Yazılım yaşam döngüsü , yazılım geliştirme aşamalarını ve bu aşamaların yürütüldüğü sırayı açıklar .Her aşama kendinden sonraki aşamanın temelini oluşturur ve bir sonraki aşamaya geçmek için gerekli olan çıktıları üretir .Öte yandan yazılım yaşam döngüsü her zaman ileri yönde ilerlemez ,kendisini oluşturan temel aşamalar içerisinde rotasyon oluşturur. Bu rotasyon içerisindeki temel aşamalarda bir önceki aşamaya geri dönmek ve tekrar ilerlemek mümkündür. Bu temel aşamalar planlama, analiz, tasarım , gerçekleştirme ve teslim-bakımdır. Özünde yazılım yaşam döngüsü, müşterinin zaman ve maliyet beklentilerini karşılamaya yönelik yüksek kaliteli yazılımlar üretmeyi amaçlamaktadır.

1.Aşama (Planlama): SDLC'nin birinci aşamasıdır ve temel ihtiyaçlar belirlenip ,proje planlaması yapılır.

2.Aşama(Analiz): Yazılım yaşam döngüsünün 2. Aşaması olan analiz geliştirilicek olan üründen tam olarak ne istendendiğinin dokümantasyonunun yapıldığı ve ürürün somutlaşmaya başladığı evredir.

3.Aşama(Tasarım): Bu aşamada 2. Aşamada incelenen ihtiyaç özelliklerinden sistem ve yazılım tasarımı hazırlanır. Sistem tasarımı, donanım ve sistem gereksinimlerinin belirlenmesine ve ayrıca genel sistem mimarisinin tanımlanmasına yardımcı olur ve bir sonraki aşama için girdi görevi görür.

4.Aşama(Gerçekleştirme): Bu aşama kodlamanın gerçekleştirildiği aşamadır. Bu aşamada seçilen yazılım dili kullanılıp sistemin modüllere ayrılarak geliştiriciler tarafından inşa edilmesini içerir. Yazılım yaşam döngüsünün en çok zaman alan bölümünü kapsar.

5.Aşama(Teslim-Bakım):Tüm aşamalar tamamlandıktan sonra yazılım ürünün sahaya teslim edilebilir bir versiyonu çıkartılır ve müşteriye teslim edilir. Teslim çıktısı olarak ürün tek başına yeterli değildir. Ürün ile birlikte mutlaka son kullanıcılar için kullanım kılavuzu ve versiyon fark dokümanı oluşturulmalıdır. Teslim ile birlikte bakım aşaması başlar ve ürünün çalışır durumda kalması garanti çerçevesinde kontrol edilir ve ürünün üstünde müşterinin isteği doğrultusunda yeni eklemeler ve değişikler yapılabilir ve bunlar 3 kısma ayrılır.

-Hata düzeltme (Bug fixing):Üründe daha önce fark edilememiş hatalar düzeltilir.

-Yükseltme(Uprage):Ürünün daha yeni sürümlerine yükseltme işlemi yapılır.

-Geliştirme(Enhancement):Mevcut yazılıma yeni özellikler eklenir.

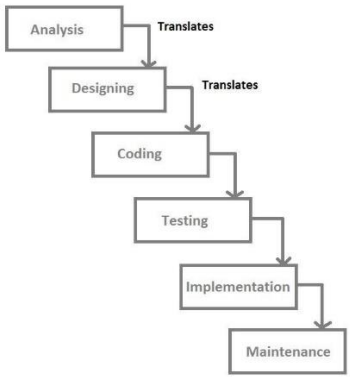
Yeni bir ürünün versiyonu v1.0.0'dır. 3 parçadan oluşan bu numaralandırmanın soldan sağa isimleri majör value , minör value ve build value'dir.Bu 3 tanımı bir örnek üzerinde anlatırsak eğer teslim edilen ürün üzerinde yapılan köklü değişiklikler sonrası ürünün majör değeri değişir ve versiyon 2.0.0 değerini alır.Ürün üzerinde çok fazla farklılık yaratmayan değişiklikler için minör value arttırılır ve ürünün versiyonu 2.1.0 olur.Ürün üzerindeki ufak tefek sorunları giderebilmek adına yapılan değişikliklerde ise ürünün build value'si arttırılır ve ürünün versiyonu 2.1.1 olarak adlandırılır.

SDLC MODELLERİ

1)Şelale(Waterfall)Modeli

En eski ve en iyi bilinen sdlc modellerinden biridir. Temel aşamalar planlamadan bakıma kadar sırasıyla birbirlerini takip ederler. İyi tanımlanmış ve anlaşılmış gereksinimleri olan sistemler , Şelale modeli için uygundur.

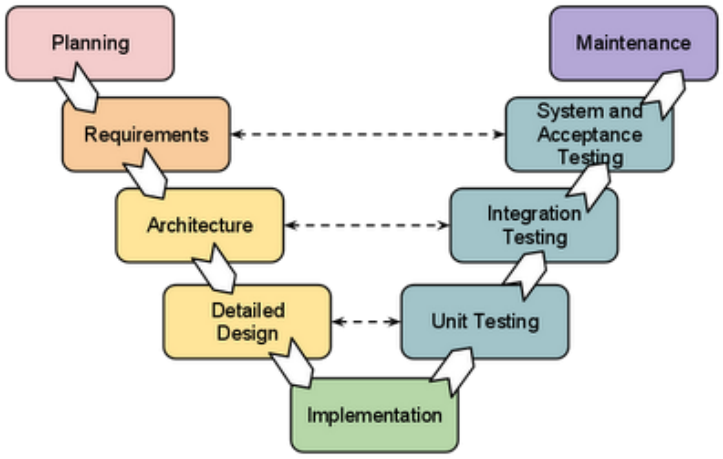
Şelale modelinin iyi yanları , anlaşılması kolay ve kullanımının kolay olmasının yanı sıra yeni ve deneyimsiz personel için yapı sağlar. Yönetim kontrolü için iyidir ve gereksinim istikrarını iyi ayalar. Gereksinimler çok iyi bilindiğinde ve müşterinin isteklerinin değişmeyeceği ve kesin anlaşıldığı zaman kullanıma uygundur. Fakat şelale modeli ,ihtiyaçların sürekli değiştiği projeler ,nesne odakları projeler için uygun değildir ve müşteri ile iletişime çok az fırsat tanır.



2)V Süreç Modeli

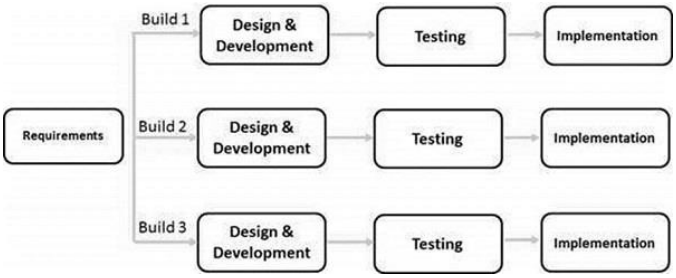
Şelale modelinin bir uzantısıdır. Doğrusal bir şekilde aşağıya inmek yerine V şeklini alarak uygulama ve kodlama aşamasından sonra işlem adımları yukarı doğru bükülür. V Süreç Modelinin, Şelale modelinden en büyük farkı modeldeki erken test planlamasıdır.

Basit ve kullanımı kolaydır. Her aşamada belirli çıktıları vardır. Yaşam döngüsüsün başlarında test planının geliştirilmesi nedeniyle Şelale Modeline göre daha fazla başarı şansı vardır ve gereksinimlerin kolayca anlaşıldığı yerde iyi çalışır. Fakat şelale modeli gibi esnek değildir. Test aşamalarında bulunan sorunlar için net bir çözüm yolu sağlamaz ve ayrıntılı bir plana ek olarak maliyetlidir ve daha fazla zaman gerektirir.



3)Yinelemeli Model

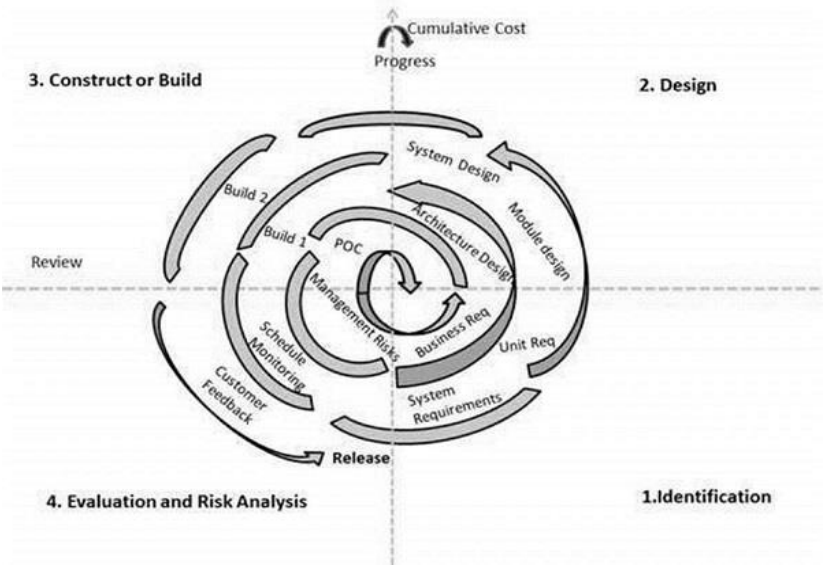
Yinelemeli Model tamamen tekrarla ilgilidir. Geliştiriciler bütün gereksinimleri tam olarak bilmeye başlamak yerine direk yazılım geliştirme aşamasından başlanır ve bunları test eder, iyileştirir. Böylece her yenilemede yazılımın yeni bir versiyonu üretilir ve bu ürün oluşana kadar tekrar olur.



4)Spiral Model

Spiral modelin dört aşaması vardır. Bir yazılım projesi, Spiral adı verilen yinelemelerde bu aşamalardan tekrar tekrar geçer. Bu aşama, iş gereksinimlerinin temel spiralde toplanmasıyla başlar. Sonraki spirallerde ürün olgunlaştıkça, sistem gereksinimlerinin, alt sistem gereksinimlerinin ve birim gereksinimlerinin belirlenmesi bu aşamada yapılır. Bu aşama aynı zamanda müşteri ve sistem analisti arasındaki sürekli iletişim yoluyla sistem gereksinimlerinin anlaşılmasını da içerir. Spiralin sonunda, ürün belirlenen pazarda konuşlandırılır.

Spiral Modelde gereksinimler daha doğru bir şekilde anlaşılır. Kullanıcılar sistemi erken görür, geliştirme daha küçük parçalara bölünebilir ve riskli bölümler daha erken geliştirilebilir, bu da daha iyi risk yönetimine yardımcı olur. Fakat bu modelde yönetim daha karmaşıktır. Oluşturulan ürünün teslim tarihi tam olarak bilinmeyebilir ve küçük ve risksiz projeler için uygun değildir maliyeti çok fazla arttırabilir. Spiral sonsuza kadar devam edebilir ve çok fazla ara aşamanın yanında aşırı dokümantasyon gerektirir.



5)Big-Bang Modeli

Big-Bang modeli ,çok az planlamayla yada hiçbir plan olmadan direk yazılım geliştirme ve kodlamaya odaklanır. Gereksinimler bu süreç ilerledikçe anlaşılır ve uygulanır. Bazen gerekli herhangi bir değişiklik için tüm yazılımı değiştirmeye gerek duyulabilir. Bu model bir veya iki kişinin birlikte çalıştığı küçük projeler için uygundur. Ayrıca akademik projeler için ve gereksinimlerin net anlaşılmadığı ve ürünün teslim tarihinin verilmediği projeler içinde uygundur.

Bu Big Bang Modelinin avantajı, çok basit olması ve çok az planlama gerektirmesi veya hiç planlama gerektirmemesidir. Yönetimi kolaydır ve resmi bir prosedür gerekmez. Bununla birlikte, Big Bang Modeli çok yüksek riskli bir modeldir ve gereksinimlerdeki değişiklikler veya yanlış anlaşılan gereksinimler, projenin tamamen baştan kodlanmasına bile yol açabilir. Minimum riskle tekrarlayan veya küçük projeler için idealdir.

6)Artımlı Model

Sistemi tek seferde teslim etmek yerine, geliştirme ve teslim parçalara bölünür. Her teslim beklenen işlevselliğin bir parçasını karşılar. Kullanıcı gereksinimleri önceliklendirilir ve öncelikli gereksinimler erken teslimlere dahil edilir. Gereksinimler önemlerine ve birbirine bağımlılıklarına göre sıralanarak her yinelemede bunların bir kısmı tamamlanır. Bir parçanın geliştirmesi başladığında, gereksinimleri dondurulur. Olası değişiklikler sonraki teslimlerde ele alınır.

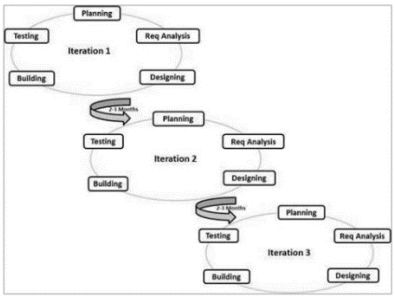
Üretilen her yazılım sürümü birbirini kapsayacak ve giderek artan sayıda işlev içerecek şekilde geliştirilir. Uzun zaman alabilecek ve sistemin eksik işlevsellikle çalışabileceği türdeki projeler bu modele uygun olabilir. Bir taraftan kullanım, diğer taraftan üretim yapılır.

Her teslimle birlikte müşteriye görünen bir değer döndüğünden, sistemin işlevselliği erken aşamalarda ortaya çıkar.Erken teslimler, sonraki teslimler için gereksinimleri çıkarmada prototip vazifesi görür. Projenin tümden batması riskini azaltır. Öncelikli gereksinimleri karşılayan sistem işlevleri daha çok test edilir

7)Çevik Model

Çevik SDLC modeli, çalışan yazılım ürününün hızlı teslimi ile süreç uyarlanabilirliği ve müşteri memnuniyetine odaklanan yinelemeli ve artımlı süreç modellerinin bir kombinasyonudur. Çevik Model, ürünü küçük artımlı yapılara böler. Bu yapılarda yinelemelere gidilir. Her yineleme tipik olarak yaklaşık bir ila üç hafta sürer.

Çevik yöntemler, son zamanlarda yazılım dünyasında yaygın olarak kabul görmektedir. Ancak bu yöntem her zaman tüm ürünler için uygun olmayabilir. Yazılım geliştirmek için çok gerçekçi bir yaklaşımdır ve takım çalışmasını teşvik eder. İşlevsellik hızla geliştirilebilir ve gösterilebilir. Sabit veya değişen ihtiyaçlara uygundur, minimal kurallar vardır ve dokümantasyon kolayca anlaşılır. Yönetimi kolaydır ve geliştiricilere esneklik sağlar fakat büyük ölçüde müşteriye bağlıdır ve müşteri net değil ise ,ekip yanlış yöne yönelebilir ve bu zaman kaybına ,maliyet artışına sebep olur. Üretilen minimum dokümantasyon yüzünden çok yüksek bireysel bağımlılığa ihtiyaç duyar. Yeni ekip üyelerine teknoloji transferi, dokümantasyon eksikliği nedeniyle oldukça zor olabilir.

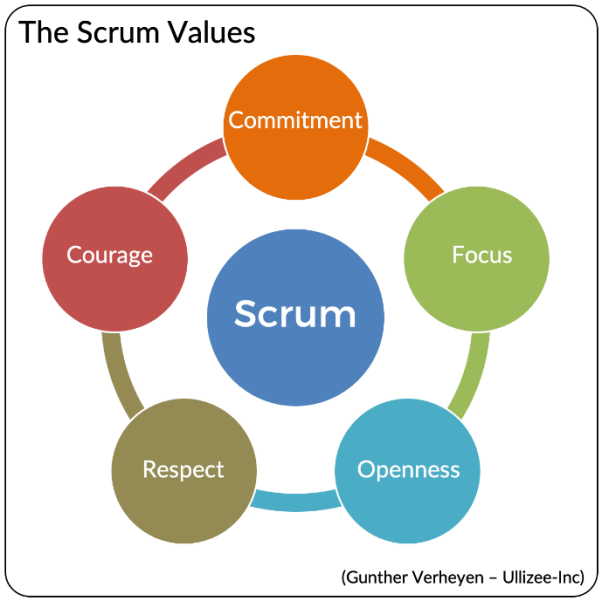


SCRUM NEDİR

Scrum, günümüzde kullanılan en popüler çevik yazılım geliştirme metodlarından biridir ve haklı olarak karmaşık ürünler ve sistemler geliştirmek için kullanılmaktadır. Scrum adı aslında bir rugby terimidir. Rugby'de bir scrum, topu almaya çalışan bir grup oyuncudur. Proje yönetimi alanında "scrum", ekip üyelerinin bir projeyle ilgili başarıları, ne kadar ileri gittikleri, sonraki adımların neler olduğu ve bekledikleri gelecekteki zorluklar hakkında konuşmak için bir araya geldiği kısa toplantıları ifade eder. Toplantılar kısa ve yoğun, daha yüksek kalitede övünen, hızlandırılmış bir ürün teslimiyle sonuçlanır ve aslında bu toplantılar geliştiricilerin kendi içlerinde sosyalleştiği alanlardır.

Önemini tam olarak anlamak için önce Çevik geliştirme sürecinin nasıl işlediğini anlamanız gerekir. Çevik, belirli bir yazılım geliştirme yöntemi veya bir çerçeve değildir, bunun yerine yazılım geliştirme yöntemlerinin sürekli gelişimini destekleyen bir dizi ilkedir. Çevik geliştirme, yinelemeli geliştirme üzerine inşa edilmiş bir dizi yazılım geliştirme metodolojisine ev sahipliği yapar.

Her şey çeşitli yöntemleri izlemek ve yazılım geliştirmek için belirli araçları kullanmakla ilgilidir. Scrum bu yöntemlerden biridir. Scrum'ın ana uygulaması, karmaşık ürünlerin ve sistemlerin geliştirilmesidir. Daha çok "yap, kontrol et ve uyarla" ilkesine dayanmaktadır. Bu süreç optimum üretkenlik sağlar ve ortaya çıkabilecek riskler üzerinde daha fazla kontrol sağlar ve bu yalnızca iki yaklaşım kullanıldığında mümkündür - yineleme ve artış. Scrum ile Çevik Proje Yönetiminin arkasındaki fikir, son kullanıcılara tam olarak istediklerini vermektir. Bu, "Sprintler" veya sürekli geri bildirim ve yinelemeler yoluyla elde edilebilir. Sprintlerin kısa, ancak düzenli, dört haftayı geçmeyen ve önemli bir ürün artışının sunulmasının beklendiği döngüler olması amaçlanmıştır.



Kaynaklar:

https://guntherverheyen.com/the-scrum-values/

https://zeynepaygun.wordpress.com/2017/05/29/what-is-sdlc-sdlc-nedir/

https://medium.com/@denizkilinc/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-temel-a%C5%9Famalar%C4%B1-software-development-life-cycle-core-processes-197a4b503696

https://medium.com/@chandu_22532/sdlc-models-software-development-life-cycle-models-452a1e10d015