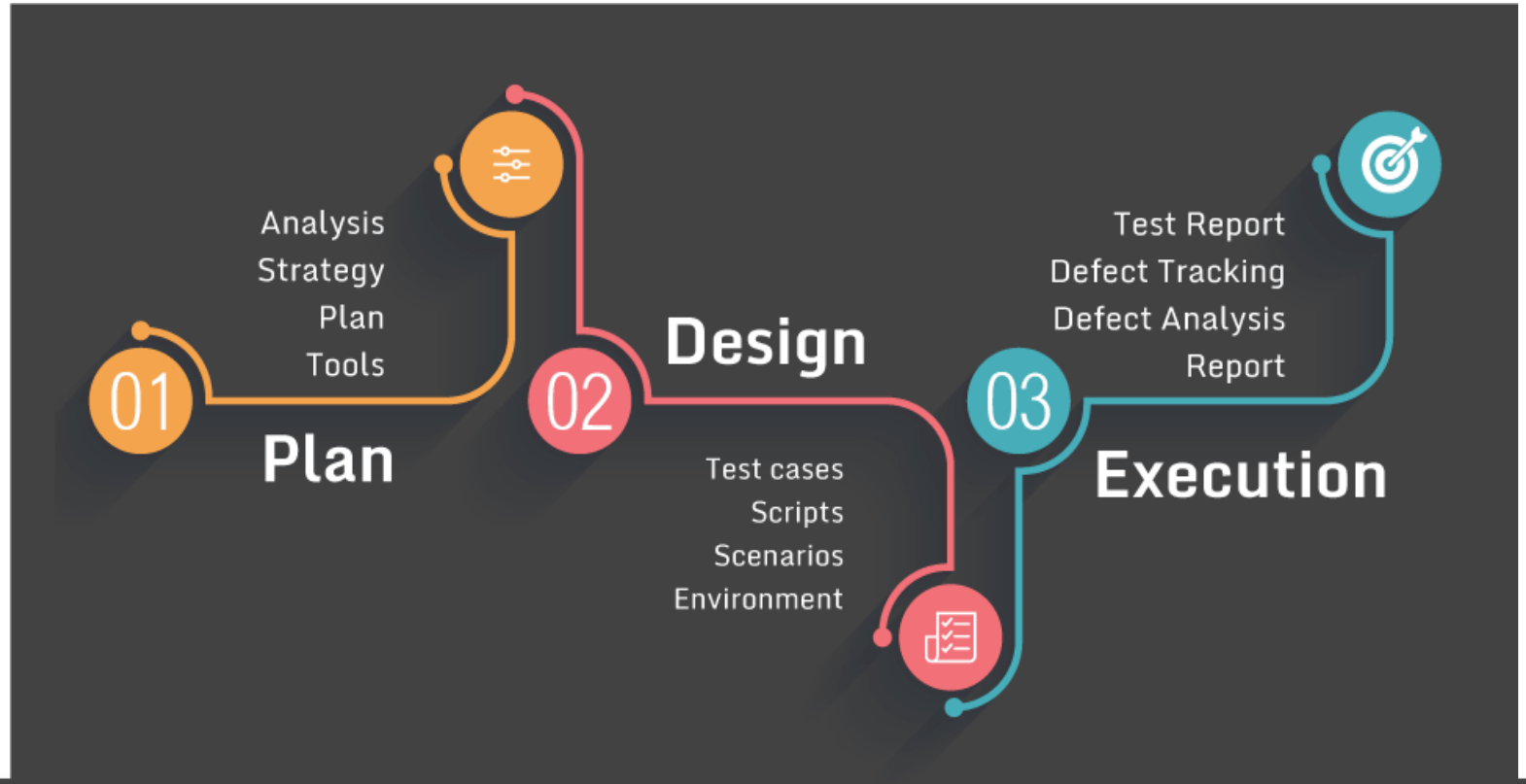


YAZILIM TESTİ



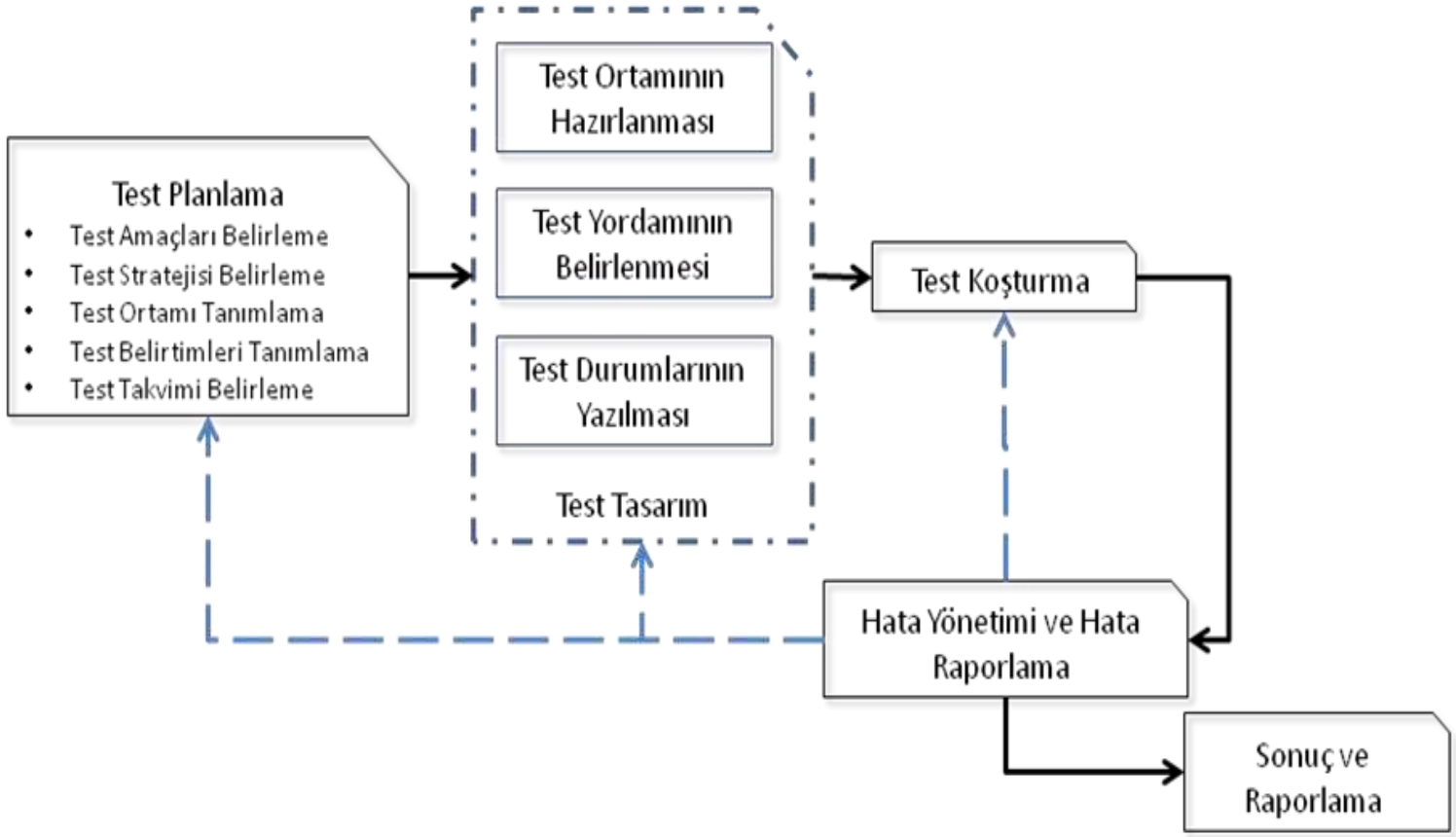
TEST SÜRECİ VE YÖNETİMİ

- Test kapsamında gerçekleştirilen işlemler temel olarak planlama, tasarım ve gerçekleştirme adımları ile yürütülmektedir.



TEST SÜRECİ VE YÖNETİMİ

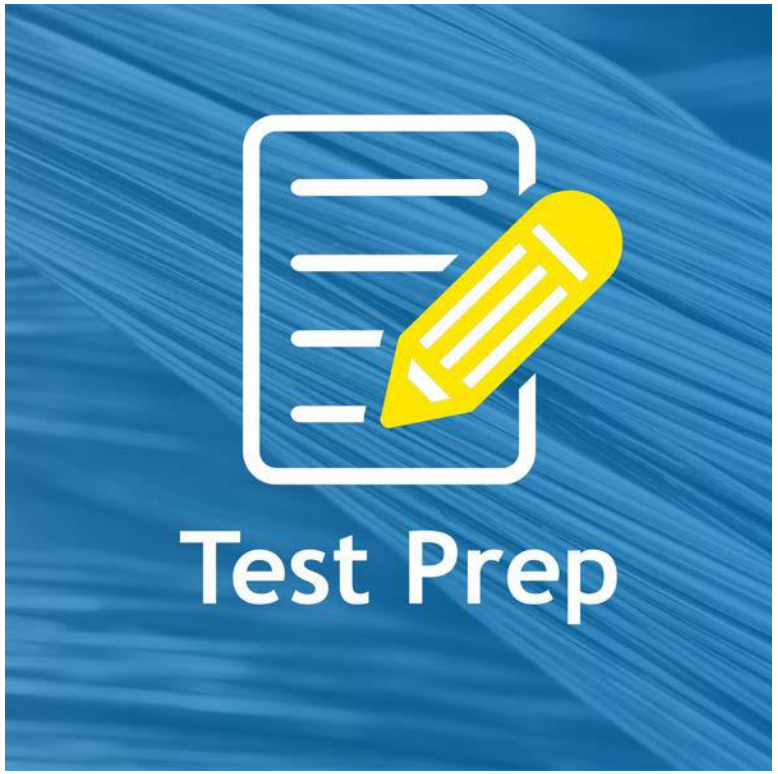
- Test kapsamında gerçekleştirilen işlemler temel olarak planlama, tasarım ve gerçekleştirme adımları ile yürütülmektedir.
- Yazılım test süreci önce planlanan, sonra icra edilip sonuçları kayıt altına alınarak belgelendirilen bir dizi eylemden oluşur. Bu süreç geliştirilen hataların yazılımdaki varlığına odaklanır.



YAZILIM TEST SÜRECİ ADIMLARI

I. TEST HAZIRLIK SÜRECİ

- Test mühendisi projenin en başından itibaren projeye dahil olarak proje amaç ve içeriğini öğrenir. Projenin kritik, riskli, kompleks kısımlarını tespit eder. Bu hazırlık projenin anlaşılması, testlerde hangi kısımların öncelikli olacağının ve hangi kısımlara yoğunlaşılacağının görülmesi açısından büyük önem taşır.



YAZILIM TEST SÜRECİ ADIMLARI

2. TEST PLANLAMA

- Yapılacak testler projenin başında planlanmalı ve belgelenmelidir. 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 dokümandır.
- Test planı geliştirilirken basit, tam, anlaşılır, güncel ve içerdiği eylemler bakımından kabul edilebilir olmalıdır.

YAZILIM TEST SÜRECİ ADIMLARI

2. TEST PLANLAMA

- Projeye ait Teknik Analiz, İş Analizi ve Gereksinim dokümanları incelenir. Roller ve sorumluluklar belirlenir. Risk analizi yapılır. Testte kritik ve öncelikli alanlar tespit edilir. Test çıkış kriterleri belirlenir.

YAZILIM TEST SÜRECİ ADIMLARI

2. TEST PLANLAMA

1. RİSKANALİZİ

- Ürün riski analizi ürününperformansı, güvenliği, kullanışlılığı, genişletilebilirliği, veri bütünlüğü, test edilebilirliği ile ilgilidir. Risk analizi testin ne kadar sürmesi gerektiğinin belirlenmesinde kritik girdi sağlar.
- Testte isabetli önceliklendirme yapılmasını sağlar.Testte hangi test tekniklerin uygulanacağı ve test yapılacağına karar verilmesinde etkili olur.
- Testin amaçlarından biri de risklerin anlaşılması en az seviyeye indirgenmesidir.
- Riskler belirlenir, riskler önceliklendirilir ve gerekli önlemler alınır.

YAZILIM TEST SÜRECİ ADIMLARI

2. TEST PLANLAMA

2. TEST ÇIKIŞ KRİTERLERİNİN BELİRLENMESİ

- Her Yazılım Test Ekibinin kendine özgü test çıkış kriterleri olabilir.
- Temel olarak kriterleri aşağıdaki gibi sıralayabiliriz.
  - Hazırlanan tüm senaryolar test edilmiş olmalıdır.
  - Hata kayıtları çözülmüş olmalıdır.
  - Uygulama gerçek ortamdan önce preprod ortamında test edilmelidir.
  - Uygulamanın gerçek ortama alınmasının risk içermediğinden emin olunmalıdır.
  - Test hedefine ulaşmış olmalıdır.
- Bunların haricinde uygulamanın özellikleri ve uygulamanın risklerine göre o projeye özgü kriterler de belirlenir.

YAZILIM TEST SÜRECİ ADIMLARI

2. TEST PLANLAMA

- Test planı içeriği,
- Test Stratejisi ve Test Edilecek Öğeler
- Doğrulama Yöntemleri
- Testlerin Tamamlanma Kriterleri
- Hata ve Test Sonuç Raporlama
- Test Sorumlulukları
- Test Ortamı
- Eleman ve Eğitim İhtiyacı
- Test Takvimi
- Risk Yönetimi
- Test Çıktıları

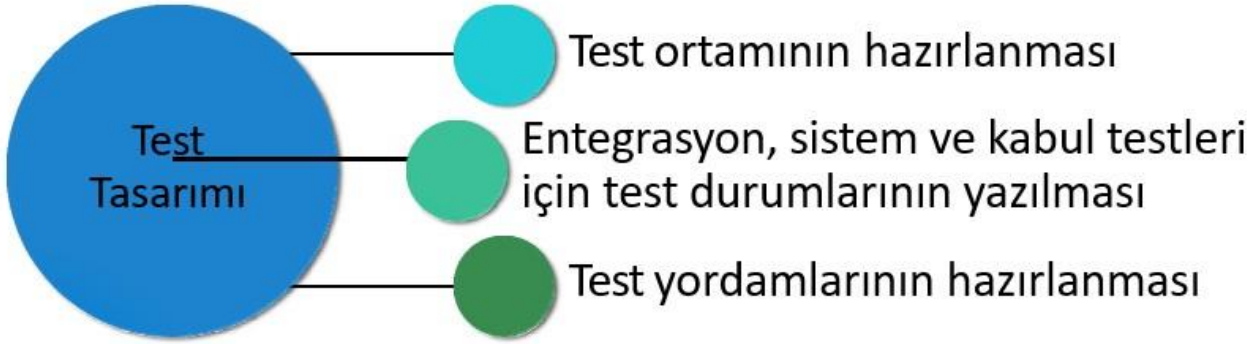
Şeklinde hazırlanmalıdır. Ancak bu plandaki ayrıntılar değişebilmektedir.

YAZILIM TEST SÜRECİ ADIMLARI

3. TESTTASARIMI

- Test planlama süreci tamamlandıktan sonra test tasarım süreci başlar. İhtiyaçlar, mimari, tasarım, kullanıcı arayüzü, test verileri, test ortamı ve test araçları dikkate alınarak uygulanacak test seviyeleri, çeşitleri ve teknikleri belirlenir. Test senaryoları hazırlanır ve proje bağlamına göre ve amaca göre çeşitlendirilir.

- Bu süreçte aşağıdaki görevler icra edilir:



YAZILIM TEST SÜRECİ ADIMLARI

3. TESTTASARIMI

TEST ORTAMI

- Testler test mühendisleri tarafından tanımlanan yazılım ve donanımdan oluşan bir ortamda gerçekleştirilir. Test ortamı hazırlanırken testlerde kullanılacak olan yardımcı test yazılımlarıda geliştirilir veya hazırlanır.Yazılım testlerinde kullanılan yardımcı yazılımlar:

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

YAZILIM TEST SÜRECİ ADIMLARI

3. TESTTASARIMI

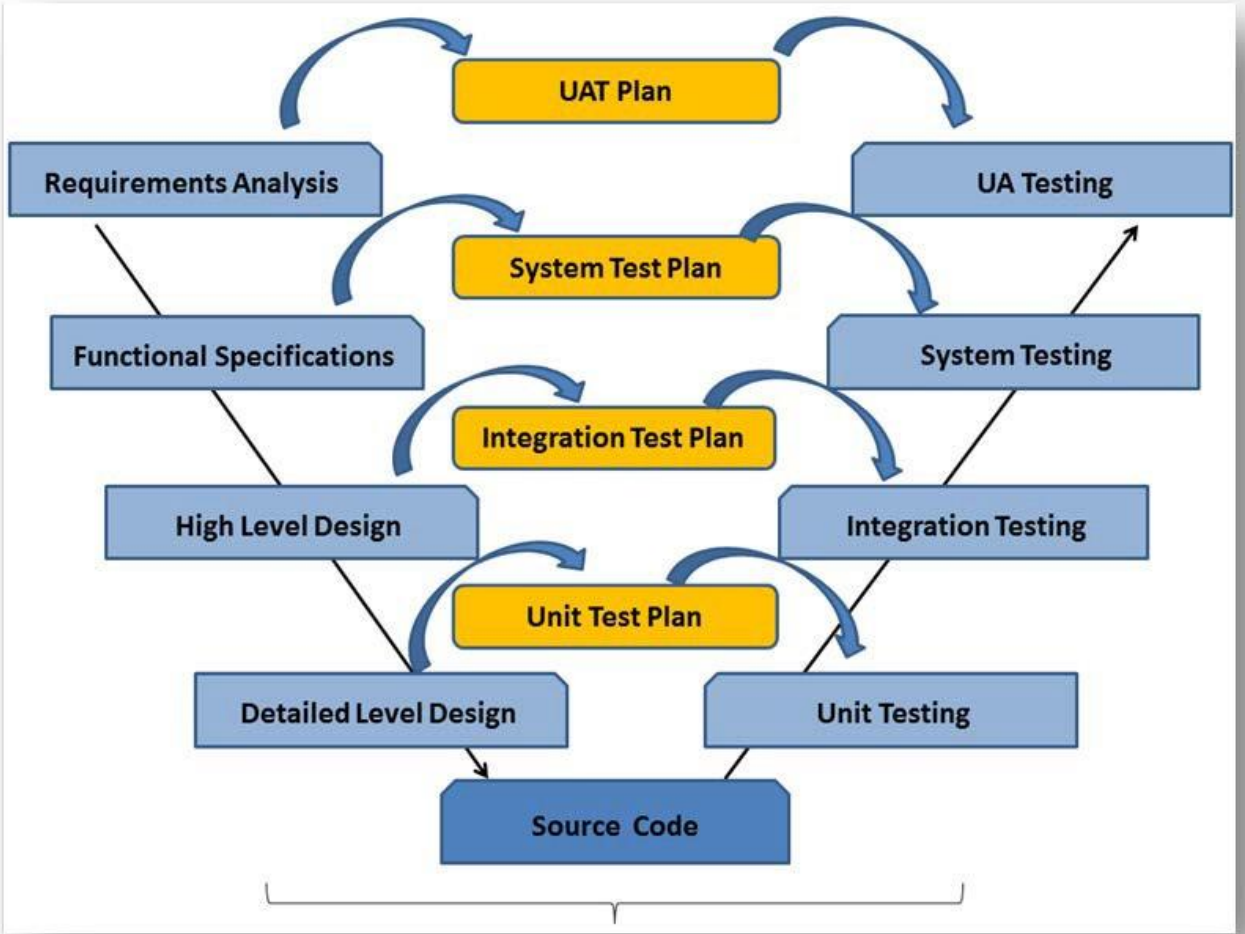
- Emülatör ve Simülatörler: Yazılımların ihtiyaç duyduğu gerçek donanımları taklit eden ve testler için gerekli olan donanım verilerini sağlayan yazılımlardır.
- Test Verisi Üreteçleri: Test durumlarının koşturulması için gerekli olan test girdi verilerini üreten yazılımlardır.
- Hata Ayıklayıcılar(Debugger): Testler sırasında karşılaşılan hataların kaynak kod üzerinde bulunmasını sağlayan yazılımlardır

YAZILIM TEST SÜRECİ ADIMLARI

3. TESTTASARIMI

TEST DURUMLARININYAZILMASI

- Test durumu, belirli bir program parçasının çalıştığının veya bir gereksinimin doğrulandığının gösterilmesi için kullanılan girdiler, gerçekleştirilmesi gereken adımlar ve beklenilen tüm sonuçların belirtilmesidir. Test durumları testin en küçük parçasıdır.



YAZILIM TEST SÜRECİ ADIMLARI

3. TESTTASARIMI

TEST DURUMLARININYAZILMASI

- Test ekibi resmi hale gelen gereksinimlerden test durumu oluşturmaya başlar.
- Bir gereksinim, bir test durumu ile doğrulanabildiği gibi birden fazla test durumu ile de doğrulanabilir. Bu amaçla, geliştirilen yazılımın kendi belirtimlerinin tümünü karşıladığını göstermek için her bir gereksinime en az bir tane test durumu yazılmalıdır.
- Geliştirilen yazılımın kendinden beklenen tüm davranışları gerçekleştirdiğinin göstergesi, yazılıma ait tüm test durumlarının testlerden geçmesidir.

YAZILIM TEST SÜRECİ ADIMLARI

3. TESTTASARIMI

TEST DURUMLARININYAZILMASI

- Test ekibi resmi hale gelen gereksinimlerden test durumu oluşturmaya başlar.
- Bir gereksinim, bir test durumu ile doğrulanabildiği gibi birden fazla test durumu ile de doğrulanabilir. Bu amaçla, geliştirilen yazılımın kendi belirtimlerinin tümünü karşıladığını göstermek için her bir gereksinime en az bir tane test durumu yazılmalıdır.
- Geliştirilen yazılımın kendinden beklenen tüm davranışları gerçekleştirdiğinin göstergesi, yazılıma ait tüm test durumlarının testlerden geçmesidir.

YAZILIM TEST SÜRECİ ADIMLARI

3. TESTTASARIMI

TEST DURUMLARININYAZILMASI

Bir test durumu adım adım bir testin nasıl icra edileceğini tanımlar. Bir test durumunda olması gerekenler:

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. Yazılım sürüm tanımı
7. Yazılımın çalışma ortamı ve test ID

YAZILIM TEST SÜRECİ ADIMLARI

3. TESTTASARIMI

TEST DURUMLARININYAZILMASI

- Test durumlarında bilinen girdilere yer verilmelidir. Bu girdiler ile beklenen çıktılar test durumları içerisinde doğrulama noktası olarak verilir. Eğer beklenen girdiler ile beklenen sonuç yazılım tarafından verildi ise ilgili test durumu geçmiştir. Test edilen öğeye ait test durumlarının tamamının veya belirlenen bir oranının testten geçmesi ile gerçekleştirilen testler başarılı sayılabilir. Ancak bu karar projenin test stratejisi kapsamında belirlenmelidir.

YAZILIM TEST SÜRECİ ADIMLARI

3. TESTTASARIMI

TESTYORDAMI

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

- Bu belge içerisinde testin gerçekleştirilmesi için adım adım tanımlanmış ayrıntılı açıklamalar vardır.

YAZILIM TEST SÜRECİ ADIMLARI

3. TESTTASARIMI

TESTYORDAMI

Örnek bir test yordamı:

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

YAZILIM TEST SÜRECİ ADIMLARI

3. TESTTASARIMI

TEST SENARYOLARI

- Test tasarımı sırasında, . Test senaryoları hazırlanır ve proje bağlamına göre ve amaca göre çeşitlendirilir.
- Test senaryoları belli test seviyelerinde belli test çeşitleri manuel veya test otomasyonları ile uygulanır.

Test Senaryoları:

- Test Senaryosu Adı, kimliği
- Yazarı, Tarih
- İlgili gereksinimler/Testin Amacı
- Ön Koşul /Varsayımlar
- Test Girdileri
- Test Senaryosu Adımları
- Beklenen sonuçlar

YAZILIM TEST SÜRECİ ADIMLARI

- **4. TEST GERÇEKLEŞTİRME**

- Test koşturmada genel olarak şu adımlar izlenir.

- Yazılım ekibi tarafından test edilecek yazılım(yük) oluşturulur.
- Testçiler gelen yazılıma uygulanacak olan test durumları
- Yazılımın test edilebilir olduğuna karar verilir.
- Yazılım teste kabul edilirse test başlar.
- Testlerde bulunan hatalar raporlanarak yazılım ekibine bildilirilir.
- Bulunan hatalar yazılım ekibi tarafından düzeltilir ve yeni sürüm hazırlanır.
- Testçiler yeni gelen yük ile yineleme testlerini gerçekleştirir.
- Bu adımlar müşteriyeson kabul edilebilir yazılım verilinceye kadar devam

TEST ÇALIŞMA EKİBİ YAPISI

- Yazılımda kalitenin sağlanması için, belirlenen standartlara uygun olarak belgelemenin düzenli bir biçimde hazırlanmasının, yazılım geliştirme çalışmalarının yönetim tarafından daha kolay izlenmesine olanak vereceği ve üretimin sürekliliğini sağlayacağı düşünülmektedir.
- Yazılımın sınanması sürecinde, sınamada bağımsızlık özelliğinin sağlanması için, yazılımın sınanmasından sorumlu olan kişiler, yazılımı tasarlayan ve oluşturan gruptan olmamalıdır. Yazılım sınamada işlemlerinin geliştirme sürecinin her aşamasında, yazılım üretim ekibi dışında hazırlanmasının sağlanması da bu belgeleme düzeni ile kolaylaşacaktır.
- Test ekibinin görevi, Planlarda belirtilen testler için gerekli test hazırlıklarını yapmak, testleri yürütmek, testlerde bulunan hataları yazılımcıya bildirmek, izlemek test sonuçlarını raporlamaktır.

TEST ÇALIŞMA EKİBİ YAPISI

- İyi bir testçi ne zaman mükemmele ulaşılmayacağını, iyinin yeterli olacağını bilir.
- Her Yazılım Projesindeki optimum test miktarı farklıdır.
- Test hataları (bug) arar, Testçiler üründe bug bırakmamaya, mümkün oldukça daha erken bulmaya çalışır, ama kalite sorumlusu değillerdir.