**Yazılım Geliştirme Yaşam Döngüsü ve Modelleri**



**Yazılım Geliştirme Yaşam Döngüsü (Software Development Life Cycle – SDLC) Nedir?**

Yazılım Geliştirme Yaşam Döngüsü, nitelikli yazılım ürünlerinin oluşturulmasında ve takip sürecinde kullanılan, tekrar eden aşamalardır. Bu aşamalar ürünün geliştiricilerine planlı bir yol haritası çizerek zamanın ve maliyetin yönetiminde önemli bir rol oynar. Temel yazılım geliştirme aşamaları ise şunlardır:

* **Planlama:** Yazılım yaşam döngüsünün ilk adımıdır. Zaman ve maliyet hesaplamaları yapılır. Uygunluk (fizibilite) raporu hazırlanır.
* **Analiz:** Yazılımın gereksinimleri tanımlanarak belgelenir.
* **Tasarım:** Yazılıma genel bir çerçeve çizilir. Tüm bileşenler ayrıntılı olarak modellenir. Tasarım aşaması, döngünün en önemli adımlarından biridir. Çünkü burada yapılacak hatalar ciddi sorunlara (maliyet aşımları veya projenin tamamen çökmesi gibi) yol açabilir.
* **Uygulama:** Şimdiye kadar planlanan her şeyin yazılıma dönüştüğü aşamadır. Öncesinde belirlenen standartlara göre kodlama işlemi yapılır.
* **Test:** Yazılımın teslim edilmeden önceki son kontrolleri ve düzeltmeleri yapılır. Uygulama aşaması ile eş zamanlı olarak da gerçekleştirilebilir.
* **Teslim ve Bakım:** Testlerden geçen ürün son aşamada teslim edilir. Test sırasında bulunamayan hatalar ve ürünün güncel tutulması için bakım süreci başlar.

**Yazılım Geliştirme Yaşam Döngüsü Modelleri**

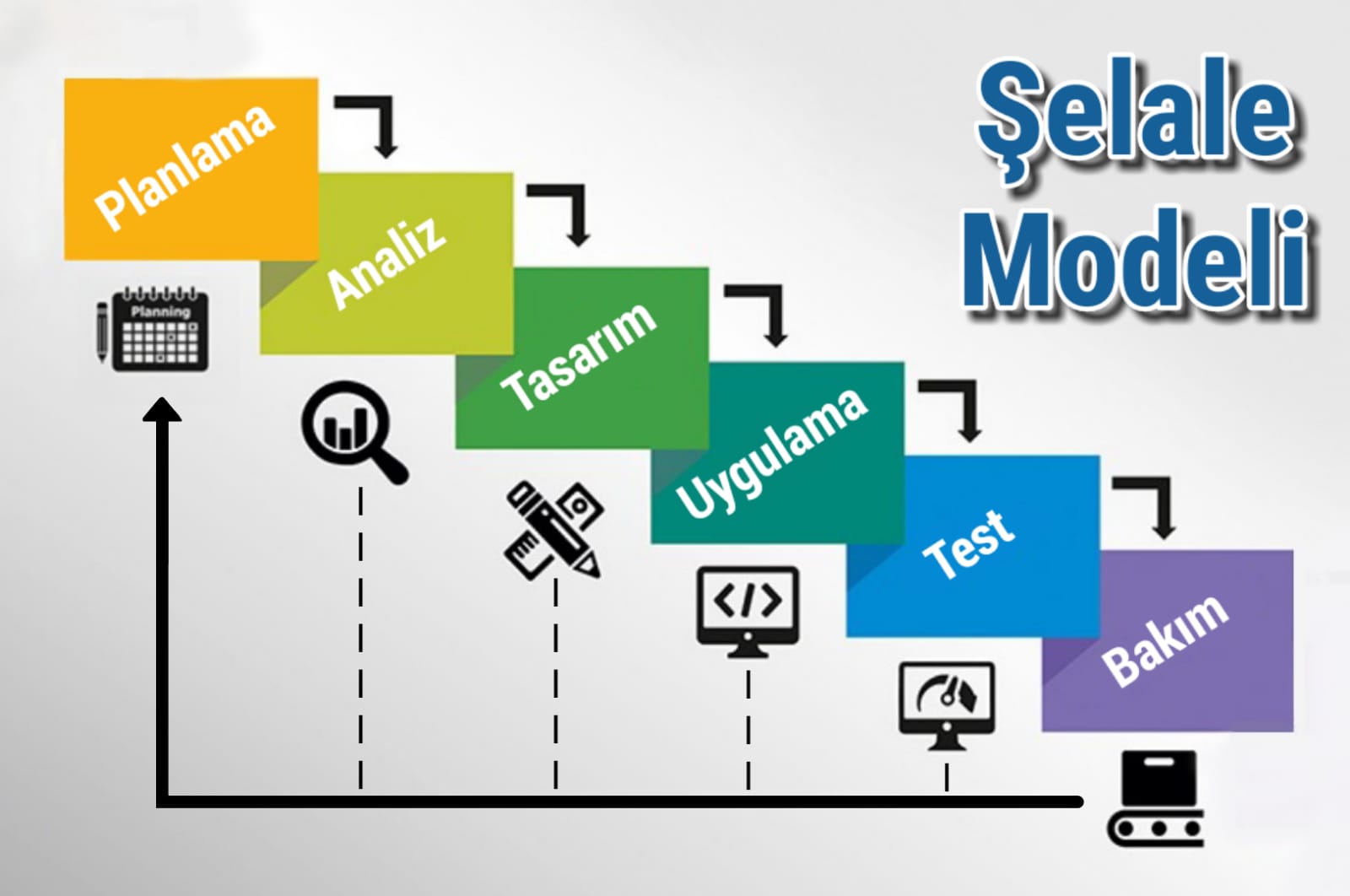
**1. Gelişigüzel Model**

Bu modelde herhangi bir yöntem söz konusu değildir, tamamen geliştiren kişiye bağlıdır. Bundan dolayı da bakımı ve takibi çok zordur. 1960’lı yıllarda; tek kişilik, basit yazılımlarda kullanılmıştır. Günümüzde geçerliliği yoktur.

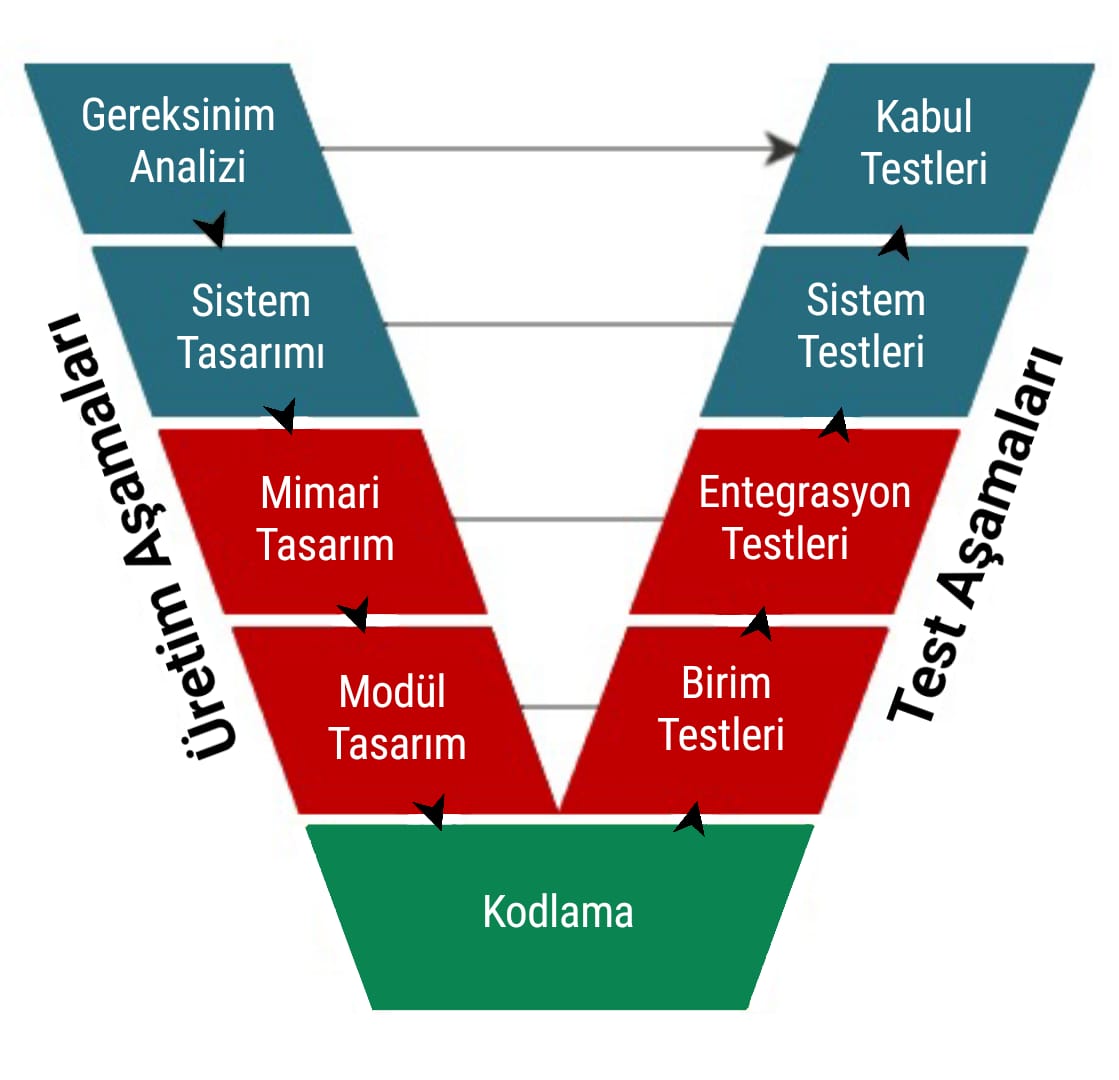
**2. Barok Modeli**

Temel yazılım geliştirme aşamalarının doğrusal olarak uygulandığı modeldir. Adımlar arasında geri dönüş yoktur. Belgeleme süreci ayrı bir aşama olarak işlenir. Uygulama evresine ağırlık veren bir model olduğundan dolayı günümüzde kullanımı önerilmemektedir.

**3. Şelale Modeli (Waterfall Model)**

Şelale modeli, yazılım geliştirme için kullanılan en eski modeldir. Diğer tüm modellerin temelini oluşturur diyebiliriz.

* Bu modelde süreç en az bir kez tekrar edilir. Statik bir yapıya sahiptir. Bir aşama tamamlanmadan sonraki aşamaya geçilmez. Her aşamanın sonunda belgeleme yapılması şarttır. Beklenmeyen kötü sonuçlarla karşılaşmamak adına, analiz sürecinin titizlikle yürütülmesi gerekir. Tasarımın da gereksinimler doğrultusunda, yazılımı tam anlamıyla yansıtacak şekilde hazırlanması önemlidir.
* Kolay uygulanabilir olması, belgeleme sürecinin zorunlu tutularak genele yayılması ve özenli çalışma gerektirmesi, şelale modelinin olumlu yönleri içinde sayılabilir. Kısa zaman alan, küçük projeler için uygun bir döngü modelidir.
* Yüksek risk ve belirsizliğin olması, değişen gereksinimleri karşılayacak bir yapıya sahip olmaması, analizi iyi yapılmamış projelerin ileriki aşamalarında çıkacak sorunların maliyetin ciddi anlamda yükseltecek olması, şelale modelinin dezavantajları arasında sayılabilir. Ayrıca, analiz ve tasarım aşamalarına daha fazla yoğunluk verilmesi; bir an önce uygulama, çalıştırma ve sonucu görme eğiliminde olan yazılım ekiplerinde mutsuzluğa da yol açmaktadır. Karmaşık ve uzun süren projeler için uygun bir döngü modeli değildir.

**4****. V Modeli**

V Modeli, her geliştirme aşamasına karşılık bir test aşaması olan yazılım geliştirme yaşam döngüsü modelidir. Şelale modeliyle arasında çokça benzerlik bulunan bu yaklaşım, adını aşamaların “V” şeklinde uygulanmasından alır.

* Bu model iki sıralı süreçten meydana gelir. Şeklin sol tarafı üretim aşamalarını, sağ tarafı ise test aşamalarını temsil eder. Kodlama aşaması, bu iki tarafı birleştiren adımdır. Dinamik olmayan bir yapıya sahiptir. Bu yüzden gereksinimlerin ayrıntılı bir şekilde tanımlanmış, belgelenmiş ve proje ekibi tarafından iyi anlaşılmış olması gerekir. Şelale modelinde olduğu gibi burada da sonraki aşama ancak bir önceki aşama tamamlandıktan sonra başlar. Şimdi bu aşamalara kısaca bir göz atalım;

**Üretim Aşamaları**

1. **Gereksinim Analizi:** V döngü modelinin ilk aşamasıdır. Müşteri ile iletişim kurularak gereksinimler belirlenir. Döngünün en önemli aşamalarından biridir.
2. **Sistem Tasarımı:** Ayrıntılı olarak belirlenmiş gereksinimler doğrultusunda, bütün sistemin tasarımı gerçekleştirilir.
3. **Mimari Tasarım:** Sistem, farklı görevlerdeki modüllere ayrılır. Bu aşama yüksek seviye tasarım olarak da adlandırılır.
4. **Modül Tasarım:** Sistem, kodlama için daha küçük ve ayrıntılı birimlere ayrılır. Bu aşama düşük seviye tasarım olarak da adlandırılır.

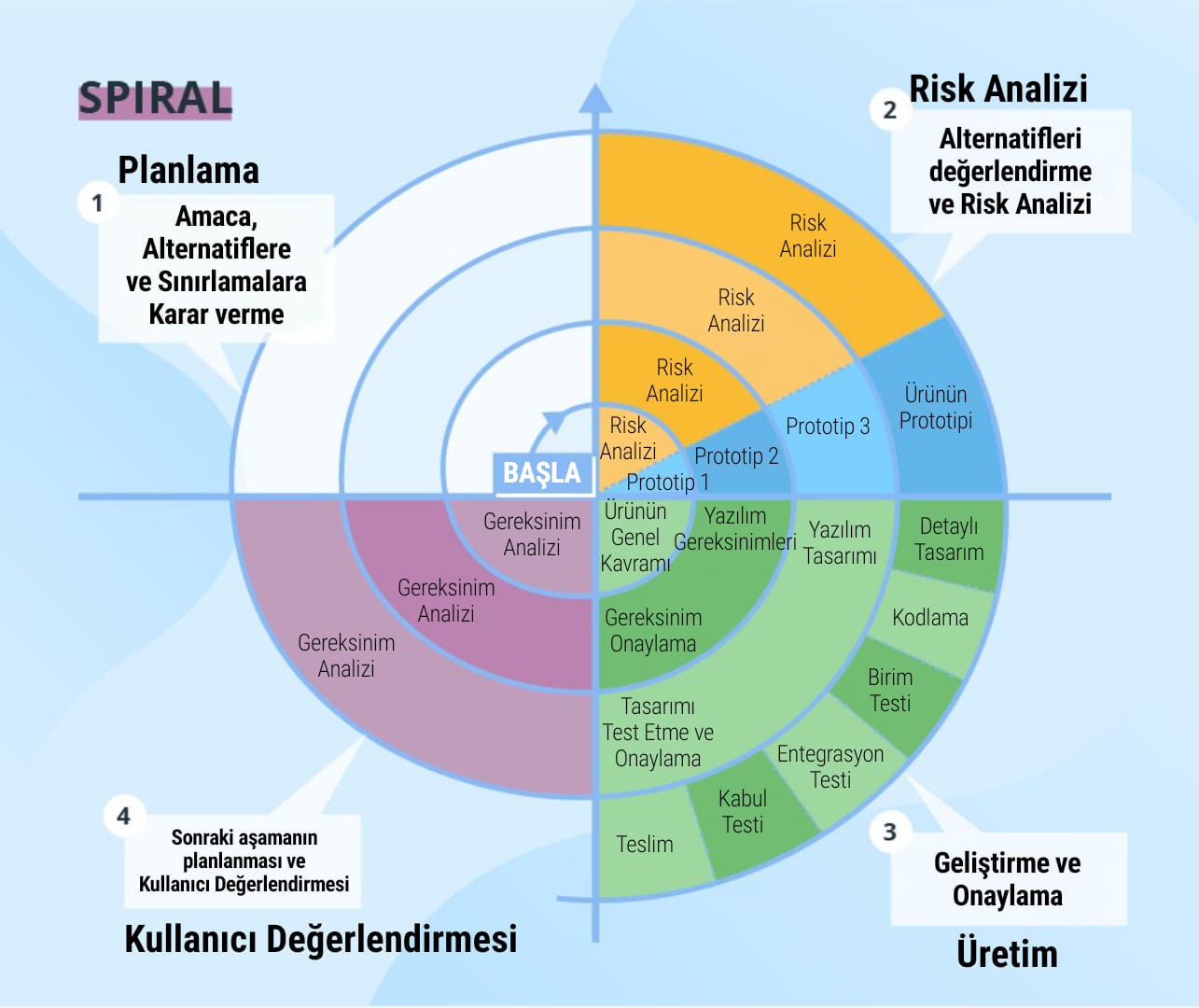
**Kodlama Aşaması**

Bu aşamada; modül tasarım evresinde ayrılan birimlerin, gereksinimlere göre seçilen en uygun programlama dili ile kodlaması yapılır. Kodlama aşaması sona erdikten sonra, artık ürünün test süreci başlar.

**Test Aşamaları**

1. **Birim Testleri:** Test aşamalarının ilk adımıdır. Kod düzeyinde yapılır, böylece hataların erken tespit edilmesi sağlanır. Ancak her hata birim testleriyle belirlenemez.
2. **Entegrasyon Testleri:** Bu aşama, mimari tasarım aşaması ile ilişkilidir. Farklı işlevlerdeki modüllerin ilişkisini test etmek için yapılır.
3. **Sistem Testleri:** Sistemin işlevselliğini ve diğer sistemlerle olan iletişimini test etmek için yapılır. Yazılım ve donanım arasındaki uyumluluk sorunlarının birçoğu, bu test sırasında ortaya çıkar.
4. **Kabul Testleri:** Yazılım ürününün kullanıcı ortamında test edilmesidir. Diğer sistemlerle olan uyumluluk, genel performans sorunları kabul testleri aşamasında ortaya çıkabilir.

* V Modelinin avantajları; kullanımı, yönetimi ve takibinin kolay olmasıdır. Her aşamanın belirli çıktıları ve inceleme süreci vardır. Aşamalar birer birer tanımlanır, bu yüzden de disiplinli bir yapıya sahiptir. Gereksinim analizlerinin iyi yapıldığı, küçük projeler için uygun bir döngü modelidir.
* V Modelinin dezavantajları; statik bir yapıya sahip olması, yüksek risk ve belirsizlik barındırmasıdır. Gereksinim analizlerinin iyi yapılmadığı projelerde, test aşamasında çıkan sorunlara çözüm üretmek çok zordur; geriye dönüş, maliyeti ciddi oranda arttırır. Şelale modelinde de olduğu gibi son ana kadar çalışan bir yazılım üretilememesi, proje ekiplerinde mutsuzluğa yol açar. Karmaşık ve uzun süreli projeler için uygun bir döngü modeli değildir.

**5. Helezonik Model (Spiral Model)**

Helezonik Model, döngüyü oluşturan aşamalardan tekrar tekrar geçilmesini ve her geçişte projenin ilerleme kat etmesini hedefleyen SDLC yaklaşımıdır.

* Bu modelde aşamalar spiral şeklinde tekrar ederek ilerlenir. Her tekrarda risk analizleri yapılır ve bir prototip elde edilir. Bu prototipler kullanıcıya sunularak, kullanıcının başından itibaren yazılımı takip etmesi amaçlanır. Böylece ileriki aşamalarda çıkabilecek sorunlara erkenden çözüm üretmek mümkün olur. Helezonik model, 4 ayrı bölümde incelenir:

ο **Planlama:** Üretilecek prototipe yönelik planlamalar yapılır ve amaç belirlenir.

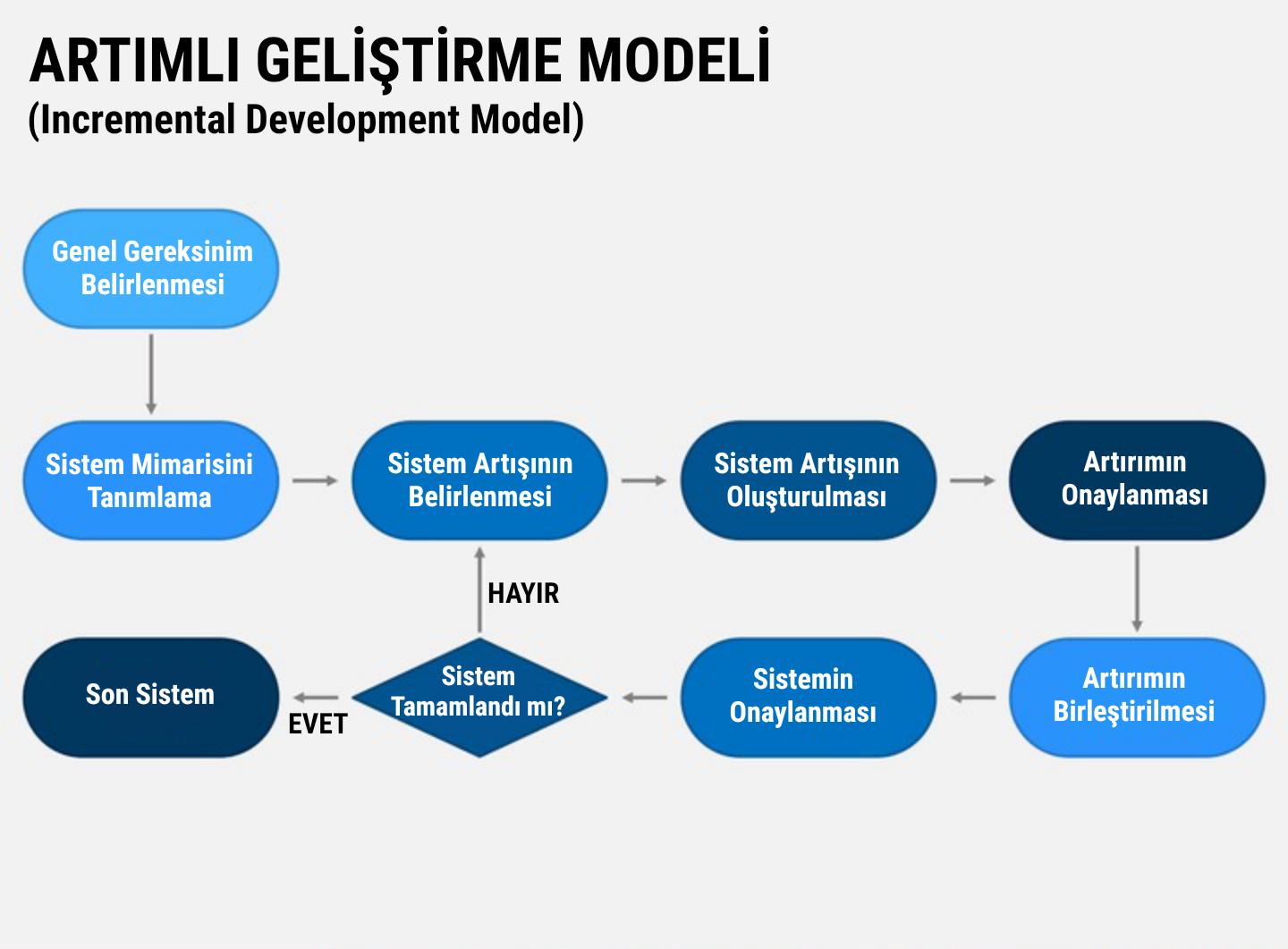
ο **Risk Analizi:** Risklerin araştırılması, değerlendirilmesi yapılır.

ο **Üretim:** Prototipin geliştirilmesi yapılır. Ayrıca bu aşama, son ürüne yapılacak testleri de kapsar.

ο **Kullanıcı Değerlendirmesi:** Prototipin kullanıcı nezdinde değerlendirildiği ve geri dönüşlerin alındığı aşamadır. Bitmemiş projeler için analizler yapılır ve yeni bir döngüye başlanır.

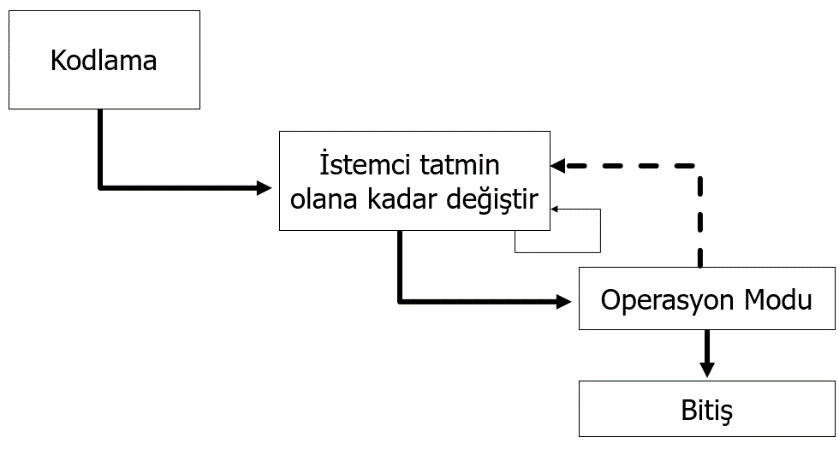
* Helezonik modelin avantajlı yönleri; esnek bir yapıya sahip olması, değişen gereksinimleri sisteme entegre etmenin kolay olmasıdır. Kullanıcıyı ilk andan itibaren işin içine katarak sistemi erkenden görmesi sağlanır. Geliştirme küçük parçalara bölünerek daha iyi bir risk yönetimi oluşturulabilir. Çağdaş modellere son derece yakındır. Karmaşık ve uzun süreli projeler için uygun bir modeldir.
* Helezonik modelin dezavantajlı yönleri; sürecin karmaşık ve yönetiminin zor olmasıdır. Ayrıca proje planlandığından daha uzun sürebilir. Çok sayıda ara aşama olduğundan aşırı belgeleme gerektirir. Küçük ve düşük riskli projeler için pahalı bir modeldir ve uygun değildir.

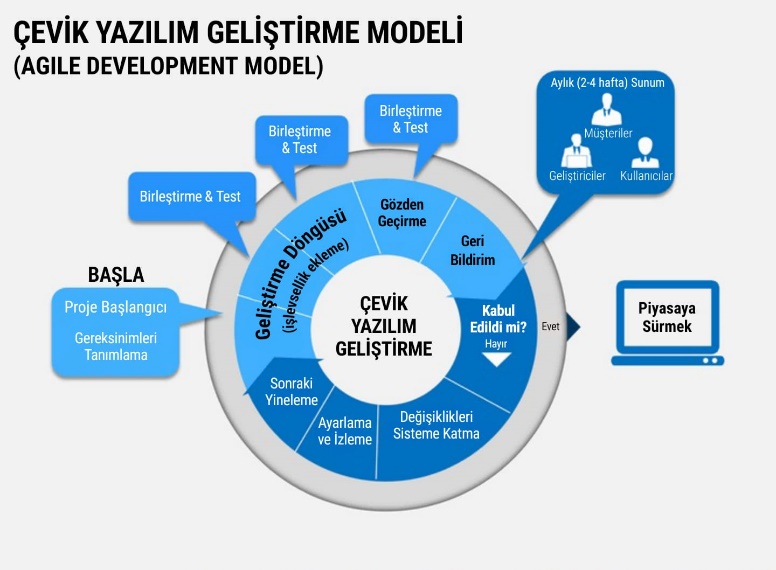
**6. Artımlı Geliştirme Modeli (Incremental Development Model)**

Artımlı Geliştirme Modeli, yazılım projesinin parça parça geliştirildiği ve teslim edildiği modeldir. Her teslim, gereksinimlerin bir kısmını karşılar.

* Bu modelde, kullanıcı gereksinimleri önceliklerine göre sıralanır. Öncelikli gereksinimler erken teslimlere dahil edilir. Gereksinimlerde olacak değişiklikler sonraki teslimlere aktarılır. Üretilen her versiyon öncekileri kapsayacak şekilde, işlevleri artırılarak teslim edilir. Bir taraftan üretim yapılırken diğer taraftan kullanıma devam edilir.
* Böl ve yönet yaklaşımı benimsendiğinden uygulanması kolaydır. Gereksinimlerin belirlenmesinde kullanıcı (müşteri) aktif rol oynar. En önemli gereksinimler erken teslim edildiği için bu özellikler daha fazla test edilmiş olur. Ayrıca, ilk önce teslim edilen ürünler prototip görevi görerek gereksinimlerin daha iyi anlaşılmasını sağlar. Uzun zaman alabilecek projeler için uygun bir modeldir.
* Artımlı Geliştirme Modeli, deneyimli personeller gerektirir. Artımları tanımlamak için tüm sistemin tanımlanmasına ihtiyaç vardır. Kısa ve küçük projeler için, maliyeti çok yükselteceğinden dolayı, uygun bir model değildir.

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

Kodla ve Düzelt Modeli, ağırlığın uygulama aşamasına verildiği, kodlama ve çıkan sorunları düzeltme yaklaşımının benimsendiği bir SDLC modelidir.

* Bu modelde, öncelikle yazılım ürününün ilk sürümü geliştirilir. Daha sonra, sistem istenilen şekle gelinceye kadar devamlı kodlama yapılır. Planlama, analiz ve tasarım aşamaları yoktur, direkt olarak uygulama aşamasına geçilir. Bakım aşaması vardır, ancak belgeleme süreci olmadığı için yapımı çok zor ve maliyetlidir. Ayrıca, emeklilik aşaması da vardır.
* Kodla ve düzelt modelinin avantajı, en kolay uygulanabilir yazılım yaşam döngü modeli olmasıdır. Programlama aşamasına çabuk geçilir. Uzman fikrine ihtiyaç yoktur, herkes bu modeli kullanabilir. Çok küçük ve kısa projeler için uygun olabilir.
* ****Kodla ve düzelt modelinin dezavantajları; planlamanın ve belgelemenin olmamasıdır. Bitiş süresi belli değildir. Esnek olmayan bir yapıdadır, değiştirilmesi güçtür. Hataların bulunması ve doğrulaması çok zordur. Her ne kadar kolay uygulanabilir olsa da en pahalı döngü modelidir. Hataları düzeltmek yüksek maliyetlidir.

**8. Çevik Modeller (Agile Models)**

20. yüzyılın sonlarına doğru tüm dünyada bir yazılım krizi yaşanmıştır. Kalitesiz, verimsiz, gereksinimleri karşılamayan, zamanında teslim edilemeyen, bütçeyi aşan, bakımı zor yazılımlar çoğalmıştır. İşte bu sorunların aşılması amacıyla, 1990’lı yılların sonlarında “çevik (agile)” olarak adlandırılan yaklaşımlar geliştirilmiştir.

* Çevik modellerde, yazılım projesi 2-4 haftalık yinelemelere (iteration) ayrılır. Her yinelemede döngü aşamaları tekrarlanır. Yinelemelerin sonunda müşteriye çalışan bir yazılım sunulur ve ürünün hangi aşamada olduğuna dair bilgilendirme yapılır.

**Çevik Yazılım Geliştirme Manifestosu**

Kent Beck ve 16 arkadaşı tarafından, daha iyi yazılım geliştirme yollarını uygulamak ve uygulanmasını sağlamak amacıyla 2001 yılında yayınlanan bildiridir. Bildiri 4 ana madde ve 12 temel prensipten oluşmaktadır. Bu 4 ana madde bireyler ve etkileşimlerin, süreçler ve araçlardan; çalışan yazılımın, kapsamlı dokümantasyondan; müşteri ile iş birliğinin, sözleşme pazarlıklarından; değişime karşılık vermenin, bir plana bağlı kalmaktan daha önemli olduğunu söyler. Çevik yazılım geliştirme sürecine çerçeve çizen 12 temel prensip ise şöyledir:

1. Yazılımın erken ve devamlı teslimi ile müşteri memnuniyeti sağlanmalıdır.
2. Yazılım sürecinin son aşamalarında olunsa bile değişen gereksinimler kabul edilmelidir.
3. Kısa zaman aralıkları belirlenerek çalışan yazılım, düzenli bir şekilde müşteriye sunulmalıdır.
4. Yazılım sürecini yöneten ve geliştiren ekibin tüm elemanları, proje boyunca her gün birlikte çalışmalıdır.
5. Projede motivasyonu yüksek bireylere yer verilmelidir. Onlara gerekli ortam, destek ve güven sağlanmalıdır.
6. Yazılım ekibinde iletişimin en verimli ve etkili yöntemi yüz yüze iletişimdir.
7. Çalışan yazılım, ilerlemenin birincil ölçüsüdür.
8. Çevik süreçler, sürdürülebilir geliştirmeyi teşvik etmektedir.
9. Teknik konulara ve tasarıma önem gösterilmelidir.
10. Yazılım sade ve basit olmalıdır.
11. En iyi mimariler, gereksinimler ve tasarımlar; kendi kendine organize olabilen ekiplerden ortaya çıkar.
12. Yazılım ekibi düzenli aralıklarla, nasıl daha etkili ve verimli olabileceğini düşünür ve buna göre gerekli iyileştirmeleri yapar. (Süreç iyileştirme)

* Çevik yazılım geliştirme modellerinin avantajları arasında, esnek bir yapıda olması ve değişime açık olması özellikleri sayılabilir. Ayrıca, kısa döngüler barındırdığından dolayı ekip elemanlarının motivasyonu hep yüksek seviyede olur. Projede çalışan üyelerin sürekli iletişim halinde olması da süreci hızlandırır. Yazılım ürünü küçük parçalara bölündüğü için geriye dönük hataların düzeltilmesi daha kolaydır, maliyeti düşüktür.
* Çevik modeller kendi içinde, temeli aynı ancak pratikleri farklı olan yaklaşımlara ayrılır. En yaygın yaklaşımlar: 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).

**Extreme Programming (XP)**

En bilinen çevik modellerden birisidir. 1999 yılında, Kent Beck tarafından ortaya atılmıştır.

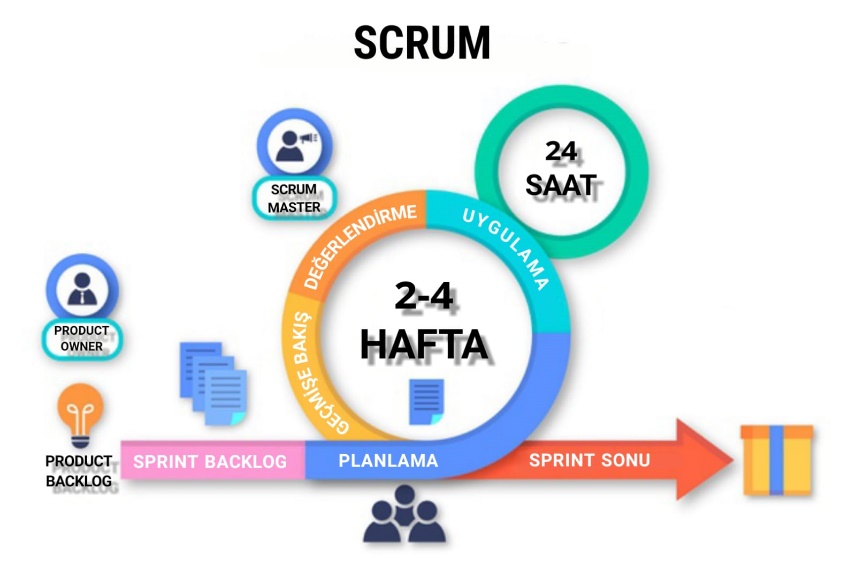
* Uygulanması kolay, iletişime önem veren, geri bildirimlerin etkin kullanımına olanak sağlayan bir modeldir. Müşteri ve müşteri gereksinimleri modelin merkezinde konumlanmıştır, değişen gereksinimlere ayak uydurmak kolaydır. XP modelinin 12 farklı pratiği ve 4 temel değeri vardır:

***4 Temel Değer***

1. İletişim: Bir projenin başarılı olabilmesi için etkili bir iletişim şarttır. Bu iletişimin açık, samimi ve yüz yüze olması gerekir.
2. Basitlik: Yazılımda en önemli konulardan birisi de basitliktir. Zorunlu olan işlemlerin yapılması ve gereksinimlere uygun en basit yapının kurulması gerekir.
3. Geri Bildirim: Sürekli ve hızlı geri bildirimler, projenin gelişimine olumlu yönde etki eder. Bu sayede oluşabilecek yanlış anlaşılmalar ve hatalar erkenden çözüme kavuşturulur.
4. Cesaret: Yazılımcı başarısız olmaktan korkmamalıdır. Sorumluluk almalı, projenin üstüne yılmadan gitmelidir. Elini korkak alıştırmamalıdır!

***XP Modelinin 12 Pratiği***

1. Planlama Oyunu: Yazılım ekibindeki üyelerin, müşteri öncelik planına göre hazırlanan yinelemeleri, ne kadar sürede gerçekleştirebileceklerini tahmin etmeye çalıştıkları toplantıdır. Müşteri, kullanıcılar ve yazılımcılar katılır.
2. Ekipte Müşteri: XP Projelerinin merkezinde müşteri gereksinimleri vardır. Dolayısıyla müşterinin de projeye dahil olması gerekir. Süreç içerisinde müşteri temsilcisi, yazılım ekibi ile aynı ortamda çalışır. Böylece, yazılımcı gerekli bilgiye en kısa yoldan ulaşabilir.
3. Önce Test: Test programı, kodlama işlemi yapılmadan önce hazırlanır. Bu sayede, çıkabilecek sorunlar erkenden tespit edilebilir, risk azalır.
4. Basit Tasarım: Yazılımda, müşteri gereksinimlerine göre en basit yapının kurulması gerekir. Böylece yazılım daha kolay anlaşılabilir, değiştirilebilir ve genişletilebilir.
5. Çiftli Programlama: XP modelinde, iki yazılımcının aynı bilgisayarda çalışabilir. Farklı bakış açıları, farklı bilgi birikimi ile proje hızlanır. Genellikle yeni başlayan yazılımcılar için önerilir. Usta-çırak ilişkisi vardır.
6. Sürekli Entegrasyon: Geliştirme sırasında sistemde yapılan değişiklikler, hemen sisteme entegre edilerek test edilir. Bu sayede yazılımcı, değişiklikleri sistem üzerinde görür ve oluşabilecek hataları erken aşamalarda tespit eder.
7. Kısa Aralıklarla Sürümler: Proje 2 ila 4 haftalık küçük yinelemelere ayrılır. Her bir yinelemenin sonunda, müşteriye çalışan bir sürüm teslim edilir. Böylece müşterilerin gidişatı kontrol edebilmesine olanak sağlanır.
8. Yeniden Yapılandırma: Yazılan kodlar ve yapılan tasarım sürekli gözden geçirilir. Yazılımı basitleştirmek ve müşteri istekleri doğrultusunda geliştirmek için yeniden yapılandırma yapılabilir.
9. Ortak Kod Sahiplenme: Üretilen kodlar, tüm ekip üyelerinin ortak sorumluluğudur. Yazılımcılar sadece kendi yazdıkları değil tüm kodlar üzerinde değişiklik yapma hakkına sahiptir.
10. Benzetim (Metafor): Yazılım projesinin her bir parçası, birbirine benzeyecek şekilde geliştirilir. Her sistem başka bir sistemle benzeşir.
11. Kodlama Standardı: Projedeki ekip üyeleri, yazılımı belirli bir standarda göre gerçekleştirirler. Bu standartlarla isimlendirmelerin nasıl olacağı, yorum satırlarının kullanımı, kodun genel yapısı gibi tanımlamalar yapılır. Amaç; kodlar üzerinde çalışan herkesin anlayabileceği şekilde yazılımı geliştirmektir.
12. Haftada 40 Saat: Projede çalışan ekibin çalışma süresi haftalık 40 saattir. Günde (hafta içi) 8 saatten fazla çalışmak verimliliği düşürür, hata oranını yükseltir. Bazı zamanlar fazla mesai yapılabilir. Ancak sürekli fazla mesai XP modeline aykırıdır ve projede bir şeylerin yanlış gittiğinin göstergesidir.

**SCRUM**

SCRUM, 1995 yılında Jeff Sutjerland ve Ken Schawaber tarafından ortaya atılan bir çevik yazılım geliştirme modelidir.

* Bu modelde yazılım, küçük parçalara (sprint) ayrılır. Bir sprint en fazla 1 ay sürer. Her gün 15-30 dakika süren bir değerlendirme toplantısı (Scrum Daily Meeting) yapılır. Bu toplantılar ile projenin genel durumu takip edilir. Esnek bir yapısı vardır. Gereksinimlerin kolaylıkla tanımlanamadığı, karmaşık projeler için uygundur. Ayrıca Scrum modeli sadece yazılım geliştirmek için değil, her alanda uygulanabilir bir modeldir. 3 temel kavram barındırır: Roller (Roles), Toplantılar (Meetings) ve Bileşenler (Artifacts).

***Roller (Roles)***

Ürün Sahibi (Product Owner): Proje ekibi ile müşteri arasındaki iletişimi sağlar. Yazılımın gidişatını takip eder, geri dönüşler sağlar. Ayrıca product backlog (önceliklerine göre gereksinim listesi) oluşturur.

Scrum Yöneticisi (Scrum Master): Scrum kurallarını ve pratiklerini iyi bilen kişidir. Ekibin bu kuralları uygulamasını sağlar. Çalışma ortamını iyileştirir.

Scrum Takımı (Scrum Team): Scrum takımı, bir sprint boyunca yapılacak olan işleri tamamlayan ekiptir. Ekip devamlı iletişim halindedir, kendi kendini yönetir. Scrum takımındaki kişilerin tek bir görevi yoktur, çapraz görev dağılımı vardır. 5 ila 9 kişiden oluşur.

***Toplantılar (Meetings)***

Sprint Planlama (Sprint Planning): Bu toplantıda, product backlog’daki gereksinimler tasklara (küçük görevlere) ayrılır, scrum takımındaki kişilere dağıtılır. Risk değerlendirmesi ve maliyet hesaplamaları yapılır.

Sprint Gözden Geçirme (Sprint Review): Her sprintin sonunda yapılan toplantıdır. Sprint boyunca yapılan işlemler ve oluşturulan yazılım incelenir.

Günlük Scrum Toplantısı (Daily Scrum Meeting): Sprint boyunca her gün yapılan 15-30 dakika arasındaki ayak üstü toplantıdır. Ekip üyeleri; “dün ne yaptın?”, “bugün ne yapacaksın?”, “seni engelleyen ne idi?”, gibi sorulara cevap verir. Eğer sorun yaşayan bir üye varsa scrum master ona yardımcı olur.

***Bileşenler (Artifacts)***

Ürün Gereksinim Dokümanı (Product Backlog): Müşteri gereksinimlerinin önceliklerine göre sıralandığı listedir. Dinamik bir dokümandır, değişen gereksinimlere göre değişiklik yapılabilir. Genellikle kullanıcı hikayelerinden (user story) oluşturulur.

Sprint Dokümanı (Sprint Backlog): Product backlog içerisindeki gereksinimler öncelik sırasına göre sprint backlog içine alınır. Bir sprint boyunca yapılacak işlemlerin listesi çıkarılır. Zaman çizelgesi çizilir. Yalnızca Scrum takımı tarafından değiştirilebilir.

Sprint Kalan Zaman Grafiği (Burndown Chart): Bir sprint süresince yapılması gereken işlerin ne kadarının yapıldığını ve ne kadar süre kaldığını gösteren grafiktir. Scrum takımının performansını takip etmek için kullanılır.

**SCRUM Günümüzde Neden Popüler?**

Scrum, günümüzde en çok kullanılan yazılım geliştirme modelidir. Bunun nedenleri;

* Uygulanması basit ve hızlıdır.
* Şeffaflık vardır.
* Değişime açıktır.
* Müşteri memnuniyeti ön plandadır.
* Ekip içinde etkili iletişimi sağlar.
* Maliyeti yüksek değildir.
* Gereksinimlerin kolaylıkla tanımlanamadığı, karmaşık projeler için uygundur.

***Kaynakça***

* İzmir Bakırçay Üniversitesi, BİL102 – Yazılım Mühendisliği Temelleri, 2. ve 3. Hafta Ders Notları
* <https://www.tutorialspoint.com/sdlc/index.htm>
* <https://phoenixnap.com/blog/software-development-life-cycle>
* [http://ybsansiklopedi.com/wp-content/uploads/2015/08/Yazılım-Geliştirme-Modelleri-Yazılım-Yaşam-DöngüsüSDLCYBS.pdf](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://agilemanifesto.org>
* <https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/>
* <https://www.quora.com/Why-is-the-Scrum-process-so-popular-in-the-software-industry>