AGİLE (ÇEVİK) YÖNTEMLER

Agile yazılım geliştirme; kendi kendini organize eden takımların, işbirliğine dayalı, yeterli düzeyde resmiyet içeren bir yönetişim çerçevesiyle, zamanında, uygun maliyetli ve değişen paydaş ihtiyaçlarını karşılayan yüksek kalitede çözümler ürettiği yinelemeli ve artımlı bir yaklaşımdır. Çevik yöntemler, geleneksel yaklaşımların yetersiz kaldığı durumlar için alternatif çözümler olarak geliştirilmiştir. HP'nin 2017 yılındaki araştırmasına göre, organizasyonlar yazılım proje yönetiminde ağırlıklı olarak çevik yöntemleri tercih etmektedir. Türkiye'de de AgileTurkey'in yayınladığı Türkiye Çeviklik Raporu'na göre incelenen organizasyonların %72'si çevik yöntemleri kullanmaktadır.

2001 yılında Çevik İttifak tarafından yayınlanan Çevik Bildiri felsefesine göre:

- Süreçler ve araçlardan ziyade bireyler ve etkileşimlere,
- Kapsamlı dokümantasyondan ziyade çalışan yazılıma,
- Sözleşme pazarlıklarından ziyade müşteri ile işbirliğine,
- Bir plana bağlı kalmaktan ziyade değişime karşılık vermeye değer verilir.

Yukarıda temel değerleri belirtilen agile yöntemlerin, bu felsefen doğan on iki ilkesi vardır. Bu ilkeler:

- En önemli öncelik, değerli yazılımın erken ve devamlı teslimini sağlayarak müşterileri memnun etmektir.
- Değişen gereksinimler yazılım sürecinin son aşamalarında bile kabul edilmelidir. Çevik süreçler değişimi müşterinin rekabet avantajı için kullanır.
- Çalışan yazılım, tercihen kısa zaman aralıkları belirlenerek birkaç haftada ya da birkaç ayda bir düzenli olarak müşteriye sunulmalıdır.
- İş süreçlerinin sahipleri ve yazılımcılar proje boyunca her gün birlikte çalışmalıdırlar.
- Projelerin temelinde motive olmuş bireyler yer almalıdır. Onlara ihtiyaçları olan ortam ve destek sağlanmalı, işi başaracakları konusunda güven duyulmalıdır.
- Bir yazılım takımında bilgi alışverişinin en verimli ve etkin yöntemi yüz yüze iletişimdir.
- Çalışan yazılım ilerlemenin birincil ölçüsüdür.
- Çevik süreçler sürdürülebilir geliştirmeyi teşvik etmektedir. Sponsorlar, yazılımcılar ve kullanıcılar sabit tempoyu sürekli devam ettirebilmelidir.
- Teknik mükemmeliyet ve iyi tasarım konusundaki sürekli özen çevikliği artırır.
- Sadelik, yapılmasına gerek olmayan işlerin mümkün olduğunca arttırılması sanatı, olmazsa olmazlardandır.
- En iyi mimariler, gereksinimler ve tasarımlar kendi kendini örgütleyen takımlardan ortaya çıkar.
- Takım, düzenli aralıklarla nasıl daha etkili ve verimli olabileceğinin üzerinde düşünür ve davranışlarını buna göre ayarlar ve düzenler

Çevik yöntemlerde uygulanan yinelemeler, önceki yinelemelerde elde edilen tecrübeler ve tespit edilen aksaklıklar doğrultusunda şekillenir. Yapılması gereken görevler, taşıdıkları iş değerine göre önceliklendirilir. Yapılacak işin en iyi nasıl yapılacağını, mevcut kaynaklar ve kısıtlar çerçevesinde proje takımı kendi belirler. Takım belli görevleri belli sürelerde (yineleme süresi içinde) tamamlamalıdır. Yineleme sonunda teslim edilecek ürünü meydana getirmekten sorumlu olan güçlü ve zayıf yanlarıyla takımdır. Bu nedenle takım içi işbirliği önem arz eder. Bu kapsamda çevik

yöntemlerin dayandığı temel esaslar, deneysellik, önceliklendirme, kendi kendini örgütleme, zaman çerçevesi, iş birliği şeklinde sıralanabilir.

Çevik Süreç Türleri

Çevik süreç türlerinde, süreç içinde aynı özellikler ve uygulamalar gibi benzerlikler paylaşılmaktadır. Ancak gerçekleştirme açısından her sürecin kendine özel bir yapısı, terminolojisi ve taktikleri bulunmaktadır. Bazı süreçlerde proje yönetimi ve işbirliği ağırlıklı çalışma önem kazanırken (FDD, Scrum, ve DSDM), diğer süreçlerde örneğin ekstrem programlamada yazılım geliştirme çalışmalarına odaklanılmaktadır.

Çevik süreçlere ait bazı örnekler aşağıdaki sıralanmıştır.

- Ekstrem Programlama - EP (Extreme Programming)
- Scrum
- Kanban
- Rational Unified Process (RUP)
- Feature-Driven Development (FDD)
- Test-Driven Development (TDD)
- Dynamic System Development Model (DSDM)

Extreme Programming (XP) : Çevik süreçlerden, Ekstrem Programlama doksanlı yılların sonunda Chrysler için Kent Beck, Ron Jeffries ve Ward Cunningham tarafından yapılan bir proje sonrasında oluşmuştur. En popüler ve tartışmalı çevik süreçlerden birisidir. Ekstrem programlama, Scrum temel alınarak geliştirilen bir süreçtir. Scrum, mevcut süreçlerin tersine yazılım metotlarına odaklanmaktadır. Bu nedenle Ekstrem programlama ve Scrum bir projede beraber kullanılabilmektedir. Ekstrem programlama hızlı, disiplinli ve sürekli olarak yüksek kaliteli yazılım teslim etme yaklaşımıdır.

Extreme Programming (XP)'nin Temel Değerleri

• İletişim

• Basitlik

• Geri Bildirim

• Cesaret: Projelerin üzerine yılmadan gidilmesi, projelerin geliştirilmesi açısından son derece önemlidir.

Extreme Programming (XP)'nin Pratikleri

• Planlama Oyunu: Müşterinin belirlediği her bir yineleme için yazılım ekibinin müşterinin de içinde bulunduğu bir toplantıda, o işin ne kadar zamanda yapılacağıyla ilgili kestirimde bulunmasıdır.

• Ekipte Müşteri: İşleri gerçekleştirmek için müşterinin varlığına ihtiyaç duyulmaktadır.

• Önce Test: Kod yazılmadan önce test programı yazılır.

• Basit Tasarım: Müşterilerin gereksinimlerini karşılayacak en basit tasarım gerçekleştirilir.

• Çiftli Programlama: Programın geliştirilme aşamasında her bir programcı sahip olduğu yetenekler ve bilgi birikimleri sayesinde projenin geliştirilmesine yardımcı olur.

• Sürekli Entegrasyon: Yazılım geliştirilirken yapılan sistem değişiklikleri ve yeni bileşenler hemen sisteme entegre edilerek günlük derlemelerle test edilir.

• Kısa Aralıklı Sürümler: Proje birbirinden ayrı zaman aralıklarına (2-4 hafta) bölünür. Her bir zaman aralığında yapılacak işin kendine ait son teslim tarihi vardır. Belirlenen son teslim tarihini aşmayarak tamamlanır ve müşteriye teslim edilir.

• Yeniden Yapılandırma: Kod ve tasarım sürekli gözden geçirilir.

• Ortak Kod Sahiplenme: Geliştirilen yazılım kodu, bütün ekip üyelerinin ortak malıdır.

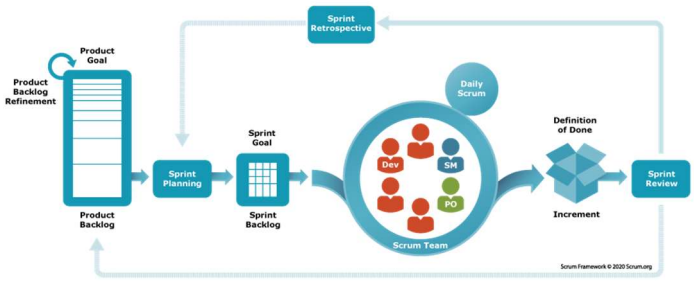
• Benzetim: Gerçekleştirilecek yazılımda sistemler birbirine benzetilerek yazılım geliştirmeye çalışılır.

• Kodlama Standardı: Ekip üyeleri önceden tanımlanmış kodlama standartlarına göre yazılımı gerçekleştirir. Burada amaç, yazılan kodun karmaşıklığını azaltarak bütün ekip üyeleri tarafından kolaylıkla anlaşabilmesini sağlamaktır.

• Haftada 40 saat: Çalışma verimliliği açısından haftada 40 saatlik bir çalışma süresi ayrılır. İşler bu zaman diliminde bitirilir.

Scrum : Çevik süreçlerden Scrum seksenli yıllarda Kent Schwaber ve Jeff Sutherland tarafından geliştirilmiş bir süreçtir. Scrum, Rugby oyununda kullanılan bir terimdir. Scrum, kelime olarak rugby oyununda oluşturulan küçük ekiplere verilen isimdir, oyuncular topu alanda hareket ettirmek için ekip çalışması yaparlar,görevleri bellidir.

Yazılım mühendisliğinde SCRUM sürecinde de aynı mantık vardır. Her ekip çalışanının belirli bir rolü vardır, takım dayanışması mantığıyla süreç ilerletilmektedir.



Scrum daha çok proje yönetim metotlarına odaklanmaktadır. Projenin yapılması için gerekli bilgilerin ayrıntılı olması gerekmez.

Scrum, bir işi yönetmek veya bir ürün geliştirmek için kullanılan artımlı ve iteratif bir süreçtir. Scrum, yazılım geliştirme dışında finansal ve medikal ürünlerin üretilmesinde de kullanılabilir. Proje başlangıcın dışında herhangi bir aşamada veya hata oluşan durumlarda uygulanabilir.

Scrum, gereksinimleri tam olarak belirli olmayan, değişime yönelik, karmaşık yazılım projelerinin yönetimi için uygulanmaktadır.

Scrum sürecini daha iyi anlamak için bazı kavramlar aşağıda açıklanmaktadır

- ➔ Koşu (sprint): Koşu, çevik yazılım geliştirmede iterasyon kavramına karşılık gelmektedir. Scrum'da genellikle koşular 1 ay süren çalışmalarla yapılmaktadır. Koşu sonunda projeye ait alt bir ürün oluşturulur.
- ➔ Müşteri (customer): Proje finanse eden kişi veya kurumdur.
- ➔ Ürün sahibi (product owner): Proje ve gereksinmler hakkında müşteri ile işbirliği yapıp takımı bilgilendiren, ürünün önceliklerini belirleyen kişidir.
- ➔ Ürün talebi (product backlog): Ürün sahibi tarafından belirlenen önceliklere göre sıralı olan proje istekleridir.
- ➔ Koşu talebi (sprint backlog): Bir koşuda, ürün talebi sonucunda yapılacak isteklere bağlı görevlerdir.
- ➔ Scrum uzmanı (scrum master): Uygulamalar konusunda uzman olan proje ekibinin ilerle
- ➔ mesini destekleyen, proje ekibinde problemlere çözüm bulan kişidir.
- ➔ Günlük scrum toplantısı (daily scrum meeting): Günlük olarak gerçekleştirilen ve çok kısa süren (15-30 Dakika) ekip toplantısıdır.

- ➔ Koşu planlama toplantısı (sprint planning meeting): Scrum ekibi ve ürün sahibinin birlikte ürün taleplerini tekrar değerlendirildiği ve isteklerin önem sırasına göre bir sonraki koşu seçilerek düzenlendiği toplantıdır.
- ➔ Koşu değerlendirme toplantısı (Sprint review meeting): Koşu bitiminde olan koşuda üretilen yazılımın değerlendirildiği toplantıdır.
- ➔ Geriye dönük koşu değerlendirme toplantısı (Scrum retrospective meeting): Scrum ekibinde, belirli sürelerde yapılan sonraki koşular için verimliliğin arttırılması için yapılan toplantılardır.

Scrum, 30 gün süreye sahip olan koşulardan oluşmaktadır. Ürün talebinde, her koşu öncesi scrum takımı ve ürün sahibi toplantı yaparak bir araya gelmektedir. Ürün talebindeki değişiklikler toplantıda görüşülerek istenilene uygun öncelikler belirlenmektedir.

Koşu başladıktan sonra amaç, toplantı sonucunda ortaya çıkan istekler ile koşu sonunda çalışan programın oluşturulmasıdır. Takım koşu süresince kendi içinde organize olup kendini iyi yöneten bir çalışma gerçekleştirir. Koşu sırasında proje takımı, günlük, ayaküstü, kısa toplantılarla bir araya gelerek uyumluluğu sağlarlar.

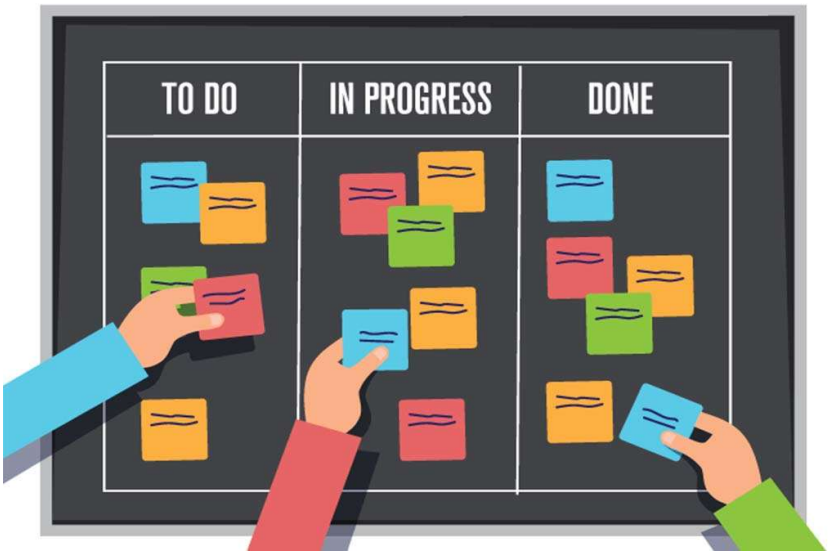
Bu toplantıda proje takımının aşağıdaki soruları cevaplayarak birbirleriyle paylaşması gereklidir.

- Dün nasıl bir çalışma yaptık?
- Bugün nasıl bir çalışma yapacağız?
- Karşılaşılan ya da karşılaşma potansiyeli gördüğümüz problemler nelerdir?

Bu toplantıda proje takımında, takım uyumunun yaratılması ve problemlerin öngörülebilmesi amaçlanmaktadır. Sonraki aşamada, çalışan programı göstermek için proje takımı, ürün sahibi ve müşteri koşu sonunda değerlendirme toplantısı yaparlar. Toplantının amacı, koşu sonuçlarının paylaşılmasıdır. Projeye devam edildiği sürece, yukarıdaki adımlar yeni bir koşu planlama toplantısı ile tekrarlanır.

(SCRUMProfessional Scrum Master™ I Certification = Bir projede profesyonel anlamda scrum master sayılabilmek için ihtiyaç duyulan sertifika.)

Kanban : Kanban, süreçlerin görselleştirilmesi ve sürekli iyileştirme üzerine odaklanan çevik bir metodolojidir. Kanban, üretkenliği artırmak, darboğazları tespit etmek ve süreci şeffaf hale getirmek için adım adım çalışmayı teşvik eder. Özellikle sürekli iş akışı gereken projelerde kullanılır.



Kanban Sürecinin Adımları ve Tahta Düzeni

Kanban tahtası, işlerin hangi aşamalarda olduğunu görselleştirir. Tahta, tipik olarak şu sütunlardan oluşur:

- **To Do (Yapılacaklar):** Tamamlanması gereken işler buraya eklenir. Ekip üyeleri buradan görevleri seçerek çalışmaya başlar.
- **In Progress (Devam Eden):** Üzerinde çalışılan işlerin bulunduğu sütundur. Buradaki işlerin sayısı, WIP (Work In Progress) sınırları dahilinde olmalıdır.
- **Done (Tamamlanan):** Tamamlanan işler bu sütuna taşınır.

Bir Kanban süreci uygularken dikkat edilmesi gerekenler:

- **WIP Sınırlarının Belirlenmesi:** Ekiplerin üstesinden gelebileceği iş yüküne göre WIP sınırları belirlenmelidir. Örneğin, "In Progress" sütununda 3'ten fazla iş olmaması gibi.
- **Görevlerin Güncellenmesi:** İşler ilerledikçe her gün tahtada güncellenmelidir. Her ekip üyesi hangi aşamada olduklarını açıkça görebilmelidir.
- **Darboğazların Tespiti:** Eğer bir sütunda aşırı birikim oluyorsa bu, bir darboğaz olduğunu gösterir. Bu durumda süreç gözden geçirilmelidir.

Kanban Araçları ve Yazılımları

Kanban metodolojisini yönetmek için kullanılabilecek bazı popüler araçlar:

- **Trello:** Basit ve görsel bir yapı sunarak Kanban tahtası olarak kullanılabilir. Sütun ekleme, kartlara yorum yapma gibi özellikleriyle süreç takibini kolaylaştırır.
- **Jira:** Özellikle yazılım projelerinde, iş yükü takibi ve WIP sınırlarının belirlenmesi için sıkça kullanılır.
- **Microsoft Planner ve Monday.com:** Özellikle takım içi iş birliğini artırmak için uygun Kanban araçlarıdır.

Ben projelerimde Trello ve Monday araçlarını kullanmıştım. Oldukça memnun kaldığımı da söyleyebilirim.

Kanban'ın Avantajları ve Zorlukları

Kanban, özellikle esneklik isteyen ekipler için mükemmel bir yöntemdir. Kanban kullanmanın avantajları ve bazı zorlukları şunlardır:

- **Avantajları:**
  - Sürecin görselleştirilmesi, ekip üyelerinin ve paydaşların süreç hakkında hızlı bilgi sahibi olmasını sağlar.
  - Ekipler, WIP sınırları sayesinde iş yükünü dengeleme şansı bulur.
  - Darboğazları erken tespit ederek daha hızlı iyileştirme yapılabilir.
- **Zorlukları:**
  - Kanban, belirli bir süre sınırı olan sprint döngüleri içermediğinden, bazı ekiplerde hız ölçümünü zorlaştırabilir.
  - Gereksiz yük birikimlerini önlemek için ekibin disiplinli bir şekilde WIP sınırlarını takip etmesi gerekir.
  - Kanban, Scrum gibi katı seremoniler barındırmadığından, kendi sürecini yönetme becerisi ve öz disiplin gerektirir.

Rational Unified Process (RUP) : RUP, yazılım geliştirme sürecini düzenlemek ve geliştirmek için kullanılan, IBM'e bağlı Rational Software Corporation tarafından geliştirilen bir yöntemdir. Nesneye

yönelik yaklaşımı benimser ve web tabanlı bir sistemdir. RUP'un amacı, yazılım projelerinde her aşama için rehberlik etmek, şablonlar ve örnekler sunarak süreci verimli hale getirmektir.

RUP, yazılım geliştirme yöntemini 4'e ayırır.

1) Başlangıç (Inception) :
    - Proje gereksinimleri toplanır,kullanıcı ihtiyaçları,problemleri ve hedefleri netleştirilir.
    - Projenin gerçekleştirilip gerçekleştirilemeyeceği analiz edilir (fizibilite çalışmaları yapılır).
    - Riskler ve tahmini maliyetler belirlenir.
2) Detaylandırma (Elaboration) :
    - Gereksinimler ayrıntılı olarak analiz edilir ve netleştirilir.
    - Yazılım mimarisi tasarlanır;hangi teknolojilerin ve araçların kullanılacağı belirlenir.
    - Daha önce tanımlanan riskler ele alınır ve azaltılmaya çalışılır.

    Yani projenin geliştirme aşamasına geçebilmesi için temel oluşturulur. Bu aşamanın sonunda, ayrıntılı bir proje planı oluşturulmuş olur.
3) Yapım (Construction) :
    - Yazılımın kodlaması yapılır.
    - Birimler (modüller) test edilir ve birleştirilir.
    - Gereksinimlere uygun olarak yazılım geliştirilir ve prototipler tamamlanır.
    - Kullanıcılara test ettirilip geri bildirimler alınır.
4) Geçiş (Transition) :
    - Yazılım kullanıcılara tanıtılır, eğitimler verilir.
    - Yazılımın gerçek ortamda kullanılması sağlanır.
    - Kullanıcılardan başarı değerlendirilmesi yapmaları beklenir.
    - Yazılım resmi olarak kullanıma sunulur.

Feature-Driven Development (FDD)

FDD, 1997'de Jeff De Luca ve Peter Coad tarafından, Singapur'da bir finans projesinin zorluklarıyla başa çıkmak için tasarlanmıştır. Özellikle o dönemde büyük ve karmaşık projelerde verimliliği artıracak bir yaklaşıma ihtiyaç duyuluyordu. FDD, o yıllarda modelleme teknikleri ve yazılımın küçük, bağımsız özelliklere bölünmesiyle ön plana çıkmıştır.

FDD, yazılım geliştirme sürecini daha öngörülebilir ve yönetilebilir hale getirir. Yazılım, küçük ve tamamlanabilir işlevsel özelliklere ayrılarak geliştirilir. Bu özellikler, kullanıcı açısından anlamlı sonuçlar üretir. Özellikle karmaşık projelerde ekiplerin üretkenliğini artırır ve yazılımın erken teslimini sağlar.

Projenin başlangıcında detaylı bir **modelleme aşaması** yapılır.

Geliştirme süreci **beş ana aşama** içerir:

1. Genel bir model oluşturma.
2. Özellik listesinin çıkarılması.
3. Özelliklerin planlanması.
4. Özellik tasarımı.
5. Özellik geliştirilmesi.

- Kısa döngüler ve sürekli teslimat prensiplerini uygular.
- Her bir özellik kullanıcı açısından anlamlı olmalıdır.

Avantajları

- Karmaşık projelerin yönetimini kolaylaştırır.
- Özellikler kısa döngülerde geliştirilip kullanıcılara sunulduğu için hızlıca proje teslimi yapılabilir.

- Her bir özellik için sorumlu ekipler belirlenir.

Dezavantajları

- Basit projelerde gereksiz karmaşıklık yaratabilir.
- Özellikle modelleme aşamasında deneyimli yazılımcılara ihtiyaç duyulur.

Test-Driven Development (TDD)

Test-Driven Development, 1990'larda Kent Beck tarafından geliştirilmiştir. İlk olarak Extreme Programming (XP) metodolojisinin bir parçası olarak tanıtılmıştır. Yazılım geliştirmede güvenliği artırma ve hata oranını azaltma hedefiyle oluşturulmuştur. TDD, o dönemde hızlı değişimlere uyum sağlamanın kritik olduğu projelerde başarıyla kullanılmıştır.

TDD'nin temel amacı, yazılım hatalarını minimum seviyeye indirgemektir. **Yazılımcılar, bir işlevi kodlamadan önce onunla ilgili testler yazar**. Bu testler, işlevin doğru çalışıp çalışmadığını kontrol eder ve yazılımın genel güvenilirliğini artırır.

Belirgin Özellikleri

- **Test-First Yaklaşımı:** Kod yazılmadan önce test yazılır.
- **Red-Green-Refactor Döngüsü:**
    1. **Red:** Testler önce başarısız olur (kod henüz yazılmadığından).
    2. **Green:** Testleri geçmek için minimum düzeyde kod yazılır.
    3. **Refactor:** Kod düzenlenir ve optimize edilir.
- Sürekli bir test süreci vardır; bu, yazılımın stabilitesini artırır.

Avantajları

- **Kod Kalitesini Artırır:** Kodun tekrar kullanılabilirliğini ve okunabilirliğini iyileştirir.
- **Hataları Erken Tespit Etme:** Geliştirme sürecinin erken aşamalarında hataların fark edilmesini sağlar.
- **Bakımı Kolaylaştırır:** Refactoring sırasında sorunları minimuma indirir.

Dezavantajları

- **Zaman Alıcıdır:** İlk aşamalarda test yazmak zaman alabilir.
- **Her Projeye Uygun Değil:** Örneğin, araştırma ve prototip geliştirme süreçlerinde pratik olmayabilir.

Lean Development

Lean Development, Toyota'nın Üretim Sistemi'nden esinlenerek geliştirilmiştir. Yazılım dünyasına uyarlanması, Mary ve Tom Poppendieck'in 2000'lerde yayınladığı *Lean Software Development: An Agile Toolkit* kitabıyla başlamıştır. Amacı, üretim sürecinde gereksiz adımları ortadan kaldırmaktır.

Lean Development, atıkları azaltmayı ve müşteri değerini artırmayı hedefler. Yazılım geliştirme sürecinde yalnızca müşteri için değer üreten faaliyetlere odaklanır. Bunun yanı sıra, sürekli iyileştirme ve öğrenme prensiplerini teşvik eder. Lean tarafında kullanılan 3M kuralı vardır.

1. Muda (İsraf)

Muda, süreçte müşteri açısından değer yaratmayan her türlü gereksiz faaliyeti ifade eder. Lean Development'ın ana hedefi, bu tür israfları ortadan kaldırmaktır. İsraf, kaynakların boşa harcanmasına ve sürecin yavaşlamasına neden olur.

2. Mura (Dengesizlik)

Mura, süreçlerdeki dengesizlik veya tutarsızlıkları ifade eder. Proje sürecinde iş yükü eşit dağıtılmadığında veya belirsizlikler ortaya çıktığında Mura oluşur. Bu dengesizlik, kaynakların verimli kullanılmasını zorlaştırır ve süreçte darboğazlar yaratabilir.

3. Muri (Aşırı Yüklenme)

Muri, çalışanlara, ekipmanlara veya süreçlere aşırı yük bindirilmesini ifade eder. Kapasiteyi aşan yüklenmeler, kalite düşüşüne ve süreçlerin bozulmasına yol açar. Muri, uzun vadede çalışanlarda tükenmişlik sendromu veya ekipmanlarda arıza gibi ciddi sorunlar doğurabilir.

Avantajları

- **Daha Hızlı Geliştirme:** Gereksiz süreçler ortadan kalkar, böylece ürün daha hızlı geliştirilir.
- **Maliyet Tasarrufu:** İsrafın azaltılması maliyetleri düşürür.
- **Esneklik:** Değişen müşteri gereksinimlerine hızla uyum sağlar.

Dezavantajları

- **Deneyimli Ekip Gereksinimi:** Sürecin etkili bir şekilde uygulanabilmesi için deneyimli bir ekip gereklidir.
- **İlerleme Ölçümü Zor:** Soyut hedefler, sürecin başarısını ölçmeyi zorlaştırabilir.

Dynamic Systems Development Method (DSDM)

DSDM, 1994 yılında Rapid Application Development (RAD) metodolojisinin bir uzantısı olarak geliştirilmiştir. Amacı, yazılım geliştirme süreçlerinde daha esnek ve hızlı bir yapı oluşturmaktı. Bu metodoloji, DSDM Consortium tarafından sürekli olarak güncellenmiştir.

DSDM, hızlı prototipleme ve sürekli teslimat yöntemlerini kullanarak yazılım geliştirme sürecini optimize eder. İş gereksinimlerini hızlıca karşılayan bir yöntemdir. "Zaman kutusu" yaklaşımı sayesinde, her iterasyon önceden belirlenen bir süre içerisinde tamamlanır.

Belirgin Özellikleri

- **MoSCoW Önceliklendirme:** Gereksinimler "Must Have", "Should Have", "Could Have" ve "Won't Have" olarak sınıflandırılır.
- **Zaman Kutusu (Time-Boxing):** İterasyonlar belirli bir zaman diliminde tamamlanır.

Avantajları

- **Hızlı Teslimat:** İterasyonlar kısa sürede tamamlanır ve değerli çıktılar sunulur.
- **Müşteri Odaklılık:** Müşteri gereksinimlerine öncelik verir.
- **Esneklik:** Değişen ihtiyaçlara kolayca adapte olur.

Dezavantajları

- **Karmaşık Yapı:** Küçük projeler için fazla karmaşık olabilir.
- **Eğitim Gereksinimi:** DSDM'nin tam anlamıyla uygulanabilmesi için ekiplerin bu metodolojiyi iyi bilmesi gerekir.

KAYNAKÇA

Ç.YAŞAR, "Yazilim-Muhendisligine-Giris" Github, [Online]. Erişim Linki: https://github.com/cyasar34/Yazilim-Muhendisligine-Giris. Son Erişim Tarihi: 21.11.2024

İren, E., & Kantarcı, A. SCRUM Yazılım Geliştirme Metodu Üzerine Bir İnceleme ve Değerlendirme.

Gencer, C., & Kayacan, A. (2017). Yazılım proje yönetimi: Şelale modeli ve çevik yöntemlerin karşılaştırılması. Bilişim Teknolojileri Dergisi, 10(3), 335-352.

https://www.linkedin.com/pulse/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-ve-modelleri-ekinsu-o%C4%9Fuz/

https://www.linkedin.com/pulse/yaz%C4%B1l%C4%B1m%C4%B1n-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-beyza-nur-karako%C3%A7/

https://dn790001.ca.archive.org/0/items/bme-vik-konyvek/Software%20Engineering%20-%20Ian%20Sommerville.pdf

https://www.linkedin.com/pulse/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-ve-modelleri-ramazan-arda-a%C5%9Fc%C4%B1/