

Yazılım Yaşam Döngüsü (SDLC) ve Modern Yaklaşımlar

Bilgisayar Mühendisliği Yüksek Lisans Programı
2026 Bahar

Dr.Öğr.Üyesi Zeynep Altan

Yazılım Mühendisliğinin Kapsamı

- Yazılım Mühendisliği aşağıdaki süreçleri gerçekleştirmek üzere bir dizi *bilgi, geliştirme araçları ve metotlardan* oluşmaktadır.
 - ❖ Yazılım Gereksinimleri (Software Requirements)
 - ❖ Yazılım Tasarımı (Software Design)
 - ❖ Yazılımın İmplementasyonu (Software Implementation)
 - ❖ Yazılım Testi (Software Testing)
 - ❖ Yazılımın Sürekliliğinin Sağlanması (Software Maintenance)

Yazılım Proje Yönetimi

- ❑ Bir yazılım projesinin hedeflerine ulaşabilmesi için zaman, maliyet ve kalite faktörlerinin önemsenmesi, mühendislik çözümlerinin doğru olarak planlanması, organizasyonu, kontrolü ve denetlenmesi ile projeye sunulan katkı proje yönetimidir.
- ❑ Bilgi teknolojilerindeki hızlı gelişim ve müşteri isteklerindeki değişimler mevcut projelerde de değişiklik gerektirmektedir.
 - ❖ Ayrıca önceki projeler uygulanırken kazanılmış deneyimler de artık yeterli gelmemektedir.
- ❑ Yazılım proje yönetimi metodolojisinin **amacı**; bir yazılım projesine ait ürünün belirli bir yöntemle iyi yönetilerek elde edilmesi; diğer bir ifade ile kaliteli bir ürün teslimi sağlamak üzere belirli bir zaman ve bütçe kısıtıyla geliştirmeyi tamamlamaktır.

Yazılım Geliştirme Metodolojileri

- ❑ Bir yazılım sisteminin geliştirilmesi sürecinde ürünün yapımını, planlamasını ve kontrolünü sağlayan bir altyapıdır.
- ❑ Her farklı altyapı zayıf ve güçlü yanlarıyla zaman içerisinde olgunlaşır; teknik, yönetsel yapı ve geliştirme takımına göre farklı özelliklere sahip olur.
- ❑ Özellikle karmaşık ve büyük bir sistem geliştirmek için, birçok işin (task) gerçekleştirilmesi gerekir. Bu da uzun ve karmaşık bir süreçtir.
 - ❖ Bu nedenle geliştiricilerin birbirleriyle daha koordineli çalışması gerekebilir.
- ❑ Metodoloji, süreçleri küçük görevlere ayırarak, ne yapılması gerektiğini belirterek proje yönetimine, planlamaya, çizelgelere, ilerlemeyi izlemeye yardım etmektedir
- ❑ Metodoloji, müşterinin isteklerine göre, projenin başarısızlığa uğramaması adına geliştiricilerin başlanacak yeni projede ya da mevcut projenin geliştirilmesinde izleyeceği yol haritasıdır.
 - ❖ Bu nedenle herhangi bir sistem geliştirme metodolojisi her proje için uygun olmayabilir.

Yazılım Geliştirme Metodolojilerinin Tarihsel Gelişimi 1970-2010

Yıl	Yazılım Geliştirme Metodolojileri
1970	1969'dan beri yapısal programlama
1980	1980'den itibaren yapısal sistem analizi ve tasarım yöntemi (SSADM) Bilgi İhtiyaç Analizi / Yazılım sistemleri metodolojisi
1990	“Nesne yönelimli programlama (Object Oriented Programming- OOP)” 1960'ların başlarında geliştirildi ve 1990'ların ortalarında baskın bir programlama yaklaşımı haline geldi. 1991'den beri hızlı uygulama geliştirme (Rapid Application Design-RAD) Dinamik sistem geliştirme yöntemi (DSDM), 1994'ten beri Scrum , 1995'ten beri kullanılmaktadır. 1998'den beri takım yazılımı süreci (Team software process) 1998'den beri IBM tarafından sürdürülen Rational Unified Process (RUP) Uç Programlama (Extreme programming)
2000	Çevik Birleştirilmiş Süreç (Agile Unified Process , AUP), 2005'ten bu yana Scott Ambler tarafından sürdürüldü. Disiplinli çevik teslimat (Disciplined agile delivery- DAD)
2010	Ölçekli Çevik Çerçeve (Scaled agile framework- SAFe) Büyük Ölçekli Scrum (Large-Scale Scrum -LeSS)

Klasik (Traditional) Yazılım Geliştirme Metodolojileri

- ❑ 1950'li yıllarda ilk yazılım geliştirme yöntemi, “kodla ve düzelt” , diğer ifadesi ile ad hoc / kovboy programlama idi.
- ❑ Problemin çözümünde herhangi bir tasarım veya planlama aşaması yoktu.
 - ❖ Kullanıcının ihtiyaçları kodlanmakta, işleyiş sırasında veya ürünün tesliminden sonra ortaya çıkan hatalar düzeltilmekteydi.
- ❑ Zaman içinde yazılım projeleri büyüdü ve bir proje yönetim metodolojisine ihtiyaç duyuldu. Kritik sistemlerde hatalara tolerans kalmadı.
 - ❖ Teknolojinin gelişmesiyle birlikte bilgisayar kapasitelerinin hızla artması ve kaliteli yazılım geliştirilememesi nedeni ile 1960'lı yıllarda yaşanan **yazılım krizi** sonucunda Almanya'da yazılım ürün geliştirilmesi yeniden tartışılmıştır (NATO Konferansı 1968) yılında
 - ❖ 1970 yılında Şelale (Waterfall) Modeli tanımlandı ve tüm yazılım geliştirme metodolojilerinin temeli oldu.

Geleneksel Yöntemlerin Yetersizliđi

□ Klasik yazılım geliştirme yaklaşımları ile proje yönetimi

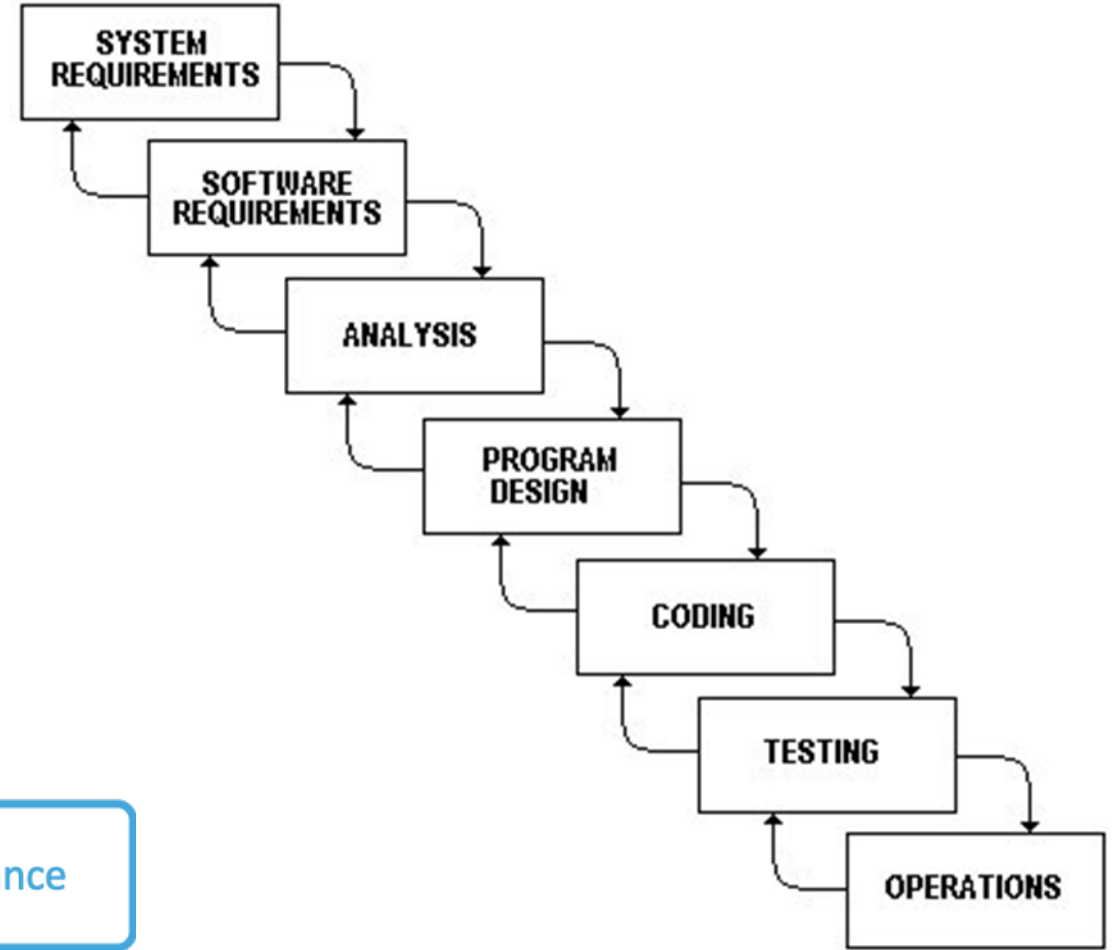
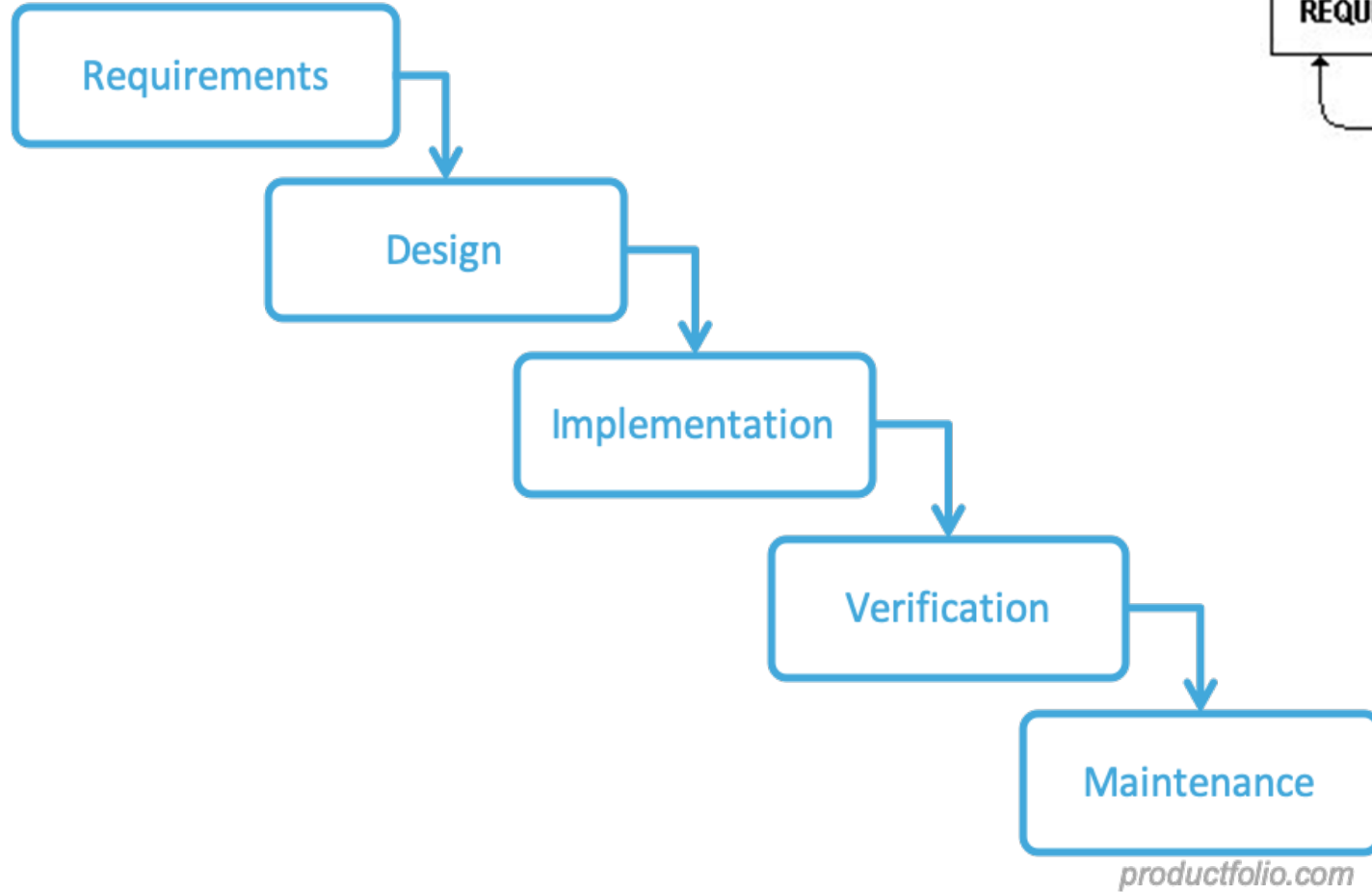
- ❖ Proje yöneticisinin süreçlerin ilerleyişini takip etmedeki güçlüğü,
- ❖ Analizlerin ve sorgulamaların yapılacağı verilerin istenilen şekilde sunulamaması ya da erişilememesi,
- ❖ İş süreçlerinin sistemin tamamlayıcı bir yapısı olarak etkin olarak dağıtılamaması,
- ❖ Proje aktivitelerinde bireylerin birbiri ile olan ilişkileri yeterince tanımlayamaması,

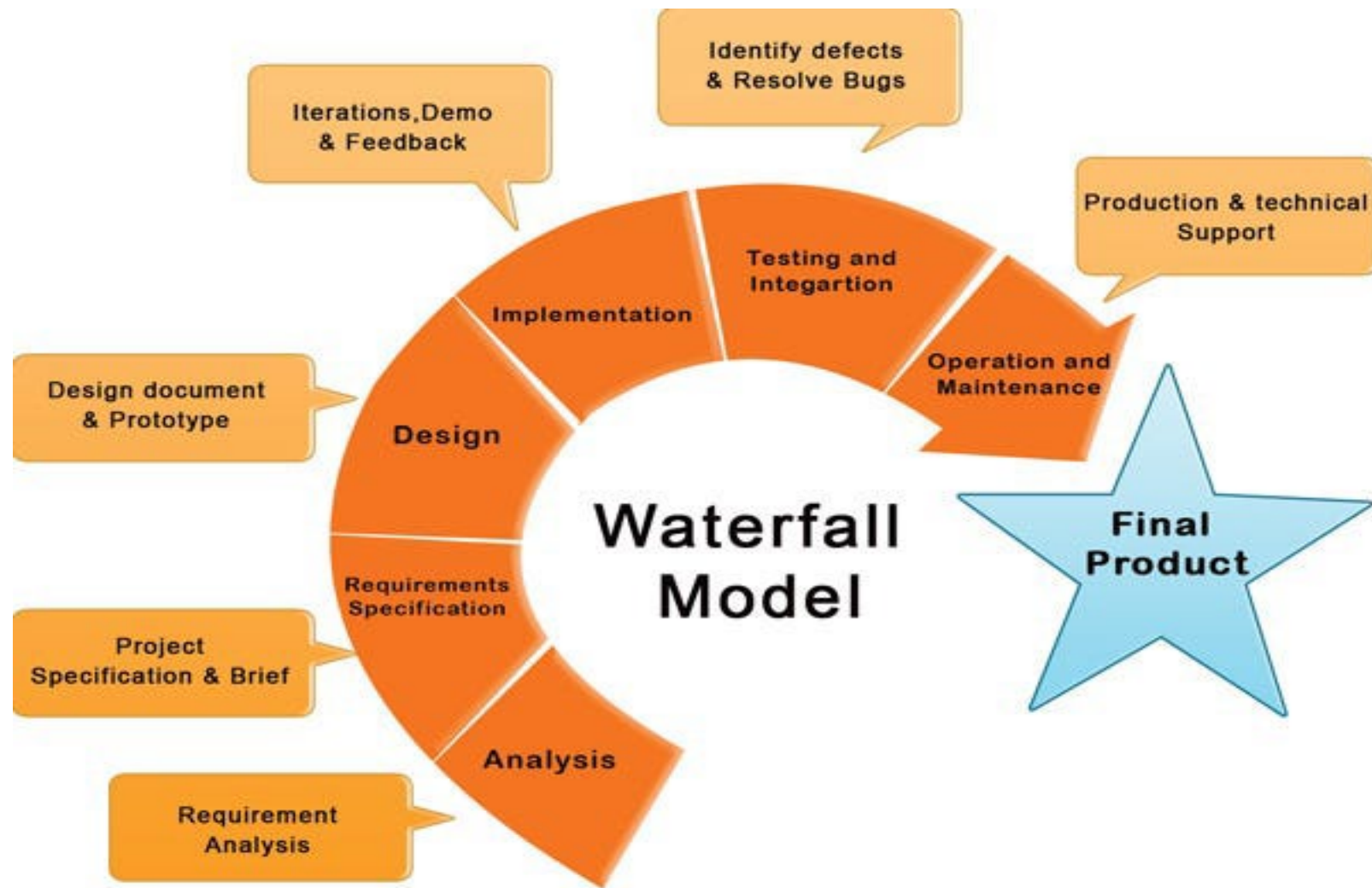
.....

gibi pek çok nedenle bu yöntemler günümüzde yetersiz kalmaktadır

Geleneksel Yazılım Geliştirme Metodolojisi –1 Şelale Yöntemi ile Yazılım Geliştirilmesi

Analiz
Tasarım
Geliştirme
Test
Uygulama

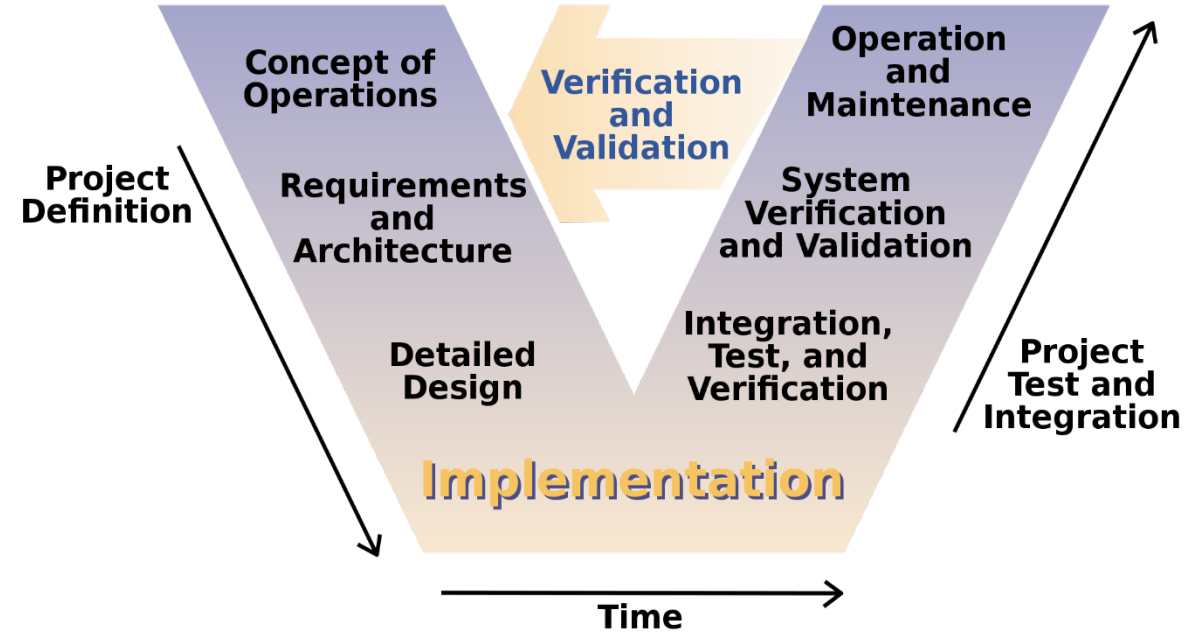




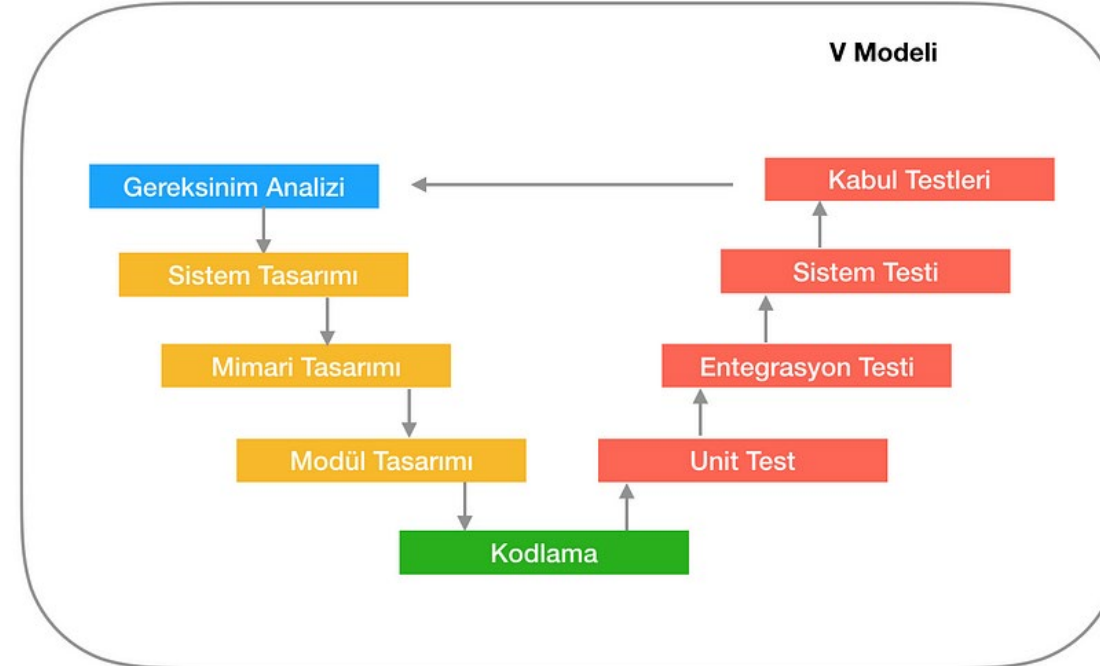
<https://medium.com/synapse-india/waterfall-model-of-software-development-a-sure-fire-practice-for-your-professional-software-needs-4c8997419800>

Geleneksel Yazılım Geliştirme Metodolojisi -2

V- modeli

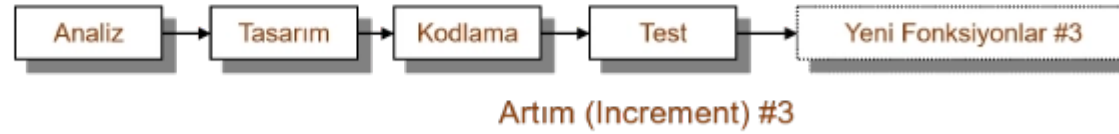
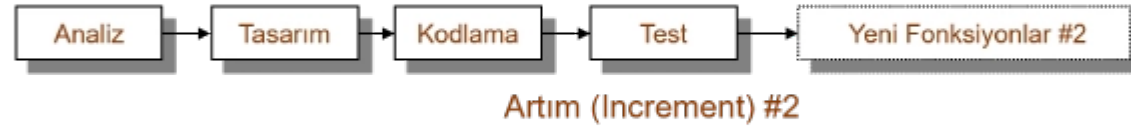
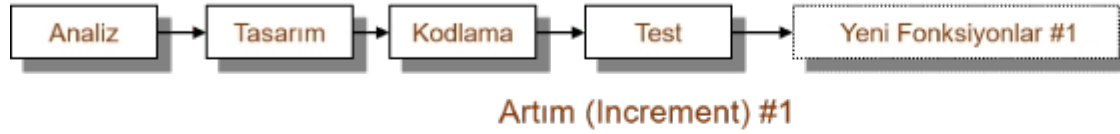


- ❑ Şelale Modeline Verification (Doğrulama) ve Validation(Onaylama) eklenerek bu yöntem değişir.
- ❑ Amaç, her aşamanın sonunda doğrulama ve onaylama gerçekleştirilerek, sistemin düzgün ve kalite çalıştığından emin olmanın hedeflendiği ürün geliştirme modelidir.



Geleneksel Yazılım Geliştirme Metodolojisi -3

Artımlı Model



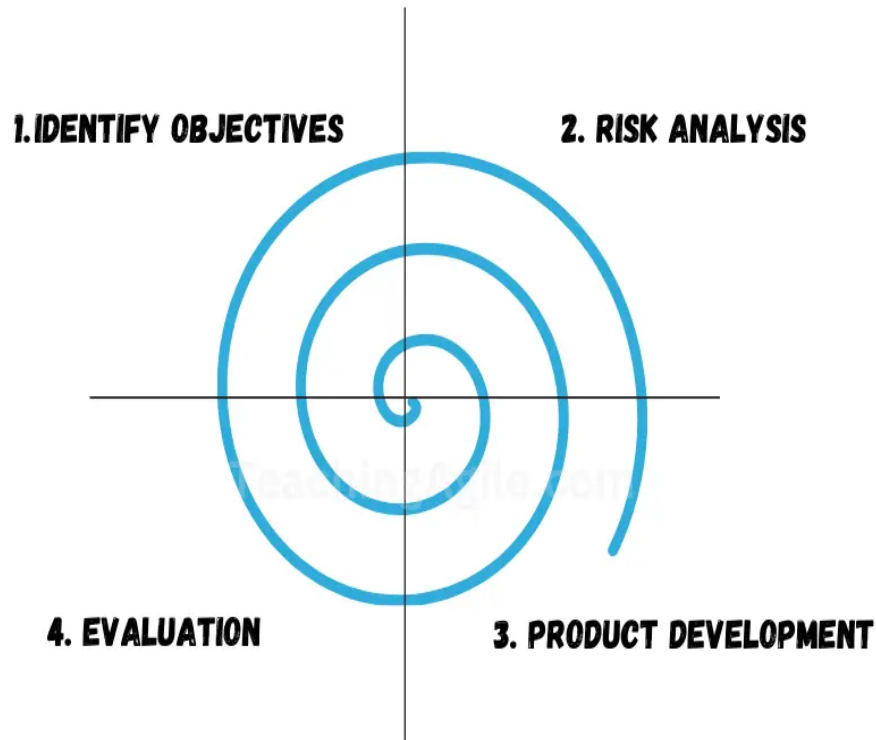
Geleneksel Yazılım Geliştirme Metodolojisi -4

Spiral Modeli

Barry Boehm tarafından 1986 yılında geliştirdi.
Risk-driven, iterative SDLC model

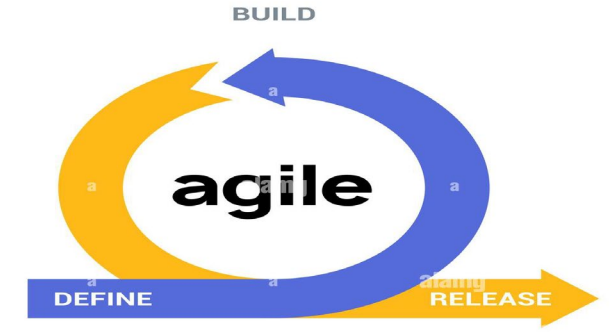
<https://teachingagile.com/sdlc/models/spiral#real-world-implementation-examples>

SPIRAL MODEL IN SOFTWARE DEVELOPMENT



- ❑ Büyük projelerde planlama ve gereksinim analizini düzgün yapabilmek çok zorlaşır.
 - ❖ Çünkü tahmin edilemeyecek pek bir çok riskle karşılaşılabilir.
- ❑ Spiral modelle bu riskleri baz alarak bir model geliştirilir.
- ❑ Amaç: Riskin seviyeler düzeyinde azalması ve projenin başarılı bir şekilde tamamlanmasını sağlamaktır.

Çevik (Agile) yaklaşımlar



- ❑ 1990'lı yıllarda ortaya çıkan çevik (agile) yaklaşımlar; yazılım süreçlerini kısaltmak, daha esnek ve güçlü kılmak için kullanılan aşamalı olarak (iteratif ve artımlı) yazılım geliştirmeyi esas alan bir dizi yöntem olarak adlandırılır.
- ❑ Çevik yazılım geliştirme metotları ile, ürünün çok hızlı şekilde müşteri ile buluşması ve değişen isteklere daha hızlı yanıt verebilmesi amaçlanmaktadır
- ❑ Geç ortaya çıkan değişim isteklerinin yönetilebilmesi, kendi kendini yöneten bir ekip, müşteri ile koordineli bir şekilde çalışılması, ekip içi iletişimin artırılması vb. özellikler çevik süreçlerin genel karakteristik özellikleridir.
- ❑ Geleneksel yazılım geliştirme yöntemleri, yeni teknolojik yazılım geliştirme aktivitelerinin gereksinimlerini karşılamada yetersiz kalmaktadır.
 - ❖ Böylece çevik yazılım geliştirme metotları daha popüler hale gelmiştir.

Çevik (Agile) yaklaşımlar

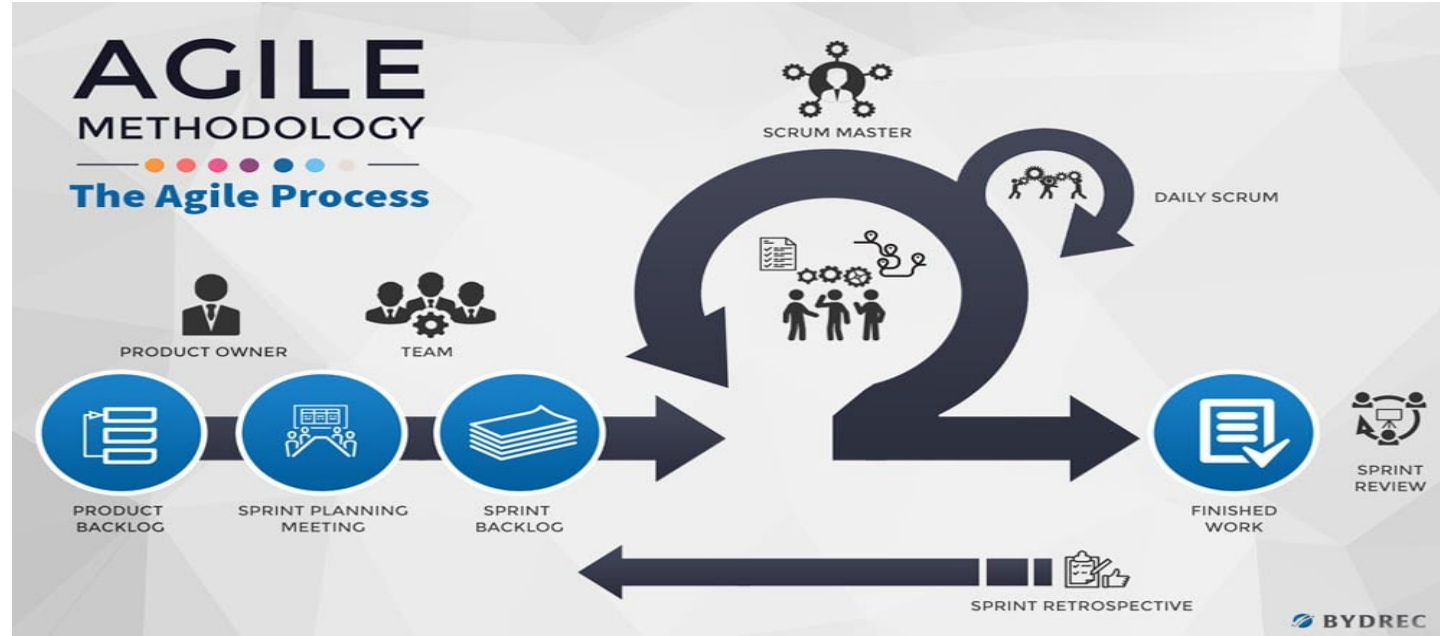


- ❑ Geleneksel yazılım geliştirme yöntemlerinde analiz ve planlama, proje başlamadan önce proaktif bir şekilde yapılmaktadır.
- ❑ Çevik yaklaşımlar ile organizasyona değişikliklere kolay adapte olabilmeyi sağlayan daha esnek bir yapı sağlanmaktadır.
- ❑ Proje gereksinimlerindeki, teknolojideki ve organizasyon yapısındaki değişiklikler çevik yaklaşımlar ile daha iyi tahmin edilebilmekte ve yönetilebilmektedir.
- ❑ Çevik yaklaşımlar, anahtar tedarikçiler ve anahtar müşteriler ile bilgi paylaşımında bulunmakta ve böylece sanal bir ağ oluşturmaktadır.
- ❑ Web destekli bu sanal ağ aracılığıyla etkin biçimde bilgi paylaşımı oluşturulabilmektedir.
 - ❖ Böylece hem müşterilerin ihtiyaçları hem de değişimler çok daha çabuk ve doğru şekilde belirlenebilmektedir.

Nesneye Yönelik Tasarım ile Yazılım Modelleme

Çevik (Agile) yaklaşımlar

- Çevik yaklaşımlarda kısa süreli geliştirme evreleri sayesinde ortaya erken ve sık sürüm çıkartıldığı için, prototip halinde dahi olsa kısa sürede çalışan bir ürün görmek mümkündür.
 - ❖ Böylece bir projede çıkacak sorunlara karşı erken müdahale edebilme olanağı verilebilmesi organizasyonlara avantaj sağlamaktadır.



Çevik Yazılım Geliştirme Manifestosu

<https://agilemanifesto.org/iso/tr/manifesto.html>

Bizler daha iyi yazılım geliştirme yollarını uygulayarak ve başkalarının da uygulamasına yardım ederek ortaya çıkartıyoruz.

Bu çalışmaların sonucunda:

Süreçler ve araçlardan ziyade

bireyler ve etkileşimlere

Kapsamlı dökümantasyondan ziyade

çalışan yazılıma

Sözleşme pazarlıklarından ziyade

müşteri ile işbirliğine

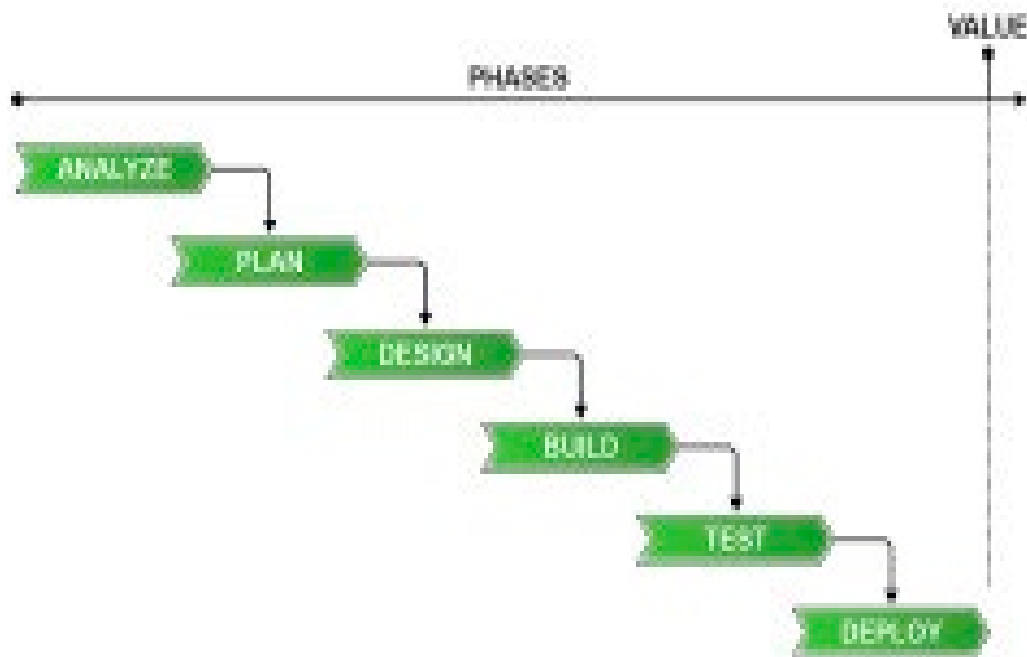
Bir plana bağlı kalmaktan ziyade

değişime karşılık vermeye

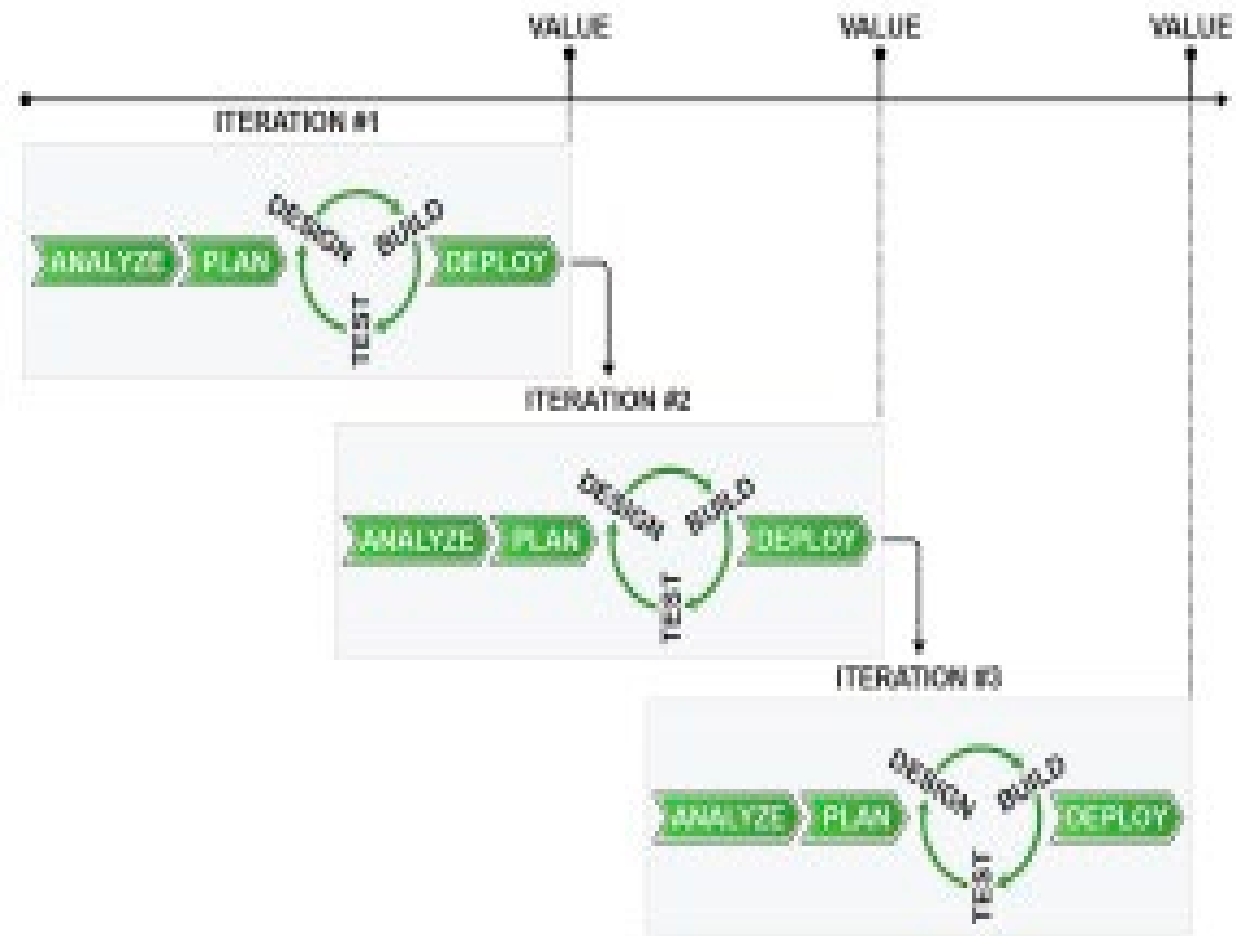
değer vermeye kanaat getirdik.

Özetle, sol taraftaki maddelerin değerini kabul etmekle birlikte, sağ taraftaki maddeleri daha değerli bulmaktavız.

WATERFALL MODEL



AGILE MODEL



Planned and stage-gated development

Well-defined requirements

Structured decision mandates

Structured and planned delivery

Validation of complete product

Focus on leveraging expertise



Incremental and iterative development

Customer-oriented backlog

Empowered teams, end-to-end responsibility

Frequent delivery and feedback

Early validation through stakeholder participation

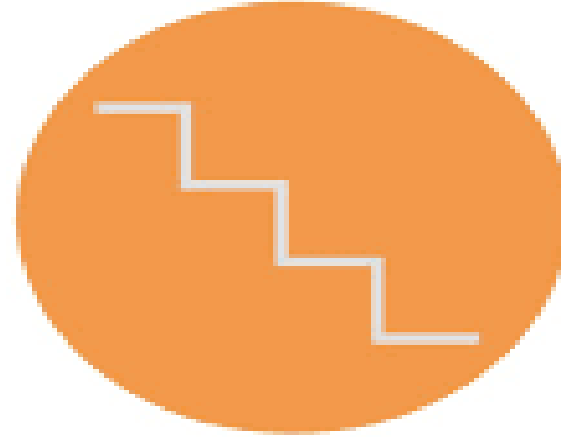
Focus on learning

ÇEVİK MODEL



VS

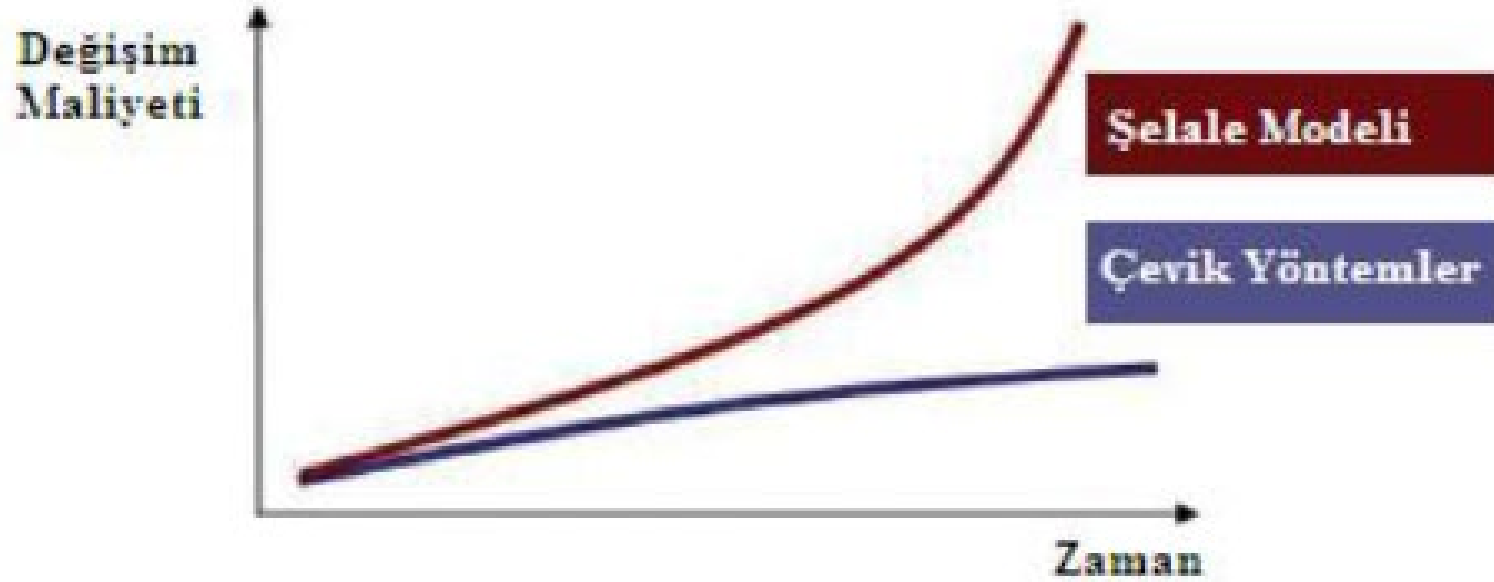
ŞELALE MODELİ



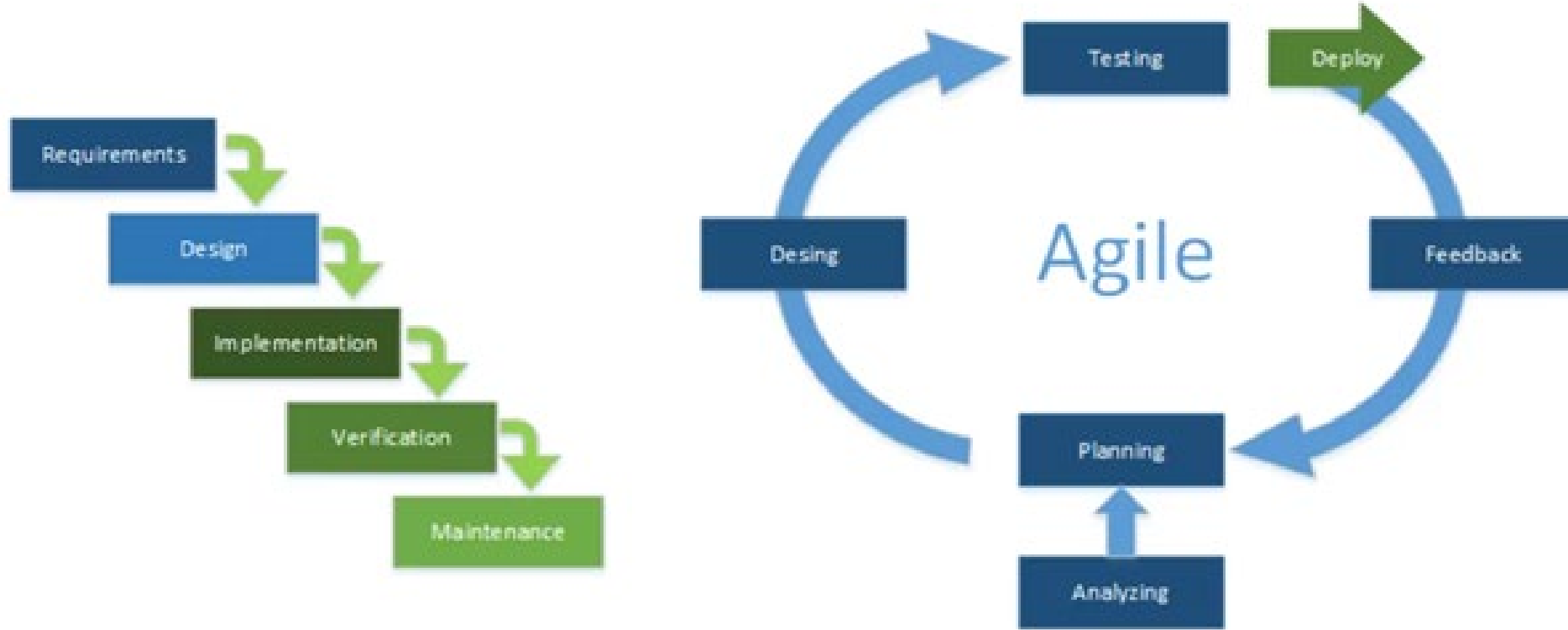
Yazılım Geliştirme Metodolojilerinin Değerlendirmeleri

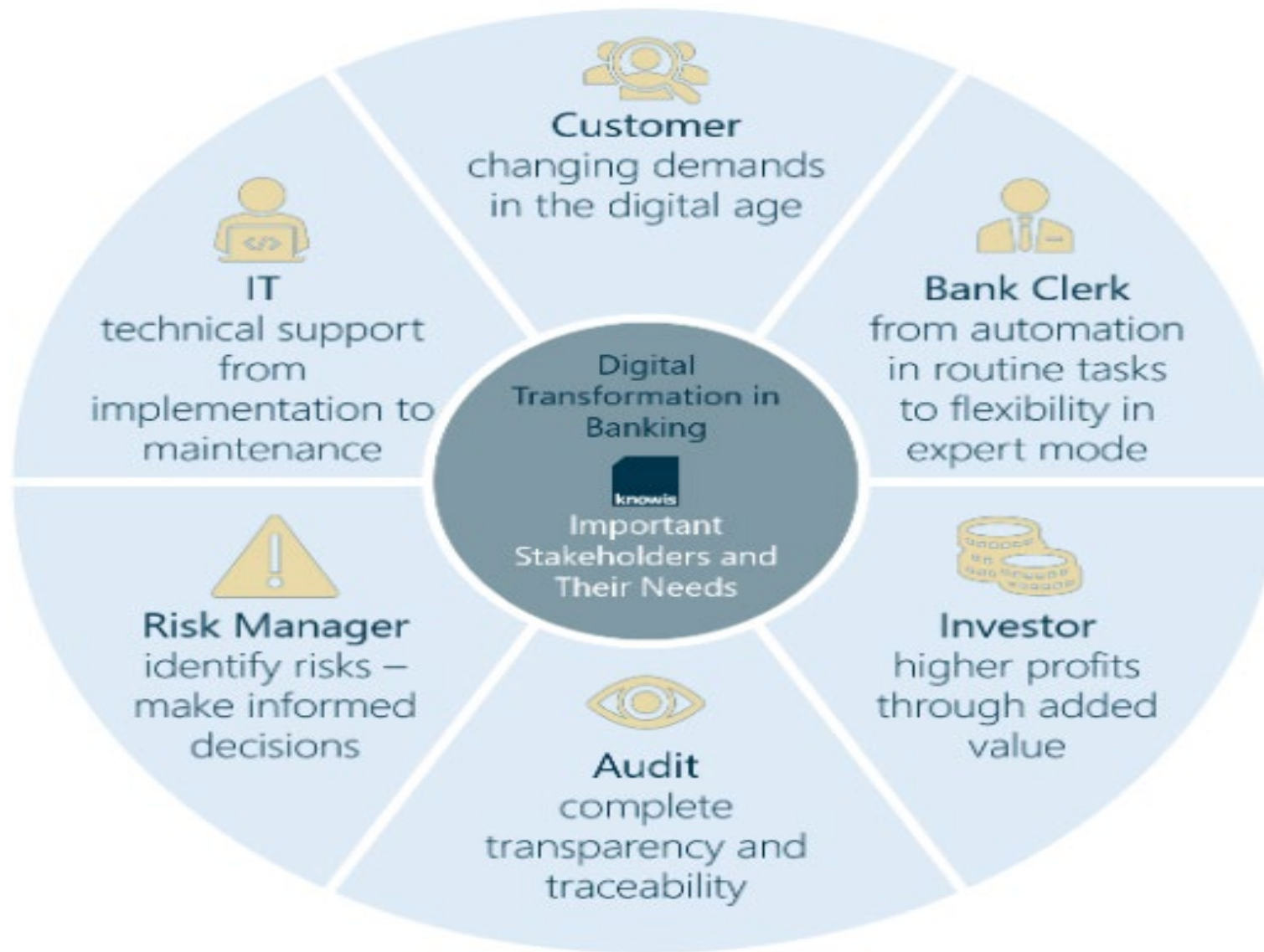
Açıklama	Klasik Metodolojiler					Çevik Metodolojiler			
	Şelale	Spiral Model	V Model	Artımlı Model	Prototipleme	Geliştirme Evrimsel	Scrum	Kanban	RUP
Gereksinimlerin Belirlenmesi	Başlangıç	Belirli Sıklıkla	Başlangıç	Belirli Sıklıkla	Belirli Sıklıkla	Başlangıç	Belirli Sıklıkla	Belirli Sıklıkla	Belirli Sıklıkla
Dokümantasyon ve Eğitim Gerekliliği	Yüksek	Orta	Orta	Orta	Orta	Orta	Yüksek	Yüksek	Yüksek
Maliyet	Yüksek	Yüksek	Yüksek	Orta	Düşük	Düşük	Yüksek	Yüksek	Yüksek
Risk Duyarlılığı	Yüksek	Düşük	Orta	Düşük	Düşük	Yüksek	Düşük	Düşük	Düşük
Başarı Garantisi	Düşük	Yüksek	Düşük	Yüksek	Orta	Yüksek	Yüksek	Yüksek	Yüksek
Esneklik	Düşük	Yüksek	Düşük	Yüksek	Yüksek	Düşük	Yüksek	Yüksek	Yüksek
Uzmanlık Gerekliliği	Orta	Yüksek	Orta	Orta	Orta	Orta	Yüksek	Yüksek	Yüksek
Zaman Uzunluğu	Yüksek	Yüksek	Yüksek	Yüksek	Yüksek	Yüksek	Düşük	Orta	Orta
Bakım	Orta	Yüksek	Orta	Yüksek	Yüksek	Düşük	Yüksek	Yüksek	Yüksek
Yönetilebilirlik/Basitlik	Yüksek	Düşük	Orta	Orta	Orta	Düşük	Düşük	Düşük	Düşük
Uygulama Derecesi	Yüksek	Orta	Yüksek	Yüksek	Yüksek	Yüksek	Yüksek	Yüksek	Yüksek

Şelale ve Çevik Yöntemi Deęiřtirmenin Maliyetleri



Şelale ve Çevik Yöntemin Karşılaştırması





The Case for Agile Methodologies against Traditional Ones in Financial Software Projects, European Journal of Business and Management Research 2021