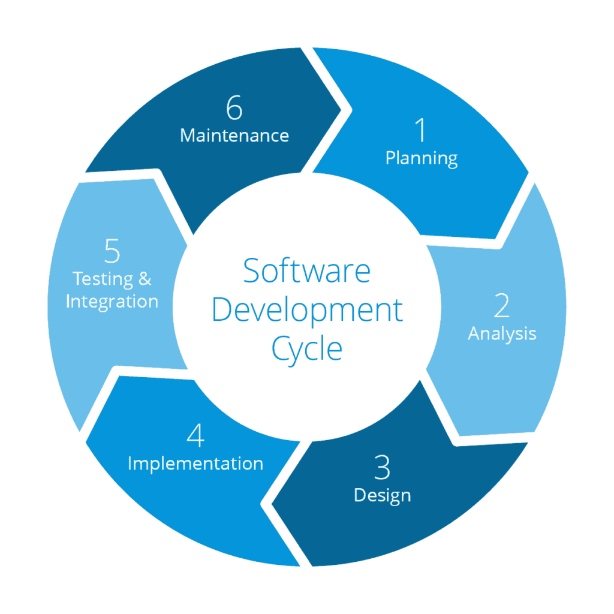
***Yazılım Yaşam Döngü Modelleri***

**1.Yazılım Yaşam Döngüsü Nedir?**

Amaçlanan yazılımın üretim, kullanım gibi kısımlarında yazılımın kalitesini arttıran, sistemli bir şekilde ilerlememizi sağlayan tüm aşamalara yazılım geliştirme yaşam döngüsü kısaca SDLC (“Software Development Life Cycle”) olarak adlandırılmaktadır. Bu süreç farklı aşamalardan oluşmaktadır. Kullanılan yazılım gereksinimleri sürekli olarak kendini yenilediği ve değiştiği için söz konusu aşamalarda mutlaka döngü biçiminde kullanılır. Döngü şeklinde kullanıldığı için geriye dönme ve tekrar ilerleme gibi durumlardan faydalanılır. SDLC aşamaları (Şekil 1) aşağıdaki şekilde sıralanabilir.

******

Şekil 1. Yazılım Yaşam Döngüsü Aşamaları

**1.1. Gereksinim Safhası (Requirement Phase)**

Planlama olarak da geçen bu aşama SDLC’nin ilk aşamasıdır. Bu aşamada bizden temel gereksinimler doğrultusunda ne istenildiğinin planlanması yapılır. İşin projelendirildiği, görev paylaşımı yapıldığı, düşüncelerin öne atıldığı ve tartışıldığı kısım da bu aşamada yapılmaktadır.

**1.2.Analiz Safhası (Analysis Phase)**

İlk safhada bulunan gereksinimlerin ve işlevlerinin en ayrıntılı bir şeklide incelendiği aşamadır. Projenin ne kadar süreceği, yazılımın eksileri ve artılarıyla belirlenir.

**1.3.Tasarım Safhası (Design Phase)**

Yapılan planlamaya ve analizlere göre bir tasarım çizilir. Bu tasarım müşteri gereksinim ve istekleri doğrultusunda ürünün özellikleri, kapasitesi vb. belirlenir. Gerçekleştirme aşamasına herhangi bir soru veya karar bırakılmaz. İki tür tasarım vardır: Yüksek/üst düzeyde tasarım- mimari tasarım ve Detaylı tasarım.

* **Yüksek/Üst Düzey Tasarım-Mimari Tasarım:** Yazılım parçalarının genel yapıları ve birbirleriyle etkileşimleri ile ilgilenir. Kısaca sistemin yapısını oluşturur. (Modüller, akış şemaları…)
* **Detaylı Tasarım:** Mimarı tasarımda kullanılan dokümanlar genelde yenilenir, düzeltilir. Kısaca yazılımı içeren bileşenler ve bunların ayrıntılarını oluşturur. (Veri yapıları, ekran tasarımları…)

**1.4.Gerçekleştirme Safhası (Implementation Phase)**

Müşteriye teslim edilecek olan ürünün programlama aşamasıdır. Kodlama (Coding) aşaması olarak da bilinen bu aşamada seçilen programlama diline bağlı olarak kod yazmaya başlanır. Yazılan kodlar yazılım geliştirme prensibi olan ***SOLID*** prensiplerine göre oluşturulmalıdır. Böylelikle kodun kalitesi, okunabilirliği ve bakımı daha anlaşılır hale gelmektedir.

* **SOLID yazılım prensipleri:** Robert C. Martin tarafından öne sürülen bu prensipler geliştirilecek olan yazılımın yeniden kullanılabilir, sürdürülebilir, esnek ve anlaşılır olmasını sağlar. İyi kod yazmak için bu prensiplere uymak gerekir.

***S*🡪 Tek sorumluluk Prensibi (Single Responsibility Principle):** Kısaca bir fonksiyonun/sınıfın (nesne) yalnızca bir işi yapıyor olmasıdır.

***O*🡪 Açık-Kapalı Prensip (Open-Closed Principle):** Kısaca bir sınıfın ya da fonksiyonun var olan özelliklerini koruyarak yeni özellik kazanabilmesidir.

***L*🡪 Liskov İkame Prensibi (Liskov Substitution Principle):** Kısaca yazılan kodda değişiklik yapmadan alt sınıfları, üst sınıfların yerine kullanabilmesidir.

***I*🡪 Arayüz Ayırma Prensibi (Interface Segregation Principle):** Kısaca tek bir arayüze gerektiğinden fazla yetenek veya özelleştirme verilmemesi onun yerine birden fazla arayüz oluşturulmasıdır.

***D*🡪 Bağımlılık Ters Çevirme Prensibi (Dependency Inversion Principle):** Kısacası sınıflar arası bağımlılığın minimum derecede olmasıdır. Özellikle üst seviye sınıflar alt seviye sınıflara bağımlı olmamalıdır.

**1.5.Test Safhası (Testing Phase)**

Kodlama süresince ve sonrasında yapılan önemli adımlardan biridir. Erken test yaklaşımı (early testing) ile hareket edip, ikinci yani analiz aşamasından itibaren bu bakış açısına sahip olmamız yapılabilecek hata oranını ve maliyetleri (para, zaman, prestij vb.) minimum seviyede tutacaktır. Çeşitli test seviyeleri vardır. Başlıca dört test seviyesi: Birim testi (unit testing), Entegrasyon testi (integration testing), Sistem testi (system testing) ve Kabul testidir (acceptance testing).

**1.6. Bakım Safhası (Maintenance Phase)**

Test aşamasından bittiğinde ve sistemde herhangi bir hata kalmadığında yapılan proje müşteriye, kullanıcıya ya da sahaya teslim edilir. Teslim ile birlikte bakım aşaması da başlamış olur. Hata giderme, yeni gereksinimler ekleme gibi işlemler bu aşamada yapılır.

**2. SDLC Yaklaşımının Avantajları**

* Risk düşürücüdür.
* Gerçek dünyaya yakındır.
* Modüler yaklaşım uygulanabilir.
* Sistematik olarak ilerler.
* Kalite yönetimi yapılabilir.
* Dinamik ortam için uygundur.
* Dinamik olduğu için sürekli yenilenebilir ve döngü içinde bir ortamdır.

**3. SDLC Yaklaşımının Dezavantajlar**

* Uzmanlık gerektirir. En az bir tane uzmanın olması projeyi hızlandırır.
* Yönetim ile ilgili tecrübeye ihtiyaç vardır.
* Genelde büyük sistemler için uygundur. Küçük projeler için zaman ve maliyet gibi durumlar açısından zarara sokabilir.
* Hızı arttırmak için genelde adım testleri atlanır.
* Hatanın devamlılığı engellenemez.
* Tasarımdaki hata diğer aşamalara da taşınmak zorundadır.

**4. Yazılım Yaşam Döngü Modelleri**

Yazılım yaşam döngüsünde birden çok model vardır. Bu modeller isteklere, uygulama alanlarına, projenin büyüklüğüne, kimler için kullanılacağına, maliyetine, zamanına vb. durumlarından dolayı birden çok model vardır. Bazılarını aşağıda verilmiştir.

**4.1. Kodla ve Düzelt (Code and Fix) Modeli**

Bu model genellikle resmi olmayan bir ürün fikriyle başlar. Ürün hazır olana kadar ya da zaman bitene kadar kod yazılmaya devam edilir. Çoğunluklar öğrenci projelerinde kullanılır.

**4.1.1. Avantajları:**

* Planlamaya ihtiyaç duyulmaz. Bu yüzden hızlı sonuca ulaşılır.
* Küçük ve kısa ömürlü projeler için uygundur.
* Uzman görüşüne ihtiyaç duyulmaz. Herkes tarafından kullanılabilir.

**4.1.2. Dezavantajları:**

* Kontrollü değildir. Kaynak planlaması yoktur.
* Bitiş süresi belli değildir.
* Kodlar esnek değildir, değiştirilmesi zordur.
* Hata düzeltme hem bulunması zor olur hem de maliyetli olabilir.
* Kodlar ihtiyaçları karşılamayabilir.

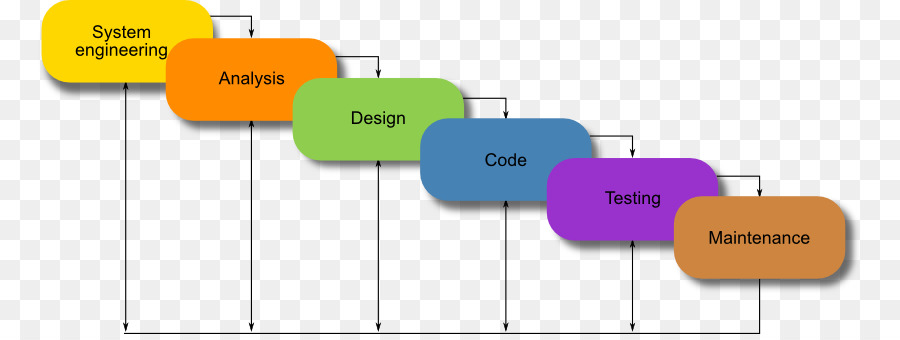
**4.2. Gelişigüzel Model**

1960’lı yıllarda uygulanan bu yöntem genellikle basit programlama içeren ve çoğunlukla tek kişi tarafından geliştirilen bir yöntemdir. Bu yöntemle oluşturulan uygulamaların çoğunda bakımı zor ve okunabilirliği düşüktür. Aslında bu yöntemi model olarak adlandırmak doğru değildir.

**4.3. Barok Modeli**

1970’li yıllarda kullanılan bu modelde ön planda tutulan kısım belgelemedir. Belgeleme model için ayrı bir süreç gibi işlenir. Yazılım yaşam döngü temel adımları doğrusal bir şeklide geliştirilir ve geri dönüş (döngü) yoktur. Günümüzde uygulanan bir model değildir.

**4.4.** **Çağlayan (Waterfall) Modeli**



Şekil 2. Çağlayan (Waterfall) Modeli

Geleneksel yazılım geliştirme modeli olarak da bilinen bu model geçmişte en popüler yazılım geliştirme modeli olmakla beraber en eski, en tanınmış ve en temel modeldir. Bu modelde aşamalar en az bir kez tekrarlanması ile geliştirilir. Adımlar incelendiğinde bir adım bitmeden bir sonrakine geçilemez. Her safhada dokümantasyon yazılmalıdır. Barok modeline göre belgeleme burada daha az bir süreci kapsar. Eğer herhangi bir safhada dokümantasyon ve test olmamışsa, o safhanın tamamlandığı kabul edilemez.

Çağlayan modelinin diğer modellere göre en önemli farkı, aşamalar arasında geri dönüşlerin daha başarılı gerçekleştirilmesidir. Çağlayan modeli daha çok iyi tanımlı ve üretimi kısa süren projeler için uygundur. Günümüzde kullanımı gittikçe azalma göstermektedir.

**4.4.1 Avantajları:**

* Yönetimi kolaydır.
* Birimlere bölünmüştür. İş bölümü ve iş planı en başından bellidir. Bu yönden proje yönetimini de kolaylaştırır.
* Kullanımı ve anlaması basittir.
* Bu model çok küçük ve gereksinimleri iyi anlaşılmış projelerde iyi çalışır.

**4.4.2.Dezavantajları:**

* Genelde yazılımın kullanıcıya ulaşma zamanı uzundur.
* Yanlışların düzeltilme ve eksiklerin giderilme maliyetleri oldukça yüksektir.
* Karmaşık ve nesne yönelimli projeler için uygun değildir.
* Kullanıcı memnuniyetini sağlamak çok zordur çünkü gelişime ve değişime açık değildir.
* Devam eden uzun projeler için zayıf kalır.
* Bir kısım bitmeden diğer kısma geçmek risklidir.

**4.5. V Süreç Modeli**

 Bu model şelale modeline benzer ancak test safhalarında daha üst düzeye çıkarılmış hali olarak düşünülmektedir. Uygulanabileceği projeler için belirsizliği az ve iş tanımlarının belirgin olması gerekir. Doğrusal ilerleme yerine, kodlama evresinden sonra yukarıya doğru çıkmaya başlar ve V şeklini oluşturur. Yatay ve dikey açılar zaman veya projenin tamamlanabilirliğini ve soyut seviyeyi gösterir. Bu modelde kullanıcının sistem gelişiminde katkısı vardır. V yazılım geliştirme süreci modeli doğrulama ve geçerlemeyi vurgulamaktadır.

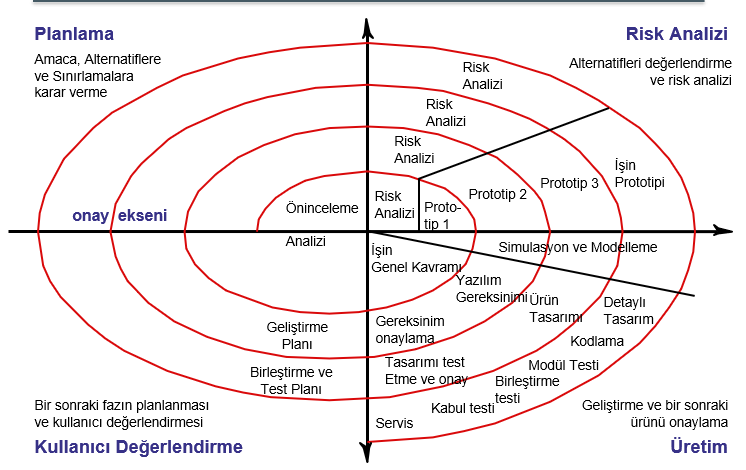
**4.5.1. Avantajları:**

* Proje yönetimi bakımından takibi kolaydır.
* Kullanımı kolaydır.
* Doğrulama​​ sadece son üründe değil tüm teslim edilebilir ürünlerde uygulanır.

**4.5.2. Dezavantajları:**

* Aşamalarda tekrar yoktur.
* Risk çözümleme içermez.
* Aynı zamanda gerçekleştirilebilecek olaylara kolay olanak tanımaz.
* Süreç içerisinde gereksinimler değişebilir.

**4.6. Helezonik (Spiral) Model**



Şekil 3. Helezonik (Spiral) Model

Yukarıdaki Şekil 3’ten de anlaşılacağı gibi sarmal model aynı safhalara geri dönülmesini zorunlu kılar, şelale modelinde yok sayılan riskleri göz önünde bulundurur. Her döngü öncesinde içerinde bulunan bölümün risk analizini yapar. Bu bakımdan hataları erken giderme imkânı tanıyabilir. Özellikle güvenlik yazılımlarının oluşumunda bu model kullanılır. Çağdaş modellere oldukça yakındır. Dört temel aşamadan oluşur:

1. **Planlama:** Üretilecek olan ara ürün için plan yapma, amaç belirleme, önceki adımla bütünleşme gibi işlemler yapılır.
2. **Risk Analizi:** Riskler araştırılır ve belirlenir.
3. **Üretim:** Ürün/ara ürün üretilir.
4. **Kullanıcı Değerlendirmesi**: Kullanıcının ara ürün hakkında yaptığı değerlendirilir ve sınanır.

Risk analizi ön plana çıkmıştır. Her döngü bir fazı ifade eder.

**4.6.1. Avantajları:**

* Ürün/ara ürünün kullanıcı tarafından sınanmasını sağlar. Bu da ileride oluşacak istenmeyen durumları engeller.
* Kullanıcı sistemi daha erken görebilmektedir.
* Birçok yazılım modelini içinde barındırır.
* Hataları erken giderme şansı tanır.
* Parçalara böler, önce en risk taşıyan parçayı çözer.
* Yazılımın kodlanması ve sınanması daha erken başlar.

**4.6.2. Dezavantajları:**

* Düşük bütçe ve küçük projeler için pahalı bir sistemdir.
* Kompleks bir yapıya sahiptir.
* Fazla belgeleme oluşturur.
* Spiral sonsuza gitme ihtimali vardır.
* Kontrat tabanlı yazılma uymaz.

**4.7. Artımsal Geliştirme Süreç Modeli**

Bu model belli bir zamana bağlı olarak yazılımı parça parça geliştirip teslim etmeye dayanır. Yani bir taraftan teslim/kullanım olurken diğer taraftan da üretim yapılır. Her teslim istenilen işlevselliğin bir bölümünü kapsar. Öncelikli olarak en önemli gereksinimleri karşılayan çekirdek bir sistem geliştirilir. Kullanıcının gereksinimleri ön plandadır ve ilk olarak bu gereksinimlerin erken teslim edilmesi amaçlanır. Bir parçanın geliştirilmesi başlandığında diğer gereksinimlere ara verilir. Uzun süreli projeler için bu model uygundur.

**4.7.1. Avantajları:**

* Projenin başarısız olma riskini azaltır.
* Daha fazla test/sınama işlemleri uygulanır.
* Böl-yönet tarzında yaklaşımı vardır.
* Gereksinimler kullanıcıyla/müşteriyle belirlenir.

**4.7.2. Dezavantajları:**

* Deneyimli çalışana ihtiyaç duyulur.
* Artımların tanımlanması için tüm sistemin iyi bir şekilde tanımlanması gerekir.
* Artımlar kendi içlerinde tekrarlanamaz.
* Artımları doğru boyutta yapmak bazı zamanlar zorlukla gerçekleşmektedir.

**4.8. Prototipleme (Prototyping)**

Bu modelde isminden de anlaşılacağı gibi prototip üretip geliştirmektir. Geliştiriciler ve kullanıcılar/müşteriler aynı masa etrafında bulunarak proje hakkında bütün detayları (örneğin girdi-çıktılar, gider-gelirler, korunumu vb.) konuşurlar. Gereksinimler hızlıca toplanır, hızlıca tasarım yapılır ve ilk prototip üretilir. Kısaca bu modelde hızlı plan, hızlı tasarım ve hızlı kodlama ön plandadır.

**4.8.1. Avantajları**

* Karmaşa ve yanlış anlamalar önlenebilmektedir.
* Risk kontrolü sağlanmaktadır.
* Yeni gereksinimlere açıktır.
* Kullanıcı sistem gereksinimlerini görebilmektedir.

**4.8.2. Dezavantajları**

* Belgelendirme yoktur.
* Prototip hedefleri net değilse güvenlik açığına neden olmaktadır.
* Düzeltme aşaması atlanırsa düşük performansa yol açmaktadır.

**4.9. Çevik Model**

Çevik yazılım (Agile Development) geliştirme metotları hızlı ürün çıkarabilme, değişikliğe hızlı tepki verebilme gibi durumları amaçlamaktadır. Bu metotlar yüksek verim, esnek, hızlı ve maliyeti az çözümler sağlamaktadırlar. Projenin boyutuna bakılmaksızın proje küçük yinelemelere ayrılır ve her ayrılan yineleme ayrı bir projeymiş gibi ele alınır. Yaklaşık olarak 2-4 hafta arası sürmesi planlanır. Bu yinelemeler sonucu kullanıcıya/müşteriye bilgi verilir. Hızlı olma sebebi ekip üyelerinin sürekli iletişim halinde olmasından dolayıdır. Geleneksel modellemelerden farklı olmasının nedeni metotların daha esnek ve kullanışlı olmasından kaynaklanır. Bu metotlar;

* Süreçler ve Araçlar yerine **Bireyler ve Etkileşimler**,
* Kapsamlı Belgeler yerine **Çalışan Yazılım**,
* Sözleşme Görüşmeleri yerine **Müşteri İlişkileri**,
* Plan İzleme yerine **Değişikliğe Açıklığın**, şeklinde gösterilebilir.

En yaygın uygulanan çevik metodolojiler şunlardır; Extreme Programming (XP), SCRUM, Agile Unified Process, Feature-Driven Development (FDD), Test-Driven Development (TDD), LEAN Development, Dynamic System Development, Methodology (DSDM), Microsoft Solution Framework (MSF).

**4.9.1. Avantajları:**

* Değişime açıklık ve esneklik en üst düzeydedir.
* Kısa döngülerden dolayı ekip üyelerinin de motivasyonu yüksektir. Bu da verimliliği büyük bir ölçüde arttırır.
* Doğal insan eğilimine yatkın olmasından dolayı fazla eğitim gerektirmez ve hızlı bir şekilde adapte olunabilmektedir.
* Kısa sürede kullanıcı/müşteri memnuniyeti sağlanır.
* Yazılım kalitesi yükselir, maliyet azalır, üretkenlik ve başarı oranında da artışı sağlar.

**4.9.2. Dezavantajları:**

* Ekip üzerinde hedef baskısı oluşturmaktadır.
* Sürekli değişen ihtiyaçlardan dolayı aşırı çalışma durumu görülmektedir.
* Kurumsal yapıda uygulanması zordur.

**5. XP ve SCRUM**

**5.1. Extreme Programming (XP)**

XP grup içi iletişime önem verir. Geri dönüşlerin daha fazla olmasına imkân sağlar. Daha çok küçük ve orta ölçekli projelerde kullanılırlar. Dört temel değer vardır: İletişim, Basitlik, Geri Bildirim ve Cesaret.

1. **Cesaret**: En zor değer olan cesaret, karşılaşılan zorluklarda pes etmemeyi, yılmadan ilerlenmesini, zorlukların üzerine gitmeyi sağlar. Başarısızlıktan korkan ekiplerde proje hızında ciddi düşüşler görünmektedir.
2. **İletişim**: Çoğu projelerde önemli sorunlardan biri iletişimdir. XP ekibin sağlıklı bir şekilde iletişim kurmasını, projenin aksamasını önlemede rol oynar. Yüz yüze ve devamlı iletişim bu konuda ön plandadır.
3. **Basitlik**: XP’de karmaşık çözümler kullanılmamaktadır. O anda hangi yöntem en basit çözümse o kullanılır.
4. **Geri Bildirim**: Müşterinin/kullanıcının da belli zaman aralıklarında ekip üyeleriyle buluşup o anki gelinen seviyeye kadar birlikte çalıştırırlar. Bu çalıştırma sonrasında geri bildirimler oluşmaktadır. İleride ortaya çıkabilecek sorunlar geri bildirim sayesinde önemli bir şekilde ortadan kaldırılır.

**5.2. SCRUM**

Scrum bir proje yönetim yaklaşımıdır. Yani her alanda uygulanabilir. Bu yaklaşımda kompleks yazılım işlerini küçük birimlere (sprint) bölerek, her birimleri de birer birer geliştirir. Bu metodolojide bir yineleme 30 günden fazladır ve günlük 15-30 dakika olarak kısa toplantılar (Scrum Daily Meeting) yapılır. Scrum’da üç temel kavram vardır. Bunlar: Roller, Toplantılar ve Bileşenler/Araçlar’dır.

**1)Roller (Roles)**

1. **Ürün Sahibi (Product Owner):** Bu kişi illa kullanıcının/müşterinin kendisi olmak zorunda değildir. Müşteri Temsilcisi ya da Scrum ekibinden biri olabilir. Önemli olan bu kişinin istenilen gereksinimleri kavrayabilmesi ve müşteriyle ilişki kurabilmesidir.
2. **Scrum Yöneticisi (Scrum Master):** Ekibin temel değerlere bağlı kalmasını sağlar. Ekibi ve organizasyonu Scrum’a uyum sağlatır böylece üretkenliğin artmasına yardımcı olur.
3. **Scrum Takımı (Scrum Team):** Sürekli iletişim halinde olan ve programlama yapan gruptur. 5-10 kişilik üyelerden oluşur. Kendi içlerinde görevleri farklılık göstermektedir.

**2) Toplantılar (Meetings)**

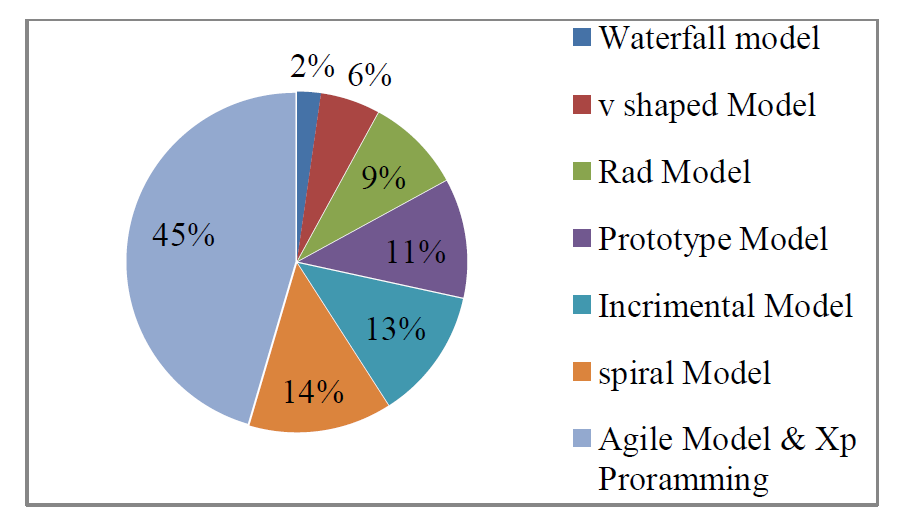
Scrum’ın olmazsa olmazıdırlar. Her gün olan bu toplantıda önceki gün neler yapıldığı, karşılaşılan sorunları ve gün içerisinde neler yapılacakları hakkında konuşurlar. Her birim(sprint) için de bir toplantı yapılmaktadır.

**3) Bileşenler/Araçlar (Artifacts)**

1. **Ürün Gereksinim Dokümanı (Product Backlog):** Proje süresince gerekli iş elemanlarının bir listesidir. Bu liste dinamik bir yapıdadır. Sürekli ekleme-çıkarma küçük alt birimlere bölme gibi birçok işlemden geçebilir. Ürün gereksinim listesi, genellikle kullanıcı hikayelerinden (User Story) oluşur ve kullanıcı bakış açısından bakılır.
2. **Sprint (Koşu) Dokümanı (Sprint Backlog):** Product Backlog’dan elde edilen iş ve görevleri içerir. Buradaki amaç sprint sonunda çalışabilir bir parça elde etmektir. Bu sprint sadece ekip tarafından değiştirilebilir.
3. **Sprint Kalan Zaman Grafiği (Burndown Chart):** Bu grafik yineleme boyunca nereye gelindiği ve nereye gelinmesi gerektiğini karşılaştıran bir grafiktir.

**6.SONUÇ**

Yazılım süreç modelleri birbirinden ayrı değildir. Hatta çoğu iç içedir. Yazılım süreç modelleri geliştirilecek yazılım, kullanım alanına ve türüne göre seçilir. Seçim için modelde oluşabilecek risklerin değerlendirilmesi, proje büyüklüğü, proje kompleksi (karmaşıklığı), gerçekleştirecek olan kurum yapısı, maliyet, zaman vb. gibi kriterler kesinlikle göz önünde bulundurulmalı ve dikkat edilmelidir. Günümüzde en çok kullanılan/popüler olan model Çevik Modeldir (Şekil 4). Tercih edilme nedeni ise kısa bir zaman diliminde esnek bir şekilde geliştirilmesi, büyük oranda başarılı sonuçlanan bir model olmasından dolayı düşünülebilir.



Şekil 4. Yazılım Yaşam Döngü Modellerinin kullanım oranları

***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> Erişim Tarihi: 26.03.2021
* <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> Erişim Tarihi: 26.03.2021
* <https://gokhana.medium.com/solid-nedir-solid-yaz%C4%B1l%C4%B1m-prensipleri-nelerdir-40fb9450408e> Erişim Tarihi: 26.03.2021
* <https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/> Erişim Tarihi: 26.03.2021
* <https://furkanalniak.com/yazilim-muhendisligi-yazilim-surec-modelleri/> Erişim Tarihi: 26.03.2021
* <https://medium.com/@omerharuncetin/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-543c7879a742> Erişim Tarihi: 26.03.2021
* <https://hayririzacimen.medium.com/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-ve-s%C3%BCre%C3%A7-modelleri-70fdfb2f8f77> Erişim Tarihi: 26.03.2021
* <https://medium.com/@batincetin44/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-2342547d0840> Erişim Tarihi: 26.03.2021
* Kılınç, D.,(2021), “Yazılım Mühendisliği Temelleri Ders Notları”, Bakırçay Üniversitesi.