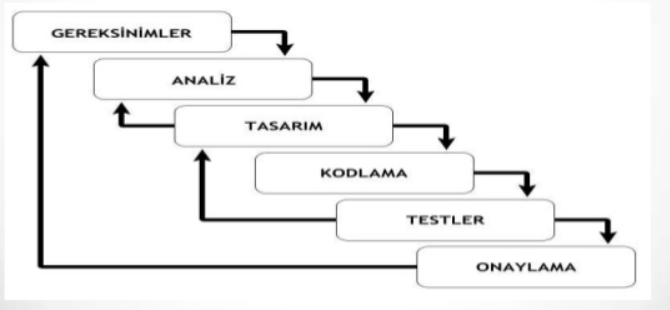
**YAZILIM YAŞAM DÖNGÜ MODELLERİ**

Yazılım yaşam döngüsünün en iyi sonuç verebilmesi için yazılım döngü modelleri kullanılır. Yazılım yaşam döngüsünde birden fazla model vardır. Bu modeller; yazılım projesinin büyüklüğü, yazılım projesini kullanacak kitle, maliyet, zaman gibi faktörler göz önünde bulundurularak kullanılır. Bu modellerden bazıları şunlardır:

**1. Gelişigüzel Model:** Geliştirme ortamında herhangi bir model ya da yöntem kullanılmaz. Bu yüzden bu yöntemi bir model olarak adlandırmak doğru değildir. 1960’lı yıllarda ortaya çıkan bu yöntem daha çok tek kişilik projelerde kullanılır ve genellikle çok basit bir programlamaya sahiptir. Genellikle kişiye bağlıdır bu yüzden takip edilebilirliği ve bakımı oldukça zordur.

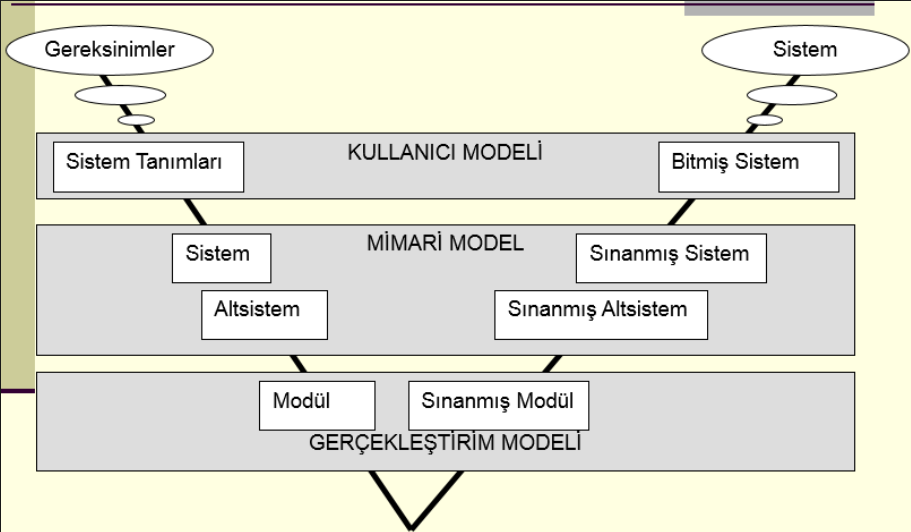
**2. Barok Modeli:** 1970’li yılların ortalarından itibaren kullanılmaya başlanan modelde, yaşam döngü adımları temel olarak doğrusal bir şekilde gözden geçirilir. Barok modelinde, dokümantasyon yazılan programın test aşamaları bittikten sonra yapılır. Günümüzde ise dokümantasyon döngünün başından sonuna kadar sürekli olarak yapılır. Aşamalar arası geri dönüşlerin nasıl yapılacağı belirsiz olan ve gerçekleştirim evresine daha çok ağırlık veren bu model günümüzde tercih edilmemektedir.

**3. Çağlayan (Waterfall) Yaşam Döngü Modeli:**



Geleneksel bir model olan çağlayan modeli en popüler ve en temel modeldir. Bu model aşamaların en az bir kez tekrarlanmasıyla gerçekleşir. Çağlayan modeli iyi tanımlı ve üretimi kısa süren projeler için uygundur. Kullanımı ve anlaşılması kolay olan bu modelde barok modelinden farklı olarak dokümantasyon ayrı bir süreç olarak değil üretimin doğal bir parçası olarak ele alınır ve aşamalar arası geri dönüşün nasıl olacağı bellidir. Kullanıcıyı sürece katmayan bu modelde olası bir sorun çok geç fark edilebilir bu da sürecin uzamasına ve maliyetin artmasına neden olur. Değişimlere elverişsiz olan bu modelin günümüzde kullanımı gittikçe azalmaktadır.

**4. V Süreç Modeli (V-Shaped Model):** Belirsizliklerin az olduğu projeler için kullanıma uygun olan model, çağlayan modelinin biraz daha gelişmiş halidir. V şeklinde şematize edilen bu modelde sol taraf üretim, sağ taraf ise sınama işlemleridir. Kullanımı ve takibinin kolay olması bu modelin avantajlarıyken risk çözümleme ile ilgili aktiviteleri içermemesi, aynı zamanda gerçekleştirilebilecek olaylara kolay imkan tanımaması ve aşamalarda tekrar bulunmaması bu modelin dezavantajlarındandır.



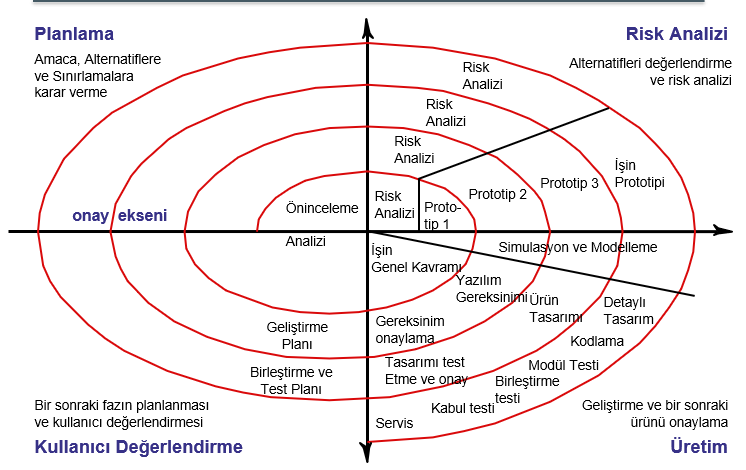
Bu döngü; kullanıcı modeli, mimari model ve gerçekleştirim modeli olmak üzere 3 modelden oluşur.

**Kullanıcı Modeli:** Kullanıcının sistemi nasıl kabul edeceğine ilişkin sınama belirtimleri bulunur.

**Mimari Model:** Projenin tasarımı ve bunların denemelerinden oluşur.

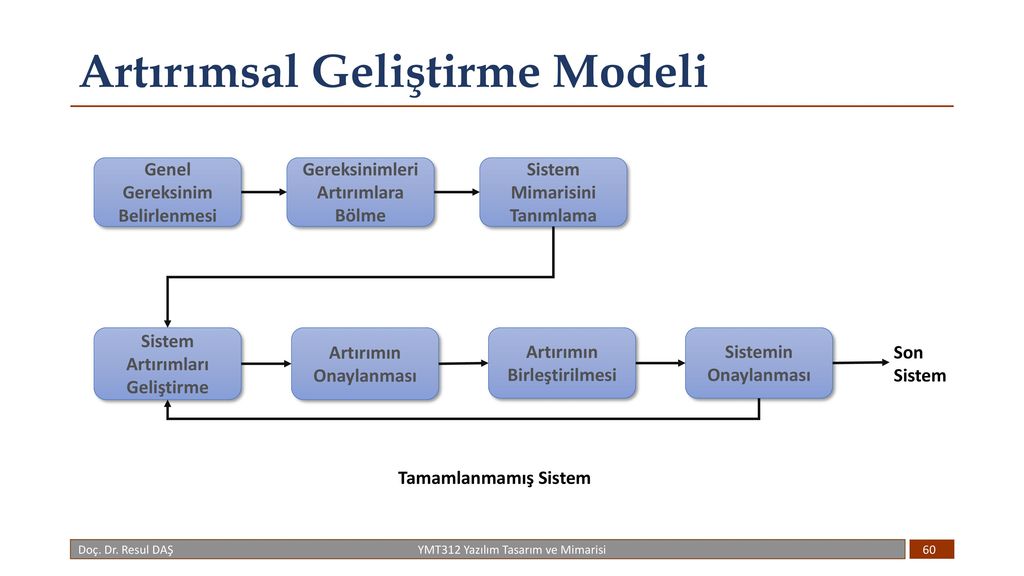
**Gerçekleştirim Modeli:** Kodlama ve bunların denenmesinden oluşur.

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



Planlama, risk analizi, üretim ve kullanıcı değerlendirmesi olarak 4 ana aşamadan oluşan bu modelde yinelemeli artımsal yaklaşım vardır. Risk analizinin ön planda olduğu ve prototiplemenin görüldüğü modelde her döngü bir fazı ifade eder. Bu model hataları erken giderme ve her aşamada yazılım projesinin bir parçasını görme imkanı sağlar. Önceden geliştirilmiş yazılım bileşenlerinin yeniden kullanıldığı projeler için kullanımı çok uygundur. Küçük ve düşük riskli projeler için oldukça pahalı bir sistem olması, kompleks bir yapıya sahip olması ve fazla dokümantasyon oluşturması bu modelin dezavantajlarındandır.

**6. Artımsal Geliştirme Süreç Modeli (Incremental Development Model):**



Bu modelde proje bir bütün olarak değil parçalar halinde ele alınır ve bu parçalar kullanıcı önceliğine göre sıralanır. Sıralanan parçalar bittiğinde birer ara ürün geliştirilmiş olur. Geliştirilen bu ara ürünler kullanıcı tarafından kullanılır. Bu ara ürünler birbirini kapsayacak ve giderek artan sayıda işlev içerecek şekilde geliştirilir. Yani bu modelde bir taraftan kullanım olurken diğer taraftan üretim yapılır. Artımsal geliştirme modeli, uzun zaman alabilecek ve sistemin eksik işlevlikle çalışabileceği türdeki projeler için uygundur (ör. Bitirme Tezleri...). Sistem gereksinimlerinin müşterilerle belirlenmesi, erken artımların prototip gibi davranarak gereksinimlerin daha iyi anlaşılmasını sağlaması, başarısızlık riskinin az olması bu modelin avantajları arasındadır. Dezavantajları ise deneyimli personel gerektirmesi, parçaların kendi içinde tekrar edilmesine izin verilmemesi ve parçaların tanımlanması için tüm sistemi detaylı bir şekilde tanımlanmayı gerektirmesidir.

**7. Kodla ve Düzelt Yaşam Yaşam-Döngü Modeli (Code and Fix Model):** Çoğunlukla öğrenciler ve bireysel geliştiriciler için uygun olan bu model büyük projeler için tercih edilmez. Emeklilik (retirement) safhası vardır. Kullanımı uzmanlık gerektirmeyen bu modelde kaynak planlaması yoktur ve bitiş süresi belirsizdir. Ayrıca dokümantasyon olmadığı için hataların tespit edilmesi ve bakım yapılabilirliği zordur. Yazılım geliştirmenin en kolay yolu olan bu model aynı zamanda maliyeti en yüksek olan yoldur.

**8. Evrimsel Geliştirme Modeli (Evolutionary Development Model):** İlk tam ölçekli olan bu model “Ne istediğimi bilmiyorum ama görsem tanırım.” mantığına dayanır. Modelin başarısı ilk evrimin başarısına bağlıdır. Diğer modeller ile kıyaslandığında ilerleyişi daha yavaş olan model, kısa süreli sistemlerde ya da büyük bir sistemin parçaları (ör. Kullanıcı Ara Yüzleri...) için kullanılabilir. Her aşamada üretilen ürünler, üretildikleri alan için tam işlevselliği içermektedirler. Gereksinimi anlamayı basitleştirmesi, risk ve hata oranının az olması bu modelin avantajları arasındadır. Bu modelin zayıf yanı ise değişiklik denetimine sahip olmamasıdır. Bu nedenle bakımı maliyetli ve zordur. Düzenli bir ürün oluşumu olmayan bu modelin keşifçi geliştirme ve atılacak prototipleme olmak üzere iki çeşidi vardır.

**9. Prototipleme (Prototyping):** Prototip, bir ürünü piyasaya sürmeden önce ürünün ne yapacağını, ürünün neye benzeyeceğini, ürünün nasıl işlediğini gösteren gerçek bir ürün simülasyonudur. Prototipleme, tasarım sürecindeki en önemli adımlardan biridir. Prototiplemedeki amaç prototip üretip geliştirmektir. Gereksinimlerin iyi bir şekilde anlaşılması bu süreç için önemlidir. Hızlı plan, hızlı tasarım ve hızlı kodlamanın ön planda olduğu bu modelde üretilen ilk prototip kullanıcının kullanımına ve değerlendirilmesine sunulur. Bu değerlendirmeler ışığında ilk prototip üzerinde gerekli değişiklikler yapılır ve prototipin yeni hali kullanıcı tarafından tekrar değerlendirilir. Böylelikle kullanıcının isteğine yaklaşmış olan son prototipe göre değerlendirmeler yapılır ve neler yapılacağı konusunda anlaşmaya varılır. Sınırlı sayıdaki yinelemeden sonra, yazılımın son hali kullanıcıya teslim edilir. Doğrusal modelin döngüye çevrilmiş hali olan bu modelin yeni gereksinimlere açık olması, risk kontrolünün olması, karmaşa ve yanlış anlaşılmayı engellemesi avantajlarındandır. Dezavantajlarına gelirsek; hedeflerin net olmaması halinde güvenlik açığı problemleri ortaya çıkabilir. Acele edildiği için dokümantasyonu olmayan, hızlı ve kirli prototipler ortaya çıkar. Kullanıcı prototipin son ürün gibi olması beklentisine girebilir. Düzeltme aşaması atlanırsa düşük performansa sebep olabilir.

**10. Formal Sistem Geliştirme (Formal System Development):** Matematiksel belirtimin farklı gösterim şekilleri ile çalıştırılabilir programa dönüştürülmesine dayalı bir modeldir. Bu model karmaşık sistemleri ve programları geliştirmeye destek verir, kullanıcı sistemi kullanmaya başladığında karşısına çıkan belirtim hatalarını minimize eder. Yazılım artımlarla geliştirilir. Sürekli tümleştirme vardır ve fonksiyonellik tümleştirilen yazılım artımları ile artar. Sistem kullanıma koyulmadan emniyet ve güvenlik durumlarının sağlanması gereken kritik sistemlerde uygulanabilir bir modeldir. Çok zaman alan ve pahalı bir yöntem olup uygulayabilmek için eğitim ve özel beceriler gerektirir. Bu model her tekrardan sonra aşamalı olarak artan efektif çözümler sunar, yazılımdaki eksiklikleri ve belirsizlikleri tespit eder.

**11. Yeniden Kullanıma Yönelik Geliştirme (Re-use Based Development):** Programı geliştirecek kuruluş tarafından daha önceden hazırlanmış veya dışarıdan temin edilmiş yazılımların kullanılmasını temel alan bu modelde kuruluşların olgunluğu arttıkça modelin kullanımı da kolaylaşır. Önceden oluşturulmuş sınıflar tekrardan kullanılabilir böylelikle kısa sürede yazılım geliştirilebilir. Bu modeli kullanmak tecrübe ve birikim gerektirir. Modelin kendisi basit ve anlaşılabilirdir. Ancak gereksinimleri anlamak güçtür. Başarım garantisi olmayan, maliyeti yüksek bu modelin son yıllarda popülaritesi artmaktadır.

**12. Birleşik Süreç (Unified Process):** Yinelemeli, arttırmalı, evrimsel ve aynı zamanda risk güdümlü olan yazılım geliştirme sürecidir. Nesne tabanlı yazılımları geliştirmek için kullanılan yöntemlerin kullanılmasıyla elde edilen deneyimlerle bu sürecin en iyi özellikleri birleştirilerek tümleştirilmiş bir yazılım geliştirme sürecidir. Değişen isteklere uygunluk, erken ürün elde etme, risklerin erken giderilebilmesi bu modelin avantajlarıyken karmaşık olması, dokümantasyon yükü nedeniyle maliyetin fazla olabilmesi ise bu modelin dezavantajlarındandır.

**13. Araştırma Tabanlı Model (Research Based Model):** Çoğunlukla proje ödevleri, yarışma projeleri gibi sistemlerde kullanılan bu modelde yapılan işlerden elde edilecek neticeler belirgin değildir. Bu model Yap-At tarzı proje olarak da isimlendirilebilir.

**14. Çevik (Agile) Yazılım Geliştirme Süreci:** Yazılım sektöründe, yazılım hatalarının geç fark edilmesi, değişimlere elverişsiz olma, değişiklik isteklerine hızlı cevap verilmemesi gibi çeşitli sorunlar ortaya çıkmıştır. Bu sorunların çözümü için çevik olarak adlandırılan metotlar geliştirilmiştir. Bu metotlar hata oranı az, yüksek verimli, hızlı, ucuz ve esnek çözümler sağlamaktadırlar. Müşteri memnuniyeti, sık aralıklarla parça parça ürün teslimi yapma, yüz yüze iletişime önem verme, iyi tasarım ve sadeliğe odaklanma, sürdürülebilir gelişmeyi destekleme, değişen ihtiyaçları karşılama, kendi kendine organize olan takımlar kurma çevik yazılım yöntemi temel prensiplerindendir. Daha çok bağımsız şirketler tarafından tercih edilen yöntem aşırı çalışma gerektirmektedir. Takım üzerinde hedef baskısı oluşturması, kariyer riski ihtimali gibi dezavantajları vardır. Bazı çevik yazılım geliştirme modelleri şunlardır:

**14. 1. Extreme Programming (XP):** Kent Beck ve arkadaşları tarafından 1996 yılında ortaya çıkarılmış bir yazılım geliştirme disiplinidir. XP’nin iletişim, basitlik, geri bildirim ve cesaret olmak üzere 4 temel değeri vardır.

**İletişim:** Projenin düzgün bir şekilde ilerlemesinde önemli bir rol oynar. Ekip içi iletişim ve kullanıcı ile yazılımcı arasındaki iletişim devamlı ve yüz yüze olmalıdır.

**Basitlik:** Zorunlu ihtiyaçları hedef alarak, esnek ve pratik yöntemlerle problemleri çözmektir.

**Geri Bildirim:** Proje başarısı için önemli bir ihtiyaçtır. Müşteri ve yazılım ekibinin belli aralıklarla bir araya gelip değerlendirmeler yapmasıyla oluşabilecek hatalar engellenir.

**Cesaret:** XP’nin 4 temel değerinden en zor olanıdır. Yapılan işlerde cesur olunmalıdır, korkmadan ilerlenmelidir. Başarısızlığın sonucuna değil sebebine ve telafisine yoğunlaşılmalıdır. Başarısızlık durumunda yazılımcılar pes etmemeli gerekirse baştan başlayabilmelidirler.

XP 12 farklı pratiği ön görür. Bunlar:

**Planlama Oyunu:** Yazılım ekibi ve müşterinin bulunduğu bir toplantıda istenilen işin ne kadar sürede yapılabileceğinin kestirilmesidir.

**Ekipte Müşteri:** Müşterinin yazılım ekibiyle yakın temas halinde bulunması, yazılımcının ihtiyaç duyduğu bilgilere kolay ve kısa bir sürede ulaşabilme olanağı sağlar.

**Önce Test:** Kod yazılmadan önce kodun testinin yazılmasıdır. Bu sayede çıkabilecek sorunlar daha erken tespit edilebilir.

**Basit Tasarım:** Müşteri gereksinimlerini karşılayacak en basit tasarım gerçekleştirilir. Tasarım gerektiğinde refactor edilmelidir.

**Çiftli Programlama (Pair Programming):** Kodun eşler halinde yazılmasıdır. Farklı bakış açılarıyla proje üzerinde çalışılması ilerleyen safhalarda oluşabilecek bazı sorunların erken tespit edilmesini sağlar. Özellikle yeni başlayan yazılımcılar için önerilir.

**Kısa Aralıklı Sürümler:** Proje parçalara bölünür. Her parçanın kendine ait teslim tarihi vardır. Bu teslim tarihine kadar belirlenen iş yapılar ve teslim edilir.

**Yeniden Yapılandırma (Refactoring):** Yazılan kodu daha anlaşılır hale getirme ve müşteriden gelecek olan yeni istekleri projeye nasıl dahil edebiliriz sorusuna cevap aramaktır.

**Ortak Kod Sahiplenme:** Yazılan kod, bütün ekip üyelerinin ortak malı kabul edilir.

**Benzetim (Metafor):** Geliştirilen yazılımdaki sistemleri birbirine benzeterek yazılım geliştirmeyi esas alan bir pratiktir.

**Kodlama Standardı:** Önceden belirlenmiş kod standartlarına göre yazılım geliştirmektir. Buradaki amaç, yazılan kodun bütün ekip üyeleri tarafından kolayca anlaşılabilmesini sağlamaktır.

**Haftada 40 Saat:** Ekiplerin verimli çalışabilmeleri için aşırı çalışma saatlerinden kaçınılması gerekmektedir. Bu uygulamaya göre 40 saatten fazla çalışmak hata yapma riskini arttırır.

**Sürekli Entegrasyon:** Sistemüzerinde yapılan değişikliklerin doğrulanıp sonrasında entegre edilmesini sağlayan yaklaşımdır. Sürekli entegrasyon oluşabilecek hataların erken tespit edilmesini ve yazılımcıların sistem üzerinde yapılan değişiklikleri görebilmesini sağlar.

**14. 2. Scrum:**

Çevik proje yönetim metodolojilerden biridir. Kompleks yazılım süreçlerinde kullanılır. Scrum, kompleks yazılım işlerini **sprint** adı verilen küçük birimlere bölerek geliştirir. Scrum; gereksinimlere hızlı cevap vermesi, zamandan ve paradan büyük ölçekte tasarruf edilmesi gibi özelliklere sahiptir. Sahip olduğu bu özellikler Scrum’ı günümüzde en çok kullanılan proje yönetim yaklaşımı yapmıştır.



Scrum’da roller, toplantılar ve bileşenler olmak üzere 3 temel kavram vardır.

**Roller: Ürün sahibi (product owner);** geliştirilecek ya da mevcut olan ürünü sahiplenen, yöneten kişidir. **Scrum yöneticisi (scrum master);** problem olduğunda onu çözen, takıma yardımcı olan, takımı ve organizasyonu Scrum’a adapte eden, takım ve ürün sahibiyle olan iletişime yardımcı olan kişidir. **Scrum takımı (scrum team);** birbiriyle sürekli iletişim halinde olan, aynı hedef uğruna çalışan kişilerden oluşan bir gruptur. Bu grup genellikle 5-9 kişiden oluşur.

**Toplantılar: Sprint (koşu) planlama (sprint planning);** sprint içinde yapılacak olan işlerin planlandığı, takımların belirlendiği, gereksinim listesinin çıkarıldığı, maliyet hesapları ve risk analizinin yapıldığı, geliştirme araçları ve alt yapısının onaylandığı toplantılardır. Bu toplantıya ürün sahibi, Scrum takımı ve Scrum yöneticisi katılır. **Günlük scrum toplantısı (daily scrum meeting);** tüm sprint boyunca Scrum takım üyelerinin her gün bir araya gelerek ayak üstü gerçekleştirildiği 15 dakikalık toplantılardır. Dün ne yaptım, bugün ne yapacağım, işimi yapmaya engel var mı sorularına her takım üyesi cevap verir. Bu cevaplar doğrultusunda herhangi bir problem var ise Scrum yöneticisi bu problemi ortadan kaldırır. Günlük Scrum toplantısı, takımdaki çoğunluğun sağlanamaması dışında her gün yapılır. **Sprint (koşu) gözden geçirme (sprint reiew);** her sprit sonunda yapılan gözden geçirme işlemidir. Yazılımın gereksinimlere uygunluğundan emin olunur. Tespit edilen hatalar düzeltilir. **Sprint (koşu) retrospektif (sprint retrospective);** sprint boyunca yapılan işlerin değerlendirildiği toplantıdır. Bu toplantı Scrum takımının kendisini geliştirebilmesi için büyük bir fırsattır.

**Bileşenler/Araçlar (Artifacts): Ürün gereksinim dokümanı (product backlog);** ürüne ait yapılması gereken işler ve üründe olması gereken özellikler listesidir. Bu listeye kullanıcının isteğine göre ekleme ve çıkarma yapılabilir. **Sprint (koşu) dokümanı (sprint backlog);** sprint boyunca yapılacak işlerin takım tarafından detaylandırılmış planıdır. Sprint backlog sadece takım tarafından değiştirilebilir. **Sprint kalan zaman grafiği (burndown chart);** bir sprint turunda yapılan görevler ile kalan iş arasındaki bağıntıyı ortaya koyan grafiktir. Kalan sürede işlerin bitip bitmeyeceği hakkında takıma bilgi verir.

**14. 3. Test Odaklı Geliştirme (Test- Driven Development-TDD):** Çevik yazılım geliştirme yaklaşımı olan TDD tekniğinin hızlı, değişime açık, verimli ve yüksek kaliteli olması nedeniyle son yıllarda tüm dünyada kullanımı hızla artmaktadır. TDD, kod yazılmadan önce test senaryolarının yazıldığı ve bu senaryolara bağlı olarak kodun yazıldığı tekniktir. TDD yaklaşımıyla programın daha az hata ile geliştirilmesi sağlanabilir. Gereksiz kod kalabalığı ortadan kaldırılır ve geriye dönük testler sürecin önemli bir parçası haline getirilir.

**14. 4. Özellik Güdümlü Geliştirme (Feature-Driven Developmend-FDD):** Bu süreç modelleme, listeleme, planlama, tasarım ve geliştirme olmak üzere 5 ana basamaktan oluşur. Jeff De Luca ve Peter Coad tarafından 1997 yılında geliştirilen ve özellik tabanlı gerçekleştirimi esas olan bir modeldir. İlerlemeli ve artımlı bir yazılım geliştirme sürecidir.

**14. 5. Dinamik Sistem Geliştirme Metodolojisi (Dynamic System Development Methodology):** Başlangıçta bir yazılım geliştirme yöntemi olarak kullanılan çevik bir proje teslim çerçevesidir. DSDM’ de kullanıcıların aktif kullanımı önemlidir. Test tüm yaşam döngüsü boyunca entegredir. DSDM, iş amacına uygunluğu teslim etmenin birincil kriteri olarak adlandırır. 1994’ten beri bu metodoloji çevik ve yinelemeli yazılım geliştirme projelerinin planlanması, yürütülmesi, ölçeklendirilmesi ve yönetimi için kapsamlı bir temel sağlamak amacıyla gelişmiş ve olgunlaşmıştır.

**Kaynaklar**

•<https://medium.com/@omerharuncetin/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-543c7879a742>

•<https://medium.com/@HayriRizaCimen/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-ve-s%C3%BCre%C3%A7-modelleri-70fdfb2f8f77>

• <https://www.codex.com.tr/yazilim-gelistirme-modelleri>

• <http://furkanalniak.com/yazilim-muhendisligi-yazilim-surec-modelleri/>

• <https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/>

• <https://medium.com/@secilcor/scrum-nedi%CC%87r-6a4326951dd8>

•<https://mmf.ders.bakircay.edu.tr/pluginfile.php/6618/mod_resource/content/1/B%C4%B0L%20102%20-%202.Hafta%20-%20Yaz%C4%B1l%C4%B1m%20Ya%C5%9Fam-D%C3%B6ng%C3%BC%20Modelleri.pptx>

•<https://mmf.ders.bakircay.edu.tr/pluginfile.php/6768/mod_resource/content/1/BI%CC%87L%20102%20-%203.Hafta%20-%20%C3%87evik%20Yaz%C4%B1l%C4%B1m%20Geli%C5%9Ftirme.pptx>

• <https://slideplayer.biz.tr/slide/12386328/>

**CEREN KAPLAN**

**190601005**