**Yazılım Yaşam Döngüsü**

Yazılım, fiziksel bir ürün olmadığı için aşınmaz fakat zamanla yetersizleşebilir. Bu gibi senaryoların gerçekleşme olasılığını asgari düzeyde tutabilmek için çoğu zaman yazılım yaşam döngüsüne ihtiyaç duyulur. Yazılım yaşam döngüsü, herhangi bir yazılım fikrinin ortaya atılıp gerçekleştirilmesine ve akabinde o yazılımın bakımına kadar geçen sürecin tümüne denir. Bu süreç içerisindeki safhaların sayısı ve niteliği kullanılan modele göre değişebilir. Bu sürecin temel aşamaları şunlardır: Gereksinim safhası (Requirement phase), Analiz safhası (Analyse phase), Tasarım safhası (Design phase), Gerçekleştirme safhası (İmplementation phase) ve Bakım safhası (Maintenance phase). Safhaların işleyişi şu şekildedir:

**Gereksinim safhası:** Ortaya atılan problemin ne olduğu anlaşılmaya çalışılır, müşterinin gereksinimleri belirlenmeye çalışılır. Bu aşamada müşterinin isteği de dikkate alınabilir.

**Analiz safhası:** Toplanan gereksinimler doğrultusunda analiz yapılır. Analizin sonucu, şartname dokümanı olarak yazılır. Yazılım proje yönetim planı da bu safhada hazırlanır. Bu aşamayı gereksinim safhasından ayıran bir önemli özellik de, müşterinin ne istediğinin değil, neye ihtiyacı olduğunun belirlenmeye çalışılmasıdır. Kısaca, geliştirilecek yazılımın, hangi işi yapacağı ile ilgili sorulara cevap bulmaya çalışılır.

**Tasarım safhası:** Bu aşama boyunca iki farklı tasarım yapılır:

* *Üst seviye ve mimari tasarım*: Kaba taslak plan, modüller…
* *Ayrıntılı tasarım*: Veritabanı tasarımı, algoritmalar, sınıf diyagramı, aktivite diyagramı, etkileşim diyagramı…

**Gerçekleştirme safhası:** Her biri için ayrı ayrı kod yazılan modüller, birbirinden bağımsız olarak test edilir. Yazılımın bir bütün olarak çalışabilmesi için modüller birleştirilir. Birleştirme sırasında (eğer varsa) meydana gelen sorun ve yanlışlıklardan ötürü yazılım, “İntegration Test” adı verilen teste tabi tutulur. Son olarak da kabul testi müşteri tarafından yapılır.

**Bakım safhası:** Bakımın iki çeşidi vardır: Düzeltici bakım ve özelliklerin arttırılmasına yönelik bakım.

* *Düzeltici bakım*: Sistemdeki hataları düzeltmek için yapılan bakımdır.
* *Özelliklerin arttırılmasına yönelik bakım*: Yazılım güncellemesini bu kategoriye örnek olarak verebileceğimiz bu bakım, şartnamedeki değişiklikleri ve değişikliklerin yazılıma uyarlanmasını içerir. Bu uyarlamanın iki çeşidi vardır: Mükemmelleştirici ve uyarlanabilir bakım. Mükemmelleştirici bakım, müşterinin düşünceleri baz alınarak ürünün işlevselliğini arttırmak için yapılırken; uyarlanabilir bakım, ürünün çalışma ortamında meydana gelebilecek değişikliklere karşı yapılır.

**Ürünün bakımı neden önemlidir?**

Öncelikle şu soruyu kendimize soralım: Bakım, kötü bir yazılıma mı yoksa iyi bir yazılıma mı yapılır? Çoğumuz bu soruya “Kötü bir yazılıma bakım yapılır” şeklinde cevap veririz. Lakin bu yanlış bir cevaptır çünkü kimse kötü bir yazılımda değişiklik yapmak istemez, meydana getirilecek en ufak bir değişiklikliğin sonucu olan zincirleme değişim reaksiyonuna sebebiyet vermek istemez. Bu yüzden kötü yazılımlar genellikle göz ardı edilme/atılma eğiliminde olurlar. Faaliyetine devam eden yazılımların bakımı yapılmadığı taktirde sistemin genişlemesi durur. Bu da yazılımlar için belli periyotlar eşliğinde sürekli bir bakımı zorunlu kılar.

Yazılım yaşam döngüsünde hatanın erken tespit edilip düzeltilmesi, düzeltme maliyetini asgari düzeyde tutar çünkü ilk aşamalarda meydana gelen hatalar kağıt üzerinde yapılır ve düzeltilmesi de daha az maliyetlidir. Son aşamalarda meydana gelen hataların düzeltilmesi için genellikle ilk aşamalarda da değişiklik yapmak gerekir. Bu da neredeyse tüm projenin gözden geçirilmesi demektir. Böylece projenin maliyeti katbekat artar.

**Yazılım Yaşam Döngüsü Modelleri**

***Gelişigüzel Model***

Geliştirme sürecinde herhangi bir model ya da yöntem kullanılmadığı için bu yöntemi bir model olarak tanımlamak doğru değildir. Geliştiriciye bağlımlı bir geliştirme yöntemi olduğu için bir süre sonra geliştirici bile sistemi anlayamaz hale gelir ve geliştirmede güçlükler yaşar. Bu yöntemin bakımı da oldukça zordur. Tek kişilik üretimde kullanılan bu yöntem 1960’lı yıllarda kullanılmıştır.

***Barok Modeli***

Yazılım yaşam döngüsü temel adımlarının doğrusal bir şekilde geliştirildiği bir modeldir. 1970’lı yılların ortalarından itibaren kullanılmaya başlanan bu modelde; belgeleme işlemi, günümüzde kullanılan modellerden farklı olarak ayrı bir işlem olarak değerlendirilir ve yazılımın geliştirilmesinden ve testinden sonra gerçekleştirilir. Lakin günümüzde belgeleme işlemi, yapılan işin doğal bir ürünü/sürecidir. Gerçekleştirme aşamasına ağırlık veren bu model, günümüzde uygulanan bir model olmaktan çıkıp güncelliğini yitirmiştir.

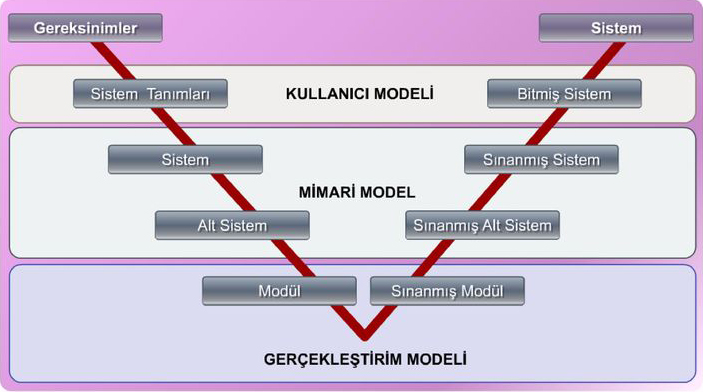
***Çağlayan Yaşam Döngü Modeli***

Bu model, barok modelinden farklı olarak projedeki belgeleme işlemini ayrı bir süreç olarak değil, üretimin doğal bir aşaması olarak ele alır. Yazılım yaşam döngüsü temel aşamaları baştan sona en az bir kere gerçekleştirilir. İyi tanımlanmış, resmi kurumlarca desteklenen ve üretiminde daha az zamana ihtiyaç duyulan yazılım projeleri için uygun bir modeldir. Barok modeline kıyasla geri dönüşlerin daha iyi tanımlandığı bu modelde; bir sonraki aşamanın başlayabilmesi için bir önceki aşamanın bitmesi gerekmektedir. Ancak bu modelin kullanımında dikkat edilmesi gereken noktalar vardır. En önemlisi, model içerisindeki aşamalar arasında daha iyi geri dönüşler yapılabilse de tüm detayın tasarıma yansıtılabilmesi için müşteri ve sistem gereksinimlerinin analiz aşamasında en ince ayrıntısına kadar belirlenmesi gerekir. Her aşamanın sonucu, bir ya da birden fazla onay alır ve gerektiğinde tekrarlamalar meydana gelebilir. Geleneksel bir model olarak bilinen bu modelin kullanımı giderek azalmaktadır.

*• Modelin bazı avantajları:* Proje yöneticileri iş dağılımını daha kolay bir şekilde gerçekleştirir. Aşamalar daha iyi anlaşılır. Bütçe ve zaman kısıtlamasından çok, kalitenin ön planda olduğu projelerde kullanılması daha uygundur.

*• Modelin bazı dezavantajları:* Bitirme kriteri olarak belgelendirmeye önem verilir, bu yüzden etkileşimli son kullanıcı uygulamaları için uygun değildir. Sistem geliştirilmesi sürecinde gereksinimler sıklıkla değiştiği için bu model, gereksinimlerin tam olarak anlaşılabildiği projelerde kullanılmalıdır. İkiden daha önceki aşamalara gitmek maliyeti arttırır. Bunun sonucu olarak da tüm aşamalar gözden geçirilebilir. Bu da kimsenin karşılaşmak istemeyeceği bir felaket senaryosudur.

***V Modeli***



Sol tarafta üretim, sağ tarafta ise test işlemleri bulunur. Bu modelin temel çıktıları;

* *Kullanıcı modeli:* Geliştirme sürecinin kullanıcı ile olana ilişkileri tanımlanır ve sistemin kabulune dair test planı ortaya çıkarılır.
* *Mimari model:* Sistemin tasarımı ve meydana gelecek olan alt sistem ile tüm sistemin test işlemlerine ilişkin işlevler bulunur.
* *Gerçekleştirim Modeli:* Yazılım modüllerinin kodlanmasına ve sınanmasına ilişkin fonksiyonlar bulunur.

Kullanıcının projeye katkısını arttırmayı hedefleyen bu model, iş tanımlarının belirgin olduğu ve belirsizliklerin asgari düzeyde tutulduğu bilgi teknolojileri projeleri için uygun bir modeldir. Bir bilgi teknolojileri projesinin iki aşamalı olarak ihale edilmesi için oldukça uygundur. İlk ihalede, iş analizi ve kabul sınamalarının tanımlarını yapmak amacıyla kullanıcı modeli hedeflenir. İkinci ihalede ise ilk ihaleden elde edilen kullanıcı modeli tasarlanır ve gerçeklenir.

*• Modelin bazı avantajları:* Kullanımı oldukça kolay olan modelin proje yönetimi tarafından takibi de kolaydır. Doğrulama ve onaylama planları erken aşamalarda vurgulanır. Ayrıca doğrulama ve onaylama işlemleri yalnızca son üründe değil teslim edilebilir tüm ürünlerde uygulanır.

*• Modelin bazı dezavatajları:* Aşamaların arasında yinelemelerin kullanılmadığı model, eş zamanlı gerçekleştirilebilecek olaylara ise kolay imkan tanımaz. Risk çözümleme ile ilgili olan etkinlikleri de içermez.

***Helezonik Model***

Bu model şu aşamalardan meydana gelir: Planlama, risk analizi, üretim ve kullanıcı değerlendirmesi.

*• Planlama:* Üretilmesine karar verilen ara ürün için planlama, amacın belirlenmesi ve bir önceki adımda üretilen ara ürün ile birleştirme esas alınır.

• *Risk analizi:* Riske senaryolarının araştırılması ve bu risklerin belirlenme aşamasıdır.

• *Üretim:* Ara ürünün üretilme aşamasıdır.

• *Kullanıcı değerlendirmesi:* Üretilen ara ürün ile ilgili olarak kullanıcının yaptığı sınama ve değerlendirmeleri kapsar.

Bu modelde risk analizi olgusu ön plana çıkmıştır. Öncelikle hedefler, alternatifler ve sınırlamalar belirlenir. Belirlenen alternatifler değerlendirilir ve riskler belirlenip çözülür. Son olarak da ürün geliştirilir. Her döngünün bir aşamayı ifade ettiği bu modelde yinelemeli artımsal bir yaklaşım söz konusudur.

*• Modelin bazı avantajları:* Kullanıcılar sistemi erken görebilir. Geliştirme küçük parçalara bölünür ve en riskli kısımlar daha önce gerçekleştirilir. Seçeneklere en erken zamanda dikkatle odaklanan bu model, hataları da erken bir şekilde gidermeye çalışır. Çok sayıda yazılım yaşam döngü modelini de bünyesinde barındırır.

*• Modelin bazı dezavatajları:* Bu model karmaşıktır ve modeldeki spiral sonsuza kadar dönebilir. Büyük ölçekteki projeler için uygun olan model, küçük ve düşük risk barındıran projeler pahalı bir yöntemdir.

Modelde ara adımlar o kadar fazladır ki çok fazla belgeleme işlemi gerektirir.

***Evrimsel Geliştirme Modeli***

İlk olarak tam ölçekli model olma özelliğini taşır. Geri bildirimler ile yazılım, çok sayıda sürüm ile yavaş bir şekilde geliştirilir. Üretilen ürünler, üretildikleri alan için tam işlevsel olma özelliği taşımaktadırlar. Dünyaya geniş bir şekilde yayılmış çok üniteli organizasyonlar için önerilir. Örnek olarak banka işlemleri. İki çeşit evrimsel geliştirme vardır: Keşifçi geliştirme ve atılacak prototipleme. Keşifçi geliştirmenin hedefi müşterinin gereksinimlerini incelemek için müşteri ile beraber çalışıp sistemin son halini kendisine teslim etmektir. Bu yüzden iyi anlaşılan gereksinimlerle başlanmalıdır. Atılacak prototiplemenin hedefi ise sistemin gerkesinimlerini anlamaktır. Bu da tam anlaşılamamış gereksinimlerle başlanmasını doğal kılar.

*• Modelin bazı avantajları:* Kullanıcılar, bu model ile kendi gereksinimlerini daha iyi anlama fırsatı elde eder. Sürekli olarak yapılan değerlendirmeler ile erken aşamalardaki geliştirme riskleri azaltılır. Hatalar da giderek azaltılır.

*• Modelin bazı dezavatajları:* Yazılım gereksinimini yenilemenin gerekebilmesi, bakımının zorluğuna işaret eder. Düzenli teslim edilebilir bir ürünün olmayışı, sürecin görünürlüğünün az olmasına neden olur. Sürekli olarak yapılan değişiklikler yazılımın yapısına zarara verir ve böylece yazılım iyi yapılandırılmaz.

***Artırımsal Geliştirme Modeli***

Sistemin tek seferde teslim edilmesi yerine, geliştirme ve teslim adımları parçalara bölünür. Her teslim beklenen işlevselliğin bir paarçasını karşılar. Önceliklendirilen gereksinimler, erken teslimler listesine dahil edilir. Geliştirmesi başlayan parçanın gereksinimleri dondurulur. Böylece meydana gelebilecek değişiklikler sonraki teslimlerde dikkate alınır. Bir taraftan kullanım olurken diğer taraftan üretim yapılır. Üretilen her yazılım sürümü birbirini kapsayacak ve giderek artan sayıda işlev içerecek şekilde geliştirilir. Uzun zaman tüketebilecek ve sistemin eksik bir işlevlikle çalışabileceği projeler için bu model uygundur.

*• Modelin bazı avantajları:* Müşterilerle birlikte gerekli olan gereksinimler belirlenir. Gereksinimlerin önemine göre teslim edilecek artımlar belirlendikten sonra en önemli gereksinimleri karşılayan çekirdek bir sistem geliştirilir. En önemli sistem özellikleri daha fazla sınanma imkanı bulur. Bu model, böl ve yönet yaklaşımına sahiptir.

*• Modelin bazı dezavatajları:* Artırımların tanımlanabilmesi için tüm sistemin tanımlanmasına ihtiyaç duyulur. Tecrübeli personel gerektirir. Bu model, artımların kendi içlerinde olabilecek tekrarlamalara izin vermez. Gereksinimlerin doğru katmandaki artımlara atanması bazen kolay olmayabilir.

***Araştırma Tabanlı Model***

Yap ve at prototipi olarak da bilinen bu modelin araştırma ortamları tamamen belirsizlik üzerine faaliyet gösteren ortamlardır. Bu nedenle yapılan işten elde edilecek sonuçlar belirsizdir. Model, zaman ve fiyat konusunda kesin bir karar olmadığı için fiyatı sabit olan sözleşmelerin bu modeli kullanması uygun değildir. Gelişimi tamamlanmış yazılımlar genellikle sınırlı sayıda kullanılır ve kullanım sona erdikten sonra yazılım işe yaramaz hale gelir ve böylece yazılım çöpe atılır.

***Kodla ve Düzelt Modeli***

Direkt olarak yazılım ürününün üzerinde işlem yapılmasını sağlayan modelde; sistem, istenilen hale gelene kadar yazılım sürekli olarak geliştirilir. İlk aşamada yazılım ürününün ilk sürümü geliştirilir. Sisteme ait bir belgeleme işlemi olmadığı için bakım safhasının gerçekleştirilmesi oldukça zordur. Birkaç yüz satır koddan oluşturulacak programlar için kullanımı uygundur. Yazılım geliştirmenin en kolay yöntemi olmasıyla birlikte en pahalı yöntemdir. Çünkü şartname ve belgelemenin olmaması ürünün bakımının yapılmasını oldukça zorlaştırır. Bu da daha fazla maliyet anlamına gelir. Yazılım geliştirmenin kolay olması, birçok küçük çaplı firmanın projesinde bu modeli kullanmasının sebeplerindendir.

***Çevik Yazılım Geliştirme***

Çevik, dünyada yazılım süreçlerini daha esnek ve güçlü kılmak için kullanılan, aynı zamanda yazılım süreçlerini de basitleştiren kavramsal bir metodolojidir. Bu metodolojide ölçeğin önemsiz olmasıyla birlikte proje, yinelemelere ayrılır ve bu yinelemeler proje gibi ele alınır. Her yineleme sonunda da müşteri, projenin tamamlanmış kısmı ile ilgili bilgilendirilir. Çevik yazılım ile ilgili her yenileme için yaklaşık 2-4 haftalık planlama yapılır. Bu sistem sayesinde müşteriye sürekli aktif bir yazılım teslim edilir ve müşteri memnuniyetinin artması sağlanır. Çevik yazılımların hızlı olmasının sebebi proje üzerinde çalışan ekip üyelerinin birbirleriyle olan sürekli iletişimidir. Projenin küçük parçalardan meydana gelmesi, geriye dönük hataların daha hızlı ve kolay bir şekilde düzeltilmesini sağlar. Genel olarak çevik yöntemler, hata oranı düşük, yüksek verimli, esnek ve hızlı çözümler sağlar. Bu sistemin avantajlarından kısaca bahsedecek olursak; üretkenlik ve yazılımın kalitesi artarken maliyet düşer. Bu da yazılımı meydana getiren ekibin motivasyonunu yüksek seviyede tutar. En yaygın kullanılan çevik metodolojiler; extreme programming, scrum, agile unified process, feature driven development, test driven development… Bu yazımızda extreme programming ve scrum metodolojilerine değineceğiz.

***Extreme Programming (XP)***

1999 yılında bir yazılım geliştirme disiplini olarak ortaya çıkarılmıştır. Kolay, grup içi iletişimlere olanak sağlayan ve geri dönüşlerin daha sık olmasını sağlayan bir geliştirme yöntemidir. Extreme programing’in dört temel öğesi; iletişim, basitlik, geri bildirim ve cesarettir. Sırasıyla bu dört öğeyi açıklayalım.

• *İletişim:* Pojelerin başarılı olabilmesi için, ekip üyelerinin birbirleriyle sağlıklı bir iletişim içerisinde bulunması gerekir. Ancak projelere bakıldığında genellikle meydana gelen problemlerin insanlar arasındaki anlaşamamazlıktan kaynaklı olduğu görülür. İşte burada XP, bu iletişim sorununu çözmeyi hedefler. Bunun sağlanmasının şartlarından birisi de iletişimin yüz yüze olmasıdır. Ekip ile kullanıcılar arasında sıkı bir iletişimin olması gerekir. Böylece sorunlar erkenden fark edilir.

• *Basitlik:* Basitlik kavramına, gereksinimlerin karşılanması için ortaya atılan en basit çözüm olması yönüyle bakıldığında bu değerin sağlanmasının zor bir konu olduğu anlaşılır. XP için karmaşık çözümlerin işlevsiz olması bu öğrenin değerini belirtir. Bu konuda XP, günün ihtiyaçlarını hedef alır ve basit bir sistem ortaya koymaya çalışarak basitliği en iyi şekilde sağlamaya çalışır.

• *Geri Bildirim:* Geri bildirim ile geriye dönüşler yapılarak meydana gelebilecek hatalar önlenir. Yazılımcılar birim testleri oluşturup sistem hakkında somut bilgiler elde etmeye çalışırlar. Müşteri de proje grubunun bir üyesi olup; müşteri, proje yöneticisi ve diğer çalışanların görüşleri alınır. 2-4 hafta aralıklarla sürümler yayınlanır ve sistemin durumu belirlenir. Müşteriler ile ekip, belirli vakitlerde biraraya gelip yazılımın son halini çalıştırırlar ve gelinen noktaya bakarlar. Ortaya çıkabilecek anlaşmazlıklar önceden giderilmeye çalışılır.

• *Cesaret:* En zor değerdir. Çünkü yılmadan projelerin üzerine gidilmesi, projelerin gelişimi açısından önemlidir. XP, başarısız olmaktan korkmayı değil, böyle bir durumda en kısa zamanda telafi etmeyi amaçladığı için yapılan işlemlerden tatmin olunmaması halinde o işlemler çöpe atılabilmeli ve yeniden yazılabilmelidir. Bu durumda yazılımcının tutumu çok önemlidir çünkü korku, yazılımcıyı yavaşlatır ve projenin gerçekleştirim süresi uzar.

***Scrum***

1990’lı yılların ortalarında geliştirilen ve diğer çevik yazılım metodolojileriyle birlikte uygulanabilecek bir proje yönetim yaklaşımıdır. Kısaca, her yere uygulanabilir bir yöntemdir. Karmaşık bölümleri daha küçük birimlere bölüp geliştirmeyi hedefler. Karmaşık ortamlarda yazılım geliştiren ekiplerin kullanımı için uygundur. Bir yinelenmenin tamamlanması 1 aydan fazla sürmez. Günlük 15 dakikalık toplantılar ile kesintisiz iş takibi yapılır.

• *Product Backlog:* Önceliklendirilmiş yüksek seviyeli bir gereksinim listesidir.

• *Sprint Backlog:* 15-30 günlük proje zaman dilimidir ve projeler birden fazla sprintten meydana gelir.

• *Scrum Daily Meeting:* Her gün takım ile birlikte en fazla 30 dakika ve ayakta olacak şekilde projenin genel durumuyla ilgili bilgi alınması için gerçekleştirilir.

Scrumda 3 temel kavram vardır: Roller (Ürün sahibi, scrum yöneticisi, scrum takımı), toplantılar (Sprint planlama, sprint gözden geçirme, günlük scrum toplantısı), bileşenler (Ürün gereksinim dokümanı, sprint dokümanı, sprint kalan zaman grafiği).

*1.Roller*

• *Ürün sahibi:* Projenin iş değeri açısından geri dönüşü ile sorumlu olan kişidir. Ekibin bir parçası olmakla birlikte müşteri tarafından görevlendrilmiştir. Detayları da takib eder.

• *Scrum yöneticisi:* Takımın dış etkilerden korunmasından ve sadece kendi işine yoğunlaşarak üretkenliğin artmasından sorumludur. Takımı scruma adapte etmekle görevlidir.

• *Scrum takımı:* Kesintisiz bir iletişim halinde olan ve kesinleştirilmiş tek bir hedefe varabilmek için mücadele eden kişilerden oluşur. Sprint hedefine ulaşabilmek için sınırlar dahilinde her şeyi yapmakta özgürlerdir. Ayrıca çalışma sonuçlarını belirli aralıklarla ürün sahibine gösterirler. 5-9 kişiden oluşur.

*2.Toplantılar*

• *Sprint planlama:* Gereksinim listesi çıkarılır. Geliştirmenin başarılı bir şekilde gerçekleştirilebilmesi için dağıtım gereksinimleri belirlenir ve bu dağıtımların gereksinim eşleştirilmesi yapılır. Dağıtımlar için takımlar belirlenir. Gözden geçirmeler yapılır ve olası gereksinim değişiklikleri belirlenir. Dağıtım, geliştirme ve pazarlamanın maliyetleri hesaplanır.

• *Sprint gözden geçirme:* Sprint başlangıcında planlama toplantısı gerçekleştirilir. İlk kısımda takım ile ürün sahibi, gereksinim listesini gözden geçirir ve gereksinim elemanlarının hedeflerini belirler. İkinci kısımda ise takım üyeleri ürün gereksinim lisetesinin en başından başlar ve sprint gereksinim listesini oluşturur. Takım, ürün sahibinin belirttiği önceliklendirilmiş gereksinimlerden ne kadarını yapacağını belirler ve onaylar.

• *Günlük scrum toplantısı:* Sprintin başlangıcından sonra bir başka anahtaar aktivite olan günlük sprint toplantıları gerçekleştirilir. Bu toplantılar 15 dakikalık toplantılardır ve her gün belirlenen vakitlerde gerçekleştirilir. Tüm takım katılım gösterir. Takımın ilerleyişinin ve engellerin görülebilmesi açısından önemli bir fırsattır.

*3.Bileşenler*

• *Ürün gereksinim dokümanı:* Proje boyunca yapılması gereken araçların listesidir. Canlı bir dokümandır ve kullanışlı olabilmesi için devamlı bakıma ihtiyaç duymaktadır. Bu liste, genellikle kullanıcı hikayelerinden oluşur, kullanıcının perspektifinden yararlanılır.

• *Sprint dokümanı:* Sprint için product backlogdan elde edilmiş iş ve görevleri kapsamaktadır. Backlogdaki işlerin hedefi, sprint sonunda son ürünün bir parçası olan bir işlevselliği elde etmektir.

• *Sprint kalan zaman grafiği:* Grafik, sprint boyunca ne kadarını yapıldığı ile normalde ne kadarının yapılması gerektiği ile ilgili bir karşılaştırılma yapılabilmesini sağlar. Takım elemanları günlük gerçekleştirdikleri iş bilgisini sisteme girerler.

**Kaynak**

• <https://caglartelef.com/yazilim-yasam-dongusu/>  
• Doç.Dr. Deniz KILINÇ, Bakırçay Üniversitesi, Yazılım Mühendisliği Temelleri; 1, 2, ve 3. Hafta Ders Sunumları  
• <https://slideplayer.biz.tr/slide/12386328/>

Ahmet AKAN | 190601044