Kalite nedir?

En basit tanımıyla kalite, müşteri isteklerine cevap verebilmektir.

- Amaçlanan kullanıma uygunluktur. (JosephM. Juran)
- Gereksinimlere uygunluktur. (Philip B. Crosby)
- Nesnenin tabiatında var olan özelliklerin gereksinimleri karşılama
- derecesidir. (ISO 9000)
- Bakımın sonucudur. (Robert Pirsig)

Standartlara sadık kalma önemlidir.
Kalite ve Test bizim için kritiktir!

Kalite Kontrol: Hata ayıklamaya yönelik, sistematik ve belirli bir anda yapılır.

Kalite güvence: Hata oluşmasını önlemeye yönelik, sistematik, zamana yayılmıştır.

Kalite, iyi proje yönetiminin ve katı mühendislik kurallarının uygulanması sonucunda ortaya çıkar. Aşağıdaki maddeler çerçevesinde uygulanır:

- Yazılım mühendisliği metotları
- Proje yönetim teknikleri
- Kalite kontrol eylemleri
- Yazılım kalite güvencesi

Kalite güvence yönetimi sayesinde:

- Geliştirilen yazılımın sonradan ortaya çıkan kusurları azaltılmış olur.
- Test ve bakım aşamalarında daha az işgücü gerekir.
- Yüksek güvenilirlik müşteri memnuniyetini artırır.
- Maliyet düşer.
- Çalışanlar arasında kurulan iş disiplini sayesinde verimlilik artar.

Kalite Ölçütleri:

- Kullanmaya yönelik ölçütler
- Taşınmaya yönelik ölçütler
- Yenileştirmeye yönelik ölçütler

İşlevsellik: Yazılım kullanıcının tüm isterlerini karşılayabilmelidir. Ana işlevler yanında yardımcı işlevlerle zenginleştirilmiş olmalıdır.

Doğruluk: Yazılımın öngörülen tüm işlevleri istenildiği şekilde, doğru ve yeterli hassaslıkta yerine getirebilmesidir.

Güvenilirlik: Yazılım güncel bir sürüme sahip olmalı, sürekli ve hatasız çalışabilmeli, tüm işlevleri doğru yapmalı, hatalı girdilere ve kullanıcı yanlışlıklarına karşı korumalı olmalıdır.

Taşınmaya Yönelik Özellikler

Taşınabilirlik: Yazılımlar farklı bilgisayar donanım ve yazılımının kullanıldığı ortamlara aktarılabilir olmalıdır. Yeni işletim sistemi sürümlerinin kullanılması, bilgisayar değiştirilmesi gibi durumlarda, daha önce geliştirilmiş olan yazılım en az çaba ile tekrar kullanılabilmelidir.

Uyumluluk: Bir yazılım ürünü daha önce üretilmiş olan veya beraber çalışan diğer ürünlerle tam uyumlu olmalıdır. Birbiriyle etkileşen sistemler ortak özelliklere sahip olarak yaratılmalıdır. Bunun için dosya biçimleri, veri yapıları, kullanıcı ara yüzleri üzerinde belirli bir standart oluşturulmalı, tüm yazılımlar bunlara uygun olarak tasarlanmalı ve geliştirilmelidir.

Tekrar kullanılabilirlik: Yeni geliştirilen yazılım daha sonra kısmen veya tamamen yeni bir uygulamada kullanılabilmelidir. Tasarım ve gerçekleştirim bu amaca hizmet edecek şekilde olmalıdır.

Garvin'in Kalite Ölçütleri:

- Performans kalitesi
- Dayanıklılık
- Özellik kalitesi
- Kullanılabilirlik
- Güvenilirlik
- Estetiklik
- Uygunluk
- Algılama

McCall' un Kalite Faktörleri:

- Ürün Taşıma
  - Taşınabilirlik
  - Yeniden Kullanılabilirlik
  - Birlikte Çalışabilirlik

- Ürün İşletme
  - Doğruluk
  - Güvenilirlik
  - Verimlilik
  - Bütünlük
  - Kullanılabilirlik

- Ürün İnceleme
  - Bakım Kolaylığı
  - Esneklik
  - Test Edilebilirlik

ISO 9126 Kalite Faktörleri:

- Kullanılabilirlik
- Taşınabilirlik
- Verimlilik
- Güvenilirlik
- Fonksiyonellik
- Bakım Kolaylığı

Kalitenin Maliyeti:

Kaliteli yazılım üretmenin maliyeti, düşük kaliteli yazılım üretmenin maliyetinden daha düşüktür.

İstatistiksel Yaklaşım:

- Yazılım kusurları toplanır ve sınıflandırılır.
- Her kusur izlenerek neden kaynaklandığı bulunur ve bu
- Bilgiler de kaydedilir.
- Neden olan ortak kusurlar bulunur.
- Bulunan kusurlar düzeltilir.

Yeni bir hata ortaya çıktığında çözümün nerede aranması gerektiği konusunda bir fikir oluşur.

Temiz Oda Süreci:

Amaç hatanın oluşabileceği ortamı ortadan kaldırmaktır.

Yardımcı Araç Desteği:

Sıkça kullanılan yazılım modüllerinin ortak hale dönüştürülmesi ve kalıpsal bir yapıya sokulması hem hataları azaltır hem de geliştirme süresini düşürür.

Nitelik Sistem Standartları:

Yetenek Olgunluk Modeli- (CMM) : Bir süreç modeli olup, örgütlerin süreçlerinin olgunluğunu değerlendirme modelidir.

- 5. Optimizing----------En İyilenen
- 4. Quantitatively Managed---------Nicel Olarak Yönetilebilen
- 3. Defined----------Tanımlı
- 2. Managed----------Yönetilen
- 1. İnitial----------Başlangıç

Yukarıya doğru çıktıkça değerlenir.

Trillium: Teknolojik olgunluk içeren ve iyileşmeyi bu olgunlukla düzenleyen yol haritası yaklaşımı getiren.

Spice: İki boyutlu bir model olup, Birinci boyutta süreçler, ikinci boyutta yetenek düzeyleri vardır.

TickIT: Değerlendirme en az iki aşamalı bir süreçtir. İlkinde örgütün nitelik sistemi, standarda göre değerlendirilir. İkincisinde, örgütün pratikte gerçekten kendi nitelik sistemine ve standarda uyumlu çalışıp çalışmadığı denetlenir, nitelik sisteminin etkinlik derecesine bakılır.

ISO 9000: ISO tarafından yayınlanan nitelik sistemi ve güvencesi standartları büyük ilgi görmüştür. Beş temel ISO standardı vardır:

- **ISO-9000---------- Nitelik yönetimi ve nitelik güvence standartları seçim ve kullanım rehberi**
- ISO-9001---------- Nitelik Sistemleri- Tasarım, geliştirme, üretim, tesis ve servis için nitelik güvence
- ISO-9002---------- Üretim ve tesis için nitelik güvence
- ISO-9003---------- Nitelik Sistemleri- Son muayene ve testlerde nitelik güvence
- ISO-9004---------- Nitelik yönetimi ve nitelik sistemleri öğeleri

AQAP-150/160: Askeri amaçlı olarak geliştirilen ürünlerin ortak özellikler arasında ileri teknoloji kullanımı, sürekli artan karmaşıklık düzeyleri, yüksek maliyet gerektirmeleri sayılabilir.

Başarısızlığın ana sebepleri:

- Müşterinin isteklerini doğru analiz edememek;
- Proje için uygun ekibi kuramamak;
- Yanlış teknoloji ve mimari seçimleri;
- Geleneksel yöntemlerin eksiklikleri;
- Müşteriyle iletişimden kaçınmak vs.

Çevik Yazılım Yöntemi:

Çok Önemli:

- Bireylerle Etkileşim
- Çalışan Bir Yazılım
- Müşterilerle İşbirliği
- Değişikliklere Uyum Sağlama

Az Önemli:

- Süreç Ve Araçlar
- Detaylı Belgelendirme
- Sözleşmedeki Kurallar
- Belirli Bir Plan

Çevik Yazılım Aşamaları

- Hızlı, devamlı ve kullanışlı yazılım üreterek müşteri memnuniyeti sağlamayı amaçlar.
- Geliştiriciler ile iş adamları arasında günlük ve yakın işbirliği bulunmalıdır.
- Çalışan yazılım gelişimin en önemli ölçüsüdür.
- Taleplerdeki geç değişikliklerin de memnuniyetle karşılanır.
- Yüz yüze görüşme iletişimin en güzel yoludur.
- Kendi kendini organize eden takım yapısı gereklidir.
- Basitlik önemlidir.

Yazılımın geliştirilmesindeki geri dönüş (feedback) ve değişikliklere uyum sağlamak son derece önemlidir.

Temel prensipler:

- Müşteriyi memnun etmek
- Değişen ihtiyaçları karşılamak
- Sık aralıklarla ürün teslimi yapmak
- Yüz yüze iletişime önem vermek
- Sürdürülebilir gelişmeyi desteklemek
- Teknik mükemmeliyete, iyi dizayna ve sadeliğe odaklanmak
- **Kendi kendine organize olan takımlar kurmak ?**

Çevik Model Takımları:

- Bir araya gelmiş,
- Kendi kendilerine organize olan,
- Çapraz fonksiyonlu,
- İşine odaklanmış,
- Hedefleri net olan,
- Teslim edilebilecek düzeyde urun
- Ortaya koyabilen
- Küçük(3-7 kişilik) gruplar

Çağlayan modeli:

Bu model yazılım projesini baştan sona planlar. Gelişim, sunulabilir işler acısından ölçülür: talep açıklamaları, tasarım dokümanları, test planları, kod incelemeleri vb. Bu durum belli aralıklara bölünmeye uygun değildir ve ilerideki değişikliklere uyum gösterilemez.

Geleneksel Yöntemler:

- Müşteriler ne istediğini iyi bilir.
- Geliştiriciler neyi, ne şekilde üreteceklerini iyi bilir.
- Bu yol boyunca hiçbir şey değişmeyecektir.

Çevik Yöntemler:

- Müşteriler ne istediğini keşfeder.
- Geliştiriciler neyi nasıl üreteceğini keşfeder.
- Bu yol boyunca bir çok değişiklik yapılabilir.

Scrum Takımı: Urun Sahibi, Geliştirme Ekibi ve Scrum Master'dan oluşur. Takım kendi kendini örgütler. Böylece kendi içerisinde uyum içinde olan takımlar daha başarılı sonuçlar alırlar. Scrum takım modeli esneklik, yaratıcılık ve verimliliği optimize etmek için tasarlanmıştır.

Backlog:

- Müşteriden ve son kullanıcıdan gelen gereksinimleri içerir.
- "Ne yapacağız" sorusunun yanıtını içerir.
- Herkese açık ve herkes tarafından müdahale edilebilir.
- Risk, iş değeri, zaman gibi kavramlara göre ürün sahibi tarafından sıralandırılır.
- User Story'lerden oluşur.

Sprint:

- Belirli bir sureye sahiptir.
- Sonunda ortada değeri olan bir çıktı olmalıdır.
- Toplantılarla içerik belirlenir.
- Sprint suresi boyunca her gün toplantılar yapılır.

User Story: Müşteri, son kullanıcı veya urun sahibi için değerli olan ve anlam ifade eden genellikle fonksiyonel özelliklerin belirtildiği ifadelerdir.

As X ---------- I want Y ---------- so that Z

Poker Kartları: Scrum takım üyeleri bir araya gelir. Scrum master bir user story okur. Takımdaki her bir üye user story için uygun gördüğü poker kartlarından birini seçer. Herkes kartları seçtikten sonra tüm kartlar açılır ve değerlendirilir. Böylece herkesin ortak görüşü sonunda user story'lerin büyüklüğü belirlenir.

Yazılım Testi:

Müşteri memnuniyetini en üst seviyeye çıkarmak için yapılır.

Burada unutulmaması gereken şey mükemmel yazılım olmadığı ve bir yazılımın asla %100 test edilemeyeceğidir.

Yazılımın doğrulanması ve geçerlenmesi süreci olarak da tanımlanabilir.

Testin Amaçları:

- Müşteriye sunmadan önce ürün kalitesinden emin olmak,
- Yeniden çalışma ve geliştirme için masrafları azaltmak,
- Geliştirme işleminin erken aşamalarında hataları saptayarak ileri aşamalara yayılmasını önlemek, böylece zaman ve maliyetten tasarruf sağlamak,
- Müşteri memnuniyetini arttırmak ve izleyen siparişler için zemin hazırlamak.

Yazılım testinde neler test edilmektedir?

- İş gereksinimleri
- İşlevsel tasarım gereksinimleri
- Teknik tasarım gereksinimleri

- Düzenleyici gereksinimler
- Yazılımın kaynak kodu
- Ortakların standartları
- Donanım yapılandırılması ve dil farklılıkları

Doğrulama: Ürün Doğru Üretiliyor mu?
Geçerleme: Ürün Doğru Çalışıyor mu?

Doğrulamayı geçse de geçerlemeyi geçemeyebilir.
Doğrulamayı geçemezse geçerlemeyi de geçemez.
Doğrulama da geliştirici çalışır.
Geçerlemede test ekibi çalışır.

Doğrulama Süreci:

Sözleşme Doğrulaması: Geliştirici ile müşteri arasında yapılan sözleşme kontrolü
Süreç Doğrulaması: Proje planlamaları uygun olup olmadığı kontrol edilir.
İsterlerin Doğrulanması: Müşteri isterleri gerçekleştirilebilir ve test edilebilir olmalıdır.
Tasarım Doğrulaması: Göze hitap eden görselleri içermesi kontrol edilmesi.
Kod Doğrulaması: Belli şartlara uygun olmalıdır. Anlaşılır olmalıdır.
Belgelendirme Doğrulaması: Belgelendirmeler açık anlaşılır ve net olmalıdır.

Yazılım Hataları:

Yazılım geliştirme süreci boyunca hatalardan kaçınmak imkansızdır.

Yazılım kalitesi konusunu anlamanın anahtarı failure, fault ve error terimlerini anlamaktır.

Error: Kodlayıcı kaynaklı, doğru olmayan bir sonuç elde edilmesi.

Failure: Sistemin veya bir parçasının gerekli fonksiyonu yeterli performansta yerine getirememesi.

Fault: Bir yazılım içerisindeki doğru olmayan adım, işlem veya veri tanımı.

Hata Ayıklama (Debugging):

Hata ayıklama başarılı bir test sürecinin sonucudur. Şöyle ki, bir test sonucunda hatalar bulunduğu zaman hataların giderilme işlemine hata ayıklama(debugging) denir.

Brute Force: Yürütme anındaki davranışlar izlenir, yazılım biriminin çeşitli noktalarına ekrana veya bir dosyaya o an akışın neresinde olduğunu, genel durumunu veya bir değişkenin değerini yazan deyimler eklenir. Bu bilgiler ışığında, hataya neden olan yazılım kusuru aranır.

Backtracking: Kodun okunarak geri izlenmesi esasına dayanır. Hatanın oluştuğu yerden itibaren geriye doğru gidilerek kod incelenir; hata yaratan deyim ya da kusurlu akış mantığı aranır. Küçük yazılımlarda yaygın olarak kullanılır. Yüksek satır ve modül sayısına sahip yazılımlarda, akış yolunun çok dallandığı noktalarda bunu yapmak oldukça zordur.

Cause elimination: Tümevarım veya tümdengelim yöntemlerine dayanarak elde edilen verilere göre hatanın nedeni araştırılır. Ortaya konan varsayımları doğrulayıcı ya da çürütücü testler tasarlanır. İlk testler olumlu sonuç verirse, daha ayrıntılı verilerle testlere devam edilerek hatanın tam yerinin saptanmasına çalışılır.

Testi Kim Yapar?

- Yazılım test ekibi
- Yazılım geliştirici
- Proje lideri
- Son kullanıcı

Test yapan kişilerin,

- Tedbirli,
- Meraklı,
- Eleştirmen
- İyi iletişim kurabilen kişilikte olmaları gerekir.

Yazılım Geliştirme Metodolojilerinde Test

Şelale Modeli: Test aşamasına gelebilmek için diğer aşamalar tamamlanmalıdır.

Agile: Test profesyonelleri, yazılım geliştirme yaşam döngüsünün en başından itibaren sürece dahil olurlar. Test güdümlü geliştirme yöntemini(TDD) kullanır.

V modeli: Bu modelde geliştirme ve test paralel şekilde yapılır. iyi tanımlandığı, belirsizliklerin az olduğu ve aşamalar halinde ilerlenmesi gereken projelerde «v» modeli iyi sonuç verir.

Spiral Model(Helozonik): Aynı safhalara geri dönülmesinin bir zorunluluk olduğunu vurgular. Proje çevrimlere ayrılır ve her bir çevrimin riskleri ayrı ayrı ele alınır.
4 Eksenden oluşur. Planlama, Risk Analizi, Üretim, Kullanıcı Değerlendirme
Çalışan sistemlerde Helozonik model kullanılır.

Test Tipleri

Manuel Test: Yazılımın herhangi bir test otomasyonu veya herhangi bir script kullanılmadan el ile ile test edilmesidir. Yazılımdaki herhangi bir beklenilmeyen davranışı veya hatayı bulmak için yapılır. Küçük projelerde kullanılması daha uygundur.

Otomasyon Testi: Otomasyon testi, test senaryolarının hızlı, art arda ve tekrarlarca uygulanabilmesini sağlar. Yük, performans, stress gibi çok kullanıcı gerektiren testlerde ve sık sık değişiklik yapılan regresyon testlerinde kolaylık sağlar.

Test Metotları:

Kara Kutu Testi: Uygulamanın iç yapısıyla ilgili hiçbir bilgiye sahip olmayan test tekniğidir. Test uzmanı, sistem mimarisiyle ilgilenmez ve kaynak kodlara erişmez. Kara kutu testinde test uzmanı sistemin kullanıcı ara yüzünde belirtilen girdileri sağlayarak çıktıların doğru olmasını bekler.

Kara kutu testi kullanılarak yakalanabilecek hatalar:

1. Doğru olmayan ya da kayıp fonksiyonlar
2. Ara yüz hataları
3. Veri yapılarındaki hatalar ya da harici veritabanı bağlantısı hataları
4. Davranış ya da performans hataları
5. Başlatma ve sonlandırma hataları

Beyaz Kutu Testi: Beyaz kutu testi kodun yapısını ve iç mantık yapılarını detaylı olarak inceler. Saydam kutu testi ya da açık kutu testi olarak da bilinir. Bir uygulamada beyaz kutu testi yapmak için, test uzmanı kodun iç çalışma yapısını bilmek zorundadır.

Beyaz kutu testi kullanılarak yapılabilecek denetimler arasında:

1. Bütün bağımsız yolların en azından bir kere sınanması,
2. Bütün mantıksal karar noktalarında iki değişik karar için sınamaların yapılması,
3. Bütün döngülerin sınır değerlerinde sınanması,
4. İç veri yapılarının denenmesi bulunur.

Gri Kutu Testi: Kara kutu ve beyaz kutu testlerinin birleşimidir. Test uzmanının veri tabanına ve dokümanlara erişimi vardır. Böylece tasarıma ve verilere uygun test dokümanı üretebilir. Yani uygulamanın iç işlemlerine kısmen erişime izin verir.

Birim(Unit) Testi: Birim testi bir bileşenin sınırları içindeki mantık ve veri yapıları gibi iç işlemler üzerinde çalışır.

Birim test durumları

- Ara yüz
- Yerel veri yapıları
- Sınır koşulları
- Bağımsız yollar
- Hata yakalama yolları

Birim ara yüzü test edilerek bilgi giriş/çıkışlarının uygun ve yeterli şekilde yapıldığı kontrol edilir.

Yerel veri yapıları verilerin saklandığı yerin bütünlüğünün bozulup bozulmadığı test edilmelidir.

Sınır koşullarının en düşük ve en yüksek değerleri, bu değerlerin biraz altı ve biraz üstü kullanılarak sınanmalıdır.

Tümleştirme(Integration) Testi: Amaç, birim testlerini başarı ile geçmiş modülleri alıp tasarımda belirtilen program yapısını ortaya çıkarmaktır.

Tümleştirme testinin yapılma nedenleri:

- Bir birimin çalışması başka bir birimin çalışmasını etkileyebilir.
- Birimler arasındaki arayüzler arasında verilerin kaybolma olasılığı vardır.
- Bir birim içinde kabul edilebilir sınırlar içinde olan kesinlik değerleri birden fazla birimin devreye girmesi ile kabul edilemeyecek değerlere ulaşabilir.
- Birimler arasında eşzamanlılığın sağlanması gerekir.
- Birimler arasında paylaşılan evrensel veri yapıları sorun çıkarabilir.

Regresyon Testi: uygulama ortamındaki yapılan tüm değişikliklerin yeni bir hata üretip üretmediğini kontrol amaçlı olarak yapılan test türüdür.

Sistem Testi:

Performans(Performance) Testi: Sistemdeki dar boğazları ortaya çıkarır. Tüm test aşamalarında yapılabilir.

Yük (Load) Testi: sistemin sınırlarını zorlayarak en fazla ne kadar veri işleme kapasitesi olduğunu belirlemek, bu yükte davranışlarını kontrol etmek amacıyla yapılır.

Yükleme Testi:

- Veri hacmi testi: Sistemin yüksek miktarda veri ile yüklenmesi
- Veri debisi testi: Tüm girişlerin yüksek hızda veri ile yüklenmesi
- Kapasite testi: Sistemin bellek ve disk kullanımının zorlanması

Germe(Stress) Testi: Sistemi normal olmayan miktarda öz kaynak gerektirecek şekilde

zorlamak amacıyla yapılan testler. Sistemin kaldırabileceği yük durumunda, ani etkilere verilecek tepki süresini ölçmek üzere yapılan testler.

Kurtarma(Recovery) Testi:

Yedekli yazılım mimarisinde, ana yazılım birimi çalışırken, yardımcı yazılım da aynı veya farklı bir donanım üzerinde paralel şekilde çalışır, fakat çıktı üretmez. Ana yazılımın çökmesi halinde bu yardımcı yazılım devreye girer.

Hataya dayanıklı yazılım ise, herhangi bir nedenle bütünüyle çökmemek üzere, kendi kendini düzeltebilen modüller halinde geliştirilen yazılımlardır.

Güvenlik Testi: Sistemin tüm potansiyel açık kapıları ve zayıflıkları araştırılır.

Temel güvenlik testi çeşitleri şunlardır:

- Zafiyet taraması
- Penetrasyon Testi
- Risk Belirleme
- Güvenlik Denetimi
- Şifre Kırma

Taşınabilirlik (Portability) Testi: Var olan bir yazılım bileşeni veya uygulamayı yeni bir ortamda test etme işlemidir.

Uygulamanın diğer ortamlarda

- Installability
- Combatibility
- Adaptability
- Replaceability yetenekleri araştırılır.

Kullanılabilirlik Testi:

Öğrenilebilirlilik: Kullanıcılar, tasarımı ilk kullandıklarında yerine getirmeleri gereken görevleri kolaylıkla yapabiliyorlar mı?

Verimlilik: Kullanıcılar, tasarımın çalışma şeklini öğrendikten sonra gerçekleştirecekleri işlemleri ne kadar hızlı yapabiliyorlar?

Memnuniyet: Tasarımı kullanmak kullanıcıları duygusal anlamda mutlu ediyor mu, kullanıcılar tasarımı kullanırken kendini rahat hissediyorlar mı?

Hatırlanabilirlilik: Kullanıcılar, bir süre tasarımı kullanmadıktan sonra tekrar kullanamaya başladıklarında tasarıma dair var olan bilgilerin ne kadarını hatırlayabiliyorlar?

Hatalar: Kullanıcılar, ne kadar hata yapıyor ve bu hataları ne sıklıkta tekrarlıyorlar, hataları ne kadar hızlı yok edebiliyorlar?

Web Uygulama Testine Giriş

Web kullanıcıları ne ister?

- Hız
- Güvenlik
- Kolay üyelik
- Düşünmemek
- Klavye kullanımı
- Açık hata mesajları
- Sadece gerekli bilgi

İçerik Testi:

1. Son kullanıcıya sunulacak içeriğin yapısı veya bütünlüğündeki hataları bulmak için,
2. Herhangi bir içerikteki sözdizimsel hataları ortaya çıkarmak için,
3. Herhangi bir içerikteki anlam hatalarını bulmak için yapılır.

Kullanıcı Arayüzü Testi:

Linkler: Her bir link, doğru içerik ve fonksiyona ulaşıp ulaşmadığından emin olmak için test edilir.

Test edilmesi gereken diğer ara yüz mekanizmaları şunlardır:

- İstemci taraflı scriptler
- Dinamik HTML
- Pop-up pencereleri
- CGI scriptleri
- Cookies
- Akıcı içerikler

Kullanılabilirlik Testi: Bir ürünün potansiyel kullanıcıları tarafından belirli bir kullanım bağlamı içinde, amaçlanan kullanım hedeflerine ulaşmak için, etkin, verimli ve tahmin edici bir şekilde kullanılabilmesi olarak tanımlanır.

5 saniye kuralı: Kullanıcı ilk 5 saniyede site hakkında bilgi edinir.
2 saniye kuralı: Kullanıcıların bir çoğu siteye girdiklerinde kalma veya gitme kararını ilk 2 saniyede verir.
3 tık kuralı: Bir çok kullanıcı sitede aradığı bilgiye ortalama 3 tık ile ulaşamazsa siteyi kullanmaktan vazgeçer.
80/20 kuralı: Tasarımda etkinin %80 i mevcut görsel tasarımın %20 sinden gelir.
7 ± 2 kuralı: İnsanın yakın zaman hafızası aynı anda 5 ila 9 olguyu hatırlamasına olanak verir. Dolayısıyla 7±2 prensibiyle menülerdeki seçenekler ortalama 7 adet olarak belirlenmelidir.

Güvenlik Testi:

Web uygulaması güvenlik testlerinde aşağıdaki denetimler yapılır.

- Siteler Arası Komut Çalıştırma (XSS)
- Enjeksion Açıkları (SQL Injection, Command Injection vs.)
- Emniyetsiz Doğrudan Nesne Erişimi
- Siteler Arası İstek Sahteciliği (CSRF)
- Oturum ve Kimlik Yönetimi Açıkları
- Mantıksal Saldırılar
- İstemci Tarafı Saldırıları
- Bilgi Sızdırma vb.

Authorization (Yetkilendirme): İstemci yada sunucu tarafında sadece uygun yetkili kişilerin girişine izin veren bir filtreleme mekanizmasıdır.

Authentication (Doğrulama): Tüm sunucu ve istemci kimliklerini doğrulayan bir mekanizmadır, her iki taraf da doğrulandığı zaman iletişime izin verir.

Performans Testi:

Yük testi, yükü çeşitli düzeylerde ve kombinasyonlarda yükleyerek uygulamanın davranışını incelemektedir.

Stres testi, Web uygulamasının ne kadar kapasitesi olduğunu ve kırılma noktasını belirlemek için yapılır.

Mobil Uygulama Testi:

Ön hazırlık aşamasında detaylı bir test süreci, hem şirketler hem de bireysel kullanıcıların, uygulamalarından daha yüksek verim almasını sağlayabilir.

Akıllı cihazlar hafıza, bağlanabilirlik, ara yüz standartları, ekran çözünürlüğü ve genişliği bakımından farklılık gösterir.

Emülatör: Bir sistemin işleyişini taklit eder ve taklit ettiği sistemin sunduğu özellikleri aynen sağlar. Böylece bu sistemi kullanan diğer sistemler için bunun gerçek sistemden bir farkı yoktur.

Simülatör: Gerçek bir sistemi sadece modeller, yani sadece işleyişini örnekler, gerçeğe benzer bir ortam oluşturmaya çalışır. Örneklediği sistemin çalışmasının anlaşılmasına yardımcı olur. Simülatör, gerçek sistemin yerine geçebilecek bir sistem değildir. Kullanım amacı bakımından emülatörden kesin bir şekilde ayrılır.

Test Süreci Ve Yönetimi

- 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.

Test eylemleri yazılım yaşam döngüsünün şu aşamalarında gerçekleştirilir;

- Gereksinim analizi:
    - Gereksinimlerin doğruluğuna karar vermek
    - Gereksinimlerin test edilebilirliğine karar vermek
    - Gereksinimin doğrulama metodolojisine karar vermek
    - Test stratejisini belirleyip test planı hazırlamak
- 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
- Kodlama:
    - Birim testlerin gerçekleştirilmesini izlemek ve sonuçlarını denetlemek
    - Kod gözden geçirmelerini izlemek ve sonuçları denetlemek
    - Bütünleştirmek ve sistem testleri için test ortamını hazırlamak
- Test:
    - Tümleştirme, sistem ve kabul testlerini gerçekleştirmek
    - Bulunan hataları bildirmek ve düzeltildiğini denetlemek
    - Yineleme testlerini gerçekleştirmek
- Bakım:
    - Yineleme testlerini gerçekleştirmek

Test Planlama

- 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.
- 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 bir plan içerisinde belirtilir. Bu yaklaşıma göre geliştirilebilecek test planları şunlardır:
        - Test Ana Planı
        - Kabul Test Planı
        - Sistem Test Planı
        - Tümleştirme Test Planı
        - Birim Test Planı
    - İkinci Yaklaşım: İcra edilecek tüm testler için tek bir test planı geliştirilir. Genel olarak Kabul Test Planı veya Sistem Test Planı diye adlandırılır. Geliştirilen plan, birim, tümleştirme, sistem ve kabul testlerinin ve diğer testlerin planlamalarını da kapsar.

Test Tasarımı

Bu süreçte aşağıdaki görevler icra edilir;

- Test ortamının hazırlanması
- Entegrasyon, sistem ve kabul testleri için test durumlarının yazılması
- Test yordamlarının hazırlanması

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

Test Durumlarının Yazılması

- 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.
- Geliştirilen yazılımın kendinden beklenen tüm davranışları gerçekleştirdiğinin göstergesi, yazılıma ait tüm test durumlarının testlerden geçmesidir.
- 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
- 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 Durumu Formu		
TD NO	**Test Durum Adı**	**İlgili İsterler**
1	İl Ekle Butonu Etiket	1.1
Ön Koşullar		
Program çalışıyor ve Yeni İl Seçme modülü açık		
Test Adımları		
1. İl ekle butonunun etiketinin «Ekle→» şeklinde olduğunu doğrula.		
Test Sonuçları		
Beklenen Sonuç	**Gerçekleşen Sonuç**	**Sonuç(Geçti/Kaldı)**
Resimde görüldüğü gibi		

Örnek test durumları

- Kayıt altına alınan test durumları üç bölümden oluşur. Her bölüm kendi içerisinde aşağıda verildiği gibi bölümlere ayrılır:
    - 1. Giriş/Genel Bakış
        - Test durumu hakkında genel bilgiler içerir.
        - Tanımlayıcı: Her bir test durumuna ait olan eşsiz/benzersiz bir tanımlayıcıya sahiptir. Bu tanımlayıcı ile test durumu ilgili test öğesi, test sonucu ve açılan hata bildirimleri ile ilişkilendirilir.
        - Test durumunu yazan: Test durumunun kimin tarafından yazıldığı bilgisidir.
        - Sürüm: Test durumunun sürüm numarasıdır.
        - Adı: Test durumunun ne ile ilgili olduğunu kolayca gösterecek olan bir ifadedir.
        - Gereksinim Tanımlayıcısı: Test durumunun hangi gereksinim için yazıldığının anlaşılması için gereksinime ait eşsiz tanımlayıcının test durumu içerisinde verilmesi gereklidir.
        - Amaç: Bu test durumu ile hangi işlevin test edildiğinin belirtilmesidir.
        - Bağlılıklar: Bu test durumunun gerçekleştirilmesi için varsa kendinden önce gerçekleştirilecek olan test durumlarının eşsiz tanımlayıcılarının belirtilmesidir.
    - 2. Test Durumu Eylemleri
        - Test Ortamı: Test durumunun koşturulabilmesi için gerekli olan yazılım, donanım ve çevresel koşulların belirlenmesidir.
        - İlklendirme: Test durumu koşturulmadan önce varsa gerekli olan değişkenlerin başlangıç değerlerinin belirtilmesidir.
        - Sonlandırma: Test durumları koşturulduktan sonra gerçekleştirilecek olan eylemler belirtilir. Örneğin, eğer test durumu veri tabanını siliyorsa, veri tabanının yeniden kayıtla doldurulması gibi.
        - Eylemler: Testin tamamlanması için yapılacakların adım adım belirtilmesidir.
    - 3. Sonuçlar
        - Beklenen Sonuç: Test durumunda belirtilen tüm adımların gerçekleştirilmesinden sonra test mühendisinin hangi sonuç ile karşılaşacağının belirtilmesidir.
        - Gerçek Sonuç: Test gerçekleştirildikten sonra ortaya çıkan sonucun belirtildiği alandır. Genellikle bu sonuç ile beklenen sonuç karşılaştırılarak testin geçip kaldığı belirlenir.

Test Yordamı

- Her bir test 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.

- 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.

Test Koşturma/Gerçekleştirme

Test koşturmada genel olarak şu adımlar izlenir;

- 1. Yazılım ekibi tarafından test edilecek yazılım(yük) oluşturulur.
- 2. Testçiler gelen yazılıma uygulanacak olan test durumlarına karar verirler.
- 3. Yazılımın test edilebilir olduğuna karar verilir.
- 4. Yazılım teste kabul edilirse test 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 gerçekleştirir.
- 8. Bu adımlar müşteriye son kabul edilebilir yazılım verilinceye kadar devam eder.

Hata Yönetimi

1. Hata Tespit Edilip Bildirilir
2. Düzeltme Faaliyetleri Gerçekleştirilir
3. Yeni Yazılım Yükü Hazırlanır
4. Yineleme Testler Gerçekleştirilir
5. Hata Yeni Yükte Düzelmiş İse Kapatılır

Hata Raporlama

- Bulunan hatalar, hataların kaynağı, yapılan düzenlemeler raporlanmalıdır.
- Bu amaç için oluşturulacak olan örnek bir hata raporu şöyledir:

Hata No	Bildiren Kişi	Hata Bildirim Tarihi
Hata Adı	**Hatanın Durumu**	**Yazan/Kodlayan**
Hata Açıklaması		
Hata Tipi	**Önem Derecesi**	**Öncelik**
Atanan Kişi	**Atanma Tarihi**	**Kapanma Tarihi**
Test Edilen Öğe		**Sürüm No**

Hata Önem Dereceleri

- **Ölümcül**: Testlerin devam etmesini engelleyecek hataları belirtmek için kullanılan derecedir. Eğer bu türden bir hata bulunmuş ise testlerin devam etmesi imkansızdır.
- **Kritik**: Testler devam edebilir ancak bu hata derecesi ile yazılım teslim edilemez.
- **Büyük**: Testler devam edebilir. Ürün bu hata ile teslim edilebilir. Ancak yazılım kullanıldığında telafisi zor sonuçlar doğurabilir.
- **Orta**: Testler devam edebilir. Ürün bu hata ile teslim edilebilir. Ancak yazılım kullanıldığında telafisi mümkün sonuçlar çıkartabilir.
- **Küçük**: Testler devam edebilir. Ürün bu hata ile teslim de edilebilir. Yazılımın işleyişinde ortaya çıkabilecek hatalar önemli bir sonuç doğurmaz.
- **Kozmetik**: Yazılım üzerindeki renk, font, büyüklük gibi görsel hatalardır. Olması durumunda ne testi durdurur ne de ürünün teslimini engeller.

Test Sonuç Raporlama Ve Değerlendirme

- Testler gerçekleştirildikten 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 test özet belgesi hazırlanır. Test özet belgesi aşağıdaki bilgileri kapsamalıdır.
    - Modül/Birim
    - İhtiyaç Numarası
    - Test Durum Numarası
    - Test Sonucu Numarası
    - Test Sonucu
    - Test Tarihi
    - Testi Yapan
    - Testi İzleyen
    - Tekrar Sayısı
    - Hata Bildirimi Numarası
    - Sürüm No
    - Donanım Sürümü
    - Yardımcı Yazılımların Sürümü
    - Notlar

Yazılım Test Riskleri

- **Tümleştirmede karmaşa**: Tek bir birimden oluşan yazılımın testi başka birimleri etkilemediği için karmaşıklık oluşmaz. Ancak, birden fazla yazılım biriminin tümleştirilmesi ve testi sırasında karşılıklı olumsuz etkilenmeler oluşabilir. Aynı anda test edilen birim sayısı arttıkça hataların nereden kaynaklandığını bulmak da güçleşir.
- **Sıralama**: Test yordamlarının belirli bir sıra takip etmeleri gerekmektedir. Uygun bir sıraya göre yapılmayan ve senaryosu iyi tanımlanmayan test durumları tekrarlara ve sonuçta da zaman kaybına yol açabilir.
- **Paralel test işlemleri**: Birden fazla öğe aynı anda test ediliyorsa ve her bir öğe içinde birden fazla birim bulunuyorsa, yürütme sırasında birbirlerine olan etkilerinden dolayı hangi öğenin doğru, hangisinin kusurlu olduğunun bulunmasında güçlük çıkabilir.

- **Yüksek maliyetli testler**: Bazı testler çok yüksek maliyet gerektirebilir. O nedenle son derece dikkatle planlanmalı, önceden yeterli bütçe ayrılmalı, iyi değerlendirme yapılabilmesi için kayıtlara önem verilmelidir.
- **Testlerin plan dışı yürütülmesi**: Testlerin bir plana göre yürütülmemesi durumunda test ortamının kullanımında, testlerin uygulanmasında, hata bulmada karmaşa yaşanabilir. Plansız yapılan testlerle zaman kaybı yaşananabilir, hataların kaynaklanma nedeninin bulunması zorlaşır.
- **Test yordamlarının yetersizliği**: Test yordamları yeterince kapsamlı olmayıp yüzeysel olarak uygulanırsa testler başarılı geçse bile atlanmış durumların gerçek kullanım sırasında sorun çıkarması beklenmelidir.

Gözden Geçirmeler

- Geliştirme aşamasında bulunan her hata, ürünü mükemmelliğe daha çok yaklaştırır. Bu nedenle de herkesin her yaptığı işe gözden geçirme (review) uygulanmalıdır.
- Yazılım mühendisliğinin pratik uygulamalarında kullanılabilecek çeşitli gözden geçirme türleri vardır:
    - **Eş düzey gözden geçirme**: Proje çalışanlarının genellikle aynı düzeyde bulunan personel ile birlikte yürüttükleri gözden geçirmelere eş düzey gözden geçirme denir. Tasarımcılar tasarımla ilgili, kodlayıcılar da kodla ilgili gözden geçirmelere katılırlar.
    - **Birleşik gözden geçirme**: Geliştirici ile müşterinin, sözleşmede yer alan yönetsel ve teknik işleri, iş adımlarını, aşamaları gözden geçirmek için beraber yaptıkları toplantıya birleşik gözden geçirme denir.
    - **Resmi teknik gözden geçirme**: 3 aşaması vardır. İnceleme, denetleme ve kod geçişleri. Bu etkinliklerin her biri toplantı şeklinde gerçekleştirilir, dikkatli bir planlama yapılır, eksiksiz katılım sağlanması, denetim altında yürütülmesi ve sonuçların kayıt altına alınıp açıklanması gerekir.
        - Denetleme(Audit): Bir yazılım ürününün, bir yazılım sürecinin veya bir dizi yazılım süreç faaliyetlerinin belirtimler , standartlar, sözleşme veya diğer unsurlar bakımından uyumunun değerlendirilmesi için yapılan sistematik değerlendirmedir.
        - İnceleme(Inspection): Bir yazılım ürünündeki hatalar ve standartlardan sapmalara neden olan anormalliklerin belirlenip tanımlanması için inceleme teknikleri konusunda eğitimli, tarafsız kişilerin rehberliğinde denk kişilerin katılımıyla gerçekleştirilen sorgulamadır.
        - Kod geçişleri(Walk-through): Bir yazılım geliştiricisi tarafından diğer geliştirme ekip üyelerine anlatılarak, yazılım ürününün iyileştirilmesine yönelik görüşlerin alınması ve standartların ihlali veya olası hataların belirlenmesidir.
        - Resmi teknik gözden geçirme süreci:
            - Planlama: Gözden geçirme sürecinin hazırlanması ve organize edilmesidir. Bu kapsamda gözden geçirme materyalleri, yordamları, toplantı takvimi, gözden geçirmeye katılacak olan kişiler ve rolleri hakkında hazırlıklar gerçekleştirilir

- Bilgilendirme: Bu aşamanın amacı gözden geçirmeye katılacak olanların gözden geçirme hakkında ve gözden geçirilecek olan ürün hakkında eğitilmesidir. Bu aşamada amaç, gözden geçirme ve ürün hakkında tüm ekibin temel bilgi düzeyine ulaşmalarını sağlamaktır.
- Bireysel Hazırlık: Ürün hakkında gerekli bilgileri öğrenen ekip elemanları daha sonra kendilerine ait roller ile gözden geçirilecek olan ürünü inceler ve ilgili gözden geçirme kayıtlarını doldurur. Bununla gözden geçirme toplantısından önce ürün üzerindeki hata, kusur ve eksikliklerin keşfedilmesi amaçlanır.
- Grup Toplantısı: Bu toplantı ile bireysel olarak tespit edilen hata, kusur ve aksaklıklar bir araya getirilir. Gözden geçirme toplantısı, genelde, gözden geçirilecek ürün sorumlusunun ürünü kısaca tanıtımı ile başlar. Daha sonra bireysel gözden geçirmelerde tespit edilen hata, kusur ve eksiklikler teker teker gündeme getirilir. 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 ilgili kişilerce ilgili 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 toplantı daha yapılabilir

- Gözden Geçirmeler;
    - Gözden geçirmeler planlanmalıdır.
    - Katılımcı sayısı ne az ne de fazla olmalıdır.
    - Bir gündem belirlenmeli ve uyulmalıdır.
    - İnatçı tartışmalar sınırlandırılmamalıdır.
    - Problemler hakkındaki düşünceler açıkça ortaya konulmalıdır.
    - Tüm problem alanlarına değinilmelidir.
    - Yazılı notlar alınmalıdır

Yazılım Ürün Değerlendirmesi

- Uygulanan nitelik güvence sistemine göre bu kapsamdaki her yazılım ürününün ilgili kişilerce gözden geçirilmesi ve Yazılım Nitelik Raporu düzenlenmesi gerekir. Bu raporda, ürünü gözden geçirecek kişilerin sorumluluk ve ilgi alanlarına göre dağıtım yapılmalıdır.

İnceleme Kontrol Listeleri

- Geliştiricilerin ve yöneticilerin deneyim eksikliği, sistemin karmaşıklığı, personel sayısının azlığı bu toplantıların etkin bir şekilde yapılmasını güçleştirir. Gözden geçirme toplantılarında dikkat edilmesi gereken çeşitli kontrol listeleri vardır. Bu listeler yandaki evreler için oluşturulmuştur:
    - Sistem mühendisliği
    - Yazılım proje planlaması
    - Yazılım tasarımı
    - Kodlama Yazılım tümleştirme ve test
    - Bakım

- Kontrol Soruları:
    - Her öğe için tasarım kısıtlamaları belirli midir?
    - Başarım ölçütleri ve nasıl ölçüleceği belirlenmiş midir?
    - Sistemin öğeleri arasında herhangi bir tutarsızlık var mıdır?
    - Sistemin gerçekleştirim çözümünü teknik olarak uygulamak mümkün müdür?
    - Önemli işlevler belirgin, anlaşılır ve sınırlandırılmış şekilde tanımlanmış mıdır?
    - Sistemin doğrulama ve geçerlemesinin yapılabilmesi için yöntemler belirlenmiş midir?
    - Sistemi oluşturan alt sistemlerin ve ana öğelerin işlevleri, bunların birbiriyle olan ara yüzleri tanımlanmış mıdır?
    - Sistemi gerçekleştirmek üzere seçilen yol ve geliştirme yöntemi tanımlanmış mıdır, bu konuda diğer alternatif seçenekler göz önüne alınmış mıdır?

Yazılım Ürün Metriklerinin İncelenmesi

- Ürün metrikleri:
    - Proje durumunun izlenmesi ve gözlenmesinde kullanılır.
    - Bir projenin öz kaynak gereksinimi için tahminlerde bulunabilme olanağı verirler.
    - Personelin başarımını değerlendirme ve sorgulayabilme olanağı sağlarlar.
    - Örgüt yapılarını, bireysel çalışma yöntemlerini değerlendirebilme olanağı verirler.
    - Yazılım geliştirme yardımcı araçlarının karşılaştırılmasında ya da onların tasarımında bir temel oluştururlar.
    - Yanlış ya da ters giden şeylerin bulunmasına yardımcı olurlar.
- **Andaç metrikler**: Bir andaç(token), kaynak kodu oluşturan basit bir birim olarak değerlendirilebilir. Andaç olarak if, for, while, switch, class, procedure gibi programlama diline bağlı bir anahtar sözcük seçilebileceği gibi özel bir tanımlayıcı ya da bir değişken ismi seçilebilir. Andaç metrikler bir sistemin tümünün, bir parçasının ya da bir yazılım biriminin kaynak kodu içinde bulunan bu tür andaçların sayılmasıyla hesaplanır. Sayım sonunda istatistiksel bilgiler elde edilerek bir karşılaştırma yapılır.
- **Denetim Akış Metrikleri**: Bu metrikle bir yazılım sistemi içinde yer alan her yazılım birimi incelenerek denetim akışı belirlenir. Çözümleme yapılan birimin denetim yapısının karmaşıklığı grafiksel gösterimle yapılır. Giren ve çıkan bilgi akışının, iletilerin sayısı bir metrik oluşturur.
- **Bileşik Metrikler**: Andaç ve denetim akış metriklerinin birleştirilerek kullanılmasıyla oluşturulur.
- **Sistem Metrikleri**: Diğer metrikler yazılım kodunun çeşitli özellikleriyle ilgilenirken sistem metrikleri bir sistemin daha çok tasarım niteliğiyle ilgili süreç, boyut, zamanlama, ürün niteliği, bakılabilirlik, üretkenlik gibi büyük ölçekli özellikleri üzerinde yoğunlaşır. Bu metrikler bir projenin erken evrelerinde başladığı için geleceğini tahmin etmede büyük yarar sağlarlar.

Metrik Tanımları

Büyük projelerde ise sistem metrikleri büyük önem taşır. Bu nedenle yaygın olarak kullanılan sistem metriklerinden bazılarının tanımını göreceğiz;

- **Süreç Metrikleri**: Yazılım geliştirme sürecinin niteliğini izlemek üzere, kullanılan tekniklerin, araçların insanların, örgütün ve altyapının özelliklerinden oluşan metrikler kullanılır. Bu şekilde, bir sürecin doğru uygulanıp uygulanmadığı, bir yazılım aracının etkinliği ve üretkenliği, kodlayıcıların yetenekleri, örgütün içsel ve dışsal iletişim yetenekleri, altyapının ve tesislerin yeterlilikleri değerlendirilebilir.
- **Boyut Metrikleri**: Boyut metriklerinin hedefi projenin büyüklüğüne ilişkin durumun önceden yapılan planla uyuşmakta olduğunun gözlemlenmesidir. Planlanan toplam ister sayısı ile değerlendirmenin yapıldığı anda karşılanmış olan ister sayısı karşılaştırılarak isterlerin durumu gözlemlenir.
- **Zamanlama Metrikleri**: Zamanlama metriklerinin toplanma amacı projenin takvime uygun yürütülmekte olduğunun gözlemlenmesine yardımcı olmaktır. Kullanılan en önemli ölçüt zamandır. Projenin tahmin edilen toplam süresi ile o ana kadar harcanan süre projenin durumu hakkında bilgi veren önemli bir göstergedir.
- **Maliyet Ve Kaynak Metrikleri**: Projenin belirli bir aşamasında, önceden kestirilen kişi- zaman cinsinden emek ile o tarihe kadar gerçekleşen emek karşılaştırılarak bir metrik elde edilebilir. Benzer şekilde harcanan para da karşılaştırılabilir
- **Ürün Nitelik Metrikleri**: Amacı ürünün doğru ve kusursuz olarak üretildiğinden emin olmaktır. Bu maksatla, karşılaşılan yazılım kusurlarının doğaları, türleri, oluşma nedenleri ve ne zaman oluştukları kayıt altına alınır. Kusurların önem dereceleri, oluşma yoğunlukları ve bunların yazılımın büyüklüğüne oranları her bir aşama için belirlenir. Kusur belirleme sürecinin de etkinliği ve niteliği ürün piyasaya sürülmeden önce ortaya çıkarılan kusur sayısı ile ölçülür.
- **Bakım Ve Okunabilirlik Metrikleri**: Yapılan çeşitli araştırmalara göre yazılım bakım ve okunabilirliğini artırabilmek üzere kodlama sırasında dikkate alınabilecek bazı sayısal değerler verilebilir.
- **Üretkenlik Metrikleri**: Her proje için önemli bir etken olan üretkenlik durumu, tüm proje boyunca dikkatle takip edilmeli, bu amaçla her aşamadaki genel üretkenliği ölçebilecek metrikler tanımlanmalıdır. Bunlardan bazıları şunlardır:
    - Kazanılmış değer hesaplamaları
    - Bir aşamada ortaya çıkan hataların harcanan kişi-zaman cinsinden emeğe oranı
    - Bir aşamada elde edilen boyut metriklerinin emeğe oranı

Geleneksel Metrikler

- Kod Büyüklüğü (Lines Of Code LOC)
    - Geleneksel olarak yazılımın boyutu satır sayısı ile ölçülür. Bu ölçümün çeşitli şekilleri vardır.
        - Satır Sayısı (Lines of Code-LOC): Programın tüm satırlarının sayılmasıdır.
        - Yorum Ve Boşluk İçermeyen Satır Sayısı (Non-comment Non-blankNCNB): Programın yorum satırları ve boş satırlardan arındırılmış halidir.
        - Çalıştırılabilir Yordam Sayısı (Executable Statements-EXEC) : Program içinde yer alan yordam sayısıdır.
- Yorum Oranı (Comment Percentage - CP)
    - Yorum oranı, program için hazırlanmış yorum satırlarının, toplam programın yorum ve boşluk içermeyen satır sayısına bölümü ile bulunur. Yüksek yorum oranı programların anlaşılabilirliğini arttıran ve bakımını kolaylaştıran bir faktördür.

    - Yorum oranının %20 ila %30 arası olması tavsiye edilen bir durumdur.
- Döngüsel Karmaşıklık (Cyclomatic Complexity -CC)
    - Metotların az karmaşıklığa sahip olması tercih edilir. Bu kodun anlaşılırlığının ve test edilmesinin kolay olduğunu gösterir. Bir metot için karmaşıklığın 10'u aşmaması tercih edilmelidir. Ancak 20'ye kadar kabul edilebilir. Bunu aşan değerler anlaşılması oldukça güç kodları gösterir.

Nesne Yönelimli Metrikler

Chidamber ve Kemerer Metrikleri;

1. Sınıf Başına Ağırlıklı Method (Weighted Methods per Class -WMC)
    a. Döngüsel karmaşıklık değerleri toplamının sınıf sayısına bölümü ile bulunur.
    b. WMC değerinin 100'ün altında olması tercih edilmelidir.
    c. WMC değerinin 100'ün altında olması kabul edilebilir bir değerdir.
    d. WMC değeri NOM'dan farklı olarak sınıfın karmaşıklığı hakkında daha net bir fikir verir.
2. KALITIM AĞACININ DERINLIĞI (DEPTH OF INHERITANCE TREE - DIT)
    a. DIT metriği sınıfın ebeveyn sınıflarının sayısını gösterir. Eğer çoklu kalıtım durumu söz konusu ise bu durumda hiyerarşideki en uzun yol kabul edilir.
    b. DIT bize kısaca kaç tane ana sınıfın potansiyel olarak sınıfımızı etkileyebileceğini gösterir.
    c. Sınıf hiyerarşisinin derinliği arttıkça daha fazla metot kalıtım alınır ve sınıfın davranışlarını öngörmek; anlamak zorlaşır.
    d. DIT için önerilen rakam genellikle 5'in altında olmasıdır. 5'in üzerindeki derinlikler oldukça karmaşık yapılar doğurabilir. DIT'in 0 olması sınıfın kök olduğunu gösterir. DIT'in ortalama 2-3 arası bir değerde olması yeniden kullanımın iyi seviyede olduğunu gösterir.2'den küçük derinlikler ise yeniden kullanımın zayıf olduğu alanları işaret eder
3. Alt Sınıf Sayısı (Number of Children - NOC)
    a. NOC metriği sınıftan türemiş alt sınıflarının sayısını verir.
    b. Alt sınıf sayısı çoğaldıkça kalıtım özelliğine bağlı olarak yeniden kullanımın arttığı anlaşılır.
    c. NOC sınıfın nüfus alanı hakkında bir fikir verir. NOC metriği yüksek sınıflar gözden geçirme, test gibi süreçlerin daha dikkatli ve uzun tutulması gereken yerlerdir
4. Nesne sınıfları arasındaki bağımlılık (Coupling Between Object Classes - CBO)
    a. Bir sınıf içindeki özellik ya da metotların diğer sınıfta kullanılması ve sınıflar arasında kalıtımın olmaması durumunda iki sınıf arasında bağımlılıktan bahsedilebilir. CBO; verimliliği ve yeniden kullanılabilirliği ölçmede kullanılır
5. Sınıfın tetiklediği metot sayısı (Response for a class - RFC)
    a. Bir sınıftan bir nesnenin metotları çağrılması durumunda, bu nesnenin tetikleyebileceği tüm metotların sayısı RFC değerini verir. Yani, bir sınıfta yazılan ve çağırılan toplam metot sayısıdır. Bu metrik; sınıf seviye tasarım metriklerinden olup; anlaşılabilirliği, dayanıklılığı, karmaşıklığı ve test edilebilirliği ölçmede kullanılır

6. Metotlardaki uyum eksikliği (Lack of cohesion in methods - LCOM)
    a. LCOM, n adet kümenin kesişiminden oluşan kümelerdeki uyumsuzlukların sayısıdır ve metotlardaki benzerlik derecesini ölçer. Metotlardaki uyum eksikliği; bir sınıfın, iki veya daha fazla alt sınıfa ayrıldığını gösterir ve karmaşıklığı arttırır. Yapılan bir çalışmada, LCOM ölçütünün uyum özelliğini çok da iyi ayırt edemediği ispatlanmıştır. LCOM metriği test ediciye; verimlilik ve yeniden kullanılabilirlik derecesi hakkında bilgi verir.

Test Araçları

Sızma (Penetrasyon) Test Araçları

- Metasploit
    - içinde pek çok hazır exploit bulunduran aynı zamanda bu exploitleri yerel ve uzak erişim için yapılandırabilen özel bir penetrasyon test yazılımıdır. Güvenlik uzmanları ve Hackerlar için exploit uygulama ve geliştirmede kullanılabilen bir alt yapıdır. Metasploit, Metasploit firması tarafından yönetilen ve pek çok kişi tarafından geliştirilen bir topluluk projesidir.

Güvenlik Test Araçları

- Nessus
    - kapsamlı bir güvenlik açığı tarama yazılımıdır. Genel amacı, bilgisayar sistemlerinde ve bilgisayar ağlarında potansiyel güvenlik açıklarını tespit etmektir. Çalışma prensibi istemci/sunucu biçimini kullanır ve test edilecek sistemde nessus sunucu yazılımının açık olması daha derinlemesine test ve analiz imkânı sunar.
- Nmap
    - Gelişmiş bir güvenlik tarayıcısıdır. Taranan ağ üzerindeki sistemler hakkında bilgi sahibi olunmasında, ağ topolojisinin çıkarılmasında, sızma testlerinin gerçekleştirilmesinde, herhangi bir ağ hazırlanırken gerekli ayarların test edilmesinde, ağ envanteri tutulması, haritalaması, bakımında ve yönetiminde kullanılır.
- Hping
    - Kullanıldığı alanlar
        - İsteğe göre düzenlenmiş TCP, UDP, ICMP, Raw-ip paketler üretme
        - Güvenlik Duvarı işlevsellik ve performans testleri

- DOS Engelleme sistemi
- Saldırı Tespit Ve Engelleme Sistemleri
- Gelişmiş Port Tarama
- Gelişmiş Dosya Transferi
- TCP/İP üzerinden hedef sistemlerden bilgi toplama

Web Güvenliği Test Araçları

- Acunetix
  - Dinamik oluşturulan web uygulamalarındaki güvenlik açıklarını tarayarak, çok detaylı analiz yapılmasını ve gelişmiş raporlar alınmasını sağlayan araçtır. Bu sayede, web uygulamasında, veri tabanı bağlantısında ya da servisteki güvenlik açıklarını, güvenlik sorunu ortaya çıkmadan bulmanızı ve kapatmanızı sağlar.
- Sqlmap
  - Açık kaynak kodlu bir test aracıdır. Saldırı tespiti, korunmasızlık sömürücü ve Sql enjeksiyon açıklarını bulma gibi işlemlerde otomatik olarak işlem yapan çok güçlü bir araçtır.
- Havij
  - Bir web sitedeki Sql injection açıklarını bulmaya yardım eden bir SQL injection otomasyonudur. Web uygulamasının savunmasızlığından faydalanır.
- Netsparker
  - Bir web uygulaması güvenlik tarayıcısıdır. Otomatik olarak bir web sitesini uygulama seviyesindeki güvenlik açıklarına karşı analiz edip güvenlik açıklarını raporlar. Ek olarak raporlamanın bir adım da ötesine geçip güvenlik açıklarını kullanarak aynı bir saldırgan gibi sistemden data çıkartabilir ya da sisteme tam erişim sağlayabiliyor.
- Sucuri SiteCheck
  - Sitenizin iframe virüsü, malware ve kullanıcılar için tehdit oluşturabilecek diğer zararlı kodları içerip içermediğini, kara listelere girip girmediğini tek tıklamayla kontrol etmenize olanak sağlayan güzide güvenlik servislerinden biridir.

Mobil Test Araçları

- Crittercism
  - Mobil uygulama performansınızı her açıdan değerlendirebilmenize olanak sağlıyor. Bununlar birlikte Crittercism anlık olarak iOS, Android, Windows Phone ve Html5

platformlarındaki uygulamalarınızın hatalarını görmenize olanak sağlayarak detaylı raporlama sunar.

- AppThwack
    - Mobil uygulamaları 100'den fazla gerçek cihazla test etmeye yarayan bulut tabanlı bir test aracıdır. Uygulamanın hatalarını, cihazın hafıza ve işlemci kullanımını ve benzeri birçok özelliği her bir cihaz için ayrıntılı rapor halinde sunar.
- Bugsense
    - BugSense mobil uygulama geliştiricilerin tercih edebileceği bir diğer uygulama performansını ölçme (application performance management) araçlarından bir tanesidir. Mobil uygulamaların hata loglarını size bildirerek güncelleme yapmanızı sağlamaktadır. BugSense uygulamanızın büyüklüğü ölçüsünde değişik aylık abonelik paketleri ve iOS, Android, Windows Phone ve HTML5 desteği sunmaktadır.

- Instabug
    - SDK'sını uygulamanıza entegre ettiğiniz andan itibaren kullanıcılar uygulamanın açık olduğu anda telefonlarını sağ ve sola sallayarak uygulamanız hakkında size geri bildirimde bulunabiliyorlar. Bunun haricinde uygulamada yer alan fonksiyonlar içinde ayrı ayrı yorumda bulunabiliyorlar. Gelen yorumlar instabug'ın uygulama yayıncıları için sağladığı ekrandan görülerek kontrol edilebiliyor. Bunun haricinde şirket uygulamanız için gelişmiş bir hata raporlama imkanı hizmetide vermektedir.
- Pulse.io
    - Mobil uygulamaların test aşamasında gözden kaçan hatalarını yakalamak ve kod performansı konusunda ölçümleme ve analiz yapan bir servistir. Uygulamadaki anlık oturumları analiz ederek o andaki sorunlar ve ileriye dönük oluşabilecek hatalar konusunda raporlar çıkartan servis dönüşüm, verimlilik konusunda önemli ölçütleri geliştiriciyle paylaşmaktadır

Kullanılabilirlik Test Araçları

- YouEye
    - Uzaktan kullanıcı testi yapabildiğiniz YouEye'da kullanıcıların nereye baktıklarını takip ederek göz izleme testi yapabilirsiniz. Aynı zamanda kullanıcı ses kaydı da bulunuyor. Duygularını da ölçebilen bir servisi olan test aracında, önce görevler oluşturuyor, ardından katılımcılarla test yapıyor ve elde ettiğiniz verileri analiz edebiliyorsunuz.
- Session Cam
    - Kullanıcıların hareketlerini izlemeye yarayan bir kullanılabilirlik test aracı olarak Session Cam, sıcaklık haritası, tıklama haritası, form analizi ve dönüşüm hunisi oluşturmayı sağlıyor. Kullanıcıların hareketlerini kayıt ederek tekrar tekrar izleme şansı bulunuyor.
- Try My UI

    - Kullanıcıların demografik olarak profillerini seçerek kullanılabilirlik testi uygulanabilir. Ekran ve mouse hareketleri, sesli düşünme testleri ve yazılı geri bildirimler dahil her kullanıcının verileri video halinde sunulur. Kendi veritabanına ait kullanıcıları kullanmak isteyenler için ise ayrı bir ücret sunuluyor.
- Optimizely
    - Bağlılık, tıklama, dönüşüm ve kayıt gibi aşama ve kıstaslarda hangi yöntemin daha iyi işe yaradığının ölçmeye yarayan bir araçtır. Hedef takibi ile web sitenin kullanıcı deneyimi açısından daha iyiye ulaşması için A/B testlerinde kullanılır.
- Userlytics
    - Web site, mobil - kullanıcı etkileşimini ölçmek amacıyla uzaktan kullanılabilirlik testi uygulamayı sağlar. Kullanıcı deneyimini pozitifleştirmek için raporlar halinde sunulan veriler kullanıcı kayıtlarından elde edilir. Prototiplerin veya canlı web sitesinin kullanılabilirlik testlerini yaparken kullanıcılara sorular sorarak anket yapabilmeyi de sağlıyor.

- Userzoom
    - iPhone, iPad, ve Android telefon ve tabletler için mobil websiteleri ve prototiplerini kullanılabilirlik yönünden test etmemizi sağlayan bir araçtır. Kullanıcı deneyimi konusunda bize bir çok zengin özellik sunar.

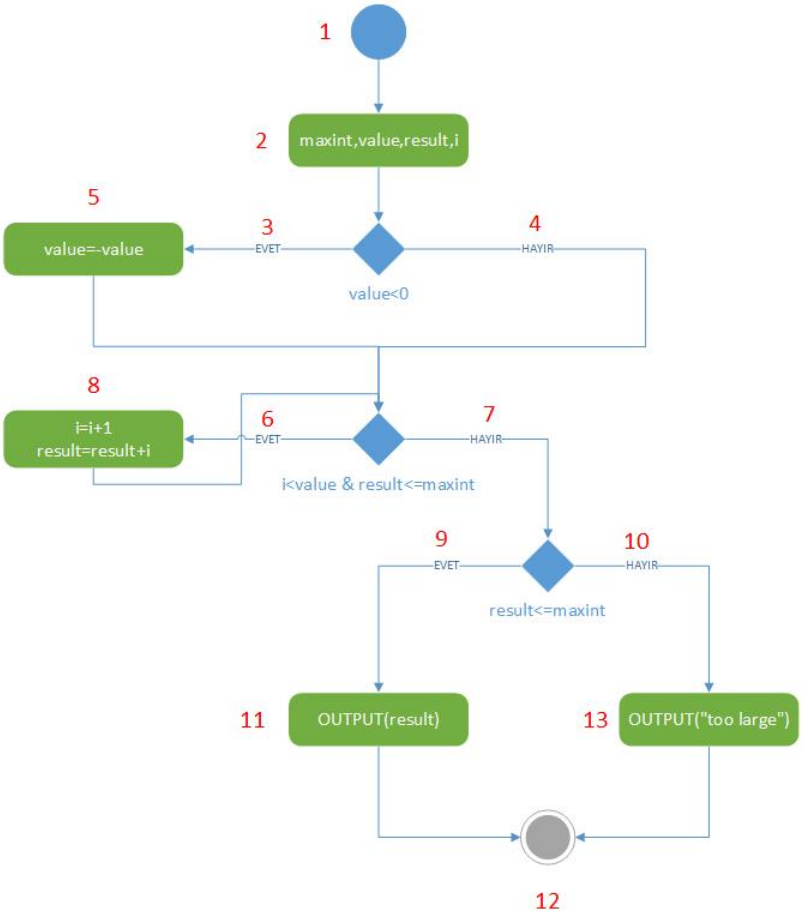
Web Performans Test Araçları

- GTmetrix
    - Web sitelerinin:
        - Kod yapısını,
        - Yer alan resimleri,
        - Css ve js dosyalarını,
        - Sayfa boyutunu,
        - Sorgu sayısını,
        - Kodlama standartlarına uyumluluğunu,
        - Açılma hızı gibi arama motorlarındaki sıralaması ile doğrudan orantılı olan düzenlemeleri kontrol edip size rapor sunan bir servistir.
- Google PageSpeed
    - sitenizin daha da hızlanması için site içinde küçük bir araştırma yapan ve size tavsiyelerde bulunan bir uygulamadır. Sitenizin hızını arttırmak için sunulan tavsiyeler arasında CSS ve Javascript dosyalarını sıkıştırma önerileri yer alır. Sitenizin hem SEO bazında, hem kullanıcı tarafında daha da iyi hizmet vermesini sağlayan uygulama; tarafınıza oldukça gelişmiş ve detaylı bir rapor sunar.
    - Search Engine Optimization-SEO, arama motorlarının web sayfalarını daha kolay bir şekilde taramasına olanak sağlayan tekniktir. Bununla birlikte arama motorları, arama sonuçlarını listelerken algoritmik bir yapı kullanmaktadır. Bu nedenle web geliştiricileri yazmış oldukları sayfaları bu ayrıntıya dikkat ederek oluşturmalıdır.
- Pingdom

    - sunucunuza bağlı tüm servisleri istediğiniz zaman aralıklarında takip eden ve sunucunun cevap vermediği durumlarda size e-mail ya da SMS ile uyarı gönderen bir servistir. Temel olarak bir web sitenin kaç saniye açıldığını hesaplamakta ve hangi unsurların bu açılış hızını etkilediğini grafik ve tablolarla göstermektedir.
- Yottaa
    - Diğerlerinden ayıran özellik, aynı anda birçok farklı konumdan sorgulama yapabilmesidir. Amerika, Avrupa ve Asya kıtalarında birden fazla konumda onlarca sunucu desteği sayesinde, dünyanın farklı ülke ve merkezlerden sitenizin açılış hızını test edebilirsiniz. Aynı anda pek çok ölçütü bu servis aracılığıyla değerlendirebilir ve bunlara bağlı olarak sitenizin performans sonucunu anlaşılır grafiklerle izleyebilirsiniz.

Beyaz Kutu Testi

```
PROGRAM maxsum ( maxint, value : INT )
        INT   result := 0 ;  i := 0 ;
        IF   value < 0
        THEN  value  :=  - value ;
        WHILE  ( i < value )  AND  ( result <= maxint )
        DO          i  :=  i + 1 ;
                    result  :=  result + i ;
        OD;
        IF   result <= maxint
        THEN  OUTPUT ( result )
        ELSE  OUTPUT ( "too large" )
END.
```



1,2,3,5,6,8,7,9,11,12
(-1),(1),(1,1)

1,2,3,5,7,9,11,12
(-1),(-1),(0,-1)

1,2,3,5,7,10,13,12
(-1),(-1),(0,-1)

1,2,4,6,8,7,9,11,12
(1),(1),(1,1)

1,2,4,7,9,11,12
(1),(-1),(0,-1)

1,2,4,7,10,13,12
(1),(-1),(0,-1)

LCOM HESAPLAMA

P -> Boş Küme, Q -> Dolu Küme,

P<Q ise LCOM = 0

P>Q ise LCOM = P - Q

1) Aşağıda verilen kodun LCOM değerini hesaplayınız **(25p).**

```
public class Canlılar{
        public void insanSesi () { a = ''selam''; }
        public void kuşSesi () { a=''cik cik''; b = ''cibili cibili şak şak'';}
        public void eşekSesi () { a=''Ai ai''; c=''Iııı'';}
        public void kediSesi () {a=''Miyav''; c=''Tııss'' ;}
        public void yılanSesi () {c=''Tııss'';}
        public void köpekSesi() {c=''hav hav''; a=''auuuu'';}
private:
        string a,b,c;
```

$I_{1:\text{ insanSesi}}$, $I_{2:\text{ kuşSesi}}$, $I_{3:\text{ eşekSesi}}$, $I_{4:\text{ kediSesi}}$, $I_{5:\text{ yılanSesi}}$, $I_{6:\text{ köpekSesi}}$

$I_1 \cap I_2 = \{1\}$ $I_2 \cap I_3 = \{2\}$ $I_3 \cap I_5 = \{1\}$

$I_1 \cap I_3 = \{1\}$ $I_2 \cap I_4 = \{2\}$ $I_3 \cap I_6 = \{2\}$

$I_1 \cap I_4 = \{1\}$ $I_2 \cap I_5 = \varnothing$ $I_4 \cap I_5 = \{1\}$

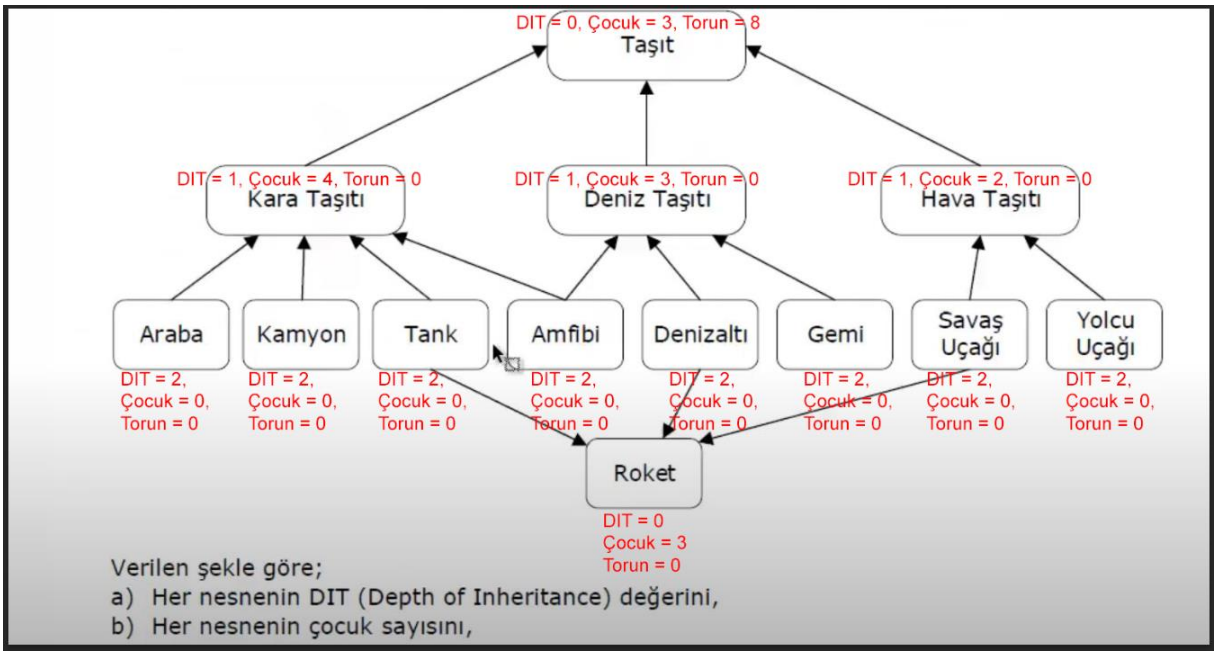
$I_1 \cap I_5 = \varnothing$ $I_2 \cap I_6 = \{2\}$ $I_4 \cap I_6 = \{2\}$

$I_1 \cap I_6 = \{1\}$ $I_3 \cap I_4 = \{2\}$ $I_5 \cap I_6 = \{1\}$

$P = 2$ ve $Q = 13$, $P<Q$

LCOM=0

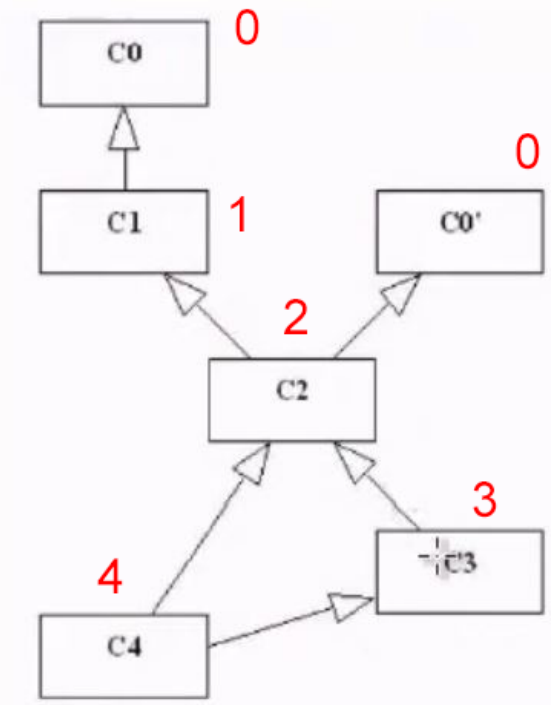
DIT


DIT = 0, Çocuk = 3, Torun = 8
Taşıt
DIT = 1, Çocuk = 4, Torun = 0
Kara Taşıtı
DIT = 1, Çocuk = 3, Torun = 0
Deniz Taşıtı
DIT = 1, Çocuk = 2, Torun = 0
Hava Taşıtı
Araba
Kamyon
Tank
Amfibi
Denizaltı
Gemi
Savaş Uçağı
Yolcu Uçağı
DIT = 2, Çocuk = 0, Torun = 0
Roket
DIT = 0
Çocuk = 3
Torun = 0
Verilen şekle göre;
a) Her nesnenin DIT (Depth of Inheritance) değerini,
b) Her nesnenin çocuk sayısını,

ÖRNEK UYGULAMALAR (DIT)

Derinlik Hesaplama
En uzun Yol Alınır

C0 -> 0
C1 -> 1
C0' -> 0
C2 -> 2
C3 -> 3
C4 -> 4



RFC

ÖRNEK UYGULAMALAR (RFC)

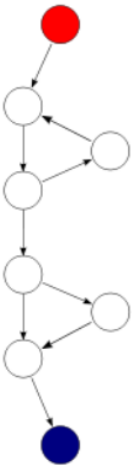
Cevap = 4

```
public class A {
  private B aB;

                          +1
  public void methodA1() {
    return aB.methodB1(); +1
  }

                          +1
  public void methodA2(C aC) {
    return aC.methodC1(); +1
  }
}
```

DÖNGÜSEL KARMAŞIKLIK

DK = Y – N + 2

Y = Yol, N = Düğüm



Basit bir programın kontrol akış grafiği. Program kırmızı düğümde çalışmaya başlar, ardından bir döngüye girer (kırmızı düğümür hemen altındaki üç düğüm grubu). Döngüden çıkıldığında, koşullu bir ifade (döngünün altındaki grup) vardır ve son olarak program mavi düğümden çıkar. Bu grafiğin 9 kenarı, 8 düğümü ve 1 bağlantılı bileşeni vardır , bu nedenle programın döngüsel karmaşıklığı 9 - 8 + 2*1 = 3'tür.

LOC(Lines of Code) = Toplam Satır Sayısı + Yorum Satır Sayısı (Boş Satırlar Hariç)

SLOC(Source LOC) = Toplam Satır Sayısı (Yorum ve Boş Satırlar Hariç)

LSLOC(Logical SLOC) = print, for, while, if-else, vb. yapılar Toplanır

PSLOC(Physical SLOC) = Toplam Satır Sayısı + Yorum Satır Sayısı + Boş Satır Sayısı