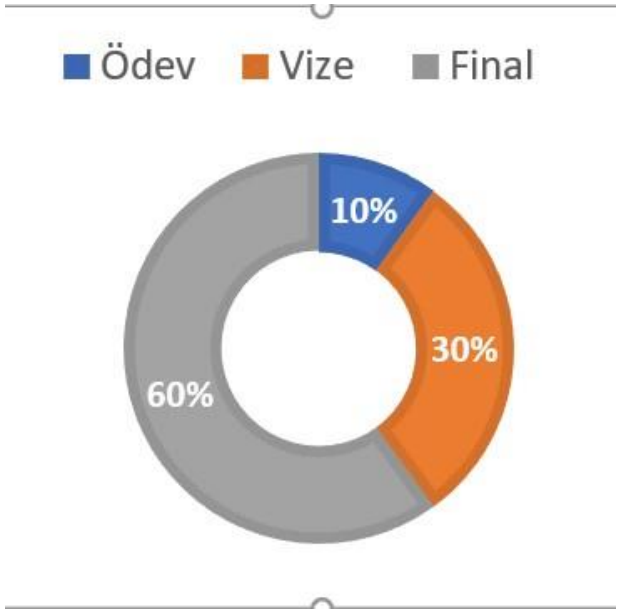
Yazılım Mühendisliği
1906003082015

Dr. Öğr. Üy. Önder EYECİOĞLU
Bilgisayar Mühendisliği

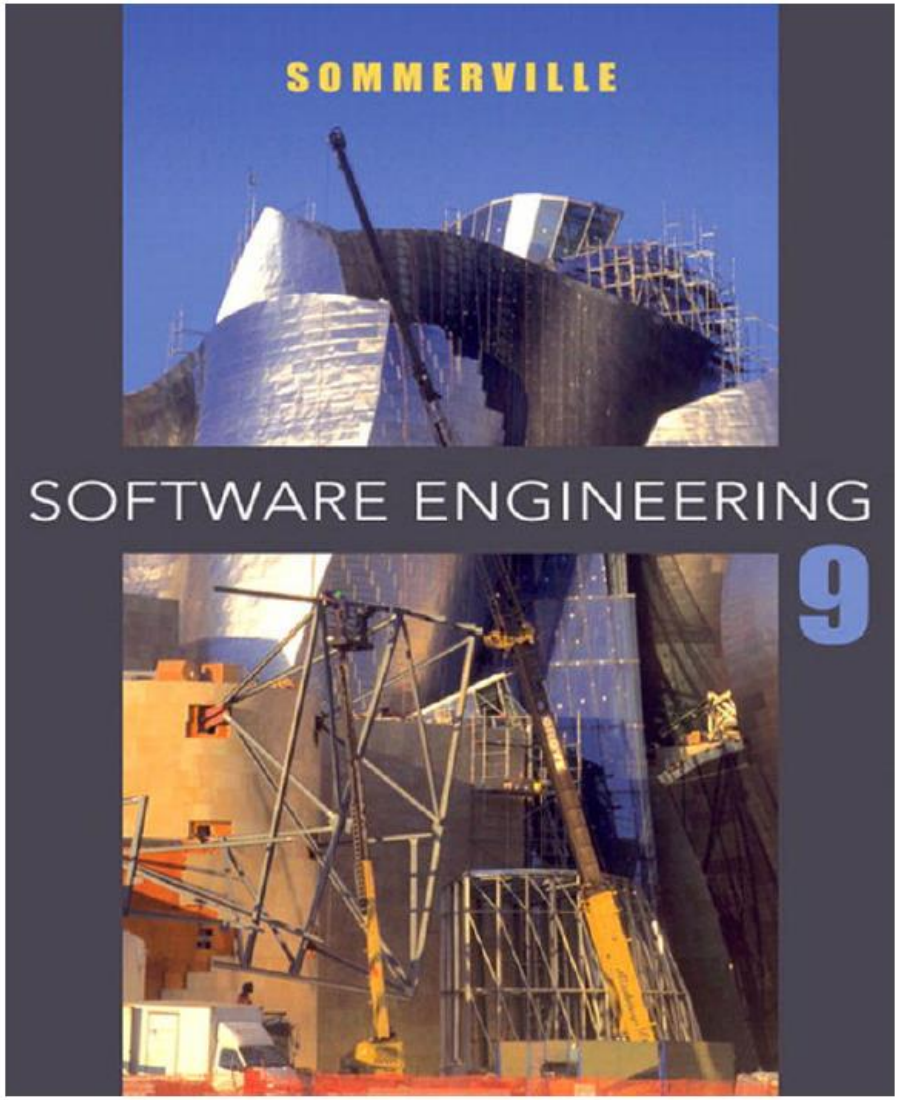
Giriş

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



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

Kaynaklar



1.BÖLÜM

GİRİŞ

Yazılım mühendisliği devletin, toplumun ve ulusal ve uluslararası iş dünyası ve kurumların işleyişi için gereklidir. Modern dünyayı yazılım olmadan yürütemeyiz. Ulusal altyapılar ve servisler bilgisayar tabanlı sistemler tarafından kontrol edilir ve elektrikli ürünlerin çoğu bilgisayar ve kontrol yazılımı içerirler.

Endüstriyel üretim ve dağıtım, iktisadi sistem gibi tamamen bilgisayarlaştırılmıştır. Eğlence, müzik endüstrisi dahil, bilgisayar oyunları, film ve televizyon, yazılım yoğunlukludur. Hâlâ çok sayıda kötü giden yazılım projesi ve “yazılım arıza”ları raporlanmaktadır. Yazılım mühendisliği modern yazılım geliştirme için yetersiz olmakla eleştirilmektedir.

Fakat kanımca, bir çok yazılım arızası iki etmenin sonucudur:

1. Artan sistem karmaşıklığı.

2. Yazılım mühendisliği yöntemlerini kullanmadaki başarısızlık.

Yazılım mühendisliğinin tarihçesi

Yazılım mühendisliği kavramı ilk olarak 1968'de o zaman yazılım krizi (Naur ve Randell 1969) olarak adlandırılan konuları tartışmak için düzenlenen bir konferansta önerilmiştir. Program geliştirmedeki birbirinden farklı yaklaşımların büyük ve karmaşık yazılım sistemlerinin geliştirilmesinde ölçeklenmediği anlaşılmıştı. Yazılım sistemleri güvenilir olmamakta, beklenenden daha fazla maliyetle geç teslim edilmekteydi.

1970'ler ve 1980'ler boyunca, yapısal programlama, bilgi saklama ve nesneye yönelik geliştirme gibi çeşitli yeni yazılım mühendisliği teknikleri ve yöntemleri geliştirilmiştir. Günümüzdeki yazılım mühendisliğinin temeli olan araçlar ve standart gösterimler geliştirilmiştir.

PROFESYONEL YAZILIM GELİŞTİRME

- Birçok kişi program yazar. İşteki insanlar işlerini kolaylaştırmak için hesap çizelgesi (spreadsheet) programları yazarlar; bilim adamları ve mühendisler deneysel verilerini işlemek için program yazarlar; amatörler kendi ilgileri ve merakları için program yazarlar.

- Ancak çoğu yazılım geliştirme, yazılımın iş amaçları için geliştirildiği, diğer cihazlara dahil edilmek üzere veya bilgi sistemleri ve bilgisayar destekli tasarım sistemleri gibi yazılım ürünleri olarak geliştirildiği profesyonel bir etkinliktir.

- Temel farklılıklar, profesyonel yazılımın geliştiricisinden ayrı bir kişi tarafından kullanılmasının amaçlanması ve genellikle yazılımın bireyler yerine takımlar tarafından geliştirilmesidir. Yazılım tüm yaşamı boyunca değiştirilir ve bakımı yapılır.

- Yazılım mühendisliğinin, bireysel programlamadan çok profesyonel yazılım geliştirmeyi desteklemesi amaçlanmıştır. Program spesifikasyonu, tasarımı ve evrimi gibi normal kişisel yazılım geliştirme için önemli olmayan teknikler içerir.

Soru	Cevap
Yazılım nedir?	Bilgisayar programları ve ilişkili belgeleme. Yazılım ürünleri belirli bir müşteri için geliştirilebilir veya genel bir pazar için geliştirilebilir.
İyi yazılımın özellikleri nelerdir?	İyi yazılım kullanıcıya gereken fonksiyonelliği ve performansı sağlamalı ve bakımı yapılabilir, güvenilebilir ve kullanılabilir olmalıdır.
Yazılım mühendisliği nedir?	Yazılım mühendisliği yazılım üretiminin başlangıçtaki ilk fikirden işletim ve bakıma kadar olan tüm yönleriyle ilgilenen bir mühendislik disiplinidir.
Temel yazılım mühendisliği etkinlikleri nelerdir?	Yazılım spesifikasyonu, yazılım geliştirme, yazılım doğrulama ve yazılımın evrimi.
Yazılım mühendisliği ve bilgisayar bilimleri arasındaki fark nedir?	Bilgisayar bilimleri teori ve temeller üzerine odaklanır; yazılım mühendisliği ise kullanışlı yazılım üretim ve tesliminin uygulamalarıyla ilgilenir.
Yazılım mühendisliği ve sistem mühendisliği arasındaki fark nedir?	Sistem mühendisliği donanım, yazılım ve süreç mühendisliği dahil olmak üzere bilgisayar tabanlı sistem geliştirmenin tüm yönleriyle ilgilenir. Yazılım mühendisliği bu daha genel sürecin parçasıdır.
Yazılım mühendisliği ile ilgili esas zorluklar nelerdir?	Artan çeşitlilikle başa çıkma, daha kısa teslim süresi talepleri ve güvenilir yazılım geliştirme.
Yazılım mühendisliğinin maliyetleri nelerdir?	Yazılım maliyetlerinin kabaca %60'ı geliştirme maliyetleridir, %40'ı sınama maliyetleridir. Özel ısmarlama yazılım için, evrim maliyetleri genellikle geliştirme maliyetlerini aşar.
En iyi yazılım mühendisliği teknikleri ve yöntemleri nelerdir?	Tüm yazılım projelerinin profesyonel olarak yönetilmesi ve geliştirilmesi gerekir ama farklı sistem türleri için farklı teknikler uygundur. Örneğin oyunlar her zaman bir dizi prototip kullanılarak geliştirilmeliyken, hayati tehlike arz eden kontrol sistemleri tam ve analiz edilebilir bir spesifikasyonun geliştirilmesini gerektirirler. Her şey için iyi olan yöntemler ve teknikler yoktur.
İnternet yazılım mühendisliğinde ne değişiklik yapmıştır?	İnternet sadece çok büyük, çok dağıtık servis-tabanlı sistemlerin geliştirilmesine yol açmamıştır, aynı zamanda yazılımın ekonomisini değiştiren mobil cihazlar için bir uygulama (*app*) endüstrisi yaratılmasını desteklemiştir.

Şekil 1.1 Yazılım mühendisliği hakkında sık sorulan sorular.

Yazılım Mühendisliği

Yazılım mühendisliği sistem spesifikasyonunun ilk aşamalarından sistemin kullanıma verildikten sonraki bakımına kadar yazılım üretiminin tüm yönleriyle ilgilenen bir mühendislik disiplinidir. Bu tanımda iki anahtar ifade vardır:

a) Mühendislik Disiplini
b) Yazılım Üretiminin Tüm Yönleri

Yazılım mühendisliği iki nedenden dolayı önemlidir:

1. Kişiler ve toplum, gittikçe daha fazla gelişmiş yazılım sistemlerine güvenmektedirler. Emniyetli ve güvenilir sistemleri ekonomik ve hızlı olarak üretebilmeliyiz.

2. Genellikle, uzun vadede, profesyonel yazılım sistemleri için yazılım mühendisliği yöntemlerini ve tekniklerini kullanmak, kişisel bir programlama projesiymiş gibi sadece programlar yazmaktan daha ucuzdur. Yazılım mühendisliği yöntemini kullanmamak sınama, kalite güvence ve uzun dönemli bakım için daha yüksek maliyetlere neden olur.

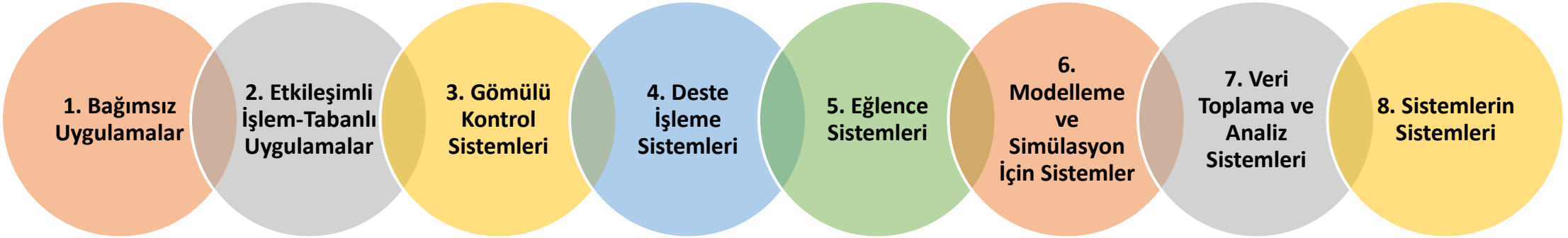
Ürün özelliği	Tanım
Kabul edilebilirlik	Yazılım tasarlandığı kullanıcı tipi tarafından kabul edilebilir olmalıdır. Bunun anlamı, anlaşılabilir, kullanılabilir ve kullandıkları diğer sistemlerle uyumlu olması gerektiğidir.
Güvenilebilirlik ve güvenlik	Yazılım güvenilebilirliği, güvenirlik, güvenlik ve emniyet dahil olmak üzere bir dizi özelliği içerir. Güvenilebilir yazılım bir sistem arızası durumunda fiziksel veya ekonomik bir zarara neden olmamalıdır. Yazılım zararlı kullanıcıların sisteme erişemeyeceği veya zarar veremeyeceği şekilde emniyetli olmalıdır.
Verimlilik	Yazılım bellek ve işlemci döngüleri gibi sistem kaynaklarını gereksiz yere kullanmamalıdır. Bu nedenle verimlilik, çabuk yanıt verme yeteneği, işlem zamanı, kaynak kullanımı vb. içerir.
Bakım kolaylığı	Yazılım müşterilerin değişen gereksinimlerini karşılamak için evrilebileceği şekilde geliştirilmelidir. Bu hayati bir özelliktir çünkü yazılım değişimi, değişen bir iş ortamının kaçınılmaz bir gereksinimidir.

Şekil 1.2 İyi yazılımın gerekli özellikleri.

Yazılım Mühendisliği Çeşitliliği

Yazılım mühendisliği, yazılım üretiminde yazılım müşterilerinin ve üreticilerinin gereksinimleri kadar pratikteki maliyet, zamanlama ve güvenilirlik konularını da dikkate alan sistematik bir yaklaşımdır. Kullanılan yöntemler, araçlar ve teknikler yazılımı geliştiren kuruluşa, yazılımın türüne ve geliştirme sürecine katılan kişilere bağlıdır.

Herhalde hangi yazılım mühendisliği yöntemlerinin ve tekniklerinin en önemli olduğunun belirlenmesinde en önemli faktör geliştirilen uygulamanın türüdür. Çok farklı uygulama türleri vardır:



İnternet Yazılım Mühendisliği

- İnternet'in ve World Wide Web'in geliştirilmesinin hepimizin yaşamlarında derin etkileri olmuştur. Başlangıçta, web evrensel olarak erişilebilen bir bilgi deposuydu ve yazılım sistemleri üzerinde küçük bir etkisi olmuştu.
- Bu sistemler yerel bilgisayarda çalıştı ve sadece bir organizasyondan erişilebilirdi. 2000 yılı civarında web değişmeye başladı ve tarayıcılara gittikçe daha fazla fonksiyonellik eklendi.
- Bunun anlamı, özel amaçlı bir kullanıcı arayüzü yerine, bu sistemlerin bir web tarayıcı aracılığıyla erişilebildiği yerde web tabanlı sistemler geliştirilebilirdi.
- Bu durum web üzerinden erişilen ve yenilikçi servisler sunan çok büyük çeşitlilikte yeni sistemlerin geliştirilmesine yol açtı. Bunlar genellikle kullanıcının ekranında görüntülenen reklamlarla desteklendi ve kullanıcılardan doğrudan ödeme içermedi.

YAZILIM MÜHENDİSLİĞİ ETİĞİ

Diğer mühendislik disiplinleri gibi, yazılım mühendisliği, bu alanda çalışan kişilerin özgürlüğünü sınırlayan sosyal ve yasal bir çerçevede yürütülür. Bir yazılım mühendisi olarak, mesleğinizin sadece teknik becerilerin uygulanmasından daha geniş sorumluluklar içerdiğini kabul etmelisiniz.

Yeteneklerinizi ve becerilerinizi dürüst olmayan bir şekilde veya yazılım mühendisliği mesleğine itibarsızlaştıracak şekilde kullanmamalısınız. Ancak kabul edilebilir davranışların standartlarının yasalara bağlı olmadığı fakat daha zayıf mesleki sorumluluk kavramına bağlı olduğu alanlar vardır. Bunların bazıları:

Yazılım Mühendisliği Etik kurallar ve Profesyonel uygulamalar

ACM/IEEE-CS Yazılım Mühendisliği Etiği ve Profesyonel Uygulamalar Ortak Çalışması

ÖNSÖZ

Kuralların kısa sürümü istekleri yüksek bir soyutlama düzeyinde özetler; tam sürümde eklenen ifadeler örnekler verir ve bu isteklerin profesyonel yazılım mühendisleri olarak davranışımızı nasıl değiştirdiğini ayrıntılandırır. Bu istekler olmadan, ayrıntılar yasaya benzeyebilir ve bıktırıcı olabilir; ayrıntılar olmadan istekler kulağa iyi ama boş gelebilir; istekler ve ayrıntılar birlikte uyumlu kurallar oluştururlar.

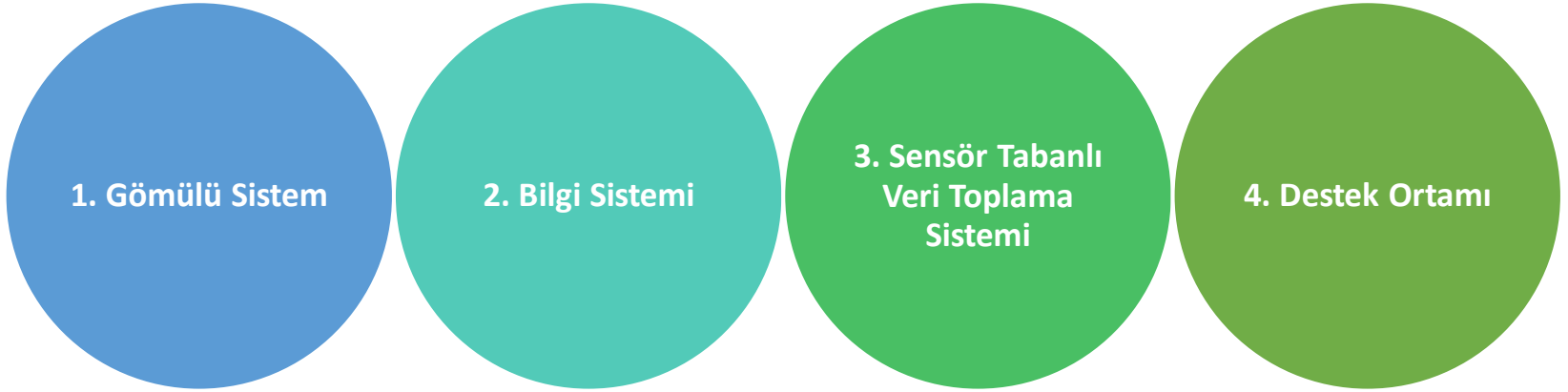
Yazılım mühendisleri yazılımın analiz, spesifikasyon, tasarım, geliştirme, test ve bakımını yararlı ve saygı duyulan bir meslek olarak yapma sorumluluğunu üstlenirler. Bu sorumlulukla uyumlu olarak, toplumun sağlığı, güvenliği ve refahı için yazılım mühendisleri aşağıdaki sekiz ilkeye sadık kalacaklardır:

1. TOPLUM – Yazılım mühendisleri toplumun ilgisiyle tutarlı hareket edeceklerdir.
2. MÜŞTERİ ve İŞVEREN – Yazılım mühendisleri toplumun ilgisiyle tutarlı olarak müşterilerin ve işverenin ilgisine en uygun şekilde davranacaklardır.
3. ÜRÜN – Yazılım mühendisleri ürünlerinin ve ürünlerle ilgili değişikliklerin mümkün olan en yüksek mesleki standartları karşılamasını sağlarlar.
4. DEĞERLENDİRME – Yazılım mühendisleri kendi mesleki değerlendirmelerinde bütünlüğü ve bağımsızlığı sağlayacaklardır.
5. YÖNETİM – Yazılım mühendisliği yöneticileri ve liderleri yazılım geliştirme ve bakımının yönetimine etik bir yaklaşıma uyacak ve onu daha da geliştireceklerdir.
6. MESLEK – Yazılım mühendisleri toplum ilgisiyle tutarlı olarak mesleğin bütünlüğünü ve itibarını geliştireceklerdir.
7. MESLEKTAŞLAR – Yazılım mühendisleri meslektaşlarına karşı adil ve destekleyici olacaklardır.
8. KENDİSİ – Yazılım mühendisleri mesleklerinin uygulamalarını göz önüne alarak yaşam boyu öğrenmeye katılacak ve mesleklerinin uygulanmasında etik bir yaklaşım izleyeceklerdir.

Şekil 1.3 ACM/IEEE Etik Kurallar. (ACM/IEEE-CS Yazılım Mühendisliği Etiği ve Profesyonel Uygulamalar Ortak Çalışması, kısa sürüm. http://www.acm.org/about/se-code) (© 1999 ACM, Inc. ve IEEE, Inc.)

DURUM ÇALIŞMALARI

Yazılım mühendisliği kavramlarını örnekle açıklamak için dört farklı tür sistemden örnekler kullanıyorum. Bu kitaptaki anahtar mesajlardan biri yazılım mühendisliği uygulamasının üretilen sistemin türüne bağlı olması olduğu için kasten tek bir durum çalışması kullanmadım. Bu nedenle emniyet, güvenilirlik, sistem modelleme, yeniden kullanım vb. gibi kavramları tartışırken uygun bir örnek seçiyorum. Durum çalışmaları olarak kullandığım sistem türleri:



Yazılım Kalitesi nedir?

- Yazılım Kalitesi; somut olarak tanımlanamayan Soyut ve yoruma açık bir kavramdır.
- Her yazılım projesi müşteri istekleri doğrultusunda geliştirilir.!?
- Gereksinimlerin karşılanma başarısıdır.
- Yazılım kalitesi, yazılımın çözmeye çalıştığı problemlerin başarı derecesidir
- **Yazılım mühendisliği çerçevesinde tanımlanan yöntem ve araçların amacı, üretilecek yazılımda yüksek kalite düzeylerine erişmektir. Kalite ise bir çok boyutu olan bir özelliktir.**

Yazılım Kalitesi nedir?

- **Kalite çok boyutludur.**
  - **Hata az**
    - Yazılımın uygulamada bulunduğu fazın gerektirdiği çalışma şeklinden sapma durumu yazılım hatasıdır..
  - **Kullanıcı / müşteri ? Gereksinimlerini karşılayan**
  - **Bakım süreçleri iyi**

Yazılım Kalitesi nedir?

KALİTE GEREKSİNİMLERİ (Kurnaz,2003)

- 1. Doğruluk: Yazılımın belirtilmiş gereksinimleri karşılaması
- 2. Güvenirlilik: Üretilen yazılımın gereksinim duyulan işlevleri ne hassasliyetle yerine getirebileceği beklentisi
- 3. Verimlilik: İşlevlerin gerçekleştirilmesi için kullanılması gereken bilgisayar kaynakları, zaman ve kod miktarı
- 4. Bütünlük: Yazılım ve bilgilerine istenmeyen insanlarca ulaşımın ne derece engellenebildiği, yazılımın değişmeden etkilenmeden, amacı için kullanılabilmesi
- 5. Kullanılabilirlik: Programın öğrenilmesi, çalıştırılması, girdi ve çıktı işlemleri için gerekli çaba miktarı, hatasız verimli kullanılabilirliği
- **6. Bakım Kolaylığı:** Bir programda hata bulma ve onarma için gereken çaba
- 7. **Esneklik:** Çalışan bir programda değişiklik yapma kolaylığı

Yazılım Kalitesi nedir?

KALİTE GEREKSİNİMLERİ (Kurnaz,2003)

- **Denetlenebilirlik:** Standartlara uygunluğun kontrol edilmesi kolaylığı
- 13. **Doğruluk:**Sistemin fiziksel yapısının oluşturulan mantıksal modele ne derece uygun olduğu
- 14. **Tamlık:** Gereken işlevlerin tam olarak tanımlanma derecesi
- 16. **Kesinlik:** Programın tasarımda satır sayısı olarak küçüklüğü
- 17. **Tutarlılık:** Proje süresince tasarım ve belgeleme tekniklerinde uygulanan biçem benzerliği

YAZILIM MÜHENDİSLİĞİ

YAZILIM

- Yazılım bir üründür, ancak başka ürünler geliştirmeye veya elde etmeye yarayan bir araç da olabilir.
- Yazılım fiziksel bir ürün olmadığı için aşınmaz, ancak zamanla yetersizleşebilir.
  - Değişim kaçınılmazdır: Yazılım, yaşam döngüsü süresince değişikliklere uğrar.
  - Değişiklikler, yazılımda yeni hatalar oluşturabilir.
- Yeni hatalar tam olarak düzeltilmeden yeni değişiklikler gerekebilir.

Kaliteli yazılım; Yazılım mühendisliği ilkelerine uyularak yazılım yaşam döngüsü içiersinde daha iyi tasarlanmış yazılım

Yaşam döngüsü : Yazılımın bir fikir olarak doğmasından, kullanım dışı bırakılmasına kadar geçen süreç.

YAZILIM MÜHENDİSLİĞİ

YAZILIM MÜHENDİSLİĞİ

- Yazılım mühendisliği, bilgisayar bilimi, yönetim bilimi, ekonomi ve bilişim bilim dallarından yararlanmakta ve sorun çözümünde mühendislik yöntemlerini kullanmaktadır. Yazılım mühendisliğinin, temelinde kalite bakış açısı olan, mühendislik yöntemleri, süreçler, metodolojiler ve yazılım araçları (CASE tool) olan katmanlı bir yapısı vardır

YAZILIM MÜHENDİSLİĞİ

YAZILIM MÜHENDİSLİĞİ

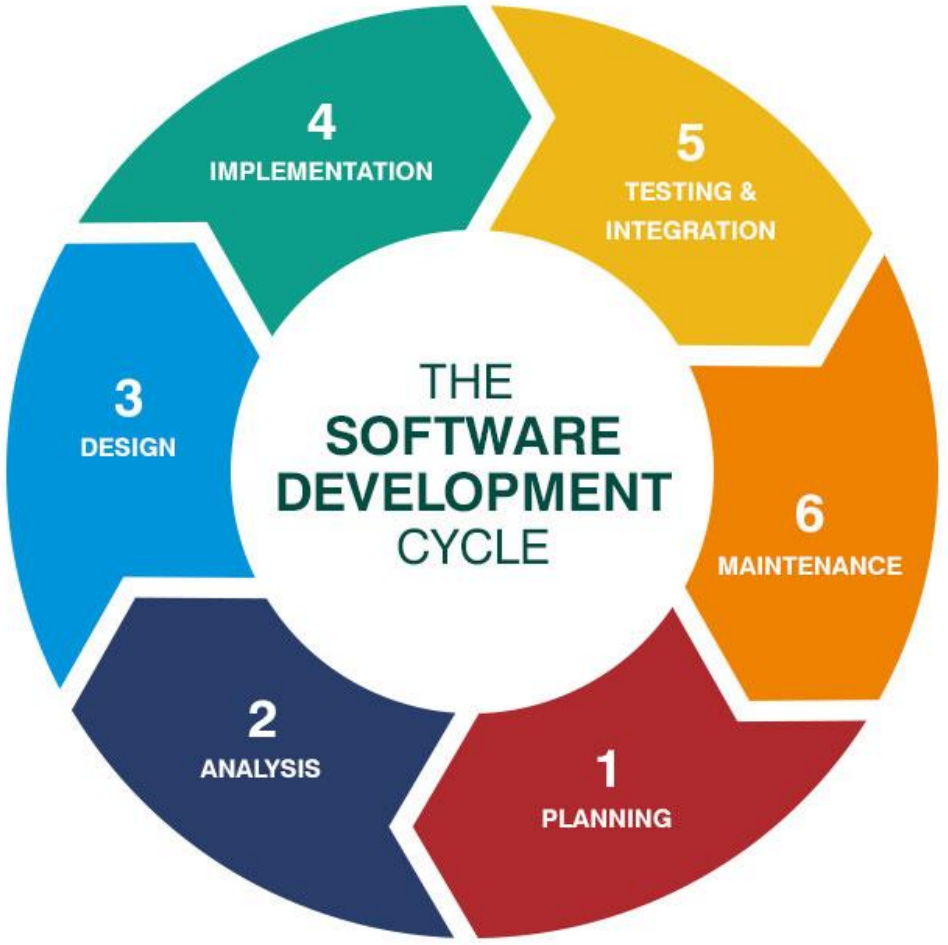
Yazılım Mühendisliği destek süreç adımları ise,

- Proje Yönetimi
- Kalite Yönetimi
- Geçerleme Doğrulama
- Formal ve teknik gözden geçirme
- Yazılım Konfigürasyon Yönetimi
- Yeniden kullanılabilirlik Yönetimi
- Yazılım Ölçümü
- Risk Yönetimi

Yazılım Geliştirme Yaşam Döngüsü (SDLC)

Yazılım Geliştirme Yaşam Döngüsü (SDLC) kavramı bir döngüsel (cyclic) yaklaşımdır (SEKER, 2014). Bir sarmal hareket halinde olan bir yaklaşımdır. SDLC aşamaları aşağıdaki şekilde sıralanabilir.

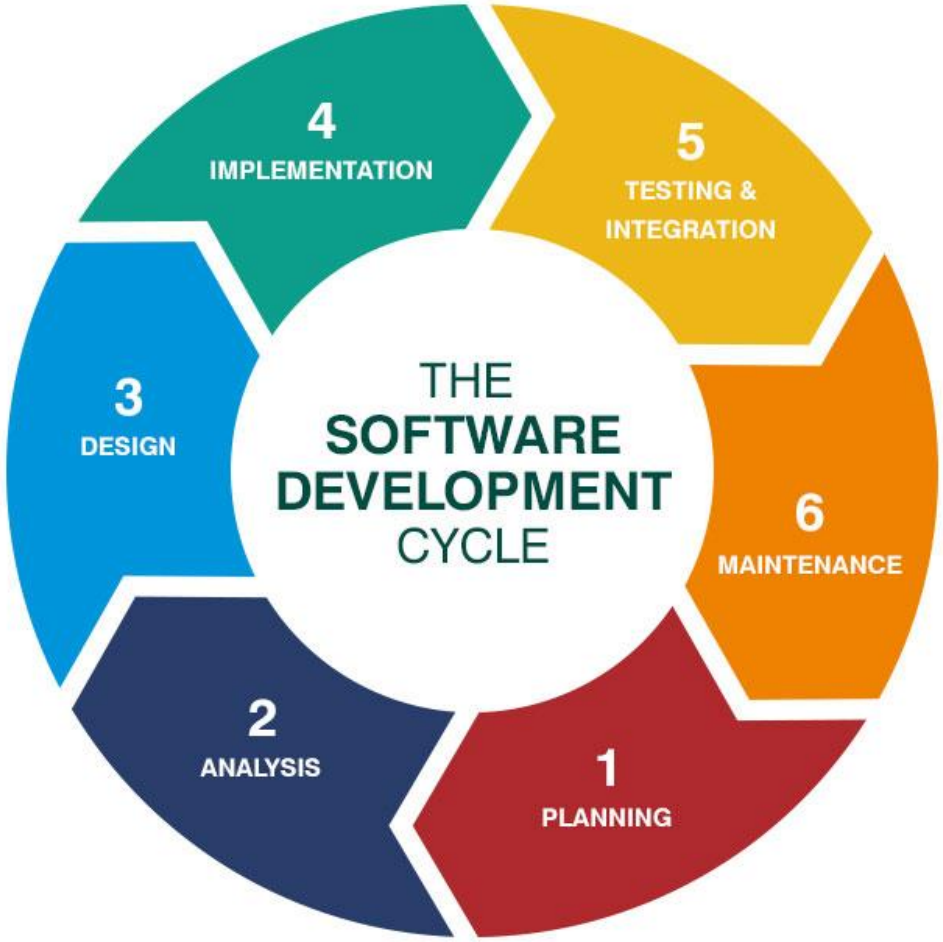
1. Planlama
2. Analiz -Tanımlama
3. Tasarım
4. Geliştirme Uygulama
5. Entegrasyon ve testler
6. Bakım



Yazılım Geliştirme Yaşam Döngüsü (SDLC)

1. Planlama:

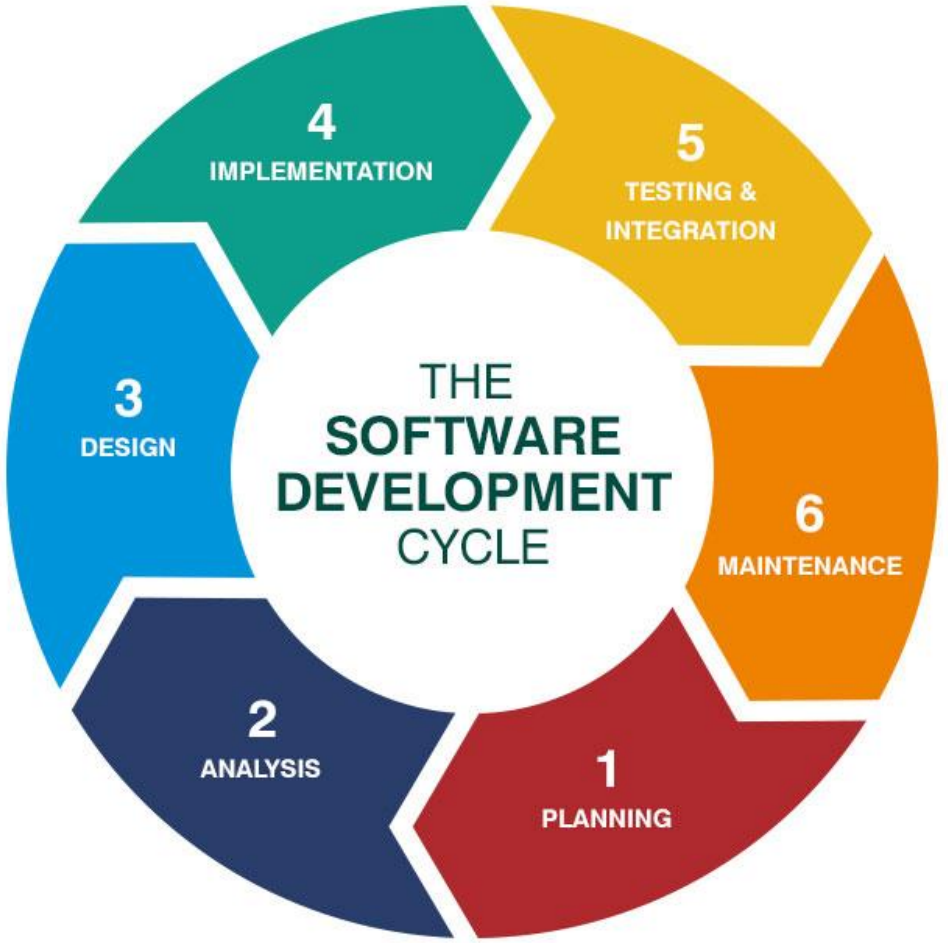
- Plan aşaması aslında proje yönetimlerinde en önemli aşamasıdır. Diğer adımların da başlangıç aşamasıdır. İşin projelendirildiği, fikrin ortaya çıkarıldığı ve fikrin tartışıldığı aşamadır.
- Personel ve donanım gereksinimlerinin çıkarıldığı, fizibilite çalışmasının yapıldığı ve proje planının oluşturulduğu aşamadır.



Yazılım Geliştirme Yaşam Döngüsü (SDLC)

2. Analiz –Tanımlama

- Sistem gereksinimlerinin ve işlevlerinin ayrıntılı olarak çıkarıldığı aşama. Var olan işler incelenir, temel sorunlar ortaya çıkarılır
- Genelde istenilen fikrin ne olduğu ve temel tanımlar üzerinde konuşulacak kavramların tanımlandığı aşamadır.
- Çünkü aynı fikirden bahsedilmiyorsa farklı sonuçlara varılabilir. Bu tanımlar üzerine bir tasarım vardır. Bu analiz aşaması olarak da düşünülebilir.
- Problemin tanımlandığı veya yaşam döngüsünün tanımlandığı sistemin veyahut yazılımın tanımlandığı aşamadır.
- Tanım aşamasında projede nelerin istenildiği ile ilgili analiz çalışmaları da yapılabilir.



Yazılım Geliştirme Yaşam Döngüsü (SDLC)

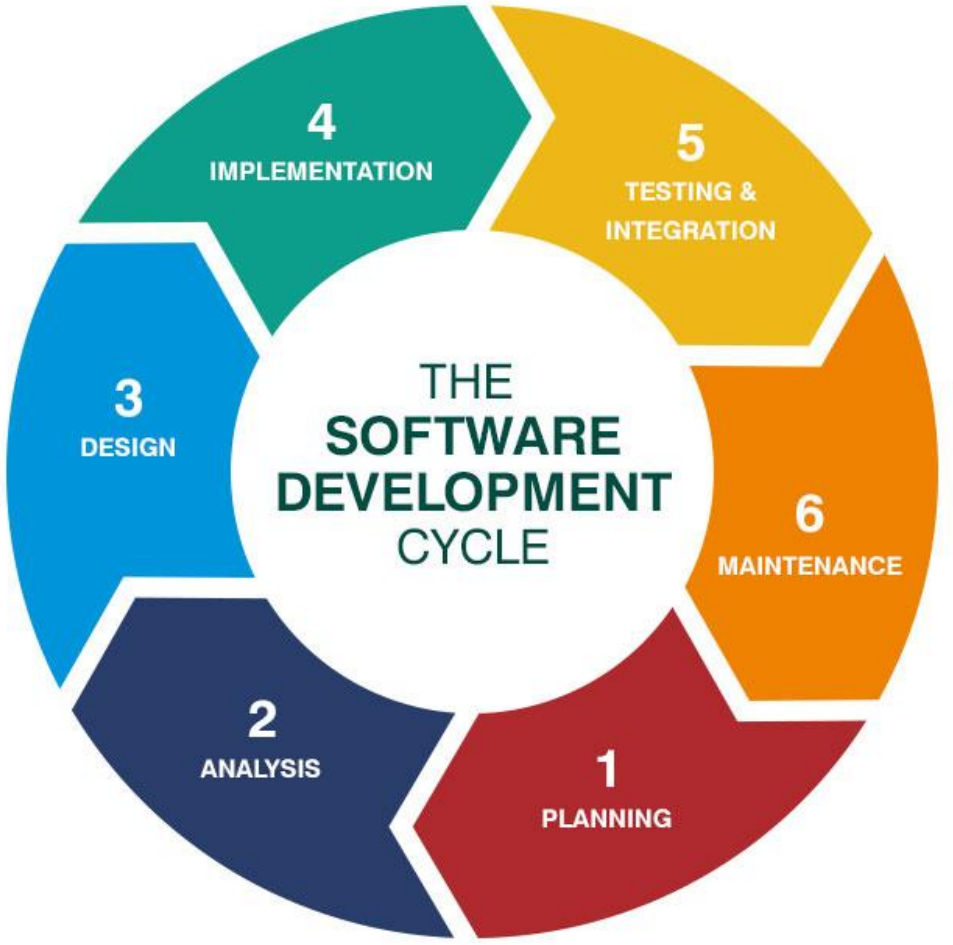
3. Tasarım

- Belirlenen gereksinimlere yanıt verecek yazılım sisteminin temel yapısının oluşturulduğu aşamadır.

  mantıksal; önerilen sistemin yapısı anlatılır,

  fiziksel; yazılımı içeren bileşenler ve bunların ayrıntıları.

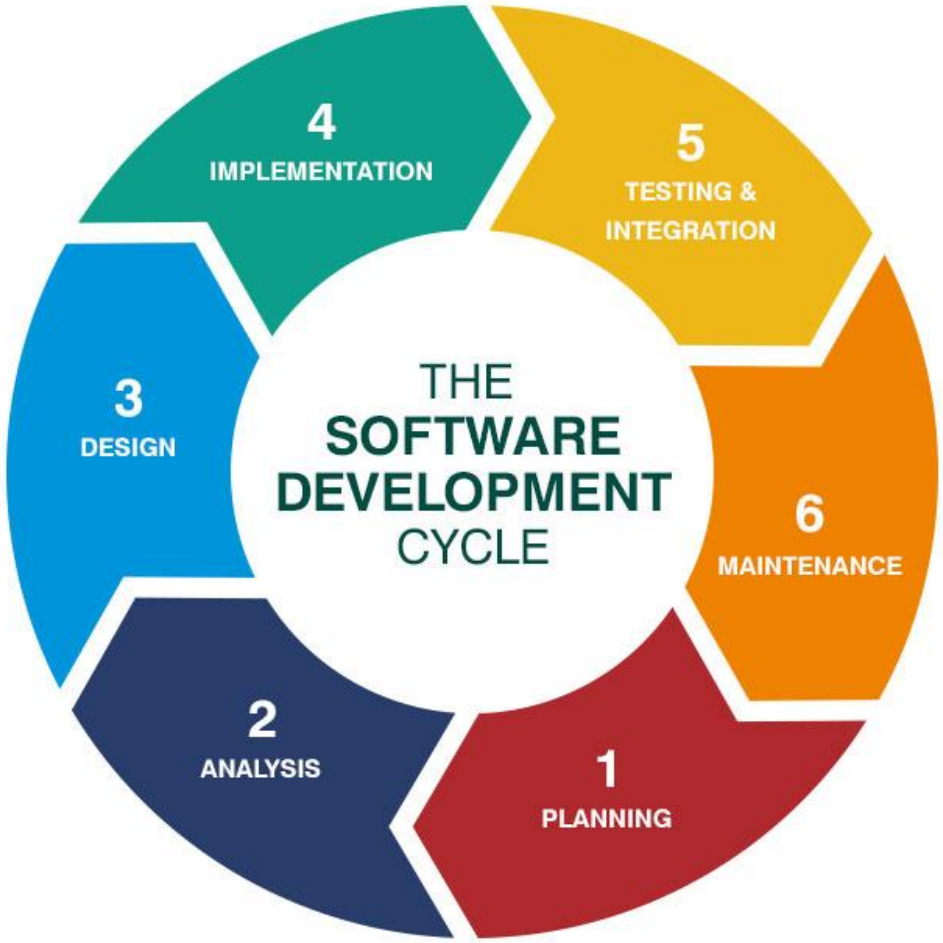
- Yazılımımızın veya sistemimizin tasarımları yapılır. Projeleri çizilir. İnşaat projeleri gibi düşünülebilir.
- Planlama ve tanımlaya göre bir tasarım çizilir. Kararlar verilir, seçimler yapılır, örneğin yazılımın ekranları,
- ekranlarda neler bulunacağı, hangi ekranlara nasıl geçileceği, fonksiyonel olarak hangi adımların oluşturulacağı,
- hatta yazılımın bileşenleri, modülleri bu aşamada tasarlanır. Bir sonraki adım olan uygulma/geliştirmeye bütün
- kararlar verilmiş olarak geçilmesi beklenir. Geliştirme aşamasında herhangi bir soru veya karar bırakılmaz.



Yazılım Geliştirme Yaşam Döngüsü (SDLC)

4. Geliştirme Uygulama

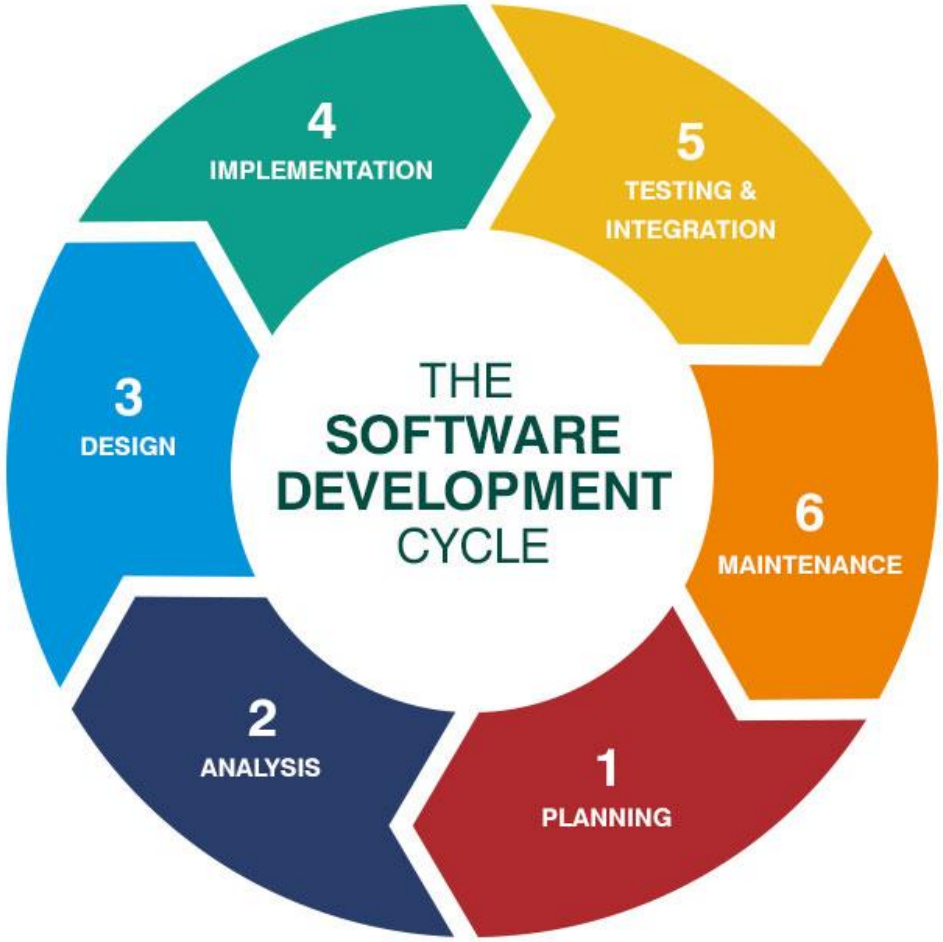
- Bu aşama, yazılım projeleri için kodlama olarak düşünülebilir veya sistemin yaşatılmaya başlandığı ilk örneklerinin çıkmaya başlandığı aşama olarak düşünülebilir.
- Daha önceden tasarım aşamasında karar verilen ortama uygun olarak, yine tasarım aşamasında verilen kararlar doğrultusunda projenin gerçekleştirilmesine başlanır.
- Yazılım Projelerinin kodlandığı aşamadır.



Yazılım Geliştirme Yaşam Döngüsü (SDLC)

5. Entegrasyon ve testler

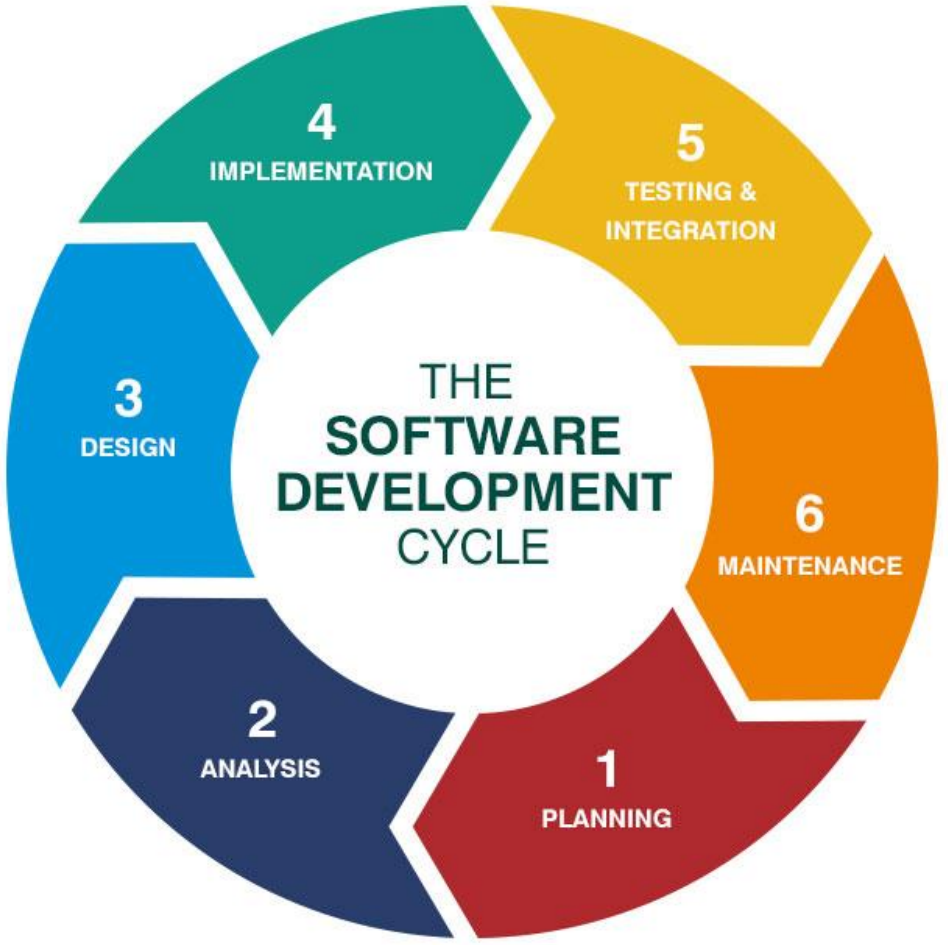
- Sistemin artık gerçek hayata geçtiği aşamadır. Sistem veya yazılım hayat içinde bir yer edinir.
- Bir şirkette kullanılacak yazılımdan bahsediliyorsa şirketle ilgili birimlerin yazılımla ilgili eğitimlerin alınması, donanımlarının temin edilmesi, bağlantılı olduğu birimlerin buna entegre edilmesi, başka yazılımlarla bir entegrasyonu varsa bunun sağlanması ve veri tabanı bağlantıları gibi pek çok adım bu aşamada ele alınır.



Yazılım Geliştirme Yaşam Döngüsü (SDLC)

5. Entegrasyon ve testler

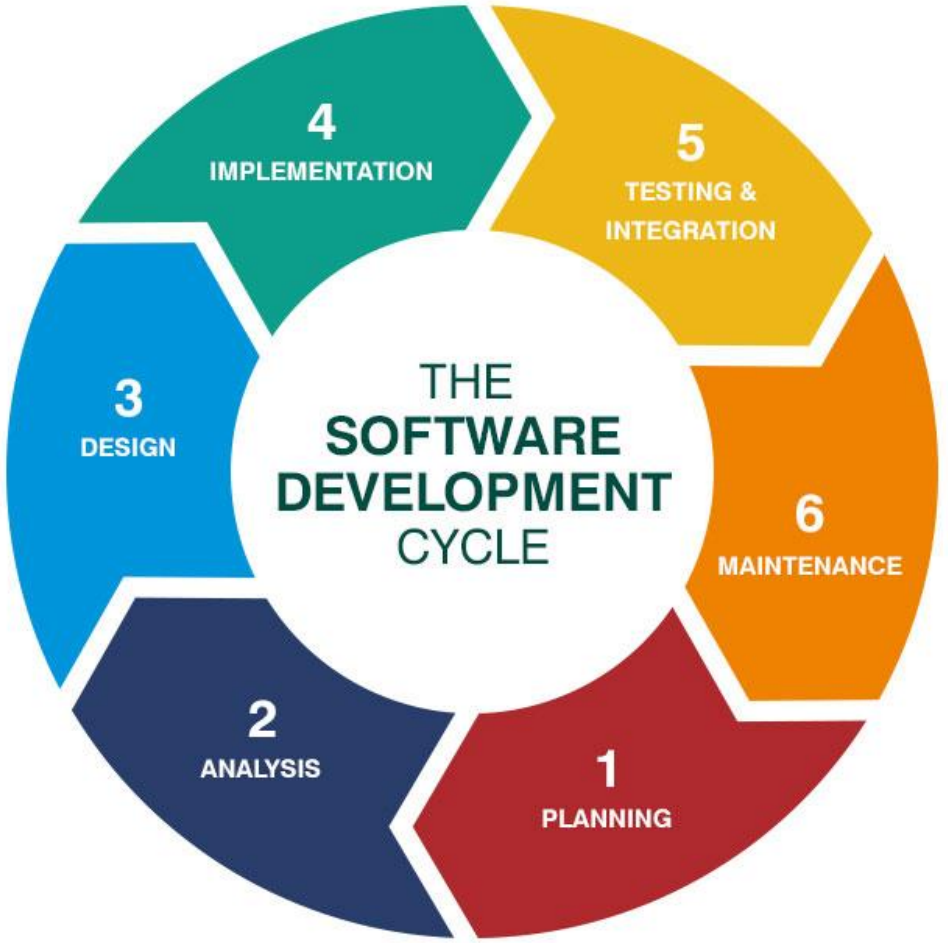
- Test aşaması yazılım geliştirme çevriminde her aşamada yapılmalıdır.!
- Yazılım sürecinde ilerledikçe, ortaya çıkabilecek hataların giderilme maliyeti üstel olarak artar.
- Aksi gibi, hataların büyük çoğunluğunun kökenleri isteklerin belirlenmesi ve tasarım aşamalarındaki sorunlara dayanır.
- Kalite ve maliyet açısından erkenden, ve sık sık test yapılmalıdır



Yazılım Geliştirme Yaşam Döngüsü (SDLC)

6. Bakım

- Bu aşama yazılım tesliminden sonra gerçekleştirilen aşamadır.
- Hata giderme ve yeni eklentiler yapma aşamasıdır.
- Yazılımın faaliyete geçirilmesinden sonra sistemde yapılan değişikliklerdir.
  - Yazılım hatalarının düzeltilmesi:
  - Kodlama hataları
  - Tasarım hataları (!)
  - Gereksinim ve analiz hataları (!!)
- Sistemin işlevlerini değiştirme veya işlevlere eklemeler/çıkarmalar,
- Yazılımın farklı bir ortama taşınması (programlama dili, işletim sistemi, donanım, iklim, vb.)



Yazılım Geliştirme Yaşam Döngüsü (SDLC)

- Yazılım yaşayan bir organizmadır.
- Belli bir yaşam süresi vardır. Belli bir süre sonra tükenir.
- Yazılım bir tüketim ürünüdür.
- Yazılım, bir mühendislik projesidir.
- Yazılım Geliştirme Yaşam Döngüsü (SDLC) bu iki yaklaşımın (tüketim ürünü ve mühendislik ürünü yaklaşımlarının) ortasındadır. (Seker, 2015).
- Uzun yıllar boyunca yazılımın yaşaması istenilir. Ama bu yazılımının hayatını devam ettirebilmesi için belli aralıklarla ufak dokunuşlar yapılması ve düzeltmeler yapılması da gerekir. Yazılım Geliştirme Yaşam Döngüsü işte tam da bu noktadadır. SDLC yapılacak düzenlemelerin nasıl yapılacağını organize eder. Proje yönetimi açısından bakıldığında yazılımcı belli aşamalardan geçtikçe biriktirdiği birikimleri bir sonraki aşamada kullanmaya başlar. Bir problem ortaya çıktığında SDLC yol göstericidir.

SDLC Yaklaşımına Farklı Bakış Açıları

SDLC aşamaları dörde indirgenebilir (Seker 2015).

Bunlar:

- **Planlama:** Bu aşama, planlamanın bütün gereklerini tamamlar. Zaman kaynak ayrımı, personel, kurumun mevcut bilişim seviyesi ve yeni sistemin kullanılabilirliği, uzun vadede sistemin bakımı ve sürdürülebilirliği gibi çok sayıda parametre bu aşamada incelenir. Bu aşamanın tamamlanmasının ardından bir uygunluk raporu (feasibility report) hazırlanarak kurumdaki irade sahiplerine (yönetim) sunulur ve bilişim sisteminin yapacağı olumlu katkıların maliyet analizine göre katma değeri olduğu kesinleştirilir.
- **Sistem Tahlili (Analiz) :** Bu aşamada sistem analizi yapılır ve geliştirilmekte olan bilgi sisteminin etkilediği diğer birimler ve mevcut işleyiş tahlil edilir. Örneğin geliştirilmekte olan sistem ödeme emirleri ile ilgili olsun, bu durumda girilen emirlerin satış, defteri kebir, stok kayıtları gibi çok sayıdaki etkilediği diğer kayıt tahlil edilerek bu sistemlerle olan etkileşimi çalışılır.

SDLC Yaklaşımına Farklı Bakış Açıları

- **Sistem Tasarımı (Design):** Bu aşamada sistemin bütün bileşenleri ve genel yapısı tasarlanır. Çok çeşitli tasarım yöntemleri ve aşamaları olmakla beraber genelde mantıksal bir tasarım aşamasından geçilerek sistemin hedefleri ve etkileşimde olduğu alt bileşenleri arasındaki uyum gözetilir. Ardından bu mantıksal tasarımın fiziksel tasarıma çevrilmesi aşaması gelir. Fiziksel tasarıma çevrilim aşamasında, programlama dilinden kullanılacak olan teknolojilere ve sistemin bütün parçalarına kadar olan tasarımı tamamlanır. (veri tabanı tasarımı, kullanıcı ekranları, veri akış yolları gibi).
- **Uyarlama ve işletme aşaması (implementation and operation):** Bu aşama, şimdiye kadar yapılan bütün kağıt üzerindeki işlemlerin gerçeğe dönüştürüldüğü ve yazılım halini aldığı aşamadır. Kodlama, kod kontrolleri, testleri, kurulumu, yönetimi gibi işlemlerin tamamı bu aşamada yapılır.

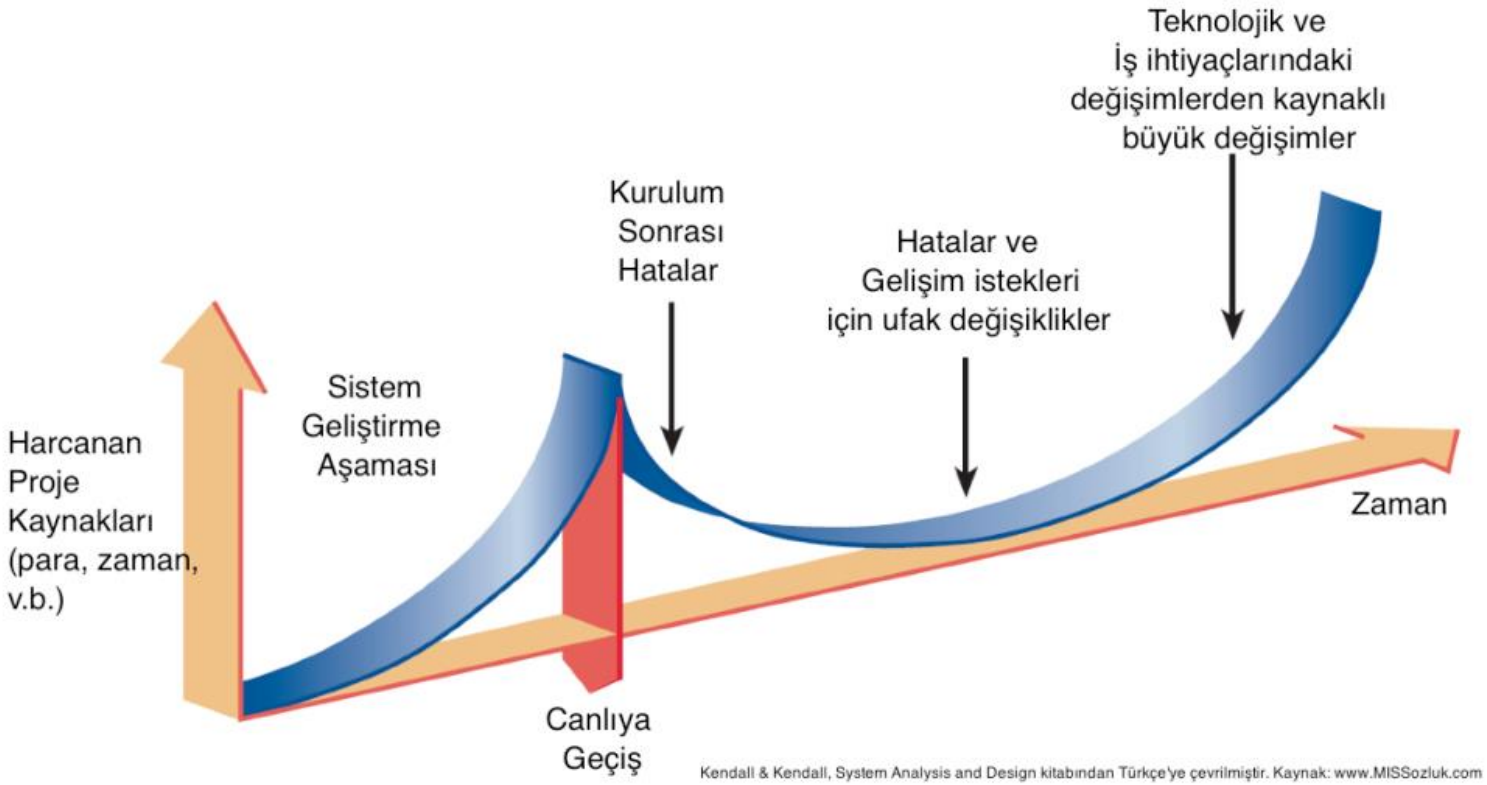
SDLC Yaklaşımına Farklı Bakış Açıları

- Kendall and Kendall tarafından sunulan 7 adımlı adımları

1. Problem, fırsatlar ve hedeflerin belirlenmesi
2. İnsan seviyesi enformasyon ihtiyaçlarının belirlenmesi
3. Sistem ihtiyaçlarının (isterlerinin) belirlenmesi
4. Şimdiye kadar olan adımlar ışında elde edilen isteklere cevap veren bir sistemin tasarlanmsası
5. Yazılımın geliştirilmesi ve dokümantasyonu.
6. Sistemin testleri ve bakım adımlarının belirlenmesi.
7. Sistemin gerçeğe alınması ve değerlendirilmesi

SDLC Yaklaşımına Farklı Bakış Açıları

- Kendall, SDLC Proje maliyet Grafiği



Kendall & Kendall, System Analysis and Design kitabından Türkçe'ye çevrilmiştir. Kaynak: www.MISSozluk.com

KAYNAKLAR

•[1] Tian , J. (2005), Software Quality Engineering Testing, Quality Assurance, and Quantifiable Improvement, IEEE Computer Society/ John Wiley & Sons, Inc.
•[2] S.E. Seker, (2015), Yazılım Geliştirme Modelleri ve Sistem Yazılım Yaşam Döngüsü, YBS Ansiklopedi, 2,3,18.
•[3] S. Kurnaz, (2003), Yazılım Mühendisliğinde Kalite Ve Uml, Havacılık Ve Uzay Teknolojileri Dergisi, 1,2,1.