**YAZILIM GELİŞTİRME MODELLERİ VE SİSTEM/YAZILIM YAŞAM DÖNGÜSÜ** (**Software Development Lifecycle)**

Yazılım geliştirme yaşam döngüsü (SDLC), bir yazılım oluşturmak için oluşturulan yazılımın kalitesini ve doğruluğunu sağlayan sistematik bir süreçtir. Yazılım geliştirme yaşam döngüsü süreci, müşteri beklentilerini karşılayan yüksek kaliteli yazılımlar üretmeyi amaçlamaktadır. Sistem geliştirme, önceden belirlenmiş bir süre dilimi ve maliyetle tamamlanmalıdır. SDLC, yazılım projelerinin nasıl planlanacağını, oluşturulacağını ve sürdürüleceğini açıklayan ayrıntılı bir planlamadan oluşur. SDLC yaşam döngüsünün her aşamasının kendi süreci ve bir sonraki aşamaya giren çıktıları vardır. Yazılım işlevleri ile ilgili gereksinimler sürekli olarak değişebileceği için, bu aşamalar sürekli bir döngü biçiminde ele alınmalıdır. Bir döngü içerisinde herhangi bir aşamada geriye dönmek ve tekrar ilerlemek mümkün olmalıdır. Temel yazılım geliştirme aşamaları aşağıdaki gibidir:

|  |
| --- |
|  |

* Aşama 1: Gereksinim toplama ve analiz
* Aşama 2: Fizibilite çalışması
* Aşama 3: Tasarım
* Aşama 4: Kodlama
* Aşama 5: Test
* Aşama 6: Kurulum / Dağıtım
* Aşama 7: Bakım

### Aşama 1: İhtiyaçların toplanması ve analizi (Requirement collection and analysis):

### Gereksinimlerin toplanması, SDLC işleminin ilk aşamasıdır. Kalite ve güvence gerekliliklerinin planlanması ve risklerin tespit edilmesi de bu aşamada yapılır. Bu aşama, tüm projenin kapsamı ve projede beklenen sorunlar, fırsatlar ve direktiflerin net şekilde bir resmini verir. Gereksinimlerin toplanma aşaması, ayrıntılı ve kesin olarak gereksinimleri elde edebilmek için ekiplere ihtiyaç duyar. Ekiplerde rol paylaşımı yapılır. Sonuca daha hızlı ulaşırken en üst düzeyde verimlilik almak ve kalite esastır.

### Aşama 2: Fizibilite çalışması (Feasibility study):

İhtiyaç analizi aşaması tamamlandıktan sonraki adım, yazılım ihtiyaçlarını tanımlamak ve belgelendirmektir. Bu aşama, yazılım yaşam döngüsü boyunca tasarlanması ve geliştirilmesi gereken her şeyi içerir. Temel olarak beş tür fizibilite kontrolü vardır.

* **Ekonomik:** Projeyi bütçe dahilinde tamamlayabilir miyiz tamamlayamayabilir miyiz?
* **Yasal:** Bu projeyi siber yasa ve diğer düzenleyici çerçeve / uyumluluk olarak ele alabilir miyiz?
* **Operasyon fizibilitesi:** Müşteri tarafından beklenen operasyonlar yaratabilir miyiz?
* **Teknik:** Mevcut bilgisayar sisteminin yazılımı destekleyip desteklemediğini kontrol etmeniz gerekiyor
* **Takvim/Program:** Projenin verilen program dahilinde tamamlanıp tamamlanamayacağına karar verin.

**Aşama 3: Dizayn:**

Bu üçüncü aşamada sistem ve yazılım tasarım dokümanları, şartname dokümanına göre hazırlanır. Bu, genel sistem mimarisinin tanımlanmasına da yardımcı olur. Bu tasarım aşaması, modelin bir sonraki aşaması için girdi görevi görür. Bu aşamada geliştirilen iki tür tasarım belgesi vardır:

1. **Üst Düzey Tasarım (High-level design) (HLD):**

* Her bir modülün ismi ve kısa açıklaması yazılır.
* Her modülün işlevselliği hakkında kısa taslaklar oluşturulur.
* Ara yüz ilişkileri ve modüller arasındaki bağımlılıklar incelenir ve tespit edilir.
* Temel unsurlarıyla birlikte tanımlanan, veri tabanı tabloları oluşturulur.
* Teknolojik detaylarıyla birlikte eksiksiz olarak mimari diyagramlar oluşturulur

1. **Alt/Düşük Düzey Tasarım** (**Low-Level Design) (LLD):**

* Tür ve boyut içeren veri tabanı tabloları oluşturulur.
* Arayüzün tüm detayları tanımlanır.
* Tüm bağımlılık/bağlılık sorunları giderilir.
* Hata mesajlarının listesi yapılır.
* Her modül için girdi ve çıktılar tanımlanır.

**Aşama 4: Kodlama (Coding):**

* Sistem tasarımı aşaması bittiğinde, bir sonraki aşama kodlama aşamasıdır. Bu aşamada, geliştiriciler seçilen programlama dilini kullanıp kod yazarak tüm sistemi oluşturmaya başlarlar. Kodlama aşamasında görevler birimlere veya modüllere ayrılır ve çeşitli geliştiricilere paylaştırılır. Yazılım Geliştirme Yaşam Döngüsü sürecinin en uzun aşamasıdır. Bu aşamada geliştiricinin önceden tanımlanmış belirli kodlama yönergelerine de uyması gerekir. Ayrıca, kodu oluşturmak ve uygulamak için derleyici, tercümanlar, hata ayıklayıcı gibi programlama araçlarının da kullanılması gerekir.

**Aşama 5: Test (Testing):**

* Yazılım tamamlandığında ve test ortamına dağıtıldığında. Test ekibi tüm sistemin işlevselliğini test etmeye başlar. Bu prosedür, uygulamanın müşteri gereksinimlerine göre çalıştığını doğrulamak için yapılır. Bu aşamada kalite güvencesi (QA) ve test ekibi bazı hatalar / kusurlar bulabilir. Geliştirme ekibi hatayı düzeltir ve yeniden test için KG'ye geri gönderir. Bu süreç, yazılım hatasız, kararlı ve o sistemin iş ihtiyaçlarına göre çalışana kadar devam eder.

### Aşama 6: Kurulum / Dağıtım (Installation/Deployment):

### Yazılım projesinin test aşaması bittiğinde ve sistemde herhangi bir hata kalmadığında, son dağıtım işlemi başlar. Proje yöneticisi tarafından verilen geri bildirimlere dayanarak, nihai yazılım yayınlanır.

**Aşama 7: Bakım (Maintenance):**

Yazılım/program dağıtıldıktan ve müşteriler geliştirilen bu sistemi kullanmaya başladığında, aşağıdaki 3 etkinlik gerçekleştirilir.

* **Hata düzeltme (Bug fixing):** Hiç test edilmeyen bazı senaryolar nedeniyle veya tespit edilemeyen hataların oluşması sonucu yapılır.
* **Yükseltme (Upgrade):** Uygulamayı yazılımın daha yeni sürümlerine yükseltme işlemi yapılır.
* **Geliştirme (Enhancement):** Mevcut yazılıma bazı yeni özellikler eklenir.

Bu SDLC aşamasının temel amacı, ihtiyaçların karşılanmaya devam etmesini ve sistemin ilk aşamada belirtilen spesifikasyona göre çalışmaya devam etmesini sağlamaktır.

**Popüler Yazılım Süreç Modelleri:**

Yazılım geliştirme yaşam döngüsü veya sistem geliştirme yaşam döngüsü olarak bilinen SDLC kavramına bağlı olarak literatürde sıkça geçen ve temel olarak kabul edilebilecek çeşitli yazılım geliştirme metotları bulunmaktadır. Bu yazılım geliştirme modelleri: Şelale modeli (waterfall model), Artımlı Geliştirme (Incremental Development), V modeli ( V-Shaped Model), Artımlı Geliştirme Modeli(Incremental Development), Çevik Modelleme (Agile), Spiral model, Big Bang Modeli, Kodla ve Düzelt (Code and fix) ve Prototipleme (Prototyping).

* **Şelale Modeli (Waterfall model):**

Şelale modeli yaygın olarak kabul gören bir SDLC modelidir. Bu metotta, yazılım geliştirmenin tüm süreci çeşitli aşamalara ayrılmıştır. Şelale yönteminde yazılım geliştirme süreci analiz, tasarım,​ kodlama, test, sürüm ve bakım​​ gibi safhalardan oluşur. Bu SDLC modelinde, bir aşamanın sonucu bir sonraki aşamanın girdisi olarak işlev görür. Bu SDLC modelinde dokümantasyon işlemi çok yoğundur ve önceki aşamalar sonraki aşamalarda ne yapılması gerektiğini belgelemektedir.

* **Artımlı Geliştirme (Incremental Development):**

Artımlı model ayrı bir model değildir. Aslında bir dizi şelale döngüsüdür. Projenin başlangıcında gereksinimler gruplara ayrılmıştır. Her grup için SDLC modeli takip edilir. SDLC işlemi tekrarlanır, her sürüm tüm gereksinimler karşılanıncaya kadar daha fazla işlevsellik katar. Bu yöntemde, her döngü önceki yazılım sürümünün bakım aşaması olarak işlev görür. Artımlı modelde değişiklik yapılması, geliştirme döngülerinin çakışmasına izin verir ve bir sonraki döngü önceki döngü tamamlanmadan başlayabilir.

* **V Modeli (V-shaped Model):**

Şelale modelinin kontrol safhasının daha organize edilmiş hali olarak görülebilir. Her aşama kendi kontrol aşamasıyla eşleştirerek “V” harfine benzer şekilde gösterildiği için bu ismi almıştır. Yatay ve dikey açılar zaman veya projenin tamamlanabilirliğini ve soyut seviyeyi gösterir.

|  |
| --- |
|  |

* **Çevik Modelleme (Agile Model):**

Çevik metodoloji, bir projenin SDLC süreci boyunca geliştirme ve test etkileşimini sürdürmeyi destekleyen bir uygulamadır. Burada gereksinimler ve çözümler kendinden örgütlü olan farklı grupların ortak çalışmasıyla olgunlaşır. Agile yönteminde, tüm proje küçük parçalı yapılara ayrılmıştır. Bu yapıların tümü tekrarlı programlarla sağlanır ve her yineleme bir ila üç hafta sürer. Agile metotları bir yazılım geliştirme yaklaşımı değil, geliştirme süreçleri topluluğudur. Kısa döngülerden oluştuğu için geliştiricilerin motivasyonu yüksek olur ve verim üst seviyelerdedir. Fakat takım üzerindeki hedef baskısı bazen bireyleri demoralize edebilir. Sık sık değişen ihtiyaçlar dolayısıyla çok yoğun ve yorucu çalışma gerektirebilir.

* **Spiral (Helezonik) Model:**

Spiral model risk odaklı bir süreç modelidir. Bu SDLC modeli, ekibin bir şelale, artımlı, şelale vb. gibi bir veya daha fazla işlem modelinin öğelerini benimsememize yardımcı olur. Bu model, prototipleme modelinin ve şelale modelinin en iyi özelliklerini taşır. Spiral metodoloji, tasarım ve geliştirme faaliyetlerinde hızlı prototipleme ve eşzamanlılığın başarılı bir kombinasyonudur.

* **Big Bang Modeli:**

Büyük patlama modeli, çok az planlama ile yazılım geliştirme ve kodlamadaki her türlü kaynağa odaklanmaktadır. İhtiyaçlar geldiklerinde anlaşılır ve uygulanır. Bu model, birlikte çalışan daha küçük boyutlu geliştirme ekibine sahip küçük projeler için en iyi sonucu verir. Akademik yazılım geliştirme projeleri için de yararlıdır. Gereksinimlerin bilinmediği veya son çıkış tarihinin verilmediği ideal bir modeldir.

* **Kodla ve Düzelt (Code and Fix) Model:**

Bu model genellikle resmi olmayan bir ürün fikriyle başlar ve program ürün “hazır” olana kadar ya da gerekli zaman bitene kadar kodlama yapılarak devam eder. Öncesinde herhangi bir planlama yapılmasına gerek duyulmadığı için süreçler çok hızlı işler ve sonuca daha çabuk ulaşılabilir. Uzman görüşüne ihtiyaç duyulmadığı için neredeyse herkes yapabilir bu yönden elverişliyken, bitiş zamanı/süresi net olarak belli olmadığı için birtakım problemler ve yüksek maliyetler oluşabilir. Sonrasında yaşanabilecek sorunlarda geri dönüşler çok zorlu olabilir.

* **Prototipleme (Prototyping):**

Bu modellemede ilk iş olarak gereksinimler hızlıca toplanır. Geliştiriciler ve müşteri bir araya gelerek yazılımdan elde edilecek bütün çıktılara, bu çıktılar için gerekli girdilerin nasıl sağlanacağına, nasıl korunacağına ve hangi işlemlere uğrayacağına karar verirler. Sonrasında ise hızlıca yapılan bir tasarım ile yazılımın kullanıcıya yansıyacak yönünü aktaran bir​​ örnek​ ürün(prototip) ​üretilir. Bu prototip tekrar kullanıcıya sunulur ve geri dönüşler alınır. Böylelikle kullanıcının talep ettiği özelliklere sahip en yakın örnek elde edilmiş olur. Sınırlı sayıdaki yinelemelerden sonra, son yazılım paketi müşteriye verilir. Bu metodolojide yazılım ,müşteri ve geliştirici arasında periyodik bilgi ve görüş alışverişlerinden sonra sonuca ulaşılmış olunur.

* **KARŞILAŞTIRMA:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **No** | ​​​​ **Model/Özellik Çevik Modelleme** | **Kodla ve Düzelt** | **Çağlayan Modeli** | | **V Modeli** | |  |
|  |  |  |  | |  | |  |
| **1.** | Gereksinin Belirleme Belirli Sıklıkla | Başlangıç | Başlangıç | | Başlangıç | |  |
| **2.** | Maliyet Çok yüksek | Düşük | Maliyetli | | Maliyetli | |  |
| **3**. | Başarı Garantisi Çok yüksek | Düşük | Düşük | | Orta | |  |
| **4**. | Uzmanlık Gerekliliği Çok Yüksek | Düşük | Orta | | Orta | |  |
| **5**. | Fazların Örtüşmesi Evet | Hayır | Hayır | | Hayır | |  |
| **6.** | Maliyet Kontrolü Evet | Hayır | Evet | | Evet | |  |
| **7**. | Basitlik Karmaşık | Basit | Basit | | Orta | |  |
| **8**. | Risk Duyarlılığı Azaltılmış | Yüksek | Yüksek | | Yüksek Değil | |  |
| **9**. | Esneklik Çok Esnek | Katı | Katı | | Düşük​​ | |  |
| **10**. | Bakım Evet | Düşük | Düşük Bakımlı | | Düşük Bakımlı | |  |
| **11**. | Değişiklik​​ Yapma Zor | Kolay | Zor | | Zor | |  |
| **12** | Yeniden Kullanılabilirlik  Evet | Düşük İhtimal |  | Düşük İhtimal | Düşük |
| **13** | Dökümantasyon ve Eğitim Gerekliliği Evet | Evet (Zorunlu) |  | Evet | Evet |
| **14** | Zaman Kısa | Çok Uzun |  | Uzun | Uzun |
|  |  |  |  |  |  |
|  | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **No** | **​​​​ Model/Özellik** | **Prototipleme** | **Spiral Model** | **Formal Sistem** | **Yeniden Kullanıma Yönelik** |  | | **1** | Gereksinin Belirleme | Belirli Sıklıkla | Belirli Sıklıkla | Başlangıç | Başlangıç |  | | **2** | Maliyet | Düşük | Maliyetli | Yüksek | Düşük |  | | **3** | Başarı Garantisi | Orta | Yüksek | - | Çok Yüksek |  | | **4** | Uzmanlık Gerekliliği | Orta | Yüksek | Yüksek | Yüksek |  | | **5** | Fazların Örtüşmesi | Evet | Hayır | Hayır | Hayır |  | | **6** | Maliyet Kontrolü | Hayır | Evet | Evet | Hayır |  | | **7** | Basitlik | Orta | Karmaşık | Karmaşık | Karmaşık |  | | **8** | Risk Duyarlılığı | Düşük | Düşük | Düşük | Düşük |  | | **9** | Esneklik | Çok Esnek | Çok Esnek | Katı | Çok Esnek |  | | **10** | Bakım | Evet | Evet | Evet | Zor |  | | **11** | Değişiklik Yapma | Kolay | Kolay | Zor | Kolay |  | | **12** | Yeniden Kullanılabilirlik | Mümkün | Mümkün | Zor | Evet |  | | **13** | Dökümantasyon ve Eğitim Gerekliliği | Evet Ama Çok Değil | Evet Ama Çok Değil | Evet | Çok Sınırlı |  | | **14** | Zaman | Uzun | Uzun | Çok Uzun | Uzun |  | | **15** | Uygulama | Kolay | Karmaşık | Karmaşık | Karmaşık |  | |  |  | |  | |  |
|  | * SCRUM GÜNÜMÜZDE NEDEN POPÜLER?   Öncelikle scrum kavramından biraz bahsetmek gerekirse; Scrum bir analiz değildir, Scrum bir uygulamadır, daha doğrusu bir dizi uygulamadır. Bir scrum uygulaması yapılmadan önce, içersinde görev dağılımı yapılan bir ekip kurulmalıdır. Ekip üyelerinden bir miktar farklı olarak Scrum Master da bulunmalıdır. Bu ekipler her gün mutlaka, daily scrum yada stand-up diye adlandırılan ayaküstü,kısa süreli toplantılar yapar.Bu toplantılarda takımdaki herkes dün yaptıklarını, bugün ne yapacaklarını ve en önemlisi onları engelleyen şeyleri , çıkan problemleri anlatırlar. Bu durum Scrum masterın engelleri/problemleri tanımlamasının ana yollarından birisidir. Bu aktivite sayesinde, gelişmeler günü birlik takip edilebildiği için çözümlerde daha hızlı oluşturulur. Bu durum, zaman ve maddiyat açısından çok olumlu dönütler alınmasını sağlar. Scrum kullanmanın bir büyük olumlu tarafı ise, geliştirici ekibin müşteriyle iç içe olmasıdır. Her sprint başlangıcında bir araya gelinir ve o sprint'te yapılması gereken çalışmalara ortak karar verilir. O sprintin sonunda ise, müşteri (PO) ortaya çıkan çalışan yazılımı gözden geçirir ve geri bildirim sağlar. Eğer son sprintin başlangıcından bu yana projede bazı değişiklikler olması gerekiyorsa, bu değişiklikler bir sonraki sprint için planlananlara dahil edilebilir. Son olarak, Sadelik, netlik ve her şeyin planlı-programlı yapılması ekiplerin Scrum'ı kullanmaya başlamasını kolaylaştırır. Ve bence, bu yüzden Scrum Dünya üzerinde bu kadar popüler durumda.   * **Hangi Projede Hangi Modeli Kullanmalıyız?** |  |  | |  | |  |
|  | **Şelale Modeli:** |  |  | |  | |  |
|  | Proje yönetim modelleri genelde şelale modeli (waterfall model) ile başlar. Yazılım mühendisliğinde de ilk anlatılan modellerden birisi budur. Sonradan eklenti yapılmayacak/sabit projelerde kullanılabilir. Yeniliklere açık değildir. Örneğin inşaat sektörü.  **V Modeli:** Genellikle büyük projelerde kullanılır. Özellikle mimari sektör. |  |  | |  | |  |
|  | **Arttırımlı Model (Incremental Model):** Daha önce bahsettiğim gibi, bu model kullanılırken yeni gereksinimler sonucu ekstra eklentiler yapmamız mümkündür. Bu nedenle, e-ticaret yapan tüm firmalar, mobil aplikasyonu bulunan şirketler, bankalar vb. sık sık bazı eklemelere ihtiyaç duydukları için bu modeli kullanırlar.  **Kodla ve Düzelt (Code and Fix) Model:** Genellikle küçük projelerde kullanılır.  **Çevik Modelleme (Agile Model):** Agile modelleme işlemlerinin temel prensibi, büyük parçayı küçük parçalara bölüp daha hızlı bir zaman diliminde sonuca ulaşmaktır. Bu işlem de genellikle büyük şirketlerin karmaşık, uzun projelerinde kullanılır.  **KAYNAKLAR:**   * <http://ybsansiklopedi.com/wp-content/uploads/2015/08/Yaz%C4%B1l%C4%B1m-Geli%C5%9Ftirme-Modelleri-Yaz%C4%B1l%C4%B1m-Ya%C5%9Fam-D%C3%B6ng%C3%BCs%C3%BCSDLCYBS.pdf> * <https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/> * <https://medium.com/@omerharuncetin/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-543c7879a742> * <https://www.techwell.com/2013/02/why-scrum-so-popular> * <https://www.guru99.com/software-development-life-cycle-tutorial.html> * <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>   **BATIN ÇETİN**  **190601076** |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |
|  |  |  |  | |  | |  |