YAZILIM KALİTE VE GÜVENCE TESTİ DERS NOTLARI

(BÖLÜM-1 KALİTE KONSEPTLERİ)

- **Kalite nedir?**
  - Müşteri isteklerine cevap verebilmektir. Genel bir tanım olarak ise bir ürün veya hizmetin belirlenen ihtiyaçları karşılama kabiliyetine dayanan özelliklerin toplamıdır
  - Nesnenin tabiatında var olan özelliklerin gereksinimleri karşılama derecesidir. (ISO 9000)
  - **Yazılım Kalitesi:**
    - ✓ Açıkça tanımlanmış işlev,
    - ✓ Kullanıcı isteklerine yanıt verebilme,
    - ✓ Standartlara sadık kalma,
    - ✓ Teslim sonrası destek olarak tanımlanabilir.
  - **Kalite Kontrol:** Geliştirilen yada üretilmiş bir ürünün kalitesini değerlendirmek için tasarlanmış aktiviteler kümesidir. Hata ayıklamaya yönelik, sistematik ve belirli bir anda yapılır.
  - **Kalite güvence:** Ürünün nitelik için belirlenen özelliklerini karşılamak maksadıyla yeterli güveni sağlamak için gereken planlı etkinlikler kümesidir. Hata oluşmasını önlemeye yönelik, sistematik, zamana yayılmıştır.
- **Kalite Ölçütleri**
  - **Kullanıma 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.
    - ✓ **Sağlamlık:** Yazılım, normal olmayan çalışma koşullarında da güvenle çalışabilmeli, sisteme hata hoşgörülü bir yapı kazandırılmalıdır. Tasarım ve çözümleme aşamasında belirlenemeyen bazı koşullar sonradan ortaya çıksa da sistem işlevlerini yerine getirebilmelidir.
    - ✓ **Verimli çalışma:** Yazılım işlevlerini yerine getirirken sistem öz kaynaklarını uygun şekilde kullanmalıdır. İşlemci, bellek, disk ve diğer kaynakların kullanımı etkinlikle yapılmalı, iletişim, grafik sergileme ve yazıcı çıktıları yüksek başarıma sahip olmalıdır.
    - ✓ **Kullanım kolaylığı:** Üretilen yazılımı insanların rahatlıkla kullanabilmesi için gerekli kolaylıklar sağlanmalı, özellikle grafiksel kullanıcı ara yüzü düzenli, estetik ve kullanımı kolay olmalıdır.
    - ✓ **Korunmalı Olma:** Yazılım yetkisiz kişilerin yapabileceği değişikliklere izin vermeyecek şekilde nesne kodunu, veri yapılarını, yapılandırma dosyalarını ve belgeleri elektronik ortamda koruma altına alabilmelidir.
    - ✓ **Ekonomiklik**: Yazılıma uygun fiyata sahip olmak, sürekli kullanabilmek, eldeki donanım olanaklarını değerlendirebilmek kullanıcı için önemli ölçütlerdir.
    - ✓ **İşletim Sürekliliği:** Yazılımın sürekli kapatma açma gerekmeden çok uzun süre aynı başarım ve doğruluk düzeyinde çalışabilmesidir.

- **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.
- Y**enileştirmeye Yönelik Özellikler**
  - ✓ **Bakım kolaylığı:** Başka bir yazılım geliştirici kişi ya da grup tarafından yazılımın bakımının yapılabilmesi için kaynak kodun anlaşılabilir bir şekilde yazılmış olması, iyi belgelendirilmesi, sorun çözümlemesinin ve testinin kolay olması gereklidir.
  - ✓ **Doğrulanabilirlik:** Yazılımın doğru çalıştığının test edebilme kolaylığıdır. Testi mümkün olmayan yazılımın gerektiği zaman doğru çalışması da garanti edilemez.
  - ✓ **Genişleyebilirlik:** Bir yazılım her zaman için kullanıcı isteklerine göre yenden uyarlanabilir bir özelliğe sahip olmalıdır. Ancak yazlımlar büyüdükçe bunu sağlamak güçleşir. Sistemde gerekli değişiklikleri uygun bir şekilde ve kolayca yapabilmek, ona yeni işlevler ekleyebilmek için tasarım ve gerçekleştirimin uygun şekilde yapılması gereklidir.
- **Garvin'in Kalite Ölçütleri**
  - ✓ **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
  - ✓ Güvenirlik
  - ✓ Fonksiyonellik
  - ✓ Bakım Kolaylığı

- **Kalitenin Maliyeti**
    - Bunu cevaplayabilmek için kaliteye ulaşmanın maliyeti ve düşük kaliteli yazılımın maliyetini anlayabilmemiz gerekir. Hataların ortaya çıktığı aşama ile bize getirdiği maliyetin arasında doğru bir orantı vardır. Bu yüzden **kaliteli yazılım üretmenin maliyeti, düşük kaliteli yazılım üretmenin maliyetinden daha düşüktür.**
- **Resmi Kalite Güvence Yöntemleri**
    - **Doğruluğun Kanıtlanması**
        - ✓ Bir yazılımın doğruluğunu kanıtlamanın yöntemlerinden biri, kullanılan programlama dili ile yazılan kodun algoritmalara uygun olduğunu göstermektir.
        - ✓ Otomatik kod üreten yazılım araçlarının geliştirilmesinde bu yaklaşımın da kullanımında yarar vardır.
    - **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.
    - **Temiz Oda Süreci**
        - ✓ Geliştirme ortamının her türlü zararlı etmenden arındırıldığı kabul edilir. Amaç hatanın oluşabileceği ortamı ortadan kaldırmaktır. Fakat bunun için matematik modellerinin çok iyi geliştirilmiş ve oturtulmuş olması gerekir. Bu nedenle temiz oda süreci daha çok, küçük ölçekli yazılım projelerinde kullanılmaktadır.
    - **Yardımcı Araç Desteği**
        - ✓ Sıkça kullanılan yazılım modüllerinin ortak hale dönüştürülmesi ve kalıpsal bir yapıya sokulması hem hataları azaltır hem de geliştirme süresini düşürür.
        - ✓ Nitelik güvence ekibi tarafından benimsenmiş araçlarla yapılan yazılım geliştirme sonucu elde edilen ürün yüksek nitelikli kabul edilir.

- **Nitelik Sistem Standartları**
    - **Yetenek Olgunluk Modeli- (CMM)**
        1. **Başlangıç(İnitial):** Başarı sadece bireylere bağlıdır. Proje denetimi için resmi yordamlar bulunsa da bunların düzenli olarak kullanılmasını sağlayacak bir örgütsel düzenek yoktur. Örgüt başarılı bir yazılım geliştirebilir, ancak yazılım öznitelikleri ve yazılım süreci kestirilemez. Herhangi bir kriz anında işi bırakmak olasıdır.
        2. **Yönetilen(Managed):** Yazılı olmayan ve kısmen tutarlı süreçler vardır. Nitelik güvencesi ve düzenleşim denetim yordamları oluşturulmuştur. Projenin başarısı yöneticilerin bireysel özendirme çalışmalarının sonucudur.
        3. **Tanımlı(Defined):** Firma kültürü yazılı hale gelmiştir. Bir örgüt kendine ait süreçlerle tanımlanır. Resmi yordamları vardır ve tanımlanmış süreçler bütün projelerde izlenmektedir.
        4. **Nicel Olarak Yönetilebilen(Quantitatively Managed):** Tanımlı süreçler ölçülmekte, başarım göstergeleri değerlendirilmektedir. Niceliksel veri toplulukları tanımlanmış, süreç ve ürün ölçütleri

toplanmış ve süreç iyileştirme çalışmasına geçilmiştir. Başarımı kestirmek mümkündür.

5. **En iyilenen(Optimizing):** Kurumsallaşma gerçekleşmiştir. Örgüt süreç iyileştirmeyi sürekli bir hedef haline getirmiştir. Süreç iyileştirme için öz kaynak ayrılmaktadır. Geri beslemelerin sistematik bir şekilde değerlendirilmesine başlanmıştır.

- **Trillium:** Teknolojik olgunluk içeren ve iyileşmeyi bu olgunlukla düzenleyen yol haritası yaklaşımı getiren Trillium modeli, ürün geliştirme süreci içerisinde bir örgüt alanına, bir gereksinime ya da bir ögeye odaklanan uygulamalar seti olarak tanımlanır.
- **Spice:** İki boyutlu bir model olup içe dönük süreç iyileştirme ile dışa dönük yetenek belirleme amacını taşır. Birinci boyutta süreçler, ikinci boyutta yetenek düzeyleri vardır. Yazılım edinme, geliştirme, işletim, bakım ve destek için gerekli olan planlama, yönetim, icra, denetim, ve iyileştirme aracı konularını kapsar.
- **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** 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. Bu ortak özellikler neticesinde Nato'ya Üye ülkeler arasında asgari amaçlı malzeme, parça araç ve gerecin temininde bir seri standart prosedür olarak tanımlanmıştır.

(BÖLÜM-2 ÇEVİK YAZILIM GELİŞTİRME)

- **Günümüzde Yazılım Projelerinin Durumu**
  - 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**
  - 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**
  - ✓ Biraraya gelmiş,
  - ✓ Kendi kendilerine organize olan,
  - ✓ Çapraz fonksiyonlu,
  - ✓ İşine odaklanmış,
  - ✓ Hedefleri net olan,
  - ✓ Teslim edilebilecek düzeyde ürün ortaya koyabilen
  - ✓ Küçük(3-7 kişilik) gruplar.

- **Geleneksel Model vs. Agile**
  - Çağlayan modeli
    - ✓ Yazılım projesini baştan sona planlar.
    - ✓ Gelişim, sunulabilir işler açısından 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 hiç birşey değişmeyecektir.
    - ✓ **Çevik Yöntemler**
      - Müşteriler ne istediğini keşfeder.
      - Geliştiriciler neyi nasıl üreteceğini keşfeder.
      - Bu yol boyunca bir çok değişiklik yapılabilir.

- **Scrum Modeli**
  - **Scrum Takımı**
    - ✓ Ürün 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 süreye sahiptir.
  - ✓ Sonunda ortada değeri olan bir çıktı olmalıdır.
  - ✓ Toplantılarla içerik belirlenir.
  - ✓ Sprint süresi boyunca her gün toplantılar yapılır.
- **User Story:** Müşteri, son kullanıcı veya ürün sahibi için değerli olan ve anlam ifade eden genellikle fonksiyonel özelliklerin belirtildiği ifadelerdir.
- **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.

(BÖLÜM-3 YAZIIM TESTİNE GİRİŞ)

- **Yazılım Testİ**
  - **Yazılım testi,** yazılımın daha önce tanımlanmış teknik ve işlevsel gereksinimleri karşılayıp karşılamadığının ve yazılımın beklendiği gibi çalışıp çalışmadığının kontrolüdür.
  - Yazılım test süreci de temel olarak elde edilen ürünün beklenen kalitede olduğunu belirlemek, değilse istenilen kaliteye ulaştırılmasını sağlamayı amaçlayan bir süreçtir.
- **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 Ve Geçerleme**

DOĞRULAMA	GEÇERLEME
Yazılımı doğru mu üretiyoruz?	Üretilen Yazılım Doğru mu?
Sistemin hatasız ve iyi bir mühendislik ürünü olup olmadığını ölçer.	Sistemin kullanıcı gereksinimlerine uygunluğu ölçer
Geliştiriciler veya QA ekibi tarafından gerçekleştirilir.	Test ekibi tarafından gerçekleştirilir.
Doğrulama aşamasında bulunan hataların maliyeti daha azdır	Geçerleme aşamasında bulunan hataların maliyeti daha fazladır.

- **Doğrulama Süreci**
  - ✓ **Sözleşme Doğrulaması**
    - Geliştirici tüm isterleri karşılayabileceği güvenini vermektedir.
    - İsterler tutarlı olup kullanıcı gereksinimlerini kapsamaktadır. İsterlere yapılacak değişiklikleri ve ortaya çıkabilecek problemleri kontrol edebilmek üzere yordamlar öngörülmüştür.
    - Taraflar arasında sahiplik, garanti, telif hakları ve gizlilik gibi konuları da içerecek şekilde işbirliği yapılmaktadır.
    - İsterlerle birlikte kabul kıstasları ve yordamları öngörülmüştür
  - ✓ **Süreç Doğrulaması**
    - Proje planlamaları yeterli ve takvime uygundur.
    - Proje için seçilen süreçler yeterlidir, planlandığı şekilde yürütülmektedir.
    - Proje süreçleri için seçilmiş standartlar, yordamlar ve ortamlar yeterlidir.
  - ✓ **İsterlerin Doğrulanması**
    - Sistem isterleri tutarlı, gerçekleştirilebilir ve test edilebilir durumdadırlar.
    - Sistem isterleri tasarım ölçütlerine uygun şekilde donanım öğelerine, yazılım öğelerine atanmıştır.
    - Yazılım isterler tutarlı, gerçekleştirilebilir ve test edilebilir durumda olup sistem isterlerine uymaktadır.
  - ✓ **Tasarım Doğrulaması**
    - Seçilen tasarım isterlerden türetilebilmektedir.
    - Tasarım isterlere göre tutarlıdır ve izlenebilir durumdadır.
    - Tasarım, olayların, girdilerin, ara yüzlerin, mantık akışının uygun dizilişlerini, zaman ve büyüklük tahsislerini, hata tanımlarını, hataya dayanıklılığını gerçekleştirmektedir.
  - ✓ **Kod Doğrulaması**
    - Kod, tasarıma ve isterlere göre izlenebilir, türetilebilir ve test edilebilir.
    - Kod doğru ve kodlama standartlarına uygun olmalıdır.
    - Kod, emniyet, güvenlik ve diğer kritik durumlarla ilgili isterleri gerçekleştirmektedir.
    - Kod, yazan kişiden başkası tarafından rahatça okunabilir, anlaşılabilir ve bakım uygulanabilir.
  - ✓ **Belgelendirme Doğrulaması**
    - Belgelendirme yeterli, uygun, tam, anlaşılabilir ve tutarlıdır.
    - Belgelendirme hazırlıkları takvime uygundur.
    - Belgelerin sürüm, baskı ve dağıtım denetimi belirlenmiş yordamlara göre yapılmaktadır.
    - Belgeler gizlilik derecelerine uygun olarak saklanmaktadır.

- **Yazılım Hataları**
  - **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**
    - **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 yada 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.
- **Yazılım Geliştirme Metodolojilerinde Test**
    - **Şelale Modeli**
        - ✓ Bir sonraki safhaya geçebilmek için bir önceki safhada yer alan aktivitelerin tamamlanmış olması gerekir. Yani test aşamasına gelebilmek için diğer aşamalar tamamlanmalıdır.
        - ✓ Hatalar sadece 5. aşamada giderilebildiğinden yazılımın maliyetini artırır ve başarısını azaltır
        - ✓ Kullanıcı katılımı başlangıç safhasında mümkündür. Kullanıcı isterleri bu safhada tespit edilir ve detaylandırılır. Daha sonra gelen tasarım ve kodlama safhalarında müşteri ve kullanıcılar ile diyaloğa girilmez. Bu yüzden kullanıcı testlerinde başarı yakalama oranları azdır.
    - **Agile**
        - ✓ Test profesyonelleri, yazılım geliştirme yaşam döngüsünün en başından itibaren sürece dahil olurlar.
        - ✓ Kullanıcının geliştirici ve test sorumlusu ile aynı ortamı paylaşması prensibi, geliştirilen ürünün kullanıcının gerçekten istediği bir biçimde geliştirilmesi ve gerçekçi kullanım durumlarına göre test edilmesini sağlar.
        - ✓ Test güdümlü geliştirme yöntemini(TDD) kullanır.
        - ✓ **TDD(TEST DRIVEN DEVELOPER):**
            - Tek satır kod yazmadan kodun testini yaz.
            - Testi çalıştır ve testin geçemediğini gör.
            - Testi geçecek en basit kodu yaz ve testin geçtiğini gör.
            - Kodu düzenle
            - Başa dön
    - **V modeli**
        - ✓ Test işlemlerinin ne zaman yapılacağını ön plana çıkarır.
        - ✓ Sol kanat üretim etkinliklerini, sağ kanat da test etkinliklerini gösterir.
        - ✓ Bu modelde geliştirme ve test paralel şekilde yapılır.
        - ✓ Her aşama sonunda test edilecek ürün, test grubu tarafından sınanır, onay verildikçe bir sonraki aşamaya geçilir.
        - ✓ İsterlerin iyi tanımlandığı, belirsizliklerin az olduğu ve aşamalar halinde ilerlenmesi gereken projelerde **«v» modeli** iyi sonuç verir

- **Spiral Model**
  - ✓ Aynı safhalara geri dönülmesinin bir zorunluluk olduğunu vurgular. Proje çevrimlere ayrılır ve her bir çevrimin riskleri ayrı ayrı ele alınır.
  - ✓ Üretim süreci boyunca ara ürün üretme ve üretilen ara ürünün kullanıcı tarafından sınanması temeline dayanır.
  - ✓ Gerek proje sahibi, gerekse yüklenici tarafındaki yöneticiler, çalışan yazılımlarla proje boyunca karşılaştıkları için daha kolay izleme planlaması yapılır.

(BÖLÜM-4 TEST STRATEJİLERİ)

- **Test Tipleri**

Otomasyon Testi	Manuel Test
Testler daha hızlı çalıştırılır.	Otomasyon testinden daha yavaştır.
Bir çok testi defalarca çalıştırabilir.	Bir veya iki kez çalıştırılacak olan testlerde kullanılması uygundur.
Sık sık değişiklik içeren regresyon testlerinde verimli çalışır.	Regresyon testlerini manuel olarak yapmak zordur.
Karmaşık projelerde kolaylık sağlar.	Karmaşık projeler manuel olarak test yapılmaz.
Test otomasyonlarını satın almak maliyetlidir	Daha az maliyetlidir.
Kullanıcı ara yüzü testlerinde bazen verimli olabilir	Kullanıcı ara yüzü testlerinde çok verimlidir.
Daha doğru sonuçlar üretir.	Otomasyon testinden daha az güvenilirdir.

- **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:**
    - ✓ Doğru olmayan ya da kayıp fonksiyonlar
    - ✓ Ara yüz hataları
    - ✓ Veri yapılarındaki hatalar ya da harici veritabanı bağlantısı hataları
    - ✓ Davranış ya da performans hataları
    - ✓ Başlatma ve sonlandırma hataları

AVANTAJLARI	DEZAVANTAJLARI
Kod erişimi gerektirmediği için daha kolaydır.	Test durumu tasarlamak zordur.
Birçok orta vasıflı test uzmanı, uygulamanın içeriği, programlama dili yada çalıştığı işletim sistemi hakkında bilgi sahibi olmadan uygulamayı test edebilir.	Sadece birkaç test senaryosu seçilip uygulandığı için kapsamı dardır.
	Test uzmanı özel kod bölümlerini veya hata eğilimli alanları hedef alamadığı için kapsam yetersizdir.

- **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:**
    - Bütün bağımsız yolların en azından bir kere sınanması,
    - Bütün mantıksal karar noktalarında iki değişik karar için sınamaların yapılması,
    - Bütün döngülerin sınır değerlerinde sınanması,
    - İç veri yapılarının denenmesi

AVANTAJLARI	DEZAVANTAJLARI
Kodun optimize edilmesine yardım eder	Yetenekli bir test uzmanı gerektirdiği için maliyet artar.
Gizli hatalara sebep olabilecek gereksiz satırlar kaldırılabilir.	Gizli hataları bulmak için her uç noktaya bakmak mümkün olmadığı zaman ufak problemler ortaya çıkabilir.
Test uzmanının kod hakkında bilgili olması sebebiyle, test senaryosunun kapsamı çok geniştir.	Kod analizcisi ve hata ayıklayıcı gibi bazı özel araçların kullanımını gerektirir.

- **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.

AVANTAJLARI	DEZAVANTAJLARI
Beyaz kutu ve kara kutu testinin yararlarının birleşimini sunar.	Kaynak koda erişim sınırlı olduğundan kod inceleme ve test kapsamı sınırlıdır.
Gri kutu test uzmanları, kaynak kod yerine ara yüz tanımlamaları ve fonksiyonel özellikleri kullanır.	Yazılım tasarımcısı halihazırda bir test çalıştırıyorsa gri kutu testi gereksiz olabilir.
Test, tasarımcı bakış açısıyla değil kullanıcı bakış açısıyla yapılır.	Mümkün olan her giriş sisteminin test edilebilmesi gerçekçi değildir çünkü çok fazla zaman alır. Bu yüzden bazı program yolları test edilemez.

- **Test Seviyeleri**

- **Birim(Unit) Testi**
  - ✓ Birim testi, yazılım tasarımının en küçük biriminin (yazılım bileşeni yada modül) doğrulanmasıdır. Önemli kontrol yolları, modülün sınırları içerisindeki hataları ortaya çıkarmak için test edilir.
  - ✓ 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. Örneğin, programa giren ve çıkan tüm iletilerin doğru tipte oldukları gerçeklenir.
  - ✓ **Yerel veri yapıları** incelenerek algoritmanın çalışması boyunca ya da yordamların çağrılması sırasında verilerin saklandığı yerin bütünlüğünün bozulup bozulmadığı test edilmelidir.
  - ✓ **Sınır koşulları**nın en düşük ve en yüksek değerleri, bu değerlerin biraz altı ve biraz üstü kullanılarak sınanmalıdır.
  - ✓ Birim içindeki birbirinden **bağımsız tüm çalışma yolları**, tüm dallanmalar tek tek sınanmalıdır.
  - ✓ Birim içindeki tüm **hata yakalayıcılar** birer birer denenmelidir.
- **Tümleştirme(Integration) Testi**
  - ✓ Birden fazla biriminin bir araya getirilerek uyumlu bir şekilde ve hatasız çalışması, her birinin tek tek değil de bir bütün içinde, tasarımda belirtildiği şekilde kendi üzerlerine düşen görevleri yerine getirip getirmediği **tümleştirme testi** ile kontrol edilir.
  - ✓ 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.
  - ✓ **Yukarıdan Aşağı Tümleştirme**
    - Bu stratejide, önce ana denetim biriminin testi yapılır, sonra ona en yakın düzeydeki birimlerden biri ile beraber test yapılır.
  - ✓ **Aşağıdan Yukarıya Tümleştirme**
    - Alt düzey birimler birleştirilerek kümeler haline getirilir. Bu kümeler test edilir. Daha sonra bu kümelerin birleştirilmesinden oluşan daha üstü düzeyde kümeler

meydana getirilir. Bu şekilde en üstte bulunan ana birime kadar ulaşılır.

- ✓ **Regresyon Testi**
  - Modül eklendiği veya değiştirildiği zaman yazılım değişir. Yeni veri akış yolları oluşur, yeni giriş/çıkışlar meydana gelir ve yeni mantık yapıları çağırılır. Bu değişiklikler daha önce sorunsuz olarak çalışan fonksiyonlarda problemlere sebep olabilir.
  - Tümleştirme test stratejisi kapsamında regresyon testi; uygulama ortamındaki yapılan tüm değişikliklerin yeni bir hata üretip üretmediğini kontrol amaçlı olarak yapılan test türüdür.

- **Sistem Testi**
  - ✓ **Performans(Performance) Testi**
    - Performans testi, sistemin belirli durumlarda, belirlenen beklentileri verip vermediğini kontrol etmek amacıyla yapılan testtir.
    - Performans testi sistemdeki hataların bulunmasını amaçlamaz ancak sistemdeki darboğazları ortaya çıkarır.
    - Performans testlerinde amaç sistemin bir açığını bulmak değildir. Asıl amaç sisteme yapılan girdilerden alınan çıktılarla, olması gereken sonuçların uygunluğunu tespit etmektir.
  - ✓ **Yük (Load) Testi**
    - Yük testleri, sistemin sınırlarını zorlayarak en fazla ne kadar veri işleme kapasitesi olduğunu belirlemek, bu yükte davranışlarını kontrol etmek amacıyla yapılır. Hatta, bazı durumlarda, isterlerde belirtilen değerlerin de üzerine çıkılarak kaldırabilecek en fazla yükün ne olabileceği araştırılır.
    - **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**
    - Normal olmayan koşullarda, hem yazılım hem de donanımın ne şekilde davranacağını görmek üzere germe(stress) testleri yapılmalıdır.
    - 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**
  - Bilgisayar sistemlerinin çoğunda bir hata durumunda kendini toparlayarak tekrar çalışmaya devam etmesi beklenir. Aşağıdaki yöntemler kullanılarak zararlar en aza indirilebilir.
  - **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**
  - Güvenlik testi, bir bilgi sisteminin verileri ve işlevselliğini korumak için tasarlanmış bir süreçtir. Herhangi bir bilgi sızıntısı olup olmadığını kontrol edilir. 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**
  - Taşınabilirlik testi, var olan bir yazılım bileşeni veya uygulamayı yeni bir ortamda test etme işlemidir.
  - Uygulamanın diğer ortamlarda; Installability , Combatibility, Adaptability , Replaceability yetenekleri araştırılır. Sonuçlar rapor edilir.
- ✓ **Kullanılabilirlik(Usability) Testi**
  - Tasarımların veya ara yüzlerin kullanıcı ile buluşmasından önce tasarımın kullanılabilirliğini ölçmek amacıyla yapılan testlere denir.
  - Kullanılabilirlik testleri kullanılabilirlik problemleri hakkında bize bilgi verir ve kullanıcıların uygulama ile nasıl etkileşimde bulunduğuna bakar.
  - **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?
    - **Kabul Testi**
        - ✓ Çalıştırılmadan önce yazılımın son sınanmasıdır. Artık yapay veriler yerine gerçek veriler kullanılır. Bu sınama türü;
        - ✓ **Alfa sınaması**
            - Geliştiricinin kendi yerinde müşteri tarafından yapılır. Geliştirici bu testleri gözlemleyerek gerçek kullanım hakkında bilgi sahibi olmaya çalışır; kusur buldukça not alır ve düzenleme işlemlerini yürütür. Alfa testlerinin en önemli özelliği, denetim altındaki bir ortamda, asıl kullanıcılardan biri tarafından yapılıyor olmasıdır.
        - ✓ **Beta sınaması**
            - Birçok kullanıcının kendi ortamında yapılır. Geliştirici genellikle bu testlere katılmaz; yalnızca belirli aralıklarla sonuçları ve yorumları alır. Bu testin özelliği de geliştirici tarafından kontrol edilemeyen gerçek uygulama ortamı koşullarında yazılımın denenmesidir. Beta testi sonunda geliştirici, bulunan kusurları düzelterek tüm kullanıcılar için yeni bir sürüm çıkartır.

(BÖLÜM-5 WEB UYGULAMASI TESTİ)

- Web kullanıcıları ne ister?
    - Hız
    - Güvenlik
    - Kolay üyelik
    - Düşünmemek
    - Klavye kullanımı
    - Açık hata mesajları
    - Sadece gerekli bilgi
- **Web Uygulaması Testi**
    - **İçerik Testi**
        - ✓ Son kullanıcıya sunulacak içeriğin yapısı veya bütünlüğündeki hataları bulmak için,
        - ✓ Herhangi bir içerikteki sözdizimsel hataları ortaya çıkarmak için,
        - ✓ Herhangi bir içerikteki anlam hatalarını bulmak için yapılır.
    - **Veri Tabanı Testi**
        - ✓ Veri tabanı kaynakları sorgulanır.
        - ✓ İlgili veriler veri tabanından ayıklanır.
        - ✓ Ayıklanan veriler içerik nesnesi olarak düzenlenmiş olmalıdır.

- ✓ İçerik nesnesi, görüntülenmek için istemci ortamına aktarılır.

- **Kullanıcı Arayüzü Testi**
  - ✓ Bir kullanıcı, web uygulamasıyla etkileşime geçtiği zaman bir yada daha fazla ara yüzle etkileşim meydana gelir. Her bir ara yüz mekanizması için aşağıdaki denetimler yapılmalıdır.
  - ✓ **Linkler:** Her bir link, doğru içerik ve fonksiyona ulaşıp ulaşmadığından emin olmak için test edilir.
  - ✓ **Formlar:**
    - Her bir form alanı uygun genişlik ve veri tipinde oluşturulup oluşturulmadığından emin olmak için test edilir.
    - Formlar, ön tanımlı uzunluktan daha uzun veriler girildiğinde kullanıcıyı uyarmalıdır.
    - Tüm form elemanları kullanıcının anlayabileceği, uygun, kullanışlı bir biçimde sıralanmalıdır.
    - Tarayıcıların otomatik doldurma özelliğiyle yanlış veri girilmesine engel olunmalıdır.
    - Tab tuşu(veya diğer klavye tuşları), form elemanları üzerinde uygun şekilde hareket etmelidir.
    - Form üzerinde hata denetimi yapan scriptler, hata oluştuğunda anlaşılabilir ve yönlendirmeler için hata mesajı yayımlamalıdır.
  - ✓ **Uygulamaya Özgü Arayüzler:** Testler arayüz mekanizması tarafından tanımlanan özellikler ve fonksiyonların kontrol listesine göre uygulanır.
- **Kullanılabilirlik Testi**
  - ✓ **Kullanılabilirlik:** 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. ISO 9241
  - ✓ **Kişiselleştirme**: Kullanıcılar siteyi isteklerine göre kişiselleştirebiliyor mu?
  - ✓ **Görüntü Karakteristikleri**: Uygulama uygun boyutta ve uygun çözünürlükte görüntülenebiliyor mu?
  - ✓ **Etkileşim**: Menüler, butonlar, işaretleyiciler gibi etkileşim mekanizmalarını anlamak ve kullanmak kolay mı?
  - ✓ **Düzen**: Navigasyonlar, içerik ve fonksiyonlar kullanıcının kolayca bulabileceği şekilde yerleştirilmiş mi?
  - ✓ **Okunabilirlik:** Yazılar ve Grafik sunumları kolay anlaşılabilir mi?
  - ✓ **Estetik:** Resimler, renkler, yazı tipi uyumlu mu? Kullanıcı uygulamaya bakınca iyi hissediyor mu?
  - ✓ **Zaman duyarlılığı**: Önemli özellikler, fonksiyonlar ve içerikler kısa zamanda görüntülenebiliyor mu?
  - ✓ **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.
- ✓ **A/B Testi :** A/B testleri ile bir sayfanın iki (ya da daha fazla) değişik versiyonu test edilir. İki değişik tasarımdan hangisinin daha başarılı olduğu ya da bir sayfa elemanın hangi versiyonunun daha verimli olduğu A/B testleri kullanılarak belirlenebilir.
- ✓ **Ağaç Testi:** Ağaç testi, bir web sitesindeki konuların, başlıkların bulunabilirliğini ölçmek için yapılan bir kullanılabilirlik testidir. Geniş içerikli bir web site, genellikle konular ve alt konuların bulunduğu bir hiyerarşik bir yapıda oluşturulur. Ağaç testi, kullanıcıların bu hiyerarşide aradıkları nesneleri kolayca bulabilmeleri için yapılır.
- ✓ **Gerilla Testi:** Gerilla test, kullanıcıların göz izleme cihazı olmadan ve laboratuvar dışında test edildiği bir metottur. Test kullanıcın yanında bir moderatör tarafından yönetilir ve kullanıcıdan sesli düşünmesi istenir. Test süresince kullanıcının mouse hareketleri, mimik ve sesli düşünceleri kayıt altına alınır. Gerilla test, göz izleme testine kıyasla daha ekonomik bir metottur.
- ✓ **Göz İzleme Cihazı İle Kullanılabilirlik Testi:** Kullanıcının nereye, ne kadar süre ve kaç kere baktığına, anlık ve geçmiş dikkatinin nerede yoğunlaştığına, niyetine, zihinsel durumuna ilişkin bilgi sağlamakta kullanılan bir yöntemdir.

- **Uyumluluk Testi**
  - ✓ Çözünürlük, bağlantı hızı, istemci işlem hızı gibi faktörlerden dolayı farklı sonuç verebilir. Buradan doğacak istenmeyen sonuçları engellemek için uyumluluk testi yapılır. Her farklı kombinasyon için var olan arayüz, navigasyon, performans ve güvenlik testleri uygulanır.
  - ✓ **Donanım:** CPU, memory, storage, and printing devices ¬
  - ✓ **İşletim Sistemleri**: Linux, Macintosh OS, Microsoft Windows, a mobilebased OS
  - ✓ **Tarayıcılar**: Firefox, Safari, Internet Explorer, Opera, Chrome vb.
  - ✓ **Kullanıcı Arayüz Bileşenleri**: Active X, Java applets vb.
  - ✓ **Plug-ins**: QuickTime, RealPlayer vb.
  - ✓ **Bağlantılar**: Cable, DSL, regular modem, T1, WiFi
- **Bileşen Seviyeli Test**
  - ✓ **Bileşen seviyeli test**, web uygulaması fonksiyonlarındaki hataları ortaya çıkarmak için yapılır. Her bir fonksiyon yazılımın bir bileşenidir ve kara kutu (bazen beyaz kutu) yöntemiyle test edilir. Formlara değerler girilerek doğru çıktının alınması beklenir.

- **Navigasyon Testi**
  - ✓ **Navigasyon linkleri**: Bu mekanizmalar web uygulaması içindeki iç bağlantıları, diğer web uygulamaları içindeki dış bağlantıları ve spesifik bir web sayfası içindeki **anchor** linkleri oluşturur. Her bir link tıklandığı zaman doğru içerik ve fonksiyona gidiyor mu diye test edilmelidir.
    - **Anchor link:** Bir web sayfası içinde, aynı sayfanın belirli bir bölümüne yönlendirme yapan linklerdir
  - ✓ **Yeniden yönlendirme:** kullanıcı var olmayan bir URL yi açmaya çalıştığında ya da içeriği kaldırılmış bir linke tıkladığında yeniden yönlendirmeler çalışmalıdır. Web sayfasının böyle bir durumda nasıl davrandığı test edilir. Kullanıcıya bir mesaj gösterilip diğer bir sayfaya yönlendirilmelidir.
  - ✓ **Yer imleri:** Yer imleri bir tarayıcı fonksiyonu olmasına rağmen, web uygulamasının yer imi oluşturulurken anlamlı bir başlığının olup olmaması test edilmelidir.
  - ✓ **Site haritası:** Site haritası, sitedeki tüm web sayfaların içeriklerinin haritasıdır. Site haritasındaki her bir elemanın doğru sayfadan yönlendirilip yönlendirilmediği test edilir.
  - ✓ **İç arama motorları:** Kompleks web uygulamaları yüzlerce hatta binlerce içerik nesnesi içerebilir. İç arama motorları kullanıcıların anahtar kelimelerle ihtiyacı olan içeriğe ulaşmasına izin verir. Arama motoru testi, aramanın tamlığını ve doğrulunu, istisna yakalama özelliğini ve gelişmiş arama özelliklerini doğrular.
- **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. (kullanıcıID ve şifre gibi)
  - ✓ **Authentication (Doğrulama):** Tüm sunucu ve istemci kimliklerini doğrulayan bir mekanizmadır, her iki taraf da doğrulandığı zaman iletişime izin verir.
  - ✓ **Firewall:** İnternetten gelen bilgileri denetleyen ve ardından güvenlik duvarı ayarlarınıza göre engelleyen veya geçişine izin veren bir yazılım veya donanımdır.

        - ✓ **Encryption:** Gizlenmek istenen bir bilginin bir algoritma yardımıyla bir başkası tarafından okunmasını ya da değiştirilmesini engellemek için veri üzerinde yapılan işleme şifreleme denir.
    - **Performans Testi**
        - ✓ Performans testleri, sunucu tarafı kaynaklarının eksikliği neden olan performans sorunları ortaya çıkarmak için kullanılır.
        - ✓ **Yük testi**
            - Yükü çeşitli düzeylerde ve kombinasyonlarda yükleyerek uygulamanın davranışını incelemektedir.
            - Yük testinin amacı, web uygulamasının çeşitli yükleme koşullarında nasıl davrandığını belirlemektir.
        - ✓ **Stres testi**, Web uygulamasının ne kadar kapasitesi olduğunu ve kırılma noktasını belirlemek için yapılır.

(BÖLÜM-6 MOBİL UYGULAMA TESTİ)

- **Mobil Uygulama Türleri**
    - **Native Uygulamalar**
        - ✓ Native bir uygulama, belli bir platforma özel, genellikle platform sağlayıcısının üretmiş/belirtmiş olduğu platform SDK'sı, araçları ve programlama dili yardımıyla geliştirilmiş uygulamadır. Örneğin Objective-C programlama dilini kullanarak IOS işletim sistemine özel ya da Java programlama dili ile Android işletim sistemine özel geliştirilen bir uygulama Native uygulamadır.
    - **Web Tabanlı Uygulamalar**
        - ✓ Mobil web uygulamaları ise HTML5, CSS3 ve Javascript gibi web teknolojileri kullanılarak gerçekleştirilir ve adından anlaşılacağı üzere "Web" tabanlıdırlar. Mobil işletim sistemlerine özgü üretilmiş mobil web tarayıcıları üzerinde sorunsuz çalışmaları için optimize edilmişlerdir. Mobil Web uygulamaların en öne çıkan özelliği, web tabanlı oldukları için tek seferde geliştirilip, tüm mobil platformda çalışabilir olmalarıdır.
    - **Hibrid Uygulamalar**
        - ✓ Hibrit uygulamalar Native ve HTML5'in karışımından meydana gelen uygulamalardır. Tıpkı Native uygulamalarda olduğu gibi cihaza özel geliştirilir ve HTML 5'de olduğu gibi web teknolojisi kullanılarak yazılır
- **Mobil Uygulama Testi**
    - **Kurulum Testi**
        - ✓ Herhangi bir mobil uygulamanın kurulumu basit olmalı ve kurulumun ilerle durumu hakkında bilgiler vermelidir. Test uzmanları kurulum testi ile bu durumu kontrol eder. Böylece kullanıcı uygulamanın yüklenme zamanını tahmin etmek zorunda kalmaz.
    - **Fonksiyonel Test**
        - ✓ Uygulama tüm cihazlarda test edilmelidir.
        - ✓ Uygulamanın tüm fonksiyonları test edilmelidir.
        - ✓ Bellek tamamen doluysa da uygulama düzgün çalışmalıdır.

- ✓ Uygulama, sayfa yüklenirken veya yönlendirilirken zorla durdurulmuş ise bilgilendirici mesaj vermelidir.
- ✓ Uygulama herhangi e-ödeme sistemi veya bilgi içeriyorsa verilerin son derece güvenli olduğundan emin olunmalıdır.
- ✓ Monkey testi gerçekleştirilmelidir.

- **Güç Tüketim Testi**
  - ✓ Uygulamanın çalışma sürecinde cihazın bataryasını ne oranda kullandığı test edilir. Çok şarj tüketen bir uygulama kullanıcı için verimli olmayacaktır
- **Kesme Testi**
  - ✓ **Ağ bağlantısı çeşidi:** Uygulama Wi-Fi, 4g, 3g ve ya 2g bağlantılarında kolaylıkla çalışmalıdır.
    - Cihaz herhangi bir bağlantıdan diğerine geçerse, uygulama bunu otomatik olarak algılayarak diğer ağa bağlanmalıdır.
    - Cihazda sim kart takılı olmasa da uygulama düzgün çalışmalıdır.
    - Uçak modu aktifken de uygulama düzgün çalışmalıdır.
  - ✓ **SD Kart Etkileşimi**: Bu faktör mobil cihazlarda hayati bir rol oynar.
    - Uygulama SD kartın takılı olmaması ve herhangi bir nedenden dolayı çalışmaması durumunda işlevleri yerine getiremediği zaman kullanıcıya bilgilendirme mesajları sunmalıdır.
    - Uygulamada, "SD karta taşı" şeklinde seçenek olmalıdır.
  - ✓ **Aramalar Ve Mesajlar:** Mobil cihazın arama, SMS gibi temel işlevlerinin devreye girdiği ve uygulamanın çalışmaya ara verdiği süreç incelenir. Yeniden başlatıldığında uygulamanın işlevini olması gerektiği gibi sürdürüp sürdürmediği test edilir.
- **Kullanılabilirlik Testi**
  - ✓ **Kullanılabilirlik:** Bir mobil uygulama geliştirilirken ekstra dikkat gerektiren faktör kullanılabilirliktir. Herhangi bir uygulamanın başarısı ya da başarısızlığı kullanıcı ara yüzüne bağlıdır.
  - ✓ Uygulamanın ara yüzü her cihaz için uyumlu olmalıdır. Ekrana sığmamış, kaymış kontroller olmamalıdır. Yazılar okunaklı olmalıdır.
  - ✓ Uygulama veya sayfa yüklenirken ilerleme göstergesi bulunmalıdır. Böylece kullanıcı bazı verilerin yüklendiğinin farkında olur.
  - ✓ Buton boyutları düzgün olmalıdır ve geniş parmak tipleri tarafından kolayca
- **Performans Testi**
  - ✓ Kullanıcı arayüzü ve fonksiyonel testlerde fark edilemeyen istisnaları bulur.
  - ✓ Uygulama beklenen maksimum yük taşıma kapasitesinden %20 daha fazlasını taşıma yeteneğine sahip olmalıdır.
  - ✓ Cihazın bataryası düşük iken uygulama çalıştırıldığında, cihazın işletim sistemi tarafından dayatılan herhangi bir aksaklık veya askıya alınma

olmadan uygulamanın problemsiz bir şekilde çalıştığı doğrulanmalıdır.

- **Mobil Yazılım Test Ortamları**
    - **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.
    - **Gerçek Cihazlar:** Gerçek cihazlarda test, bize en doğru test sonuçlarını verir. Test faaliyetlerinin tüm türleri, donanıma bağlı olanlar da dahil olmak üzere, kolayca yapılabilir. Kullanıcı deneyimini test etmek için kullanılabilir.
    - **Bulut tabanlı test ortamları:** Mobil cihazlara web arayüzü yani tarayıcı ile erişilebilir. Test faaliyetlerinin tüm türleri kolayca yapılabilir. Var olan tüm mobil cihazlarda uygulamayı kısa sürede test etmek mümkündür. Bu sayede çok büyük kolaylık sağlar. Bize hafıza kullanımı, işlemci kullanımı da dahil olmak üzere birçok konuda ayrıntılı raporlar sunar. Hataları açıkça bildirir.

(Bölüm-7 TEST SÜRECİ VE YÖNETİMİ)

- **Test Süreci Ve Yönetimi**
  - **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

- **Yazılım Test Süreci**
  - **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.
    - ✓ **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.
    - ✓ **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**
    - ✓ **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 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*
- **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

- **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.
- **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.*

- **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**
  - ✓ Kullanıcının geliştirilen yazılım ile yapmak istediklerinin yazılım tarafından yapılmaması veya eksik yapılmasına *hata* denilir
  - ✓ **Hata Raporlama**
    - En üst düzeyde hatalardan arındırılmış bir yazılımın geliştirilmesi isteniyorsa testler sırasında çıkan tüm hataların kapatıldığından emin olunmalıdır. Bulunan hatalar, hataların kaynağı, yapılan düzenlemeler raporlanmalıdır.
  - ✓ **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
  - ✓ Test özet belgesinde test sonuçları analiz edilip değerlendirilirken kaç adet ihtiyacın, kaç adet test durumu ile kaç kere test edilerek doğrulandığı, ne tür hatalar çıktığı ve bu hataların hangi yazılım sürümünde düzeltilerek testlerden geçtiği raporlanır.
- **Yazılım Test Riskleri**
  - ✓ Normal bir süreç olarak izlenen yazılımın test aşamasında da karşılaşılması olası temel riskler şunlardır:
  - ✓ **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.

(Bölüm-8 GÖZDEN GEÇİRME TEKNİKLERİ)

- **Gözden Geçirmeler**
  - Çözümleme, tasarım ve kodlama bir yada birkaç kişi tarafından ne kadar iyi yapılmış olursa olsun, bir başka kişi ya da kişilerin hata bulması her zaman olasıdır. Geliştirme aşamasında bulunan her hata, ürünü mükemmelliğe daha çok yaklaştırır. Bu nedenle de herkesin her yaptığı işe *gözden geçirme (review)* uygulanmalıdır.
  - Geliştirme sürecinin çeşitli evrelerinde yazılım ürününün yazarından başka kişiler tarafından incelenmesi şeklinde uygulanarak ;
    - ✓ Kusurların ortaya çıkarılmasını,
    - ✓ Uygun şekilde düzeltilmesini,
    - ✓ Ürünün daha da iyileştirmesini sağlarlar.
  - Kusur her zaman bir hata olmayabilir, fakat yanlış anlama yada deneyim eksikliği nedeniyle *asıl isterlerden ve standartlardan sapma* olarak da değerlendirilebilir.
  - **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.

- **Resmi teknik gözden geçirme aşamaları:**
  - ✓ **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.
  - ✓ **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.

- **Yazılım Ürün Değerlendirmesi**
    - Her yazılım aşamasının sonunda mutlaka bir çıktı bulunur. Bu çıktı bir belge ya da bir yazılım kodu olabilir. 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.

Kontrol Sorusu	Karar	Notlar
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?		

Örnek bir inceleme kontrol listesi

- **Yazılım Ürün Metrikleri**
    - Kelime anlamı olarak metrik, ölçülebilen değerlerdir. Yazılımların, test edilebilmeleri ve sonrasında müşteriye teslim edilebilmesi için belli bir olgunluğa gelmesi gereklidir. Yazılımın ve testin belli bir olgunluğa geldiğini *"Metrik"* denilen değerler ile ölçümleriz.
    - **Ürün metrikleri:**
        - ✓ Proje durumunun izlenmesi ve gözlenmesinde kullanılır.
        - ✓ Bir projenin özkaynak 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**
  - **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. Bunlardan bazıları şunlardır:
  - ✓ Yordamların içinde yer alan toplam satır sayısı 60'ı, çalışır deyimlerin sayısı da 50'yi geçmemelidir.
  - ✓ Açıklama satırları yordam satır sayısının en az yarısı kadar olmalıdır.
  - ✓ Bir değişkene yapılan atama ile o değişkenin bir yordam içinde kullanılması arasındaki satır sayısı 10'dan fazla olmamalıdır.
- **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)**
    - ✓ Thomas J. McCABE tarafından ortaya konan bu metrik bir metodun içindeki algoritmanın karmaşıklığını ölçmek için kullanılır. Aynı zamanda bir metodun test edilmesi için gerekli test durumu sayısını da verir.
    - ✓ **CC = yollar – düğümler +2** şeklinde hesaplanır. Örneğin bir IF cümlesi iki seçeneğe sahiptir. Eğer şart doğru ise birinci yol test edilmiş olur , hatalı ise ikinci yol test edilir.

- **Nesne Yönelimli Metrikler**
  - **Sınıf Başına Ağırlıklı Method (Weighted Methods per Class -WMC)**
    - ✓ Döngüsel karmaşıklık değerleri toplamının sınıf sayısına bölümü ile bulunur. WMC değerinin 100'ün altında olması tercih edilmelidir. WMC sınıf karmaşıklığı hakkında fikir veren en önemli donelerden biridir. Bu metriğin faydaları sınıf başına ortalama metot sayısı gibi değerlendirilebilir. WMC değerinin 100'ün altında olması kabul edilebilir bir değerdir. WMC değeri NOM'dan farklı olarak sınıfın karmaşıklığı hakkında daha net bir fikir verir.
  - **KALITIM AĞACININ DERINLIĞI (DEPTH OF INHERITANCE TREE - DIT)**
    - ✓ DIT metriği sınıfın ebeveyn sınıflarının sayısını gösterir. Eğer çoklu kalıtım durumu söz konusu ise bu durumda hiyerarşideki en uzun yol kabul edilir.
    - ✓ Sınıf hiyerarşisinin derinliği arttıkça daha fazla metot kalıtım alınır ve sınıfın davranışlarını öngörmek; anlamak zorlaşır.
    - ✓ Derin hiyerarşiler daha fazla metot ve sınıfı etkilendiğinden, dizaynı karmaşıklaştırır. Ancak hiyerarşi derinleştikçe kalıtım alınan metotların yeniden kullanım potansiyeli artar.
    - ✓ 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.
  - **Alt Sınıf Sayısı (Number of Children - NOC)**
    - ✓ NOC metriği sınıftan türemiş alt sınıflarının sayısını verir.
    - ✓ Alt sınıf sayısı çoğaldıkça kalıtım özelliğine bağlı olarak yeniden kullanımın arttığı anlaşılır.
    - ✓ NOC sınıfın nüfus alanı hakkında bir fikir verir. NOC metriği yüksek sınıflar gözden geçirme, test gibi süreçlerin daha dikkatli ve uzun tutulması gereken yerlerdir.
  - **Nesne sınıfları arasındaki bağımlılık (Coupling Between Object Classes - CBO)**
    - ✓ 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.
    - ✓ 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.

- **Metotlardaki uyum eksikliği (Lack of cohesion in methods - LCOM)**
  - ✓ 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.

(Bölüm-9 YAZILIM TEST ARAÇLARI)

- **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 Performans Test Araçları**
  - **GTmetrix**
    - ✓ 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 Page Speed**
  - ✓ 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**'yı diğerlerinden ayıran özellik, aynı anda birçok farklı konumdan sorgulama yapabilmesidir. . Aynı anda pek çok ölçütü bu servis aracılığıyla değerlendirebilir ve bunlara bağlı olarak sitenizin performans sonucunu anlaşılır grafiklerle izleyebilirsiniz.

- **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 yada 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.
  - **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.
- **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.
  - **Core impact Professional**

- **Mobil Test Araçları**
    - **Crittercism,** mobil uygulama performansınızı her açıdan değerlendirebilmenize olanak sağlıyor.
    - **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** mobil uygulama geliştiricilerin tercih edebileceği bir diğer uygulama performansını ölçme (application performance management) araçlarından bir tanesidir.
    - **Instabug**'ın SDK'sını uygulamanıza entegre ettiğiniz andan itibaren kullanıcılar uygulamanın açık olduğu anda telefonlarını sağ ve sola sallayarak uygulamanız hakkında size geri bildirimde bulunabiliyorlar.
    - **Pulse.io,** Mobil uygulamaların test aşamasında gözden kaçan hatalarını yakalamak ve kod performansı konusunda ölçümleme ve analiz yapan bir servistir.
- **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.
    - **SessionCam,** 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.
    - **TrymyUI,** 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.
    - **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. 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.