YAZILIM YAŞAM DÖNGÜ MODELLERİ

BEYZA YONTUCU 220601080

Bakırçay Üniversitesi

Mühendislik ve Mimarlık Fakültesi

Bilgisayar Mühendisliği

E-Mail: 220601080@bakircay.edu.tr

İÇİNDEKİLER

ÖZET

Bu makale kapsamında yazılım yaşam döngüsü kavramının, yazılım yaşam döngüsünün adımlarının, bu kavrama bağlı olarak literatürde geçen yazılım yaşam döngüsü modellerinin açıklanması ve modellerin karşılaştırılması hedeflenmektedir. Ayrıca makalede Düzenleyici Süreç Modelleri (Kodla ve Düzelt Modeli, Gelişigüzel Model, Barok Modeli, Çağlayan Modeli ve Spiral Model), Birleşik Süreç Modeli ve Çevik Yazılım Süreci Modelleri (Scrum) açıklanmıştır.

Yazılım yaşam döngüsü, herhangi bir yazılımın geliştirilmesi, onarımı ve bakımı için uygulanan adımlardır. Yazılım dünyasında kullanılan birden çok döngü modeli vardır. Bu döngü modelleri üç başlık altında incelenir: Düzenleyici Süreç Modelleri, Birleşik Süreç Modeli ve Çevik Yazılım Süreci Modelleri. Düzenleyici Süreç Modellerine Çağlayan Döngü Modeli, Kodla ve Düzelt Modeli, V Süreç Modeli, Helezonik (Spiral) Model ve Artımsal Geliştirme Döngü Modeli örnek verilebilir. Çevik Yazılım Modelinde en çok kullanılan yöntem Scrum Modelidir. Her bir döngü modelinin kendi içinde artıları ve eksileri bulunmaktadır. Bu yazıda ayrıtılı bir şekilde işlenmiştir.

1. YAZILIM YAŞAM DÖNGÜLERİ

Yazılım yaşam döngüsü; herhangi bir yazılımın başta üretim ve kullanım dahil olmak üzere hayatı boyunca geçirdiği tüm aşamaların kapsamlı bir şekilde tanımlandığı bir süreçtir. Bu süreç yazılımın gereksinimlerini planlamadan başlayarak analiz-çözümleme, tasarım, gerçekleştirme ve bakım aşamalarını içerir. Yazılım yaşam döngüsü; gelişen teknoloji, artan talepler, müşteri ihtiyaçları ve değişen dünya sorunları sebebiyle evrelerinin tek yönlü değil, bir döngü şeklinde gerçekleştirilmesi gerekmektedir.

2. YAZILIM YAŞAM DÖNGÜLERİNİN ADIMLARI

Yazılım yaşam döngüsünü ilk aşamasını gereksinimleri oluşturma yer alır. Gereksinim ve planlama döngünün en önemli aşamasıdır. Temel gereksinimler bu aşamada belirlenir. Bu aşamanın yetersiz olması veya atlanılması durumunda projelerin başarısız olma ihtimali yükselir. Üretilecek yazılım ürünü ile ilgili müşteri, personel ve donanım gereksinimleri elde edilir ve fizibilite çalışmaları yapılır. Daha sonra analiz-çözümleme evresine geçilir. Analiz evresinde proje ile ilgili bütün fonksiyonlar ayrıntılı bir şekilde belirlenir. Temel sorunları ortaya çıkararak yazılımın çözümleyebilecekleri vurgulanır. Analiz aşamasından sonra tasarım evresine geçilir. Burada analiz evresindeki sonuçlar dikkate alınır. Projedeki işlemler ve gereksinimler teker teker belirlenip proje alanı oluşturulur ve tasarım dokümanı hazırlanır. Proje için mantıksal ve fiziksel tasarımlar bu evrede meydana getirilir. Proje tasarımı oluşturulduktan sonra gerçekleştirme evresine geçilir. Bu evre; kodlama, test etme ve kurulum çalışmalarının yapıldığı aşamadır. Yazılımcılar hazırlanılmış dokümanları inceleyerek ürün ile ilgili gerekli kodlamaları yaparlar ve kurulum çalışmalarını tamamlarlar. Yapılan kodlamaların sorunsuz bir şekilde çalışabilmesi için bu aşamadan önce hazırlanan dokümanlar ve raporlar dikkatli bir şekilde incelenmeli ve birleştirilmelidir. Yazılan kodlar da test edilerek hatalar ayıklanmalı, düzeltilmeli ve eksik yanlar eklenmelidir. Bu aşamadan sonra proje artık hayata geçen bir yazılım haline gelir. Üretilen yazılım ürünün tüm aşamaları tamamlanıp teslimi yapıldıktan sonra bakım safhasına geçiş yapılır. Projenin bakımı yapılırken farklı aşamalardan geçirilir. Bu safhada ürün ile alakalı geri bildirimler değerlendirilir, değiştirilmesi gereken durumlar değiştirilir ve hatalar düzeltilir. Yeni talepler istenirse bu yönde güncellemeler yapılır. Bu durumda güncellemeler ve değişiklikler için sırasıyla planlama, analiz, tasarım, gerçekleştirme ve bakım aşamaları tekrardan ele alınır. Bu sebeple yazılım süreci devam eden ve bitmeyen bir döngü sürecidir.

3. YAZILIM YAŞAM DÖNGÜ MODELLERİ

Yazılım yaşam döngüsü, bir yazılımın başlangıcından sonuna kadar olan tüm aşamaları kapsayan bir süreçtir. Yazılım yaşam döngü modelleri, yazılımın üretim ve kullanım aşaması dahil olmak üzere her aşamasını ayrıntılı olarak ele alan ve her aşamada özel gereksinimleri tanımlayan yöntemlerdir. Modeller; geliştirme ekibinin yazılımın her aşamasının daha iyi planlanmasına, yönetmesine ve kontrol etmesine

yardımcı olur. Bu yüzden yazılım yaşam döngüsünün daha iyi sonuç verebilmesi için döngü modellerinin kullanımı önem arz etmektedir. Bu yazılım geliştirme modelleri üçe ayrılır: Düzenleyici Süreç Modelleri, Birleşik Süreç Modeli ve Çevik Yazılım Süreci Modelleri.

3.1. Düzenleyici Süreç Modelleri

Düzenleyici Süreç Modelleri, yazılımın üretiminin yapılma düzeni için yol göstericiler olarak tanımlanır. Bununla ilintili olarak bu makalede Kodla ve Düzelt Modeli, Gelişigüzel Model, Barok Modeli, Çağlayan Modeli ve Spiral Model ele alınacaktır.

3.1.1. Kodla ve Düzelt Modeli

Kodla ve Düzelt Modeli, en kolay yazılım geliştirme süreci olarak kabul edilir. Herhangi bir konu üzerinde araştırma yapmak veya bir problemi çözmek için kullanılır. En basit ürün geliştirme yöntemi olarak isimlendirilen bu modelde yapılabilecek minimum sürede prototip oluşturmak hedeflenir. Küçük projeler veya kısa ömürlü prototipler için uygun bir modeldir. Uzmanlık gerektirmediği için çoğu kişi bu modeli kullanabilir. Örneğin bilgisayar mühendisliği bölümü üniversite öğrencileri, kodlama veya yazılım ile ilgili bir proje ödevinde bu modeli kullanabilirler. Eğer ürün, onu yapan bireysel geliştiriciler tarafından kullanılacaksa avantajlıdır. Fakat dokümantasyonu olmadığı için hataların bulunması zordur ve kod birbirini takip eden değişikliklerden sonra karmaşık bir hale gelir bu yüzden bakım yapılabilirliği düşüktür.

3.1.2. Gelişigüzel Model

Gelişigüzel kelimesi Türkçede rastgele veya plansız anlamlarına gelir. Bu bağlamda “Gelişigüzel Yazılım Modeli” terimi, bir planlama veya yönetim olmaksızın yazılım geliştirme sürecinde yöntemsiz bir yaklaşımı ifade eder. 60’lı yıllarda kullanıma geçen bu model, herhangi bir kuraldan bağımsız ve sadece onu geliştiren kişiye bağlı olduğundan yazılımın izlenilebilirliği ve bakım yapılabilirliği oldukça zordur. Okunurluğu ise düşük olur. Bu süreç modeli daha çok tek kişilik projeler, basit öğrenci projeleri veya prototipler için uygun olabilir ancak daha büyük ve karmaşık projelerde kullanılması tavsiye edilmez. Çünkü projenin daha da karmaşıklaşmasına; para, zaman ve kalite kaybına yol açabilmektedir.

3.1.3. Barok Modeli

Barok yazılım modeli, yazılım tasarımında ve mimarisinde kullanılan bir terimdir. Bu model yazılım yaşam döngüsünün gerekli adımlarını doğrusal bir şekilde ele alır ve geliştirir fakat aşamalarının geri dönüşlerinin nasıl yapılacağı belirsizdir. Kullanım kılavuzu gerektiren basit projelerde kullanılan bir modeldir. 70’li yılların ortalarında kullanıma başlanılan bu eski model, dokümantasyonu günümüz modellerinden ayrı bir süreç olarak ele alır ve bu belgeleme işlemini yazılan programın test aşamalarının

bitiminden sonra yapar. Bunun gibi faktörlerden dolayı günümüzde pek tercih edilmeyen bir modeldir.

3.1.4. Çağlayan Modeli

Yazılım mühendisliğinde ilk anlatılan modellerden birisi olan ve geleneksel yazılım geliştirme modeli olarak da bilinen Çağlayan Modelinde yazılım geliştirme süreci; analiz, tasarım, kodlama, test ve bakım gibi evrelerden oluşur ve bu evreler en az bir kez tekrar edilerek yazılımın geliştirilmesi sağlanır. Bu evreler lineer bir şekilde uygulanır. Bütün işlemler aşama aşama yapılır ve biri bitmeden diğerine geçilmez. Genelde hastane ve kontrol sistem projelerinde bu modele başvurulur. Barok Modelinin aksine dokümantasyon işlemin doğal bir parçası olup başından sonuna kadar devam eder ve geri dönüşlerin nasıl yapılacağı belirlidir. Herhangi bir aşamada belgelendirme ve test işlemi yapılmamışsa o aşamanın tamamlandığı kabul edilmez. Kullanımı, anlaşılması ve yöntemi basittir. Proje safhaları ayrı olduğundan safhalar nettir ve bu durum iş bölümü ile proje yönetimini oldukça kolay hale getirmekle kalmayıp müşteriler için de avantaj sağlar. İterasyonlar (tekrarlamalar) bir önceki ve bir sonraki adımlarla gerçekleştirilir, daha uzak adımlarla olması pek beklenen bir durum değildir. Gereksinimleri iyi tanımlanmış, küçük ve yapımı az zaman gerektiren projeler için uygundur. Fakat Çağlayan Modeli statiktir bir diğer deyişle değişimlere elverişsizdir çünkü günümüzde yapılan çoğu proje yineleme gerektirir. Bu yüzden büyük ve karmaşık yazılım projelerinde yeni fikirlerin ve farklı ihtiyaçların ortaya çıktığı durumlarda geri dönüşler zordur. Örneğin analiz aşamasında yapılan bir hata; tekrardan analiz aşamasına gidilip bu aşamadaki analizlerin yeniden yapılmasına, bu doğrultuda tasarımın, kodlamanın ve testin değiştirilmesine neden olacaktır. Bu da ciddi zaman ve maddiyat kaybına yol açacaktır. İki ya da daha önceki fazlara gitmek de çok maliyetli olduğu gibi bir fazın tamamlanmadan diğerine geçilmesi hata oranını artıracaktır. Öte yandan Çağlayan Modeli basitliği ve temel bir model oluşu sayesinde diğer modellere bir zemin olma açısından önemlidir ancak günümüzde kullanımı gitgide azalmaktadır.

3.1.5. Spiral Model

Karmaşık ve büyük projelerde gereksinim analizinin ve planlamanın yapımı çok zahmetlidir ve pürüz çıkma ihtimali yüksektir çünkü birden fazla risk faktörü vardır. Bu sebeple Spiral (Helezonik) Model, proje yapım aşamasında oluşabilecek riskleri baz alarak geliştirilmiş bir modeldir. Bu model, riski adım adım azaltarak projeyi başarılı bir şekilde tamamlamayı amaçlamaktadır. Proje çevrimlere ayrılır ve her bir çevrimin riskleri ayrı ayrı değerlendirilir. Spiral Model; planlama, risk analizi, üretim ve kullanıcı değerlendirmesi olmak üzere dört aşamadan oluşur. Bu aşamalar spiral oluşturacak şekilde dönerek tekrar eder ve bir döngü yaratır. Her döngü bir fazı simgeler. Modelin planlama evresinde müşteriden alınan gereksinim bilgileri ışığında amaçlar belirlenir ve bu amaçlara çözüm önerileri getirilir, önceki adımlarda üretilen ara ürünler birleştirilir. Yapılan planlamadan sonra risk analizine başlanır. Potansiyel riskleri

belirlemek için gereksinimler ve planlamalar incelenir, risk araştırması yapılır. Riskler belirlendikten sonra risk azaltma stratejisi oluşturulur ve belgelenir. Bu aşamadan sonra prototip üretim safhasına geçilir. En sonunda oluşturulan prototipler müşteriye kullanıcı değerlendirmesi olarak sunulur ve geri bildirimlerin alınması sağlanır. Geri bildirimler için her döngünün sonunda yeniden planlama yapılarak riskler ve hedefler yeniden hesaplanır. Ayrıca Çağlayan Modelinde yok sayılan riskleri de göz önünde bulundurur. Modelin en faydalı özelliklerinden birisi prototiplerin sayesinde kullanıcıların sistemi erkenden görebilmesidir. Projenin boyutuna göre prototip sayısı artırılıp azaltılabilir. Risklere karşı duyarlı yaklaşımı doğabilecek güçlükleri engeller. Hataları az zamanda giderme şansı tanır. Yapılan her döngü aşamasını müşteri net bir şekilde görebildiği ve anlayabildiği için ihtiyaçları ve talepleri belirlemek daha kolay bir hal alıp zaman kazandırır. Yazılım ve donanım sistemini geliştirmek için uygun bir çerçeve sağlar. Bu avantajlar sayesinde Spiral Model; büyük planlamalarda, yazılımın devamlı risk değerlendirilmesine gerek duyulduğu yerlerde, gereksinimlerin karmaşık ve açıklama gerektirdiği projelerde kullanılır. Özellikle güvenlik yazılımlarının oluşturulmasında ve askeri savunma yazılım sistemlerinde tercih edilir. Diğer yandan, risk analizi önemli bir aşama olduğundan uzman kişilere ihtiyaç duyulur. Ara adımların fazlalığı nedeniyle çok fazla dokümantasyon gerektirir. Gereksinimler devamlı değerlendirildiği için spiral sonsuza kadar devam edebilir. Bu nedenle düşük riskli ve küçük projeler için pahalı bir yöntemdir. Çünkü sürekli prototip çıkarmak ve belgeleme yapmak gerekir. Öznel risk değerlendirme metoduna dayanır. Bu durum ise zaman açısından dezavantaj oluşturur.

3.2. Birleşik Süreç Modeli

Bu model, yazılım geliştirme sürecindeki farklı evreleri ve bu evrelerin birbiriyle nasıl ilişki içerinde olduğunu gösterir ve yazılım geliştirme süreçlerinin en iyi özelliklerini bir arada toplayıp bütünleştirir. Nesneye dayalı yazılım geliştirmek için kullanılan bir yöntemdir. Çevik yazılım geliştirme yöntemleri ile uyumlu olarak tasarlanmıştır. Fonksiyonel alanlardan ve değişik disiplinlerden oluşan insanları bir araya getirerek takım yaklaşımını destekler. Bu sayede Birleşik Süreç Modeli; projelerin daha esnek, verimli ve hızlı bir şekilde geliştirilip tamamlanmasına yardımcı olur.

3.3. Çevik Yazılım Süreci Modelleri

Yazılım geliştirme süreci uzun, yorucu ve karmaşık bir süreçtir. Müşteri isteklerine hemen dönüş yapılamaması, yazılım hatalarının geç fark edilmesi ve sistemin gelişen teknolojiye ve zamana göre kendini geliştirememesi büyük bir sorundur. Yazılım sürecinin bu sorunlarına çözüm bulmak için Çevik (Agile) Modelleri 1990'lı yılların sonunda geliştirilmiştir. Bu modeller, yazılım geliştirme sürecinde kullanılan esnek ve müşteri odaklı yaklaşımlardır ve oluşabilecek belirsizlikleri yönetmek için adaptasyon yeteneği sağlar. Çevik Yazılım Süreci Modelleri; müşteri memnuniyetini ve yazılım kalitesini artırmayı, ürünü hızlı bir şekilde çıkarabilmeyi ve olabilecek minimum sürede

yazılım ürününün alıcıya ulaştırmayı hedefler. Bu hedefler doğrultusunda hata oranı düşük, esnek, verimliliği fazla ve maliyeti az çözümler sağlamaktadır. Projenin boyutu gözetilmeksizin proje küçük iterasyonlara (tekrarlamalara) ayrılır ve müşteriye projenin gelişimi ile ilgili bilgi aktarılır. Takımlar kendi aralarında organize olurlar ve bu sayede devamlı iletişim halinde olarak projenin yönetimine ve zamanına katkı sağlar. Çünkü takım içerisinde etkili ve doğru bilgi akışı önem arz eder. Çevik Modelleme; değişime açıklığı ve esnekliği, iterasyon içeren planlamaların yapılması ve planla ile yürütme aşamalarının bir arada oluşu sayesinde avantaj sağlar. Bir diğer artısı ise diğer metotlar gibi zaman dilimini ay bazında değil hafta bazında değerlendirir. Bu da zaman tasarrufu sağlar. Her ne kadar takım birliği faydalı olsa da bazı durumlar takım üzerinde hedef baskısına yol açabilmektedir. Bu durum kurumsal yapıda uygulanmasını zorlaştırır. Devamlı değişiklik gösteren ihtiyaçlar nedeniyle çalışma süreçlerinin artması da bir başka dezavantajlarındandır.

Çevik Yazılım Süreci Modellerinde en yaygın uygulanan metodolojiler: Extreme Programming (XP), SCRUM, Lean Yazılım Geliştirme, Test Güdümlü Geliştirme ve Özellik Güdümlü Geliştirmedir. Bununla ilintili olarak bu makalede SCRUM modeli ele alınacaktır.

3.3.1. SCRUM

Scrum, günümüzde kullanılan en popüler Çevik (Agile) Yazılım Geliştirme yöntemidir. Bir projeyi sunmak için tasarlanmış, hızlı, esnek, tekrarlı ve etkili bir metodolojidir. Popüler olmasının sebebi, bu metodolojinin sadece yazılım geliştirme için değil, her proje için başvurulabilir olmasıdır. Scrum Modelinin şeffaflığı, hızlı sorun çözümü, uyarlanabilirliği, etkili çıktıları ve yüksek hızı da popüler olmasına katkı sağlamıştır. Scrum Modeli, kompleks yazılım süreçlerinin yöneltilmesi için kullanılır. Bunu yaparken komplike yazılım işlemlerini küçük birimlere (sprint) böler ve düzenli geri bildirim ve planlamalarla hedefe ulaşmayı amaçlar. Bahsedilen karmaşıklığı şeffaflık, denetleme ve uyarlama ilkeleriyle minimuma indirmeye çalışır. Bu yöntemleri sayesinde, gereksinimlerinin net bir şekilde tanımlanamadığı ve karmaşık yapılı projelerde kullanışlı bir modeldir. Yapılan günlük kısa süreli toplantılar sayesinde iş takibi güncel tutulur ve iterasyonun tamamlanması genelde otuz günlük süreyi aşmaz.

3.3.1.1 Scrum Modelindeki Kavramlar

Scrum Modeli üç temel kavramda ele alınır. Bunlar roller, toplantılar ve bileşenlerdir. Rolleri ürün sahibi, Scrum yöneticisi ve Scrum takımı oluşturur. Ürün sahibi proje ile ilgili stratejik üretim kararlarını vermekten ve iş değeri bakımından geri dönüşü ile sorumludur. Müşteri tarafından görevlendirilmiş olan ürün sahibi, geliştirme ekibinin işletmeye en iyi verimi sunduğundan emin olmaya odaklanır. Kullanıcının işlevselliğini baz alarak değişen ihtiyaçları değerlendirir ve öncelik sırasına koyar. Scrum yöneticisi ekibin verimliliğinden sorumludur. Ürün sahibine iş listesi maddelerinin nasıl oluşturulacağını öğretir. Her bir sprint için gerekli kaynakları planlar, takımı Scrum'a adapte eder, takımın verimliliğini artırır ve yüksek değerli ürün

oluşturmaya yardım eder. Scrum takımı ise test edicilerden, operasyon mühendislerinden, tasarımcılardan ve geliştiricilerden oluşur. Ürün sahibinin talepleri doğrultusunda doğru ürünü geliştirirler. Scrum takımı devamlı iletişim halinde olup bireysel sorumluluklar kabul edilmez, doğru ya da yanlış yapılanlar tüm takımın sorumluluğundadır. Genellikle 5 ile 9 kişiden oluşan takım, sprintlerin başarılı bir şekilde tamamlanması için iş birliğinde olur ve sürdürülebilir geliştirme programlarını destekler. Scrum Modelinde toplantılar sprint planlama, sprint gözden geçirme ve günlük Scrum toplantısı kısımlarından oluşur. Sprint planlamada takım bir sonraki sprint için tamamlanacak işler konusunda tahminlerde bulunur. Geniş kapsamlı gereksinim listesi çıkarılır ve riskler değerlendirme süzgecinden geçirilip kontrolleri belirlenir. Sprint, Scrum takımının bir ürün kısmını tamamlamak için ekipçe çalıştıkları süredir. Bir sprint çoğunlukla 2 hafta sürer fakat projenin boyutuna göre değişiklik gösterir. Sprint gözden geçirme safhasında takım üyeleri sprint gereksinim listesini oluşturur ve bu liste ürün sahibi tarafından değerlendirilir. Günlük Scrum toplantısı yapılarak takımın ilerleyişi değerlendirilir, üyeler tamamlanan işleri bildirir ve sprint amaçlarına ulaşmada yaşanan zorluklar dile getirilir. Bileşenler safhasında proje süresince yapılması gerekenler ürün gereksinim dokümanı adı altında toplanır. Bu doküman, kullanıcı hikayelerinden oluşur ve devamlı bakım gerektirir. Sprint iş listesi (sprint backlog) ürün iş listesi kalemlerini ve sprint hedeflerine ulaşma planını içerir. Sprint kalan zaman grafiğinde de tamamlanacak işlerin aldığı süre ile normal şartlarda alması gereken zamanın kıyaslanması sağlanır.

4. YAZILIM YAŞAM DÖNGÜLERİNİN KARŞILAŞTIRILMASI

Yazılım yaşam döngü modelleri birbirlerinden farklı avantaj ve dezavantajlara sahiptir. Spiral model ve Çağlayan modeli maliyet açısından yüksektirler. Öte yandan Spiral model karmaşık bir yapıya sahipken Çağlayan modeli daha basittir. Kodla ve Düzelt modelinde esneklik payı az iken Spiral model çok esnektir. Çevik modeller başarı garantisi açısından en yüksek seviyelerden birine sahipken uzmanlık gerekliliğinin de çok yüksek olması bu modele dezavantaj sağlar.

5. SONUÇ

Bu çalışma kapsamında, yazılım proje yönetimi açısından önemli bir yere sahip olan yazılım yaşam döngüleri olarak geçen kavramlar incelenmiştir. Kavramın, yazılım proje yönetim modelleri ile olan ilişkisi incelenmiş olup modellerin karşılaştırılması yapılmıştır.

Kaynakça

1.https://www.geeksforgeeks.org/software-engineering-spiral-model/

2.https://talentgrid.io/tr/yazilim-gelistirme-modelleri/

3.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

4.https://idealkoc.com/sik-sorulan-sorular/neden-scrum/#:~:text=Scrum%2C%20en%20pop%C3%BCler%20Agile%20(%C3%87evik,ile%20d%C3%BCzenli%20ilerleme%20ortam%C4%B1%20yarat%C4%B1r.

5.https://slideplayer.biz.tr/slide/12237711/

6.https://productcoalition.com/the-code-and-fix-model-2cabd4c48166

7.https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/

8.http://www.yilmazcihan.com/scrum-toplantilari-ve-rituelleri/

9.https://www.nimblework.com/agile/scrum-methodology/

10.https://www.scrumalliance.org/about-scrum

Hesaplar

Github: https://github.com/beyzayontucu

LinkedIn: https://www.linkedin.com/in/beyza-yontucu-b8638325a/

Medium: https://medium.com/@beyzayontucu