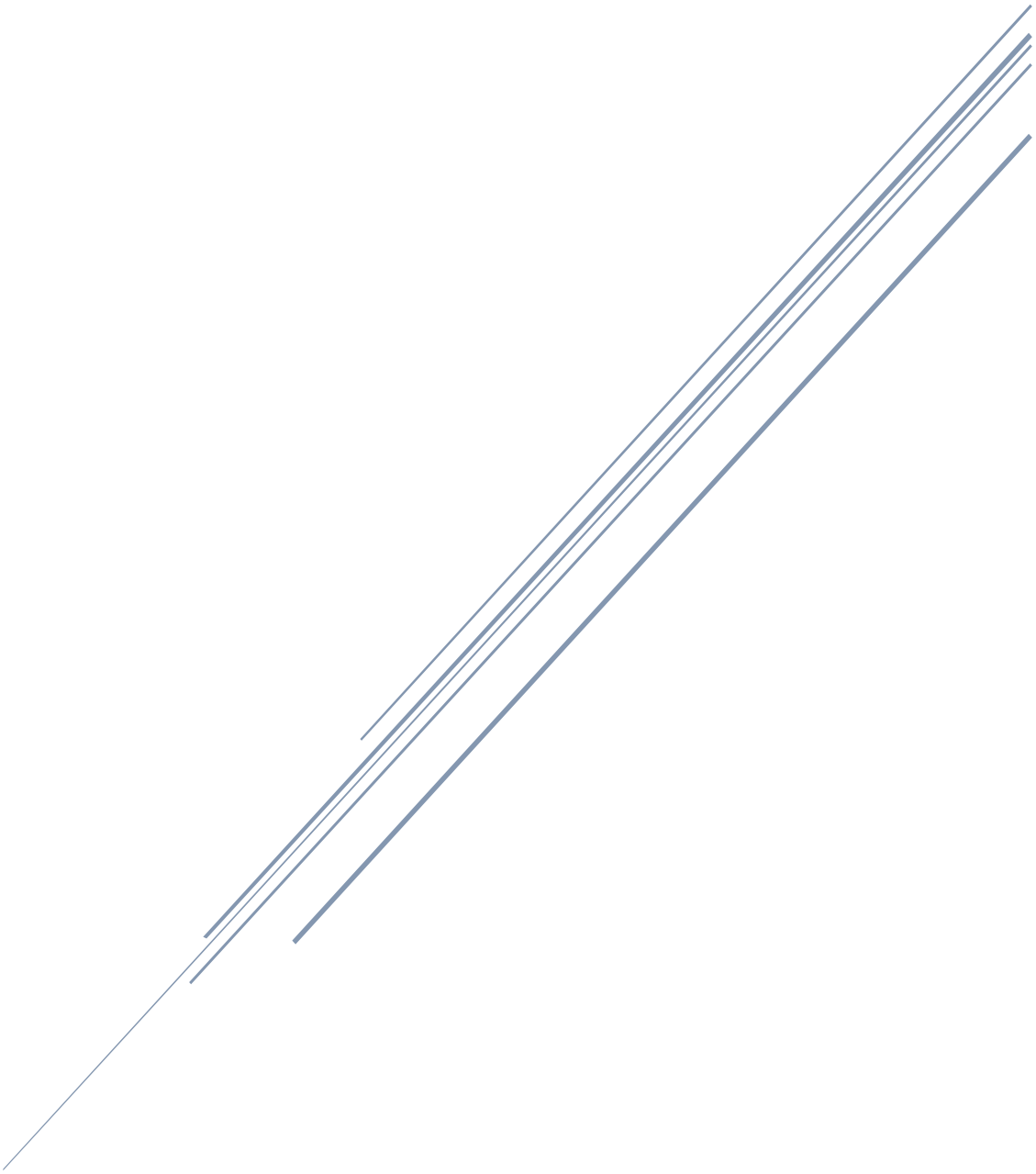
YAZILIM YAŞAM DÖNGÜSÜ MODELLERİ

DURSUN EMRE DURHAN

İZMİR BAKIRÇAY ÜNİVERSTİESİ BİLGİSAYAR MÜHENDİSLİĞİ 1. SINIF



Yazılım Yaşam Döngüsü Modelleri

Yazılım yaşam döngüsü, bir yazılım ürününün geliştirilmesi, test edilmesi ve dağıtılmasından oluşan bir süreçtir. Genellikle birden fazla aşamadan oluşur, bu aşamalar yazılım ürünün süreçteki durumuna göre farklı isimler alabilir. Yazılım yaşam döngüsünün ortaya çıkmasının sebebi teorikteki yazılım geliştirme aşamalarının pratikle pek uyuşmamasıdır. Bu uyuşmazlığın sebebi, yazılımı geliştiren kişilerin özünde insan olmaları, bundan dolayı hata yapmaya ve iletişim problemleri yaşamaya çok müsait olmaları diğer bir sebepse yazılım geliştirme süreci boyunca müşterilerin isteklerinin değişebiliyor olması. Zaman içinde yazılım sektöründe bu problemin çözümü için birçok yazılım yaşam döngüsü modeli geliştirilmiştir. Bu yazıda bu modellerden bazıları incelenecektir.

Çağlayan Modeli

Çağlayan modeli ilk olarak 1970 yılında Winston W. Royce tarafından tanımlandı. Çağlayan modeli, proje safhalarının çizgisel ardışık aşamalara bölünmesidir. Her aşama bir önceki aşamanın çıktılarına bağlı olarak gerçekleştirilir. Model yazılım geliştirme için uygulanacağı zaman 7 aşamadan oluşur: Gereksinim, analiz, tasarım, kodlama ve uygulama, test, dağıtım, bakım. Çağlayan modelin kullanılacağı yazılım projesinin, gereksinimlerinin belirli ve sabit olması, kaynakların geniş olması, sabit bir zaman çizelgesi olması ve proje devam ederken önemli değişiklikler yapılmaması gereklidir.

Avantajları

- Çok sayıda kişilerden oluşan veya değişen takımların belirli bir hedef doğrultusunda ilerlemişini sağlar
- Organizasyonun disiplinli olmasını sağlar.
- Görevlerin anlaşılmasını, takibini ve ayarlanmasını basitleştirir.
- İyi kodlama alışkanlıklarını pekiştirir.
- Yönetim kontrolünü kolaylaştırır.
- Aşamaların ve son teslim tarihlerini iyi bir şekilde tanımlar.

Dezavantajları

- Tasarım uyarlanabilir değildir.
- Test sürecini geliştirme sürecinin sonuna gelene kadar bekletir.
- Hata düzeltmeyi dikkate almaz.
- Proje yaşam döngüsünün son aşamalarına kadar çalışan bir ürün ortaya çıkmaz.
- Gelebilecek değişiklik isteklerini iyi bir şekilde karşılayamaz.

Spiral Model

Spiral model ilk olarak 1986 yılında Barry Boehm tarafından bir makalesinde tanımlanmıştır. Helezonik model, risk analizi ve prototip üretme üzerine kurulmuştur. Her döngü öncesi, içinde bulunulan aşamanın risk analizi yapılır ve o aşama için planı yapılmış olan prototip geliştirilir. Her döngünün sonunda, yeniden planlama yapılır ve yeni hedefler, yeni alternatifler ve yeni kısıtlamalar belirlenir. Model, önceden geliştirilmiş yazılım parçalarının yeniden kullanıldığı yazılım projeleri için çok uygundur. Spiral yazılım geliştirme modeli temel olarak dört ana bölüm içerir. Bunlar, planlama, risk yönetimi, üretim ve kullanıcı değerlendirmeleri olarak adlandırılabilir. Planlama, üretilecek ara ürün için işin planlanması, amaç ve alternatiflerin belirlenmesi, bir önceki adımda üretimi yapılmış olan ürün ile birleştirme yapılması faaliyetlerini içerir. Risk yönetiminde, alternatifler değerlendirilir ve risk analizi yapılır. Üretim, planlanmış ara ürünün geliştirildiği aşamadır. Bu

aşamadan sonra, kullanıcı değerlendirmesi kısmında, ara ürün hakkında kullanıcıların test ve değerlendirmeleri yapılır.

Avantajları

- Risklerin önceden belirlenmesine ve risk yönetimine odaklanır. Bu sayede olası sorunlar önceden tespit edilerek daha iyi bir kontrol sağlanır.

- Diğer modellere göre daha esnek bir yapıdadır. Geliştirme süreci boyunca değişikliklerin yapılması kolaydır.

- İteratif bir yaklaşım sunar. Bu sayede her iterasyonda yeni bir prototip oluşturulur ve bu prototip, müşteri beklentilerine daha iyi cevap verebilecek şekilde geliştirilir.

- Test sürecini iyileştirir. Her bir iterasyonda test edilerek hataların tespit edilmesi ve düzeltilmesi sağlanır.

Dezavantajları

- Diğer modellere göre daha karmaşık bir yapıya sahiptir. Bu nedenle, uygulama ve yönetimi zor olabilir.

- Spiral Model, diğer modellere göre daha fazla zaman ve kaynak gerektirir. Bu nedenle, maliyeti yüksek olabilir.

- Diğer modellere göre daha fazla plan yapılması gerekir. Geliştirme sürecindeki her adım için önceden plan yapılması gereklidir.

- Diğer modellere göre daha fazla bilgi ve tecrübe gerektirir. Bu nedenden dolayı, eğitimli ve tecrübeli personel gerektirir.

Artımlı Model

Artımlı Model, gereksinimlerin yazılım geliştirme döngüsünün birden çok bağımsız modülüne bölündüğü bir yazılım geliştirme sürecidir. Artımlı modelde bütün modüller gereksinimler, tasarım, uygulama ve test aşamalarından geçer. Modülün sonraki sürümlerinin hepsi, önceki sürüme işlev ekler. Süreç, tam sistem elde edilene kadar devam eder.

Artımlı Modeli ne zaman kullanırız?

- Gereksinimler üstün olduğunda.
- Bir projenin uzun bir geliştirme programı olduğunda.
- Yazılım ekibi çok yetenekli veya eğitimli olmadığında.
- Müşteri, ürünün hızlı bir şekilde teslim edilmesini talep ettiğinde.

Avantajları

-Yazılımın parçalar halinde geliştirilmesi, birincil özelliklerin hızlı geliştirilebilmesini ve kullanıcılardan geri bildirim alınmasını kolaylaştırır.

- Yazılım projesinin gereksinimlerinin veya hedeflerinin değişmesi durumunda esneklik sağlar.

- Her artırımın hızlı geliştirilebilmesi ve geri bildirim alabilmesi, kullanıcıların gereksinimlerinin ve beklentilerinin belirlenmesi için kolaylık sağlar.

- Büyük yazılım projeleri parçalar halinde geliştirildiği için, risk faktörünü azaltır. Her artırım, yönetimi ekibi tarafından izlenebilir.

- Kullanıcılardan geri bildirim alarak yazılımın hızlıca geliştirilebilmesi, müşteri memnuniyetini arttırır.

- Artırımların daha iyi hale gelmesi için yeniden tasarım yapılmasına izin verir. Bu, sürekli o gelişen bir yazılım ürünü için çok iyidir.

Dezavantajları

- Projenin hedeflerinin ve gereksinimlerinin belirlenmesi ve yönetilmesi zor bir hal alabilir. Artırımların her biri ayrı bir proje olarak ele alındığından, projeyi yönetmek daha karmaşık bir hale gelir.

- Geliştirme süreci birçok kez test edilmesi gereken bir dizi parça halinde gerçekleştirildiğinden, bütününün test edilmesi zor olabilir.

- Yazılımın her bir parçasının ayrı geliştirildiğinden, yazılımın uzun vadeli bakımı zorlaşabilir. Her artırımın farklı gereksinimlere uygun olması, bakımı zorlaştırabilir.

- Yazılımın parçalar halinde geliştirilmesi, yeniden kullanılabilirliği zorlaştırabilir. Bu durum yeniden tasarım ve yeniden geliştirme gerektirebilir.

- Yazılımın farklı zamanlarda farklı parçalarının geliştirilmesi nedeniyle, kaynakların verimli bir şekilde kullanılması zor bir hale gelebilir. Kaynakların planlanmasında zorluklar ortaya çıkabilir.

Scrum Modeli

Scrum, ekip çalışmasını, sorumluluğu ve iyi tanımlanmış hedefe doğru yinelemeli ilerlemeyi vurgulayan bir yazılım yaşam döngüsü modelidir. Scrum genellikle çevik yazılım geliştirmenin bir parçasıdır. Yazılım geliştirme için Scrum rolleri aşağıdaki rolleri içerir:

-Ürün sahibi: Geliştirme ekibi ile müşteri arasında bağlantıyı sağlar. Tamamlanan ürünle ilgili beklentilerin iletilmesini ve üzerinde anlaşmaya varılmasını sağlamalıdır.

-Scrum Master: Scrum Master'a proje kolaylaştırıcısı denir. Scrum modelinin iyi bir şekilde uygulandığının takip edilmesini sağlarlar. İş birliği ve süreç iyileştirme konularında yetenekli olmalıdırlar.

-Geliştirme Takımı: Scrum geliştirme ekibinin üyeleri, nihai ürünün geliştirilmesi için birlikte çalışır. Geliştiriciler, Scrum ve Agile geliştirme uygulamalarını iyi bilmelidir.

Scrum süreci, geliştiricileri sahip oldukları şeyle çalışmaya ve neyin işe yarar olduğunu sürekli olarak değerlendirmeye teşvik eder. İyi iletişim esastır ve "etkinlikler" adı verilen toplantılarla gerçekleştirilir.

Scrum etkinlikleri aşağıdaki etkinliklerden oluşur:

-Günlük Scrum. Her gün aynı vakitte ve yerde gerçekleşen kısa bir toplantıdır. Ekip, bir gün önce yapılan çalışmaları gözden geçirir ve sonraki 24 saat içinde yapılacakları planlar. Bu, projenin tamamlanmasını engelleyebilecek sorunların tartışıldığı zamandır.

-Sprint. İşin tamamlanması gereken zaman dilimidir. Genellikle 30 gün sürerler. Yeni Sprintler, öncekinin bitiminden hemen sonra başlar.

-Sprint Planlama Toplantısı. Bu toplantılarda herkes hedeflerin belirlenmesi sürecine katılır. Sonunda, en az bir artım yani kullanılabilir bir yazılım parçası üretilmelidir.

-Sprint İncelemesi. Artımı göstermenin zamanıdır.

-Sprint Retrospektifi. Sprint sona erdikten sonra yapılan toplantıdır. Toplantı sırasında bütün ekip süreci derinlemesine düşünür. Bir ekip oluşturma egzersizi de sunulabilir. Bu etkinliğin önemli bir amacı da sürekli iyileştirme yapmaktır.

Avantajları

- Proje yeni ihtiyaçlara göre esnek bir şekilde uyarlanabilir. Bu yüzden, projenin gereksinimleri değiştikçe, takım hızla uyum sağlayabilir.

- Scrum, müşterinin isteklerini ve ihtiyaçlarını karşılamak için tasarlanmıştır. Bu nedenle, müşterilerin ihtiyaçlarının verimli bir şekilde karşılanması, müşteri memnuniyetini artırır.

- İşbirliğine önem veren ve iterasyonel bir yaklaşım kullandığından, ekip, sürekli olarak kendini geliştirir ve üretkenliğini arttırır.

- İlerleme ve gelişmeler düzenli olarak takip edilir, sorunların hızlıca tespiti yapılır ve çözüme kavuşturulur.

- Takım üyelerinin çalışmalarını organize etmek için iş birliği ve diyalogu teşvik eder. Bu da takım üyelerinin işlerine olan bağlılığını ve motivasyonunu arttırır.

Dezavantajları

- Tam zamanlı katılım gerektirir. Bu, üyelerin zamanlarını iyi ayarlamaları gerektiği anlamına gelir ve bunu yapmak kimi zaman zordur.

- Scrum, birden fazla planlama aşamasına sahiptir ve bu süreçler zaman alıcıdır. Bu nedenle, zaman konusunda sıkıntısı çeken yazılım projeleri için uygun olmayabilir.

- Yazılım geliştirme işleminin tamamını kapsayan bir yöntem olduğundan, takım üyelerinin teknik bilgi ve becerileri yüksek olmalıdır.

- Scrum, iş birliği ve diyalogu teşvik etmesine rağmen, bazen takım üyeleri arasında fikir ayrılıkları veya çatışmalar yaşanabilir

Kaynakça

https://tr.wikipedia.org/wiki/Anasayfa
https://www.reddit.com/
https://www.quora.com/
https://www.scrum.org/

Hesaplar

https://medium.com/@dursunemredurhan
https://github.com/dursunemre