📘 Bölüm 6: Mimari Tasarım

Mimari tasarım bir yazılım sisteminin nasıl yapılandırılması gerektiğini anlamak ve sistemin genel yapısını tasarlamaktır. yazılım geliştirme sürecinde yazılım tasarımı aşamasının ilk basamağı mimari tasarımdır. Bu basamak tasarım ile gereksinim mühendisliği arasındaki kritik bağlantıdır, çünkü sistemin ana yapısal bileşenlerini ve bu bileşenler arasındaki ilişkileri belirlemektedir. Mimarj tasarım sürecinin çıktısı bir sistemin haberleşen bileşenler olarak nasıl yapılandığını anlatan bir mimari modeldir. Çevik süreçlerde genel eğilim, geliştirme sürecinin erken evrelerinde sistem mimarisinin genel yapısına odaklanılması yönündedir. Mimarilerin artırımlı gelişti. rimi çoğunlukla başarılı olmaz. sistem mimarisini yeniden yapılandırmak pahalıdır; zira mimari değişikliklere uyum sağlamak için pek çok siş- tem bileşenini yeniden düzenlemek gerekebilmektedir. İdeal olarak bir sistem spesifikasyonu herhangi bir tasarım bilgisi içermemelidir.

Yazılım mimarilerini iki soyutlama düzeyi

**1 Küçük çaplı mimari** tek tek programların mimarisi ile ilgilidir. Bu düzeyde belli bir programın bileşenlerine nasıl ayrıştırılabileceği

**2 Büyük çaplı mimari** başka sistemler, programlar ve program bileşenleri içerebilen karmaşık kurumsal sistemlerin bileşenlerine nasıl ayrılabileceği

**Yazılım mimarisi önemlidir,** çünkü bir sistemin **performans, sağlamlık, dağıtılabilirlik ve bakım kolaylığını etkilemektedir** fonksiyonel olmayan sistem karakteristiklerinin üzerindeki asıl belirleyici etki sistemin mimarisidir.

yazılım mimarisini doğru- dan tasarlamak ve belgelemenin **üç avantajı**

**Paydaş iletişimi, Sistem analizi, Geniş çaplı yeniden kullanım,**

Sistem mimarileri genellikle **biçimsel olmayan blok diyagramları ile modellenir**. Diyagram daki her. kutu bir bileşeni temsil eder. Blok diyagramları sistem yapısının, sistem geliştirimi sürecinde görev alan inşaların kolayca anlayabilecekleri resmi sunar

Mimari betimleme pahalı ve zaman alıcı bir süreçtir

Kalıplaşmış bir mimari tasaraım süreci yoktur süreç gereksinimlere bağlı olarak şekillenir.

Mimari desenler farklı yazılım sistemlerinde kullanışmış mimarilerin özünü yakalar mimari stil ve yapının seçimi sistemin fonksiyonel olmayan gereksinimlerine dayanmaktadır : performans güvenlik emniyet erişebilirlik bakım kolaylığı

Krutchen 4 temel görünüm :

1 mantıksal görünüm soyutlamaları **nesneler veya nesne** sınıfları olarak gösterir

2 süreç görünümü çalışma zamanında sistemin **etkileşen süreçler** halinde nasıl yapılandığı performans ve erişilebilirlik hakkında yargıda bulunmaya yardımcı olur

3 geliştirme görünümü yazılımın geliştirme amacıyla nasıl ayrıştırıldığını gösterir bileşenlere bölünüp geliştirmeyi gösterir **kod yapısı**

4 fiziksel görünüm sistem **donanımını ve yazılım bileşenlerini sistemin kurlumunu** anlatır **donanım dağılımı**

Mimari desenler

Katmanlı mimari

Ayrıştırma ve bağımsızlaştırma kavramları mimari tasarım açısından önemlidir, bu kavramlar değişikliklerin yerel kalmasına olanak sağlar her katman sadece bir altın- daki katman tarafından sağlanan imkân ve servislere yaslanmak durumundadır. katmanlı yaklaşım, sistemlerin artırımlı geliştirilmesini destekler.

Ambar mimari

Katmanlı mimari ve mgk-model görünüm kumanda- desenleri sistemin kavramsal organizasyonunu sunar ambar deseni bir grup bileşen arasında nasıl veri paylaştığını anlatır. bu model, verinin bir bileşen tarafın. dan yaratılıp bir diğer bileşen tarafından kullanıldığı uygulamalara çok uygun düşer, ambarı çok sayıda makine üzerinde dağıtmak zor olabilir. tutarlı ve güncel tutmak sisteme ek yük getirecektir. Statik yapısıyla ilgilidir çalışma zamanı yapılanmasını göstermez

İstemci sunucu mimarisi

Dağıtık sistemler için kullanılan çalışma zamanı yapılanmasını gösterir. Sistemin ana bileşenleri : bileşenlere servis sağlayan sunucular kümesi-sunucular tarafından sunulan servislere başvuran istemciler kümesi- istemcilerin bu servislere erişmlerine izin veren ağ internet protokolleri

İstemci sunucu mimarisi ayrıklık ve bağımsızlık en önemli özelliğidir diğer bölümler birbirini etkilemeden değiştirilebilir Istemci-sunucu modelinin en önemli avantajı dağıtık bir mimari olmasıdır.

Boru ve süzgeç mimarisi

Girdi verisi boru üstünde hareket ederek değiştirilen mimaridir süzgeçler boru içerisinde hareket eden veriyi değiştirmeye yarar. Dönüşümler sıralı olduğunda ve de veri yığınlar halinde işlendiğinde, bu boru ve süzgeç mimari modeli, yığın sıralı model adını alır

Uygulama mimarileri

ortaklıklar belirli tipteki yazılım sisteminin yapısını ve organizasyonunu betimleyen yazılım mimarilerinin geliştirilmesine öncülük etmiştir.uygulama mimarisi modellerini çeşitli yollarla kullanılabilir mimari tasarımın başlangıcı olarak, tasarım kontrol listesi olarak, işlemleri düzenleme yolu olarak, yeniden kullanbilirlik sağlamak amacıyla,jargon oluşturma amacıyla

1.Hareket işleme uygulamaları : Kullanıcı isteklerini işleyen ve bir veri tabanındaki bilgileri güncelleyen veri tabanı merkezli uygulamalardır. Veri bütünlüğü korunur. Hareket işleme sistemleri genellikle kullanıcıların asenkron servis talepleri yaptığı etkileşimli sistemlerdir.

2. Dil işleme sistemleri: derleyicilerdir üst düzeyli yazılan programları makine koduna çevirirler.

Bilgi sistemleri Paylaşılan bir veri tabanı ile etkileşimi gerektiren bütün sistemler hareket tabanlı bilgi sistemleri olarak görülebilirler.

Bir yazılım mimarisi sistemin nasıl yapılanması gerektiğinin anlatımıdır kararlar fonksiyonelk olmayan gereksinimlerden performans güvenlik ve erişilebilirlik gibi özelliklerden etkilenir.

Mimari desenler jenerik sistem mimarileri alanındaki bilgileri kullanmanın bir aracıdır mimariyi betimler avantaj dezavantajlarını işaret eder

. Mimari Tasarımın Çıktısı

Süreç sonunda elde edilen "mimari model", yazılım bileşenlerinin yapısını ve bu bileşenlerin nasıl etkileştiğini gösterir. Bu yapı planlı ya da çevik tüm metodolojilerde kritik bir adımdır.

**Mimari Tasarımın Avantajları**

- Paydaş iletişimini güçlendirir.

- Fonksiyonel olmayan gereksinimlerin analizi yapılabilir.

- Başka projelerde kullanılabilir bir yapı sunar.

Bölüm 7: Tasarım ve Gerçekleştirme

Bazı basit sistemler için yazılım mühendisliği yazılım tasarımı ve gerçekleştirimi anlamına gelmektedir. Büyük sistemler için yazılım tasarımı ve geliştirimi mühendislik süreçlerinden bir tanesidir. Yazılım tasarımı ve gerçekleştirim etkinlikleri her zaman birbirini takip etmektedir.

Yazılım tasarımı müşteri isterlerine bağlı olarak şekil alan yaratıcı süreçtir. Gerçekleştirim tasarımın bir program olarak çıktısını elde etme sürecidir.

Çevik metotlar çoğunlukla tasarımın biçimsel olmayan çizimlerinden çalışır ve tasarım kararlarını programcıya bırakır.

Planlama ve Karar Süreci

Hazır sistem mi geliştirilmeli mi? Bu karar maliyet, zaman ve esneklik açısından önemlidir.

UML kullanarak nesne yönelimli tasarım

Nesne yönelimli tasarım süreçleri nesnelerin sınıflarının ve bu sınıflar arasındaki ilişki tasarımlarını göstermektedir. Nesneler hem veri hem de bu veri işlemek için operasyonlar içerir. Bu nedenle bağımsız varlıklar olarak anlaşılabilir ve değiştirelebilirler. Nesneler varlıklarla bağlantılıdır çoğu zaman gerçek dünya nesneleri donanım bileşenleri gibi ve sistemde onları kontrol eden nesneler arasında net biir eşleme vardır. Bu durum tasarımın anlaşılabilirliğini ve dolayısıyla bakım kolaylığını arttırmaktadır.

**Bir sistem tasarımını kavramdan detaylı nesne yönelimli tasarım yapmak için :**

**1** bağlamı ve sistem ile olan dış etkileşimleri anlamalı tanımalı

**2** sistem mimarisi tasarlanmalı

**3** sistemdeki başlıca nesneler belirlenmeli

**4** tasarım modelleri geliştirmeli

**5** arayüzler belirlenmeli

Sistem bağlamı ve etkileşimler

**Herhangi bir yazılım tasarım sürecindeki ilk aşama, tasarlanmakta olan yazılım ile onun dış ortamı arasındaki ilişkilere ait bir anlayış geliştirmektir.** Bu, gereksinim duyulan sistem fonksiyonelliğinin nasıl sağlanacağına ve sistemin ortamı ile iletişimde bulunması için nasıl yapılandırılacağına karar vermek için gereklidir.

bağlamı anlamak aynı zamanda sistemin sınırlarını belirleyebilmenize izin verecektir.

1. **Bir sistemin Bağlam Modeli,** Geliştirmekte olan sistemin ortamında yer alan diğer sistemleri gösteren yapısal modeldir.

**Bir etkileşim modeli**, sistemin kullanımı sırasında ortamı ile nasıl etkileşimde

Sistem bağlam modelleri ve etkileşim modelleri, bir sistem ile onun ortamı arasın- daki ilişkilere ait birbirini tümleyen görünümler sunmaktadır:

* Bir sistem bağlam modeli, geliştirilmekte olan sistemin ortamında yer alan diğer sistemleri gösteren yapısal bir modeldir.
* Bir etkileşim modeli, sistemin kullanımı sırasında ortamı ile nasıl etkileşimde bulunduğunu gösteren dinamik bir modeldir.

Tasarım modelleri

tasarım veya sistem modelleri bir sistemdeki nesneleri ve nesne sınıflarını gösterir. Bu modeller sistem gereksinimleri ile sistemin gerçekleştirimi arasındaki köprüdür. Gereksiz detayların kendilerini ve sistem gereksinimleri arasındaki ilişkileri gizlememesi için soyut olmalıdır.

UML kullandığınızda, iki çeşit tasarım modeli geliştirmeniz gerekir:

* 1 Sistemin statik yapısını nesne sınıfları ve onların ilişkilerini kullanarak tanımlayan yapısal modeller.
* 2 Sistemin dinamik yapısını tanımlayan ve sistem nesneleri arasındaki beklenen çalışma zamanı etkileşimlerini gösteren dinamik modeller.

mimari modellerine ayrıntı eklemek için üç UML model türü özellikle kullanışlıdır: 1. 2. 3. Nesnelerin uyumlu alt sistemler şeklinde mantıksal gruplamalarını **gösteren alt sistem modelleri**. Alt sistem modelleri **statik** modellerdir. Nesne etkileşimlerinin sırasını gösteren **sıra modelleri.** Bu modeller, bir UML sıra veya iş birliği diyagramı ile temsil edilirler. Sıra modelleri **dinamik** modellerdir. Olaylara karşılık olarak nesnelerin davranışını nasıl değiştirdiğini gösteren **durum makinesi modelleri**. Bu modeller, UML'de durum diyagramları kullanılarak temsil edilirler. Durum makinesi modelleri **dinamik** modellerdir.

Arayüz spesifikasyonları

Nesneler ve alt sistemlerin paralel olarak tasarlanabilmesi için arayüzleri belirlemeniz gerekmektedir.

Tasarım desenleri

desen, çözümün farklı ortamlarda yeniden kullanılabilmesi için yapılan bir problem tanımı ve çözümünün özüdür.

desenlerinin dört ana elemanın: desenlerinin dört ana elemanın

2 Problem alanının, desenin ne zaman uygulanabileceğini açıklayan bir tanımı.

3 Problem alanının, desenin ne zaman uygulanabileceğini açıklayan bir tanımı.

4 Deseni uygulamanın sonuçlarının ve ödünleşimlerin (desenin kullanılmasının ir İİ er e avantaj ve dezavantajlarının) ifade edilmesi,

Tasarım Desenleri

- **Observer ( Gözlemci ) Deseni :** Çeşitli nesnelere, diğer bir nesnenin durumunun değiştiğini bildirmek.

- **Fasat – Facade Deseni :** Çoğunlukla artırımlı olarak geliştirilen belli sayıdaki ilişkili nesnelere ait arayüzleri derleyip toparlamak.

- **Iterator ( Yineleyeci ) Deseni :** Bir koleksiyondaki elemanlara, o koleksiyonun nasıl gerçekleştirildiğinden bağımsız bir şekilde erişim için standart bir yol sağlamak.

- **Decorator ( Dekörator ) Deseni :** Varolan bir sınıfın fonksiyonelliğini çalışma zamanında genişletme olasılığına izin vermek.

bir tasarım sürecinde desenleri kullan. mak genellikle bir tasarım geliştirmek, bir problemi tecrübe etmek ve sonra bir desenin kullanılabileceğinin farkına varmayı içermektedir.

Gerçekleştirim konuları

Yazılım mühendisliği, yazılım geliştirmede sistemin ilk gereksinimlerinin alınmasın- dan konuşlandırılan sistemin bakım ve yönetimine kadar yer alan tüm etkinlikleri içermektedir.

Gerçekleştirim programlama dilleri ile programlar geliştirme veya hazır sistemleri organizasyonun isterlerini karşılamak için uygun hale getirme işlemleridir.

Gercekleştirim hususları : 1.Yeniden kullanım yazılım geliştirilirken mümkün olduğu kadar var olan kodlardan yararlanmak gerekir

2.yapılandırma yönetimi bileşenlerin sürümlerinin kontrol altında olması

3. konakçı hedef geliştirme

Yazılım yeniden kullanımı bir takım farklı düzey- lerde gerçekleşebilir:

1 Soyutlama düzeyi başarılı soyutlamalara , lişkin bilgilerin kullanımı

2. nesne düzeyi nesnelerin kütüphanelerden doğrudan yeniden kullanımı

3. bileşen düzeyi bileşenler ilgili fonksiyonellikler ve servisleri sağlamak için bir arada çalışan nesne grupları

4. sistem düzeyi sistemin yeniden kullanımı

Yeniden kullanımın maliyetleri de vardır : 1. İsterleri karşılayıp karşılamadığını göreceğimiz zaman maliyeti

2.yeniden kullanım için satın alınması gereken yazılımın satın alım maliyeti

3. bileşen ve sistem yapılandırma uyarlanma maliyeti

4.bütünleştirme yapılırken yaşanılacak maliyetler

Konfigürasyon yönetimi

Yapılandırma yönetiminin amacı, tüm geliştiricilerin proje kodlarına ve dokümanlarına kontrollü bir biçimde erişebilmesi, hangi değişikliklerin yapılmış olduğunu bulabilmesi ve yeni bir sistem oluşturmak için bileşenleri derleyip, bağlayabilmesi için sistem bütünleştirme sürecini desteklemektir.

Sürüm yönetimi, yazılım bileşenlerinin farklı sürümlerinin takibinin yapılabilmesi için destek sağlanmasıdır.

bir sistemin her sürümünü oluştururken bileşenlerin hangi sürümlerinin kullanıldığının tanımlanmasında geliştiricilere yardım etmek için destek sağlanması

problem izleme oluşan problemleri raporlayabilme ve ne zaman düzeltildiğini görmek için yapılan destek sağlaması

dağıtım yönetimi yeni sürümlerin müşterilere dağıtım sürecidir.,

konakçı-hedef geliştirme

bir bilgisayarda konakçı geliştirilen yazılım farklı bir makinede hedef çalışır

açık kaynak geliştirme

bir yazılım sisteminin kaynak kodunun yayınlandığı ve gönüllülerin geliştrme sürecine katılım için davet edildikleri geliştirme yaklaşımıdır. Bu geliştiricilerin bir çoğu aynı zamanda kodun kullanıcılarıdır. Açık kaynak yazılımlar İnternet'in ve yazılım mühendisliğinin omurgasıdır.

açık kaynak lisanslama

Kodu geliştirenler, açık kaynak yazılım lisansı içine hukuki olarak bağlayıcı şartlar ekleyerek o kodun kullanımına kısıtlamalar getirebi- iler

Çoğu açık kaynak lisans biçimidir. üç genel modelden birisinin değişik biçimidir: 1.GNY Genel Kamu Lisansı açık kaynak yazılım kullanıysanız geliştirdiğiniz yazılımı da açık kaynak yampnız gerektiği anlamına gelen lisans

2.GNU Kısıtlı Genel Kamu Lisansı açık kaynak koda bağlantı sağlayan bileşenler geliştirip bu bileşenlerin kaynak kodlarının yayınlanma zorunluluğu olmayan bir çeşididir.

Berkley Standart Dağıtım BSD bu lisans açık kaynak koda yapılan değişiklikleri yayınlamak zorunda olamdığınız anlamına gelen karşılıklı olmayan lisanstır

yazılım tasarımı ve gerçekleştirimi dönüşümlü etkinliklerdir. Tasarımdaki detay düzeyi geliştirilmekte olan sistemin türüne ve plan güdümlü mü yoksa çevik bir yaklaşım mı kullandığınıza bağlıdır.

Nesne yönelimli tasarım süreci sistem mimarisini tasarlamak, sistemdeki nesneleri belirlemek, farklı nesne modelleri kullanarak tasarımı tanımlamak, ve bileşen arayüzlerini belgelemek için gerekli etkinlikleri içermektedir.

Bir nesne yönelimli tasarım süreci sırasında bir dizi farklı model üretilebilir. Bu modeller statik modelleri (sınıf modelleri, genelleştirme modelleri, ilişki modelleri) ve dinamik modelleri (sıra modelleri, durum makinesi modelleri) içermektedir.

UML Modelleri

**- Sınıf diyagramı** 🡪 Sistemdeki nesne sınıflarını ve bu sınıflar arasındaki ilişkileri gösterir

- **Sıra diyagramı**🡪 Aktörler ile sistem ve sistem bileşenleri arasındaki etkileşimleri gösterir

- **Durum diyagramı**🡪 Sistemin iç ve dış olaylara nasıl tepki verdiğini gösterir.

- **Use-case diyagramı**🡪 Sistem ile çevre arasındaki etkileşimleri gösterir.

- **Etkinlik diyagramı**🡪 Bir süreçte veya veri işlemede olan etkinlikleri gösterir.

Bölüm 8: Yazılım Testi

Yazılım testinin amacı, bir programın kendinden beklenileni yaptığını göstermek ve piyasaya sürmeden önce olası hatalarını ortaya çıkarmaktır Geçerleme testinde, sistemin beklenen kullanımını yansıtan test takımlarının bir kümesi kullanılarak sistemin doğru bir şekilde çalışıp çalışmadığı kontrol edilir. İkincisi kusur testidir. Kusur testinde, test takımları kusurları ortaya çıkarmak için tasarlanır. Kusur testindeki test takımları belirsiz olabilir ve sistemin normal koşullarda nasıl kullanıldığını yansıtmak zorunda değildir.

Test etme, yazılım doğrulama ve geçerleme sürecinin bir parçasıdır

Doğrulama: Doğru çalışan ürün geliştirildi mi? Geçerleme : Ürün doğru şekilde geliştirildi mi?

Yazılım doğrulama, yazılımın belirtilen fonksiyonel ve fonksiyonel olmayan gertk- sinimleri karşılayıp karşılamadığını kontrol etme sürecidir. Yazılım geçerleme ise daha genel bir süreçtir. Yazılım geçerlemenin amacı, yazılımın, müşterilerin beklentilerini karşıladığından emin olmaktır.

Denetlemeler daha çok programın kaynak koduna odaklanır, fakat yazılımın, herhangi bir okunabilir öğesi de kontrol edilebilir ancak, denetlemeler yazılım testinin yerini tutamaz.

ticari bir yazılım sisteminin genel olarak aşağıdaki üç test aşamasını geçmesi gerekir.

1 Geliştirme testinde, sistem olası bozukluk ve kusurları bulmak için geliştirme sürecinde test edilir. 2 dağıtım testinde, piyasaya sürülmeden önce sistemin tam sürümü ayrı bir ekip tarafından test edilir. 3 Kullanıcı testinde, sistemin kullanıcıları veya potansiyel kullanıcıları, sistemi kendi ortamlarında test eder.

“Gerileme” testi, her değişiklikten sonra o ana kadar yapılan bütün testlerin yeniden yapılmasını içerir.

Geliştirme testi

Geliştirme testi, sistemi geliştiren ekip tarafından yürütülen tüm test etme etkinliklerini içerir.

Hata ayıklama, test ederken bulunan hataların ve problemlerin düzeltilmesi sürecidir.

Geliştirme sürecinin üç aşaması vardır.

1 Birim(unit) testinde bireysel program birimleri ve nesne sınıfları test edilir. Birim testi, nesne ve metotların fonksiyonelliğini test etmeye odaklanmalıdır.

2 Bileşen testinde, birkaç ayrı birim bir araya getirilip tümleştirilerek bileşenler oluşturulur. Bileşen testi, bileşen fonksiyonlarına girişi sağlayan bileşen arayüz. lerini test etmeye odaklanmalıdır.

3 Sistem testinde sistemdeki bileşenlerin bazıları veya hepsi tümleştirilir ve sistem bir bütün olarak test edilir.

Otomatik bir test üç kısımdan oluşur:

1 Kurulum kısmı, sistemin başlangıç durumunun bir test takımı ile, yani girdi ve beklenen çıktıları ile tanımlandığı kısımdır 2 Çağrıkısmı, test edilecek metot ve nesnelerin çağrıldığı,kısımdır. 3 Bildirim kısmı, çağrı sonuçlarının beklenen sonuçlar ile Karşılaştırıldığı kısımdır

Taklit nesneler arayüzleri dış nesnelerle aynı olan ve onların fonksiyonlarını ben. zeştiren nesnelerdir.

Birim test takımlarının seçimi

Test, pahalı ve zaman alıcıdır. Bu nedenle etkili birim test takımlarının seçilmesi önemlidir. Etkililik, bu durumda iki şey demektir:

Test takımları, tanımlandıkları şekilde kullanıldıklarında test edilen bileşenlern beklenildiğini yaptığını göstermelidir. İki farklı takım tasarlanmalıdır. İlki normal işleyişi yansıtıp test etmeli diğer takım ise problemlerin yaygın olarak çıktığı durumları test tecrübelerine dayanarak denemelidir. Test takımı seçmek için etkili olabilecek iki strateji şöylece belirlenebilir:

Birim test takımlarının seçimi

- **Bölünme testi:** Girdileri anlamlı gruplara bölme

- **Kılavuz test:** Test etme kurallarını kullanır. Bileşenler geliştrilirken programcıların sık sık yaptığı hata türlerine ilişkin ön tecrübeleri yansıtır.

Eş değer bölünmeleri belirlemek için sistem spesifikasyonu kullanılırsa, buna **kara-kutu** testi denilir. Bu durumda sistemin nasıl çalıştığıyla ilgili herhangi bir bil. giye ihtiyaç duyulmaz. Kara-kutu testini, diğer olası testleri bulmak için programın kodunun incelendiği **beyaz-kutu testi** ile desteklemek faydalı olabilir. Eş değer bölünme, bölünmeleri sınırlarındaki girdiler işlendiğinde, programcının yaptığı hataların nedenini açıklamaya yardımcı olduğu için etkili bir test etme yakla- şımıdır.

, sıralar, diziler ve listeler içeren bir program test edildiğinde, şu talimatlar olası hataları ortaya çıkarmaya yardımcı olabilir: 1Yazılımı sadece tek bir değere sahip sıralar ile test et. Programcılar doğal olarak birkaç değerden oluşan sıraları dikkate alırlar ve programlarını bazen bu varsi yımlarından çıkarak gerçekleştirirler. Bunun sonucu olarak, tek değerli bir sın girilirse program beklentiye uygun bir şekilde çalışmaz. 2 Farklı testlerde, farklı büyüklükte, farklı sıralar kullan. Bu şekilde hazırlanan bi test, hatalı bir programın girdilerinin bazı rastlantısal özelliklerinden dolayı Ya” lışlıkla doğru çıktı üretme olasılığını azaltır. 3 Testleri öyle üret ki, dizinin ilk, orta ve son elemanlarına ulaşılabilsin. Bu yaklır şım, bölünmelerin sınırlarında olan problemleri ortaya çıkarır.

Bileşen testi

Yazılım bileşenleri, genellikle birbiri ile etkileşen birkaç nesneden oluşur.

Sistemtesti

Geliştirme süreci esnasında yapılan sistem testi sistemin bir versiyonunu yaratmak için bileşenlerin bütünleştirimesini ve hemen ardından tüm sistemin test edilmesini kapsar. Sistem testi bileşen testi ile örtüşür ancak iki fark vardır : Birinde tamamlanmış sistem test ediliyor birinde sistemi oluşturan parçaların farklı farklı iletişimleri test ediliyor

Test güdümlü geliştirme

Test Güdümlü Geliştirme (TGG), test etmenin ve kod geliştirmenin dönüşümlü olark | uygulandığı bir program geliştirme yaklaşımıdır kodlar adım adım geliştirilir ve artış getiren her adım için bir test küme tanımlanır. TGG yöntemi, programcıların bir kod bölütünün gerçekte ne yapması gerektiği konusundaki fikirlerinin netleştirmesine yardımcı olur. TGG (Test Güdümlü Geliştirme)

- Önce test yazılır.

- Kod testleri geçecek şekilde yazılır.

- Kod kapsamı artar, hata ayıklama kolaylaşır tecrübe gerektirir tecrübe yoksa geliştirme yapılamaz. Yararları kod kapsam yazılan her bölüm için en azından bir test tanımlıdır. Gerileme testi adım adım geliştirilmede değişikliklerin sorun çıkartıp çıkarmadığını kontrol etmeyi sağlar

Basitleşitirilmiş hata ayıklama bir test başarısızlıkla sonuçlandığında problem nerede açıkça görülebilir. Sistem dokümantasyonu testler kodun ne yapması gerektiği tanımlayan dokümantasyon sayesinde görülebilir.

TGG'nin en önemli yararlarından biri gerileme testinin maliyetini azaltmasıdır.

Gerileme testi, değişik liklerin sistemde yeni hatalar oluşturup oluşturmadığını ve yeni kodun var olan kodla beklenildiği gibi etkileşime girip girmediğini kontrol eder.

Dağıtım testi

Dağıtım testi, sistemin belli bir dağıtımın, geliştirme takımının dışındaki kişiler tara- fından kullanılması ile test edilmesi anlamına gelmektedir. Dağıtım testinin önde gelen amaçlarından birisi de, sistemin kullanım için yete- rince iyi olduğu konusunda sistem satıcılarını da ikna etmektir.

1 Gereksinimlere dayalı test etme

Gereksinimler mühendisliğinin temel prensiplerinden birisi, gereksinimlerinin test edilebilir olmasıdır.

2 Senaryo testi

Senaryo testi, bir dağıtım testi yaklaşımdır. Bu amaçla tipik kullanım durumları tasarlanır ve bu senaryolar vasıtasıyla test takımları geliştirilir.

3 Performans testi

Sistem bir kere tamamıyla bütünleştirildiğinde, sistemin performans ve güveni gibi Ea özelliklerini test etmek mümkündür. Performans testleri, sistemin amaçları, yükü kaldırabileceğinden emin olmak için tasarlanmalıdır.

Kullanıcı testi

Kullanıcı veya müşterinin test etme sürecinin sistem testi aşaması için girdi sağladığı ve tavsiyelerde bulunduğu aşamadır. Üç farklı şekli vardır :

1 alfa testi. Yazılım kullanıcıları arasından seçilen bir grup yazılımın ilk dağıtı- mını test etmek için geliştirme takımıyla birlikte çalışır.

2 Betatesti. Yazılımın bir dağıtımı daha büyük bir kullanıcı grubunun

3 Kabul testi.

Test Araçları

- Otomasyon: JUnit, Selenium

- İzleme: Bugzilla, JIRA

- CI/CD: Jenkins, GitLab

Bölüm 9: Yazılım Evrimi

Yazılım evrimi neden gereklidir ? yazılımın yaşam süresi boyunca kullanışlı kalması, rakiplerinden geri kalmaması, gelecekte kendini geliştiren yeni teknolojilere ayak uydurabilmesi için yazılımın kendini güncel tutması önemlidir. yazılım ürünleri ve uygulamaları platform değişiklikleri ve rakipleri tarafından tanıtılan yeni özelliklerle başa çıkmak için evrilmelidir. Yazılım evrimi özellikle, daha geniş bir sistemin parçası olan kurumsal sistemlerde maliyetlidir. Bir parça üzerinde değişiklerin neleri etkileyeceğini düşünmek zor ve çok maliyetlidir. Bir parçanın veya sistemin değişimi etkileşimde olduğu sistemleri de güncellemek gerektirebilir.

Brownfield = yazılım geliştirme terimidir. Yazılım sistemlerinin başka sistemlere bağımlı oldukları bir ortamda geliştirilmeleri ve yönetilmeleri gerektiği durumları tarif eder.

Yazılım mühendisliği sistemin tüm yaşam süresi boyunca gereksinimler tasarım gerçekleştirim ve sınama ile devam eden bir döngüdür.

Rekabetin artması çok hızlı geri bildirim alma evrim sürelerinde kısaltma yaratmıştır.

müşteriye özel yazılımın evrimi genellikle başka bir model izler. Sistem müşterisi bir yazılım firmasına yazılımı üretmesi için ödeme yapabilir ve daha sonra destek ve evrim için, kendi personelini kullanarak, sorumluluğu alır.

Evrim

standart bir yazılım değişimi veya evrini süreci yoktur. en uygun evrim süreci bakımı yapılan yazılım | türüne, bir kuruluşta kullanılan yazılım süreçlerine ve görev alan kişilerin yetenekle- rine bağlıdır. metin, el yazısı, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

değişiklik istekleri bazen üç nedenden dolayı acil olarak yapılmak istenebilir :

1 güvenlik hassasiyeti

2 işleyiş ortamında anomali yaratan değişiklikler

3 yeni rakiplerin yeni güncellemelerle piyasaya cıkması

Buradaki tehlikelere oluşturulan hızlı çözümler gereksinimler tasasrım ve kod tutarsızlıklar oluşturabilir. Acil durumlara verilen hızlı çözümler yaşlanma sürecini arttırır ve gelecek için bakım maliyetlerini arttırır. Ideal olarak acil çözümlerden sonra yeni kod yeniden üretilmeli ve özellikleri kaybetmenin önlenmesi için iyileştirilmelidir.

Gelişitirici çevik evrim takımısı ile plan güdümlü yaklaşım tercih ederse : evrim takımı düzenli bir belgeleme bekleyebilir bu potansiyel bir sorundur. Tam tersi takım planlamalarında ise evrim takımı çevik yöntemleri tercih ediyorsa otomasyon testler sıfırdan yazılması gerekebilir sistemdeki kod çevik yönteme uygun olmayabilir bu da yeniden muhendislik maliyeti çıkartabilir.

kalıt sistemler

kalıt sistem nedir = yeni sistemlerin geliştirilmesi için artık kullanılmayan dillere ve teknolojilere dayandıkları eski sistemlerdir. kalıt sistemler sadece yazılım sistemleri olmayıp, donanım, yazılım, kütüphanele, ve diğer destekleyen yazılım ve iş süreçlerini içeren daha geniş sosyoteknik Sistemlerdir.

COBOL iş hayatı sistemleri yazmak için geliştirilmiş bir programlama dilidir

Kalıt sistemleri yeni sistemlerle yenilemenin niçin pahalı ve riskli olduğu ile ilgili çeşitli nedenler vardır:

* 1 kalıt sistemin nadiren tam bir spesifikasyonu vardır
* 2 iş hayatı ve kalıt sistem iç içe geçmiştir.
* 3 belgelenmenin olmayışı ufak detayların gözden kaçmasına neden olur
* 4 yeni yazılım geliştirmek doğal bir risktir
* 5 farklı geliştirici ve evrim ekibinin oluşu koda olan hakimiyeti azaltması
* 6 kalıt sistemin eski kullanımı kullanıcısı olmayan dillerle yazılmış olması
* 7 uzun yıllar hayatta olmasıyla yapılan bakımların ana kodu bozması kaçınılmazdır.

Kalıt sistem yönetimi

Dört stratejik kalıt sistemlerin değerlendirmesini yapacak strateji vardır

* 1 sistemi tamamen atmak
* 2 sistemi değiştirmeden bırakmak ve normal bakıma tabii tutmak
* 3 sistemin bakıma uygunluğunu iyileştirmek için yeniden muhendislik kullanmak
* 4 sistemin tümünü veya bir bölümünü yeni sistemle değiştirmek

Kalıt sistem değerlendirme = düşük kalite düşük iş hayatına etki rezalettir yüksek maliyetlere yol açar – düşük kalite yüksek iş hayatına etki atılamaz katkı vermekte – yüksek kalite düşük iş hayatına katkı bakımları ucuzdur en azından – yüksek kalite yüksek iş hayatına etki devam süper burası

İş hayatını sistemin kullanımı desteklenen iş dsüreçleri güvenilirlik ve çıktılarından ölçebiliriz.

Bir uygulama sisteminin teknik kalitesini değerlendirmek için ana sisteme olan güvenilirliğini sisteme bakım yapmanın zorluklarını ve belgelendirmesini değerlendirebilirisiniz. Anlaşılabilirlik belgeleme veri performans kullanılan dil konfigürasyon personel yetkinlikleri kullanılan arayüz sayısı gibi etkiler değerlendirmeye yardımcı olur

Yazılım bakımı

Yazılım bakımı bir sistem teslim edildikten sonra sistemin değiştirilmesi sürecidir.

Değişikliler var olan sistem bileşenlerini değiştirerek ve gerekli olduğu yerlerde sisteme yeni bileşenler ekleyerek gerçekleştirilir. Yazılım bakımının üç farklı türü vardır :

1 **Düzeltici bakım** hataları ve zayıf noktaları gidermek için yapılan tamirler.

2 **Uyarlayıcı bakım** yazılımı yeni platformlara ve ortamlara uyarlamak için yapılan bakımlar

3 **Mükemmeliyetçi bakım** yeni özellikler eklemek ve gereksinimleri desteklemek için fonksiyonellik eklemesi. Pratikte bu bakım türleri arasında kesin bir ayrım yoktur .

Maliyet açısından işlevsellik ekleme – kusur düzeltme – ortama uyarlama biçiminde

Deneyim, bir sisteme yeni özelliklerin bakım sırasında eklenmesinin, aynı özellik- lerin ilk geliştirme sırasında gerçekleştirilmesinden genellikle daha pahalı olduğunu göstermiştir.

Bakım tahminleme

Sistemin arayüzlerinin sayısı ve karmaşıklığı- gecici sistem gereksinimlerinin sayısı- sistem kullanıldığı iş süreçleri sistemin çevresiyle ilişkisini değerlendirmeye yarar

Bakım kolaylığını değerlendirmek için düzeltme amaçlı bakım isteklerinin sayısı- etki analizi için gereken ortalama zaman- bir değişiklik isteğini gerçekleştirmek için gereken ortalama zaman- dikkat çeken değişiklik isteklerinin sayısı

Yeniden yazılım mühendisliği

Yazılım bakımı değiştirilmesi gereken programın anlaşılmasını ve daha sonra gerekli değişikliklerin gerçekleştirilmesini içerir. **Yeniden mühendislik sistemin yeniden belgelenmesini, sistem mimarisinin yeniden üretimini, programları modern bir programlama diline dönüştürmeyi veya sistem verilerinin yapısını veya değerlerini değiştirmeyi ve güncellemeyi içerebilir.**

Yeniden mühedisliğin yenisiyle değiştirmeye göre iki önemli üstünlüğü:

1 Riski azaltmak : yeni yazılım geliştirmek doğal risktir oluşturabileceği gecikmeler iş kaybına ekstra maliyetlere yol acacaktır.

2 maliyeti azaltmak : Yeniden mühendisliğin maliyeti yeni yazılım geliştirme maliyetine göre önemli derecede az olabilir.

**Yeniden mühendislik sürecindeki etkinlikler**

1Kaynak kod dönüşümü eksi dilden modern dile dönüştürme veya farklı dile dönüştürme etkinliği

2 **Tersine mühendislik** program analiz edilir ve programdan bilgi çıkarılır işlem yapısını ve fonksiyonelliği belgelemeye yardımcı olur .

3 program yapısının iyileştirilmesi

4 programın mödüllere ayrılması

5 verinin yeniden mühendisliği

Kalıt sistemlerin bu şekilde sarılması büyük ölçekli yeniden kullanılabilir servisler geliştirmek için önemli bir tekniktir.

Yeniden üretim

**Yeniden üretim** bir programın yapılan değişiklikler ile zayıflamasını yavaşlatmak için iyileştirmeler yapılması sürecidir. yapısını iyileştirmek için, karmaşıklığını azaltmak için ve anlaşılmasını kolaylaştırmak için değiştirmek anlamındadır. Yeniden üretime gelecekteki değişiklerin problemlerini azaltan önleyici bakım diyebiliriz. Yeniden üretim çevik yöntemler değişim tabanlı oldukları için bu yöntemlerin ayrılmaz bir parçasıdır.

**Yeniden mühendislik**, bir sistemin bir süre bakımı yapıldıktan sonra ve bakım maliyetleri artınca gerçekleştirilir. **Yeniden üretim** geliştirme ve evrim süreci boyunca devam eden bir iyileştirme sürecidir. Bir sistemin bakımını güçleştiren ve maliyetlerini artıran yapı ve kod bozulmasından kaçınmayı amaçlar. Yeniden üretim ile yineleyen koddan uzun yöntemlerden switch case deyimleri veri kümeleme spekülatif genellik gibi sorunlar giderilebilir.

Yazılım geliştirme ve evrim, **spiral bir model** ile gösterilebilecek bütünleşik ve yinelemeli bir süreç olarak düşünülebilir.

Yazılım evrim süreci değişiklik istekleri tarafından yönetilir ve değişiklik etki analizi, sürüm planlama ve değişiklik gerçekleştirimini içerir.

**Yeniden yazılım mühendisliği** yazılımın anlaşılmasını ve değiştirilmesini daha kolaylaştırmak için yeniden yapılandırılması ve yeniden belgelendirilmesi ile ilgilidir.

**Yeniden üretim**, fonksiyonelliği koruyan küçük program değişiklikleri yapmak, önleyici bakım olarak düşünülebilir.

CI/CD ile Evrim

Sürekli entegrasyon ve sürekli teslimat modern yazılım evriminde kritik rol oynar.

Cocomo = mikro maliyet kestirim modeline örnektir

Kullanılacak ayrıntı düzeyine göre üç ayrı model biçiminde yapılabilir : 1 Temel Model 2 Ara Model 3 Ayrıntı Model

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

metin, yazı tipi, ekran görüntüsü, beyaz içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Proje sınıfları

1 Ayrık projeler = boyutları küçük deneyimli personel tarafından LAN üzerinde çalışan İK sistemi gibi

2 Yarı gömülü hem bilgi boyutu hem donanım sürme boyutu olan projeler

3 Gömülü projeler donanım sürmeyi hedefleyen projeler insansız araç misali

1 Temel model = küçük ve orta boy projeler için hızlı kestirim yapmak amacıyla kullanılır dezavantajı yazılım ekip ve geliştirme ortamını dikkate almaz avantajı kolaylıkla uygulanır

2 Ara model = temel modelin eksikliğini kapatmak için oluşmuş. Yazılım projesinin zaman ve iş gücü maliyetlerinin kestiriminde ekibin özelliklerini ve ortam donanım gibi unsurları dikkate alır

Üç aşamadan oluşur : iş gücü hesaplama maliyet çarpanı hesaplama ilk iş gücü değerini düzeltme

Rely yazılımın güvenirliği data veri tabanının büyüklüğü cplx karmaşıklığı time işletim zamanı kısıtı stor ana bellek kısıtı virt bilgisayar platform değişim olasılığı turn bilgisayar işe dönüş zamanı

metin, ekran görüntüsü, yazı tipi, iş kartı içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Ayrıntı modeli = temel ve ara modele ek olarak iki özellik taşır

1 aşama ile ilgili işgücü katsayıları

2 üç düzey ürün sıra düzeni : maliyet kestiriminde modül-altsistem-sistem sıra düzenini dikkate alır

Yazılım projesinde maliyet kestirim modeli Cocomo’yu açıklayın.(2017 vizede çıktı)

Cevap: COCOMO, oldukça ilgi görmüş bir maliyet kestirim modelidir. COCOMO mikro model olarak belirtilen maliyet kestirim modellerine iyi bir örnektir. Uygulama, kullanılacak ayrıntı düzeyine göre üç ayrı model biçiminde yapılabilir: Temel Model, Ara Model, Ayrıntı Model .Tüm COCOMO modelleri, temel girdi olarak satır sayısı kestirimini alır ve çıktı olarak iş-gücü (Kişi-ay, Kişi-yıl vb.) ve zaman (ay, yıl, hafta vb.) çıktılarını verir.