SAKARYA ÜNİVERSİTESİ

SİSTEM ANALİZİ VE TASARIMI

Bölüm 2 – Bilişim Sistemi Geliştirme

Dr. Ömer Faruk SEYMEN

Sistem Geliştirmeye Giriş

- ✓ Bilişim sistemlerini geliştirmek ve desteklemek amacıyla belli adımları ve aşamaları olan metodolojiler kullanılır.
- ✓ Çoğu süreç konusunda olduğu gibi bilişim sistemleri geliştirme de bir «yaşam çevrimi» ile ele alınır
- ✓ Sistem Geliştirme Yaşam Çevrimi (Systems Development Life Cycle – SDLC) olarak adlandırılan bu aşamalı yapı çoğu zaman literatürde sistem geliştirme tanımının yerine kullanılır

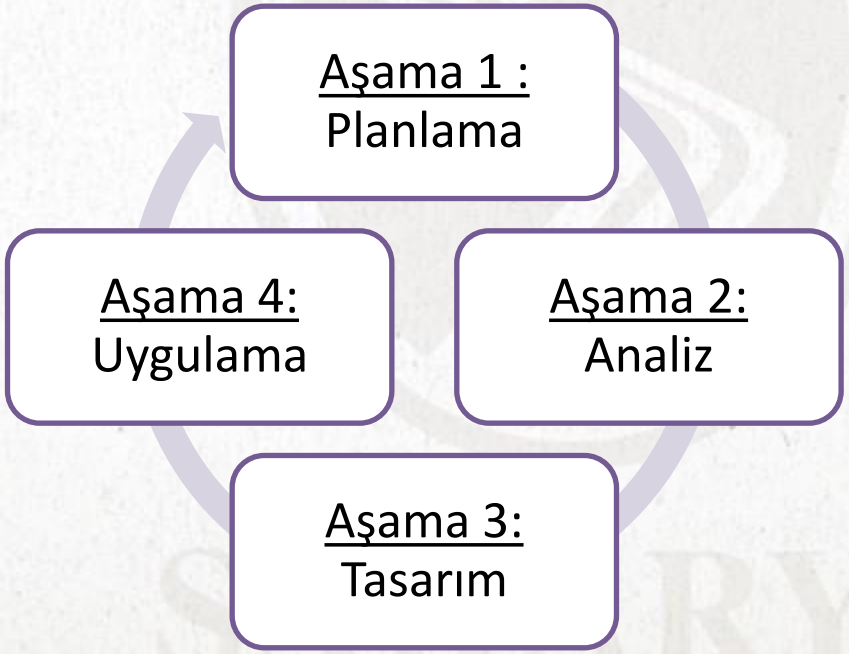
Sistem Geliştirmeye Giriş

- ✓ Organizasyonlar «özellikle yazılım geliştirirken» neden bu sistem geliştirme yaklaşımlarına ihtiyaç duyar?
- ✓ Bilişim sistemleri;
  - ➢ Veri
  - ➢ Süreçler
  - ➢ İletişimler ve ilişkiler
  - ➢ Teknolojiler

  bileşenlerini içerir.
- ✓ Bütün bu bileşenler bilişim sistemleri paydaşlarına hizmet etmektedir. Bu kadar karmaşık bir yapıyı anlamak ve tasarlamak için mutlaka standardize bir yaklaşım gerekir.

Sistem Geliştirme Yaşam Çevrimi (SDLC)

- ✓ Bir bilişim sistemi geliştirmek için gerekli olan adımları içeren bir yapıdır.
- ✓ Çeşitli kaynaklarda farklı sayıda adım olsa da genelde 4 temel aşamayı içerir.



Sistem Geliştirme Yaşam Çevrimi (SDLC)

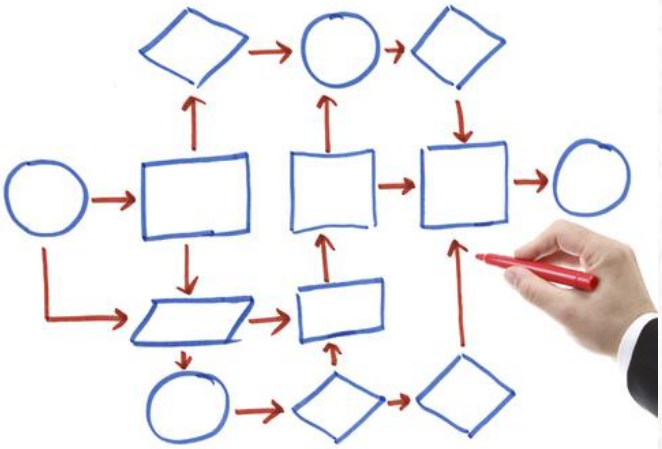
- ✓ Yaşam çevriminde bütün adımların sıralı olması zorunlu değildir.
- ✓ Her aşamada belirli girdiler ve çıktılar mevcuttur.
- ✓ Yaşam çevrimi modelleri projeden projeye, işletmeden işletmeye farklılıklar gösterebilir.
- ✓ Sonuç olarak; bu yapı bir «standart» değil, «kılavuz» olarak ele alınmalıdır.

SDLC Aşamaları / Planlama

- ✓ Bütün yaşam çevrimi metodolojileri problemin tanımlanması ve problem ifadesinin oluşturulması ile başlar.
- ✓ Proje planlama aktiviteleri planlama aşamasında yürütülür.
- ✓ Bu aşamadaki en önemli görevlerden bir tanesi de fizibilite çalışmalarıdır.

SDLC Aşamaları / Analiz

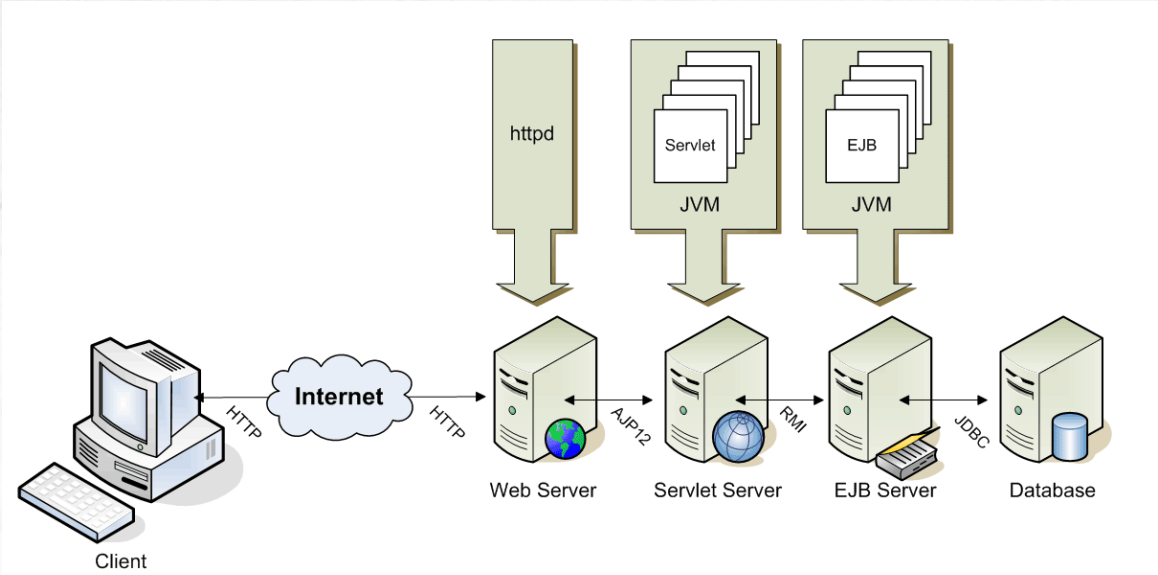
- ✓ Bu aşamada 3 temel işlem gerçekleştirilir.
  - ➢ Kullanıcı ihtiyaçları belirlenir.
  - ➢ Bu ihtiyaçlar süreç modellerine dönüştürülür
  - ➢ Süreçlerin yürütülebilmesi için gerekli veri sözlükleri oluşturulur.



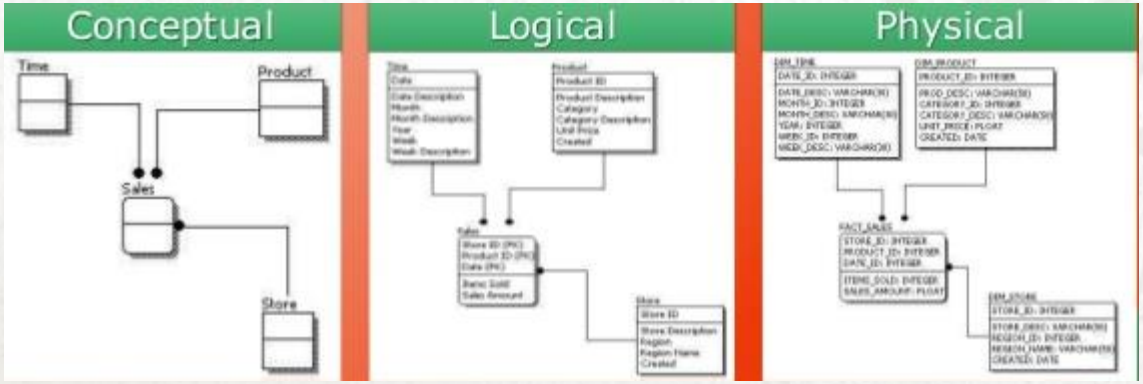
Field Name	Data Type	Data Format	Field Size	Description	Example
License ID	Integer	NNNNNN	6	Unique number ID for all drivers	12345
Surname	Text		20	Surname for Driver	Jones
First Name	Text		20	First Name for Driver	Arnold
Address	Text		50	First Name for Driver	11 Rocky st Como 2233
Phone No.	Text		10	License holders contact number	0400111222
D.O.B	Date / Time	DD/MM/YYYY	10	Drivers Date of Birth	08/05/1956

SDLC Aşamaları / Tasarım

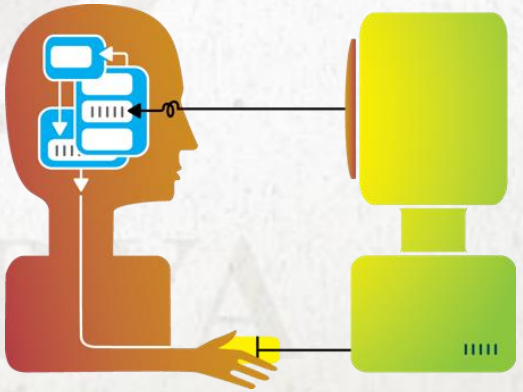
Mimari Tasarım



Veri Tabanı Tasarımı



İnsan-Bilgisayar Etkileşimi



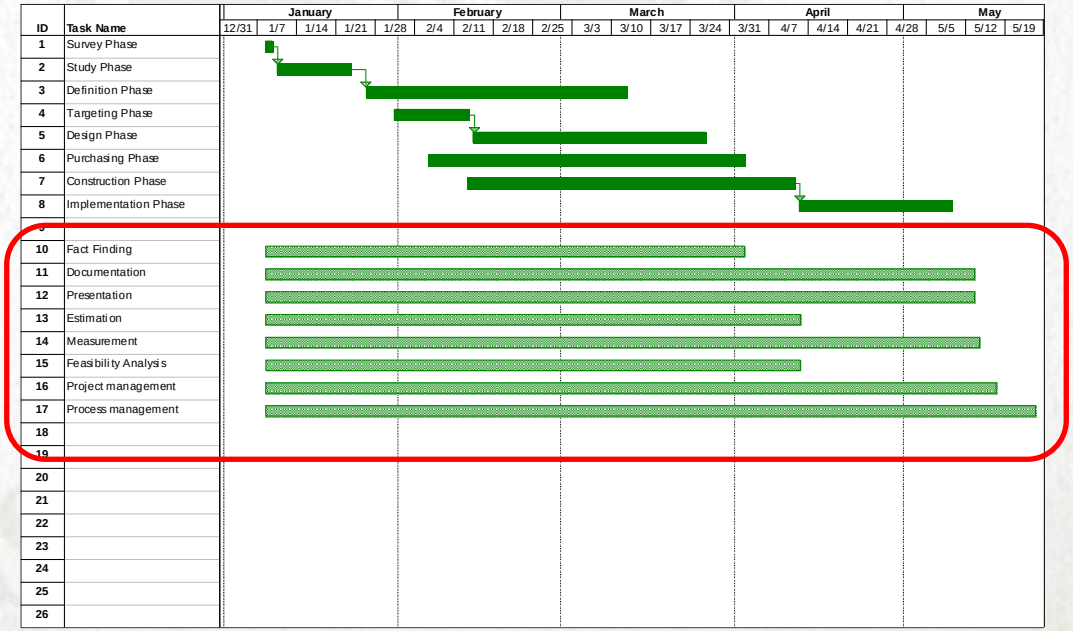
SDLC Aşamaları / Uygulama

- ✓ Bu aşamada aslında iki alt aşamadan oluşur

- ✓ Sistemin Uygulanması
  - ➢ Bilişim sistemi ile ilgili donanımların ve temel yazılımların kurulması
  - ➢ Bilişim sisteminin kodlanması
  - ➢ Dokümantasyon
- ✓ Sistem Desteği
  - ➢ Kullanıcı Eğitimleri
  - ➢ Güvenlik ile ilgili konular
  - ➢ Sistemin sürdürülebilirliği ve destek

SDLC Aşamalar Arası Çapraz İşlemler

- ✓ Bazı görevler yaşam çevrimi içerisinde birden fazla aşamada yürütülür veya tekrarlanır.
  - ➢ Bilişim Toplama
  - ➢ Dokümantasyon
  - ➢ Tahmin ve Performans Ölçümü
  - ➢ Fizibilite Analizleri
  - ➢ Proje Yönetimi
  - ➢ Süreç Yönetimi



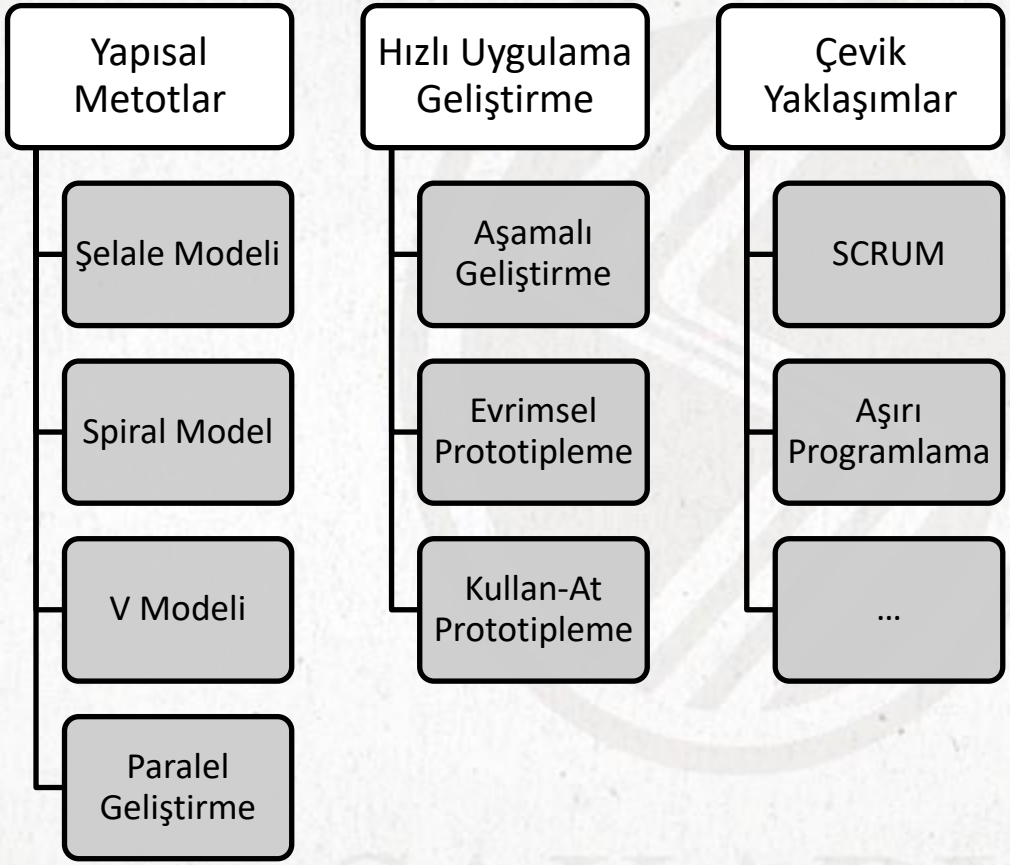
Sistem Geliştirme Metodolojileri

- ✓ Sistem geliştirme açısından **metodoloji** mantıksal yaşam çevriminin hayata geçirilmesini sağlayan araçlara karşılık gelir.
- ✓ Metodoloji aşağıdaki unsurları içeri
  - ➢ Her aşamadaki aktivitelerin adım adım gösterimi
  - ➢ Her aktivite için bireysel veya grup sorumlulukları
  - ➢ Her aktivite için çıktılar ve kalite standartları
  - ➢ Her bir aktiviteyi gerçekleştirmek için gerekli araç ve teknikler
- ✓ Bilişim sistemleri açısından metodoloji bir bilişim sisteminin analizi, tasarımı, uygulanması ve sürdürülebilirliğini sağlayan her adımın oluşturulmasını sağlayan standart bir süreçtir.

Şirketler bu metodolojileri neden kullanır?

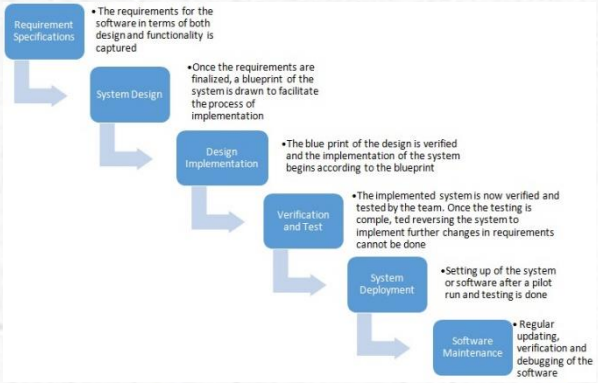
- ✓ Metodolojiler bütün projelere uygulanabilen kararlı, standardize, yeniden kullanılabilir ve güncellenebilir bir yapı sunar
- ✓ Metodolojiler proje sürecinde hatalar ve kısıtlamalardan kaynaklı risklerin azaltılmasını ve/veya yönetimini sağlar.
- ✓ Metodolojiler her bir proje için kararlı bir dokümantasyon sunar. Bu açıdan farklı projeler için başlangıç noktası olabilir.

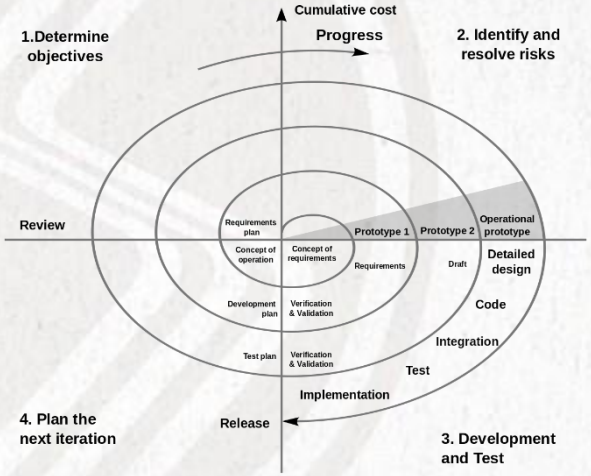
Sistem Geliştirme Metodolojileri



Yapısal Metodolojiler

- ✓ Bu metodolojiler önceden belirlenmiş adım adım bir prosedür izleyerek geliştirme sürecini yürütür.
- ✓ Çoğu yapısal modeller SDLC yaklaşımının türevleridir.





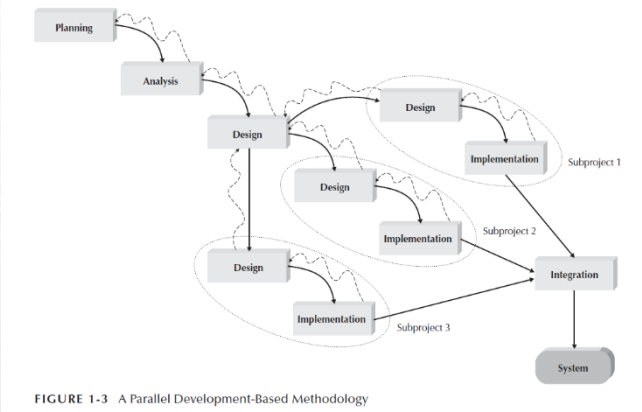
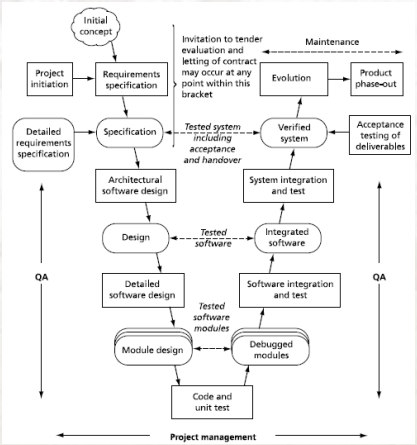


FIGURE 1-3 A Parallel Development-Based Methodology



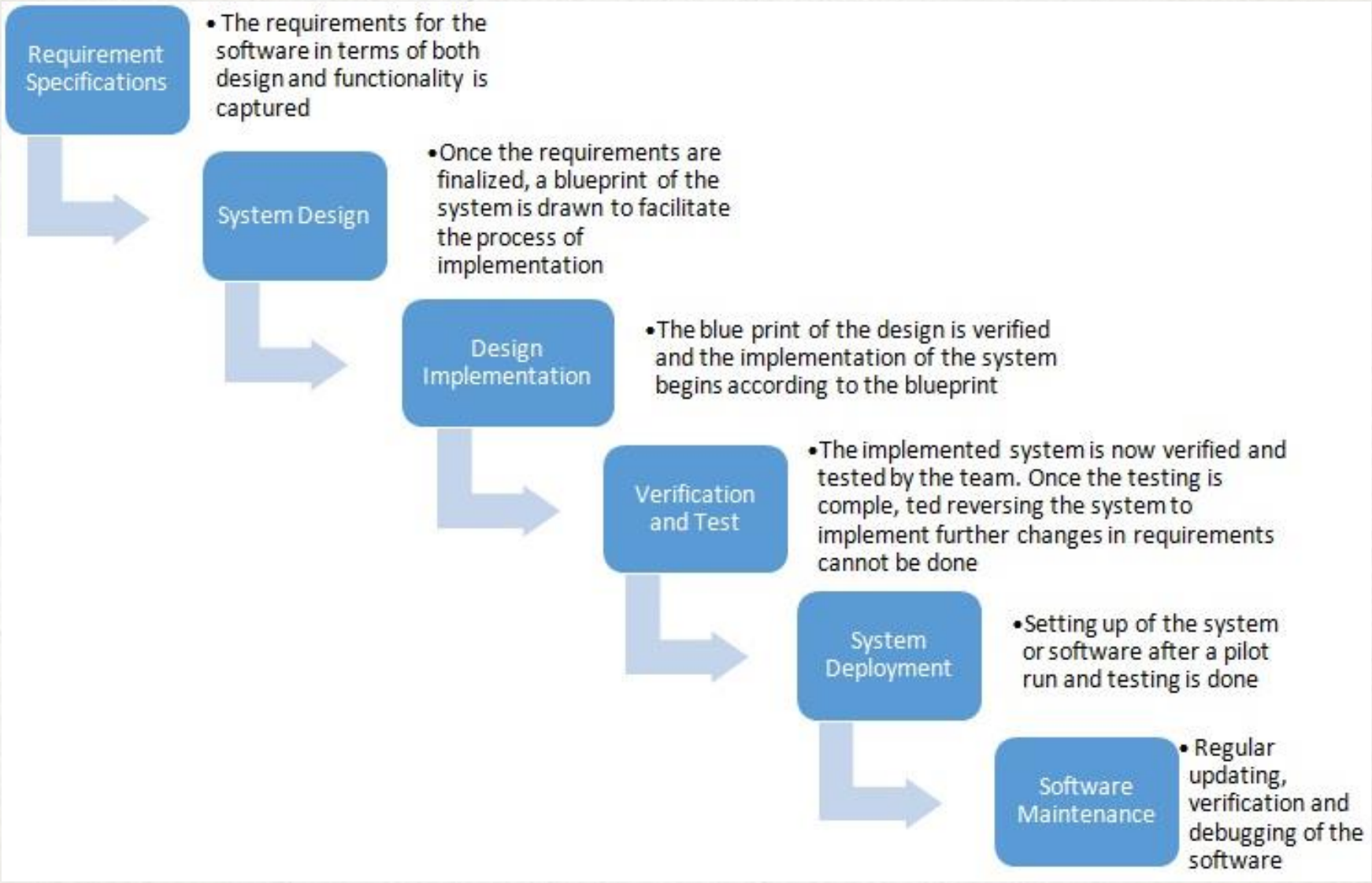
Yapısal Metodolojiler / Şelale Modeli

- ✓ Bu modelde süreç içerisinde yürütülecek işlemler birbirinden ayrı bir adım olarak sunulur.
- ✓ Süreçler bir «şelale» görseli ile sunulur ve temel şelale modelinde bir adım tamamlanmadan diğer adıma geçilmez.
- ✓ Model yaşam çevrimi boyunca adımlar arasında ortak yürütülen görevleri içermez.

Şelale Modeli ne zaman kullanılır?

- ✓ Eğer bütün gereksinimler net bir şekilde belirlenebiliyorsa
- ✓ Son ürün tanımı kararlı ve güncelleme gerektirmiyorsa
- ✓ Kullanılacak teknolojiler net bir şekilde anlaşılabilir ise
- ✓ Var olan bir ürünün yeni bir versiyonunun oluşturulmasında
- ✓ Var olan bir ürünü farklı bir platforma aktarmada
  - ➢ Android bir yazılımı IOS ortamına adapte etme

Yapısal Metodolojiler / Şelale Modeli



Avantajları

- ✓ Sistem analizcileri için kullanımı ve anlaşılması kolaydır
- ✓ Yeterli tecrübeye sahip olmayan çalışanlar için yapısal bir yaklaşım sunar
- ✓ Süreç içerisindeki «kilometre taşları» net olarak görülür
- ✓ Yönetim açısında kolaylıklar yönetilebilir bir yapı sunar
- ✓ Zaman ve maliyet kısıtlarında ziyade kalite ön planda ise çok iyi sonuçlar verir
- ✓ Kararlı yazılım projeleri için uygundur

Dezavantajları

- ✓ Değişimler zor ve maliyetlidir
- ✓ Kullanıcılar tam olarak ne elde edeceklerini bilemeyebilir
- ✓ Bütün aşamaların tam olarak sonlandırmasını gerektirir
- ✓ Uzun zaman çizelgeleri gerektirir, bu anlamda plana uymak zorlaşabilir
- ✓ Bütün gereksinimlerin açık bir şekilde başlangıçta bilinmesi çoğu zaman mümkün değildir.
- ✓ Analitik problem çözme yapısını tam olarak karşılamaz
- ✓ Proje sonunda entegrasyon problemli olabilir

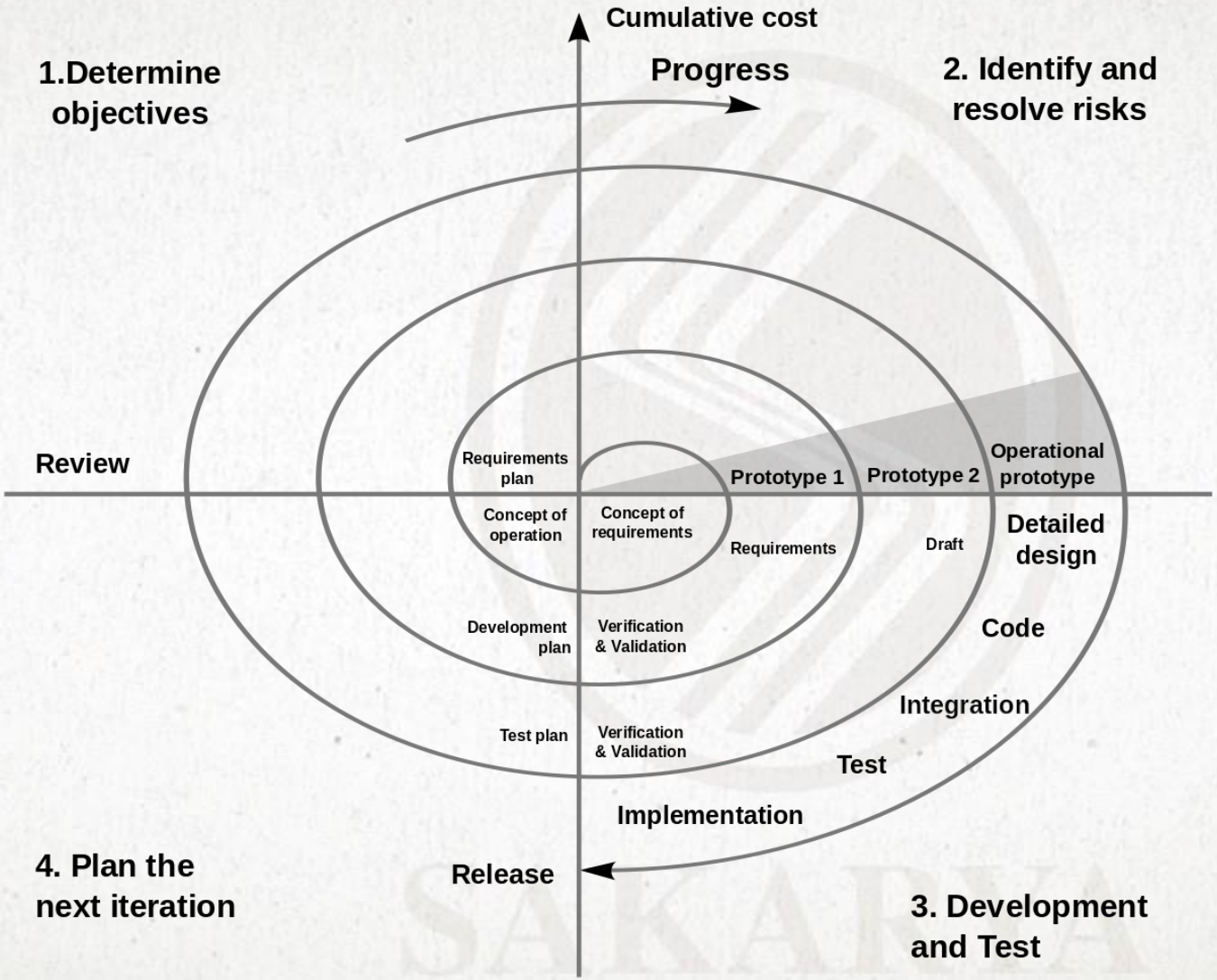
Yapısal Metodolojiler / Spiral Model

- ✓ Genelde yüksek riski kompleks projelerde kullanılan iteratif bir sistem geliştirme yaklaşımıdır.
- ✓ Bu modelde şelale modeli ile Hızlı Uygulama Geliştirme modellerinde prototipleme birlikte kullanılır.
- ✓ Bu modelde bütün aktiviteler spiral formda sıralanır.
- ✓ Spiral formun 4 bölümü vardır.
  1. Amaçların, alternatiflerin ve kısıtlamaların belirlenmesi
  2. Risk analizi ve alternatiflerin değerlendirilmesi
  3. Geliştirme ve test aşamalarının uygulanması
  4. Sonraki iterasyonun planlanması

Spiral Model ne zaman kullanılır?

- ✓ Prototip oluşturmanın uygun olduğu projelerde
- ✓ Maliyetlerin ve riskin ön planda olduğu projelerde
- ✓ Orta ve yüksek riskli projelerde
- ✓ Kullanıcıların gereksinimlerinden tam olarak emin olmadığı veya gereksinimlerin karmaşık olduğu durumlarda
- ✓ Süreç içerisinde yüksek oranda değişim ve güncelleme olacağı öngörülüyorsa

Yapısal Metodolojiler / Spiral Model



Avantajları

- ✓ Risk yönetimi görevlerini içermesi
- ✓ Süreç içerisinde değişim yönetiminin kolaylıkla sağlanması
- ✓ Yüksek riskli projelerde kullanılabilinmesi
- ✓ Kullanıcıları sistemi erkenden görebilmesi (prototipleme)
- ✓ Kritik yüksek riskli fonksiyonları belirleyip süreç içerisinde önceliklendirebilmesi
- ✓ Kullanıcıların süreç boyunca yüksek katılım göstermeleri
- ✓ Erken ve sık geri beseleme imkanları

Dezavantajları

- ✓ Maliyet genelde yüksektir. (Prototipleme ve Risk Analizi)
- ✓ Diğer yapısal yöntemlere kıyasla karmaşık bir yapıdadır
- ✓ Proje değerlendirme süreçleri için uzmanlık gerektirir
- ✓ Kurallar ve protokoller açısından oldukça katıdır.
- ✓ Düşük riskli projeler için uygun değildir.
- ✓ Iteratif yapının sonucu olarak dokümantasyon oldukça fazladır.
- ✓ Sistem geliştirme süreci için kilometre taşları belirlemek çoğu zaman zordur.

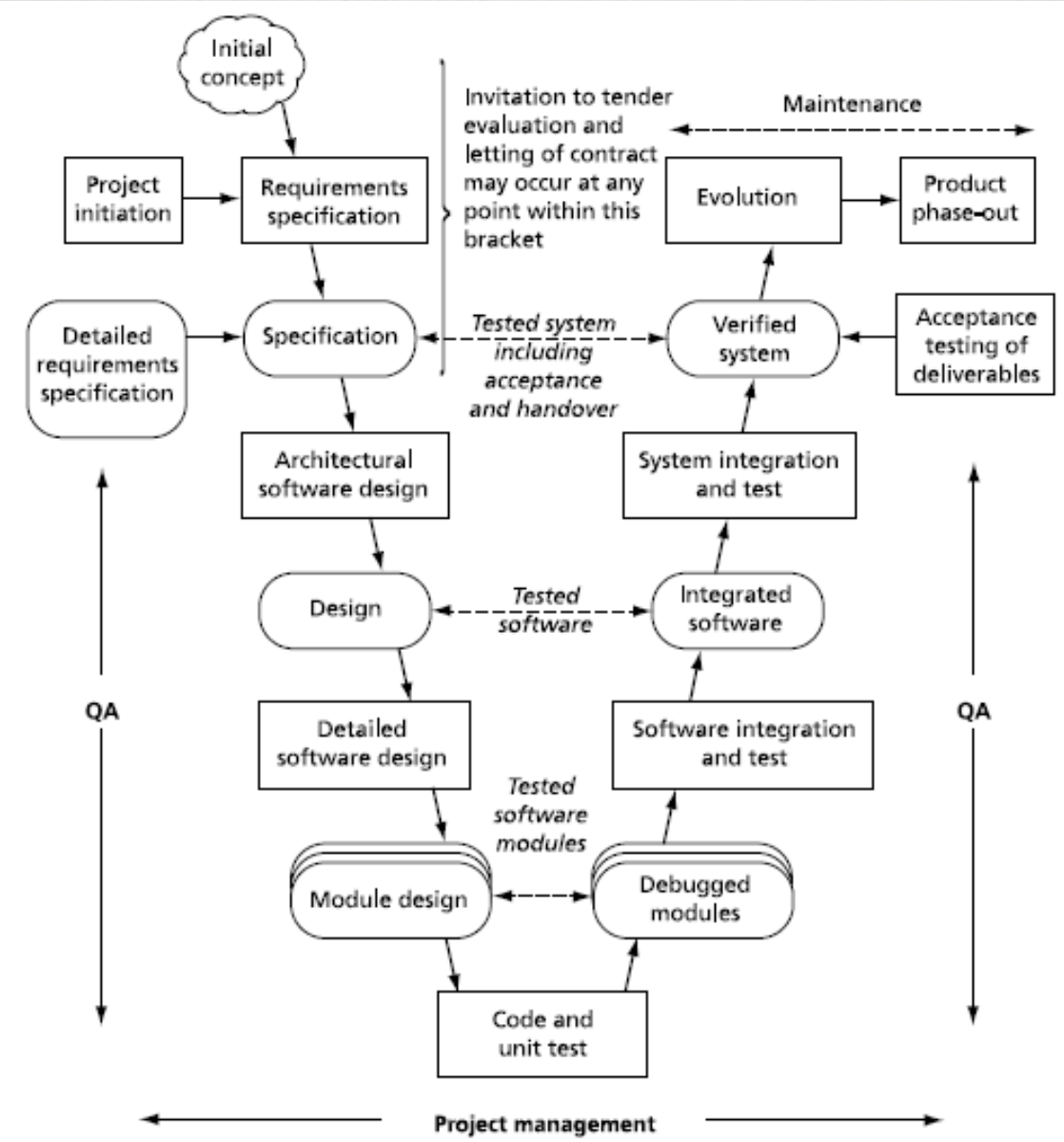
Yapısal Metodolojiler / V Modeli

- ✓ Şelale modelinin bir varyantıdır.
- ✓ Doğrulama ve Sağlamaya odaklı bir yapısı vardır.
- ✓ Şelale modeli aşamaları gerçekleştirilirken her paralel olarak test işlemleri planlanır.
- ✓ Sistem boyunca yürütülen çeşitli aşamadaki testler son ürünün kararlı yapıda olması sonucunu doğurur.

V Modeli ne zaman kullanılır?

- ✓ Yüksek güvenirlik gerektiren projeler için (Hasta takip sistemleri)
- ✓ Bütün gereksinimler başlangıçta belirli ve net ise
- ✓ Analiz süreçleri sonrasında modifikasyonların olabileceği projeler için
- ✓ Çözüm ve kullanılacak teknolojiler başlangıçta belirli ise

Yapısal Metodolojiler / V Modeli



Avantajları

- ✓ Yaşam çevriminin başlangıç aşamalarından itibaren doğrulama ve sağlama görevlerine odaklanılması
- ✓ Her aşamadaki çıktı ölçülebilir ve test edilebilirdir
- ✓ Kilometre taşlarının izlenmesi kolaydır
- ✓ Kullanımı kolaydır

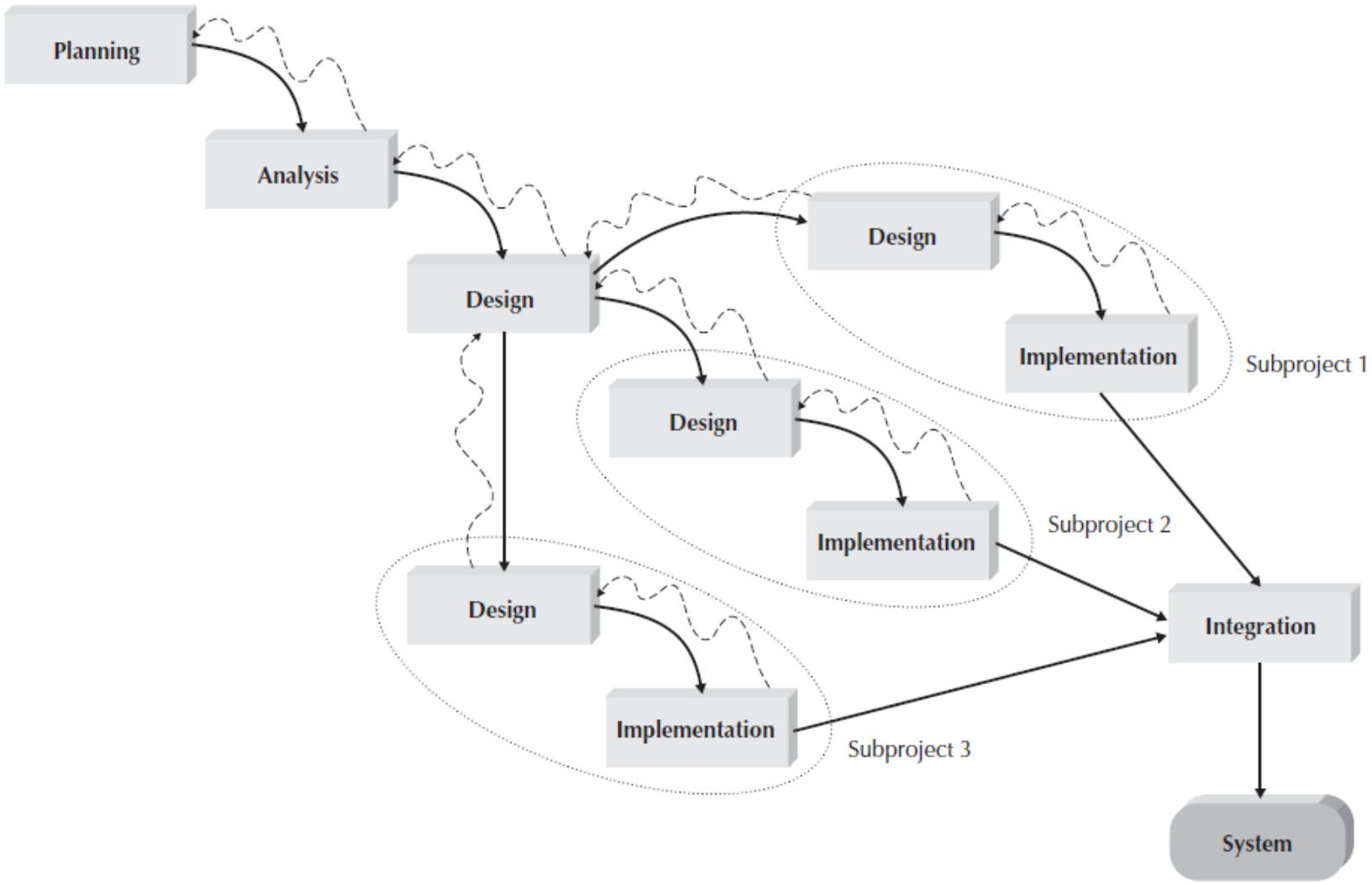
Dezavantajları

- ✓ Eş zamanlı ve kesişen görevleri yönetmekte yetersizdir.
- ✓ İteratif bir yapıda olmadığından kullanıcı katılımı sınırlıdır.
- ✓ Kullanıcı gereksinimlerindeki değişimlere cevap verebilecek dinamik bir yapıda değildir.
- ✓ Risk analizini içermediğinde büyük projeler için uygun olmayabilir.

Yapısal Metodolojiler / Paralel Geliştirme

- ✓ Sistem analizi ile bilişim sisteminin tamamlanıp, kullanıma sunulması arasındaki sürenin azaltılması amacıyla geliştirilen bir yöntemdir.
- ✓ Tasarım ve uygulama süreçlerini sıralı yürütmek yerine yazılım projesi alt projelere bölünür.
- ✓ Bütün alt projeler tamamlandığında ise entegrasyon yapılır ve son ürün olarak bilişim sistemi kullanıcılara sunulur.

Yapısal Metodolojiler / Paralel Geliştirme



Avantajları

- ✓ Toplam proje süresini kısaltır.

Dezavantajları

- ✓ Bazen bazı işlerin tekrar yapılması sonucunu doğurabilir.
- ✓ Son entegrasyon proje başlangıcında öngörüldüğü gibi sancısız olmayabilir.
- ✓ Alt projeler birbirinde tam olarak bağımsız değil ise bir proje sürecindeki alınan bir tasarım kararı diğer alt projeleri etkileyebilir.

Kademeli SDLC Modeli

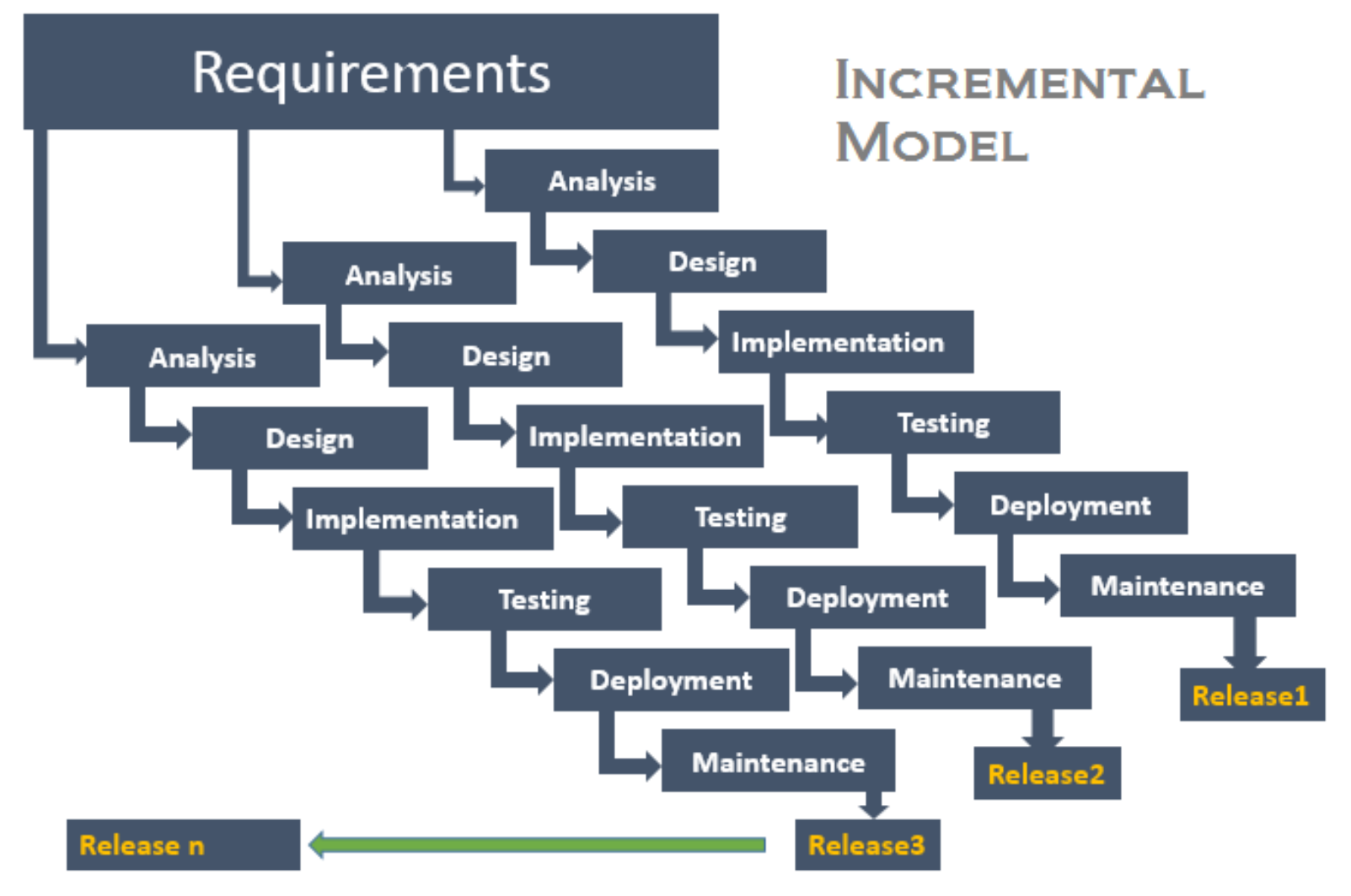
- ✓ Bu modelde sistemin temel gereksinimleri karşılayan kısmi bir uygulanması tamamlanır.
- ✓ Daha sonra her adımda yeni bir fonksiyon eklenerek sistem geliştirilir.
- ✓ Bu model gereksinimlerin önceliklendirilmesi aşaması ile başlar.

Kademeli SDLC Modeli

Kademeli SDLC Modeli ne zaman kullanılır?

- ✓ Programın temel olarak dahi erkende kullanılmaya başlanması ihtiyacı söz konusu ise
- ✓ Gereksinimlerin çoğu başlangıçta bilindiğinde
- ✓ Çok uzun geliştirme süreçleri öngörülen projelerde
- ✓ Yeni bir teknolojinin kullanımını gerektiren projelerde

Kademeli SDLC Modeli



Kademeli SDLC Modeli

Avantajları

- ✓ Yüksek riskli veya temel fonksiyonları öncelikli geliştirmeyi sağlar
- ✓ Kullanıcılar her bir versiyon için geri dönüş yapar ve bu bildirimler sürece dahil edilir.
- ✓ Görevleri «Böl ve Parçala» prensibi ile yönetir.
- ✓ İlk versiyon kullanıcıları göreceli olarak çok erken sunulur.
- ✓ Gereksinimlerin değişmesine bağlı riskler azaltılır.

Kademeli SDLC Modeli

Dezavantajları

- ✓ İyi bir planlama ve tasarım süreci gerektirir
- ✓ İyi tanımlanmış ara yüzler ve modüller gerektirir.
- ✓ Toplam sistem maliyeti göreceli olarak yüksektir.

Kademeli vs. İteratif Yaklaşımlar

- ✓ Literatürde kademeli ve iteratif yaklaşımlar zaman zaman birbirinin yerine kullanılabilmektedir.
- ✓ Kademeli modelde temel ve fonksiyonel bir versiyon tasarlanır ve bu versiyona sonradan eklemeler gerçekleştirilir.
- ✓ Bu eklemelere güncelleştirme veya hata düzeltmesi değil tamamen yeni işlevleri beraberinde getirir.

Kademeli vs. İteratif Yaklaşımlar

- ✓ İteratif yaklaşımda sistemin tam fonksiyonel bir kopyası kullanıcılara sunulur.
- ✓ Daha sonra kullanıcı dönüşlerine bağlı olarak değişiklikler ve güncellemeler gerçekleştirilir.

Hızlı Uygulama Geliştirme (RAD)

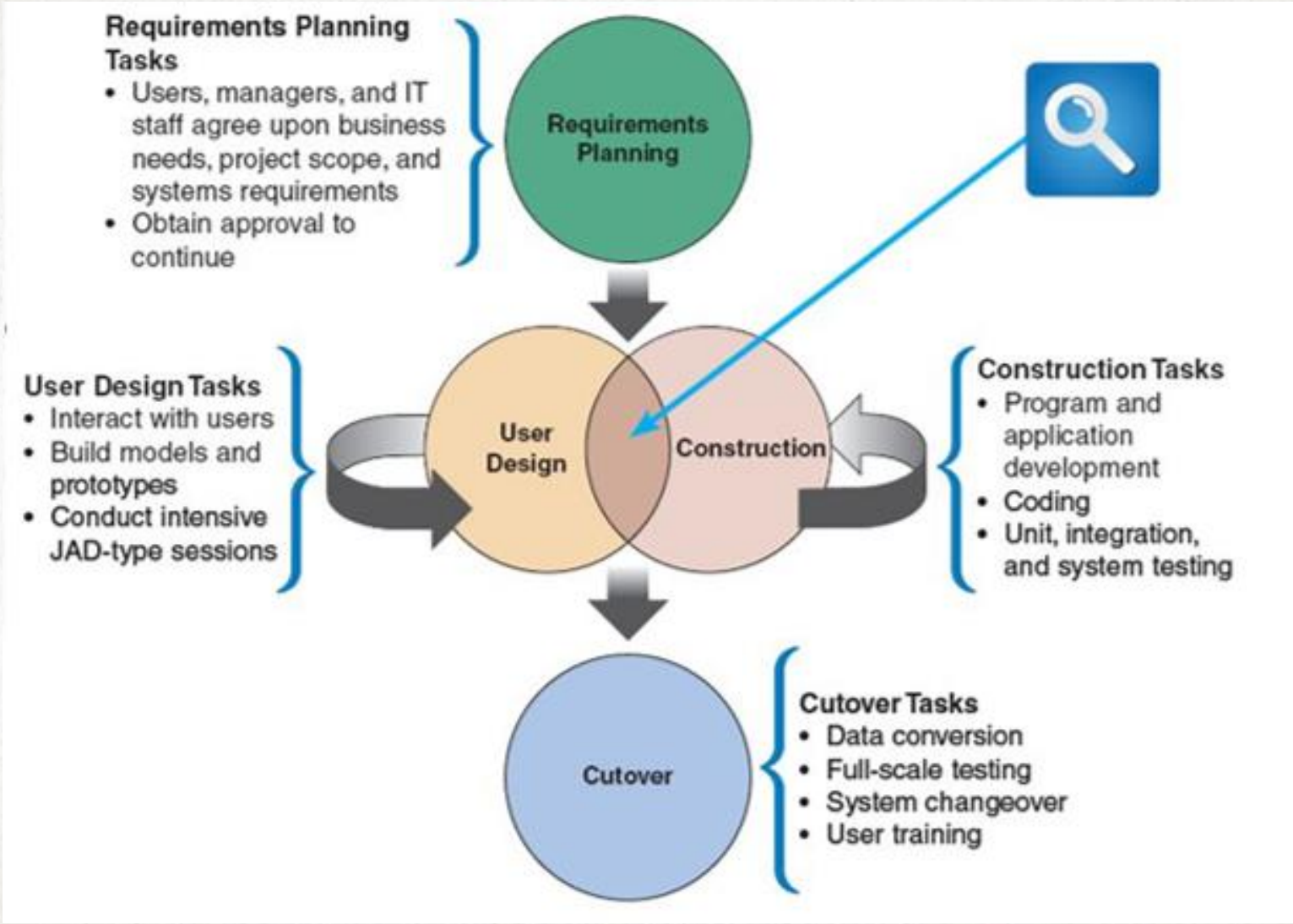
- ✓ Hızlı Uygulama Geliştirme (Rapid Application Development – RAD) planlama evresinden ziyade tasarım ve uygulama evrelerine odaklanan ve temelde son ürünü mümkün olan en kısa zamanda kullanıcıya ulaştırmayı amaçlayan modeldir.
- ✓ 4 temel adım içerir.
  - ➢ Gereksinim Planlanması: İşletme problemlerinin tartışıldığı yapısal bir çalıştay ile başlar.
  - ➢ Kullanıcı Tasarımı: Kullanıcılardan bilgileri otomatize araçlar çeker
  - ➢ Yapım: Kodlama ve tasarım araçları ile yazılımın tamamlanmasıdır.
  - ➢ Uygulamaya Geçiş: Uygulamanın kullanıcıya sunuşu, kabul testi ve kullanıcı eğitimi aşamalarını içerir.

Hızlı Uygulama Geliştirme (RAD)

RAD Modeli ne zaman kullanılır?

- ✓ Gereksinimler kabul edilebilir ölçülerde biliniyor ise
- ✓ Proje zaman kısıtlı (Time-Boxed) ise
- ✓ Yüksek performans başlangıçta talep edilmiyor ise
- ✓ Teknik risklerin düşük olduğu durumlarda
- ✓ Sistem modüler hale çevrilebiliyor ise

Hızlı Uygulama Geliştirme (RAD)



Hızlı Uygulama Geliştirme (RAD)

Avantajları

- ✓ Çok az kişi, çok kısa zamanda işlevsel bir yazılım sunar
- ✓ Zaman kısıtlı yaklaşım ile zaman çizelgesi ve maliyet ile ilgili riskler azaltılır
- ✓ Müşteri süreç içerisinde aktif katılımcıdır. Bu durum son kullanıcı tatmin olmaması riskini düşürür.
- ✓ Dokümantasyondan ziyade kodlama odaklıdır.
- ✓ İş, veri ve süreç ile ilgili bilişim toplama işleminde otomatize modelleme araçları kullanır.
- ✓ Kullanıcı ihtiyaçlarının açık olmadığı durumlarda da kullanılabilir.

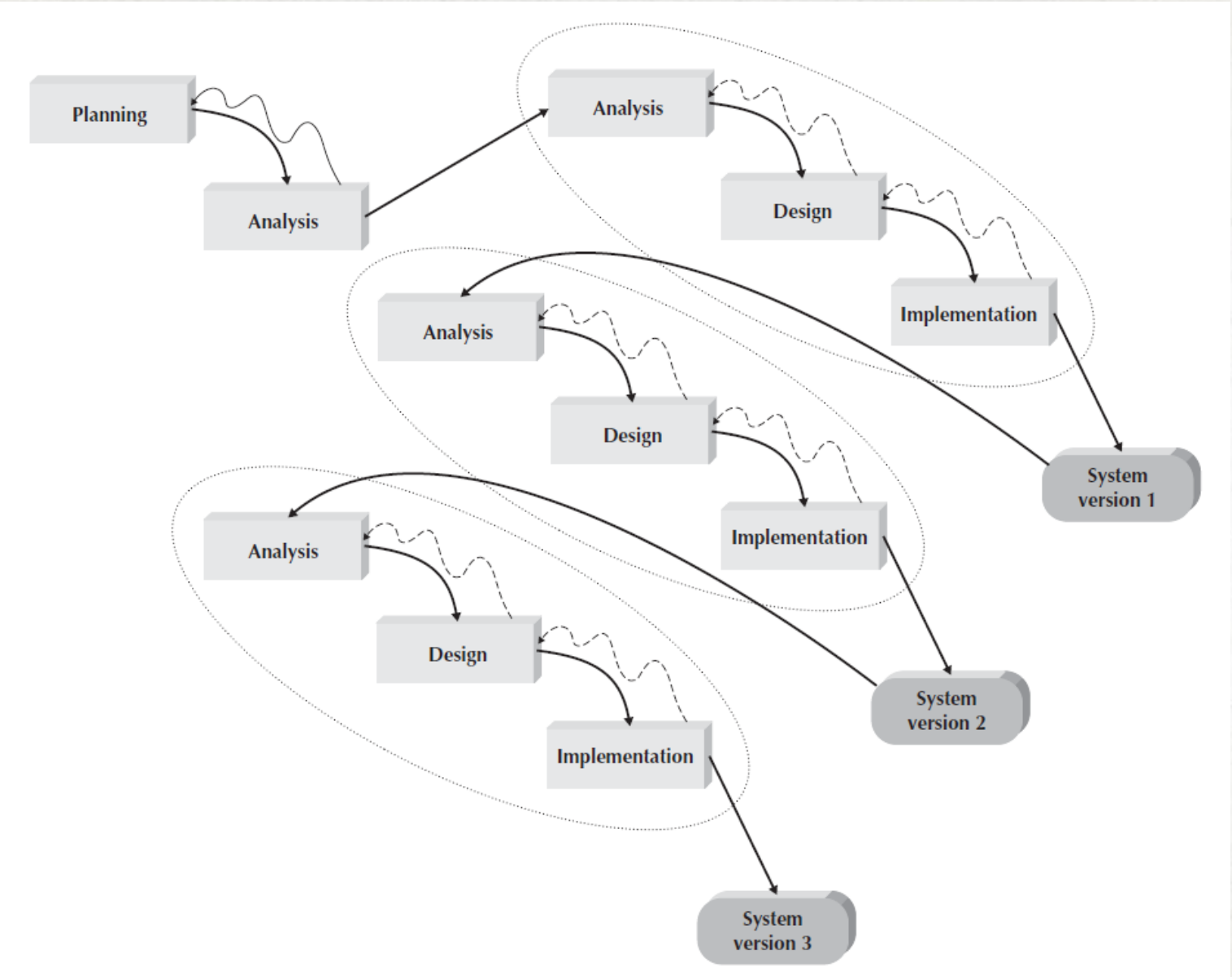
Dezavantajları

- ✓ Hızlandırılmış, zaman kısıtlı yapının bir gereği olarak hızlı müşteri geri bildirimine ihtiyaç duyar
- ✓ Projenin tam olarak sonlandırılamaması riski vardır.
- ✓ Mevcut sistemler ile adaptasyon zordur
- ✓ «Kodla, uygula, düzelt» stratejisi doğal olarak yüksek maliyetler oluşturur.

RAD Metodolojileri / Aşamalı Geliştirme

- ✓ Kademeli SDLC geliştirme olarak da literatürde kullanılır.
- ✓ Kademeli geliştirmeden farkı RAD araçlarını kullanmasıdır.
- ✓ Kullanıcıya belli fonksiyonları içeren « n » versiyonu sunulduğunda arka tarafta « n+ 1 » versiyonu ile ilgili çalışmalar devam eder.
- ✓ Çoğu zaman ikiden fazla versiyon paralel olarak geliştirilir.
- ✓ Temel amaç çevrim sürelerinin azaltılmasıdır.

RAD Metodolojileri / Aşamalı Geliştirme

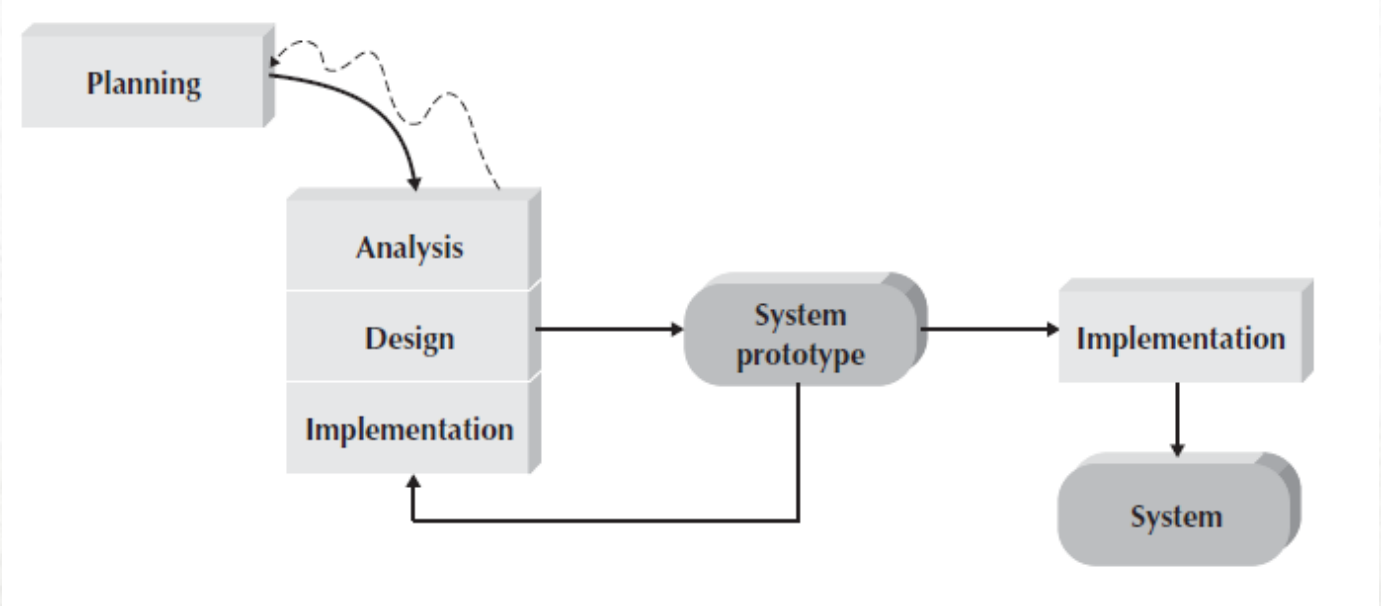


RAD Metodolojileri / Prototipleme

- ✓ İki tip prototipleme vardır.
  - ➢ Evrimsel Prototipleme
  - ➢ Kullan-At Prototipleme

- ✓ Evrimsel Prototipleme (Hızlı Prototipleme ismi ile de anılır) prototip bir sistemin kullanıcıya sunulması ve kullanıcı geri dönüşlerine bağlı olarak sistemin yeniden iteratif bir yapıda geliştirilmesini öngörür.
- ✓ İteratif SDLC modelinden farkı RAD araçları kullanılarak üretilmesidir.
- ✓ Prototipleme kullanıcı kabulü ile sonlanır ve son ürün yani bilişim sisteminde dönüşür.

- ✓ Evrimsel Prototipleme sistem riskini azaltmak için kullanılır.
  - ➢ Yanlış sistem oluşturma riski
  - ➢ Yeni teknoloji kullanmadan doğan risk
  - ➢ Farklı yöntemler kullanmadan doğan riskler



Avantajları

- ✓ Geliştiriciler ilk elden (kullanıcılardan) sistemi öğrenir.
- ✓ Son üründe daha isabetli sonuçlar meydana getirir
- ✓ Beklenmeyen ve başlangıçta belli olmayan gereksinimleri yönetir.
- ✓ Tasarım ve geliştirme aşamalarında esneklik sağlar
- ✓ Her adımda doğrulama olduğundan kararlılık sağlanır.
- ✓ Kullanıcı sistemin gelişimi ile ilgili bilgi sahibi olur.

Dezavantajları

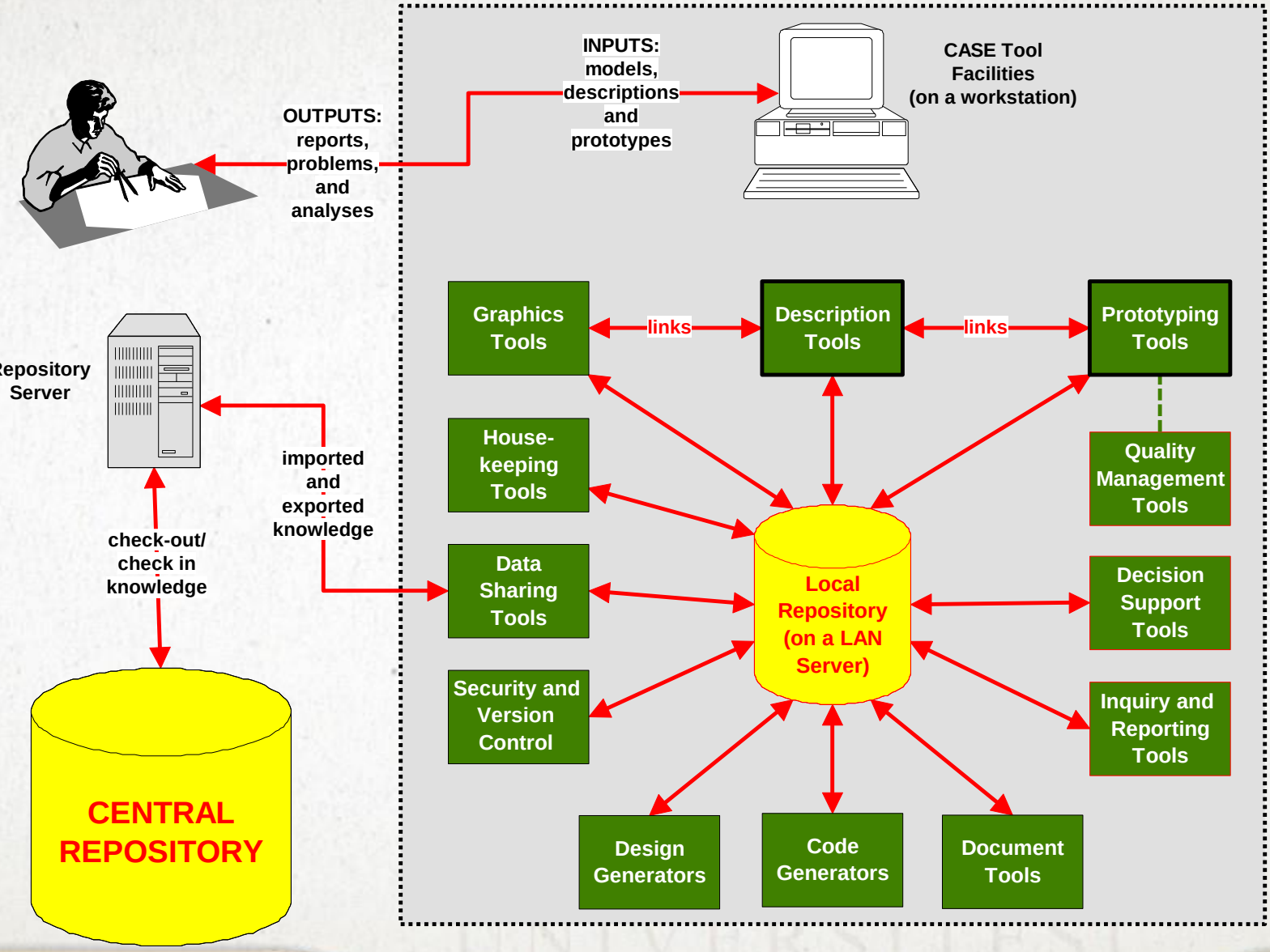
- ✓ Kodla ve Düzelt yapısının sonucu olarak ortaya çıkacak problemler (kirli kodlama)
- ✓ Sürdürülebilirlik problemleri
- ✓ Maliyet
- ✓ Sürecin sonlandırılması mümkün olmayabilir.

- ✓ Sistem analizcilerinin geliştirme süreçlerinde kullanabileceği bilgisayar destekli araçlar manasına gelir.
- ✓ CASE bir metodoloji değil, metodolojilerin uygulanmasını sağlayan bir destek sistemidir.
- ✓ CASE araçları aşağıdaki gibi sıralanabilir.

- ✓ Diagram çizme araçları
- ✓ Açıklama Araçları
- ✓ Prototipleme Araçları
- ✓ Sorgu ve Raporlama Araçları
- ✓ Kalite Yönetimi Araçları
- ✓ Karar Destek Araçları
- ✓ Dokümantasyon Araçları

- ✓ Tasarım Oluşturma Araçları
- ✓ Kodlayıcılar
- ✓ Test Araçları
- ✓ Veri Paylaşımı Araçları
- ✓ Versiyon Kontrol Araçları
- ✓ İdari Araçlar

RAD Araçları / Bilg. Destekli Sistem Müh (CASE)



RAD Araçları / Bilg. Destekli Sistem Müh (CASE)

SDLC Aşaması	Anahtar Aktivite	CASE Aracı Kullanımı
Planlama	• Üst-Seviye organizasyonel yapı gösterimi • Proje odağı oluşturma ve fizibilite çalışması	• Diyagram çizicileri • Açıklayıcılar • Dokümantasyon Araçları • Hesaplayıcılar
Analiz	• Sistem gereksinimlerini belirleme	• Diyagram çizicileri • Prototip Tasarlayıcıları
Tasarım	• Sistem tasarımları oluşturma	• Form ve Rapor Tasarlayıcıları • Prototip Tasarlayıcıları • Dokümantasyon Araçları
Uygulama	• Tasarımın bilişim sistemine dönüştürülmesi • Sistemin güncellenmesi ve geliştirilmesi	• Kodlayıcılar • Dokümantasyon Araçları • Bütün CASE araçları (iteratif yaklaşımlarda)

Avantajları

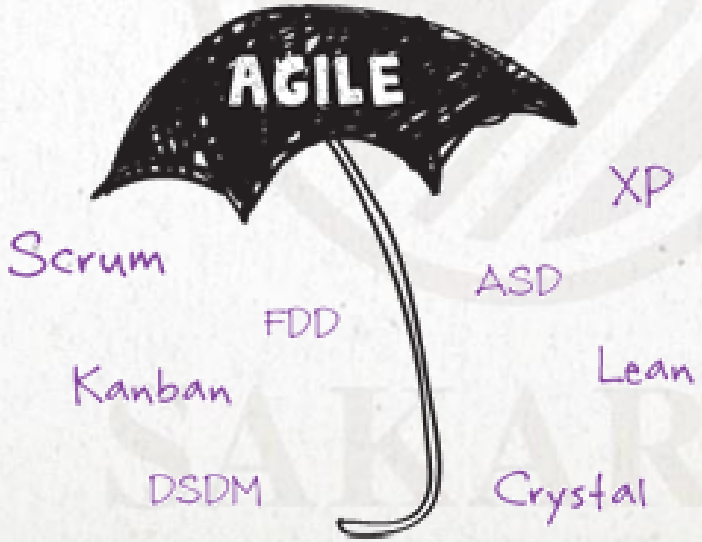
- ✓ Verimlilik artışı
- ✓ Kalitenin yükselmesi
- ✓ Daha iyi ve yönetilebilir dokümantasyon
- ✓ Kısaltılmış çevrim bakımı
- ✓ Kullanılan ve verimli çalışan metodolojileri sunması

Dezavantajları

- ✓ Özellikle dokümantasyon alanında kısıtlı esneklik
- ✓ Sistem geliştirme yaklaşımının ihtiyaçlarını tam olarak karşılayamama
- ✓ Belli seviyede uzmanlık gerektirmesi
- ✓ Başlangıç maliyetlerinin yüksek olması

Çevik Yaklaşımlar

- ✓ Bir veya daha fazla SDLC aşamasının hızlandırılması veya baypas edilmesidir.
- ✓ Genellikle daha informal bir süreçtir.
- ✓ «Kervan yolda düzülür» yaklaşımının yapısal halidir.
- ✓ Bütün çevik yaklaşımlarda amaç çalışan bir yazılım kopyasının mümkün olan en kısa sürede kullanıcıya sunulmasıdır.



Çevik Yaklaşımlar

- ✓ Çevik yaklaşımlar «**Değerlere**» bağlıdır.
  - ➢ İletişim
  - ➢ Basitlik (Yapabileceğin en basit hali ile yap)
  - ➢ Geri Beslemeler (Müşteri, sistem ve kodlayıcılar arası)
  - ➢ İnisiyatif Alma (Zor kararlar almaya hazırlık)

Çevik Yaklaşımlar

- ✓ Çevik yaklaşımlar «**Prensiplere**» bağlıdır.
  - ➢ Kademeli geliştirmeler küçük ve sıklıkla sunulan sistem versiyonları ile desteklenmeli
  - ➢ Tam zamanlı olarak müşteri ile iletişim sağlanmalı. Her an soru sormak için müşteriye ulaşılmalı
  - ➢ Kollektif ve eşli çalışma yapılmalı ama çok uzun çalışma saatleri olmamalı
  - ➢ Düzenli sistem versiyonları ile değişim desteklenmeli
  - ➢ Kodlar sürekli gözden geçirilerek basitlik sağlanmalı ve sürdürülmeli

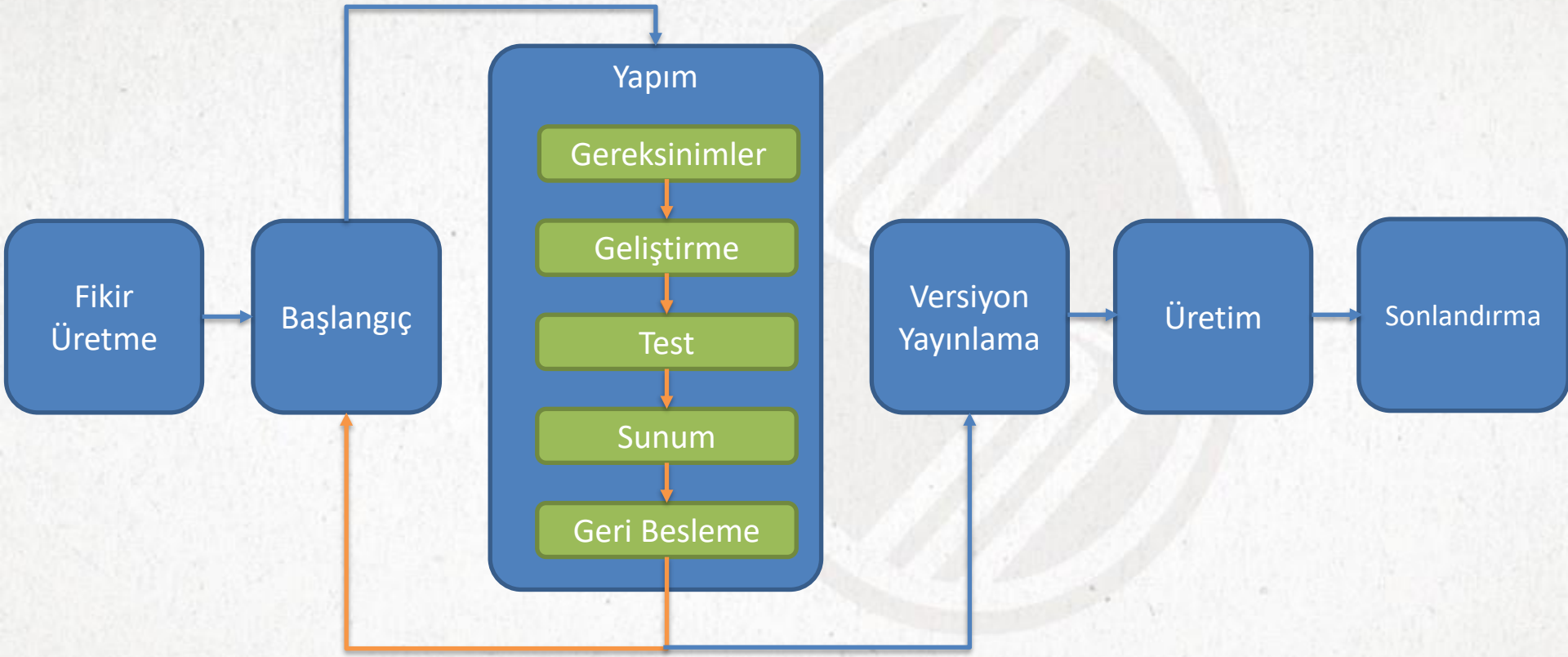
Çevik Yaklaşımlar

- ✓ Çevik yaklaşımlar «**Çekirdek Aktivitelere**» bağlıdır.
  - ➢ Planlama Oyunu (Use-Case Senaryoları)
  - ➢ Basit Tasarım
  - ➢ Metafor (Program nasıl çalışır)
  - ➢ Sürekli Test (Yazılımın doğrulanmasına odaklanma)
  - ➢ Yeniden üretim (Kodlar gözden geçirilip, temizlenmeli)
  - ➢ Eşli Programlama (Tek bir makinede, aynı kod, iki kodlayıcı)
  - ➢ Kolektif Kod Sahipliği (Kimse tek başına bir modül sahibi olmamalı)
  - ➢ Sürekli Entegrasyon
  - ➢ 40 Saat Çalışma (Kodlayıcılar aşırı çalıştırılmamalı)
  - ➢ Ulaşılabilir Müşteri (Müşteriye sürekli erişim)
  - ➢ Kodlama Standartları (bütün kodlar tek bir elden çıkmış gibi)

Çevik Yaklaşımlar

- ✓ Çevik Süreç Akışı:
  1. Fikir Üretme
  2. Başlangıç
  3. Yapım (iterasyonlar)
     1. İhtiyaç Belirleme
     2. Geliştirme
     3. Test
     4. Sunum
     5. Geri Besleme
  4. Versiyon Yayınlama
  5. Üretim (Destek)
  6. Sonlandırma

Çevik Yaklaşımlar



Çevik Yaklaşımlar

Çevik Yaklaşımlar ne zaman kullanılmalı ?

- ✓ Dinamik olarak çok kısa sürede yazılımın piyasaya sunulması ihtiyacı var ise
- ✓ Sistemde bir problem var ve hata araması için vakit yok ise
- ✓ Müşteri tatmini kademeli geliştirme için uygun ise
- ✓ Çevik prensipleri uygulayacak yönetici ve analizciler var ise
- ✓ Proje boyutu çok büyük değil ise

Hangi Metodoloji Seçilmeli

- ✓ Her biri farklı avantajlar sunan bu kadar metodolojiden uygun olanının seçilmesi her zaman analizci önündeki en büyük problemdir.
- ✓ Bu süreç çok kolay değildir.
- ✓ Metodoloji seçiminde kullanılabilecek kritik faktörler
  - ➢ Kullanıcı gereksinimlerinin belirgin olması
  - ➢ Yeni teknolojilere yatkınlık
  - ➢ Sistem Karmaşıklığı
  - ➢ Sistem Güvenilirliği
  - ➢ Kısıtlı Zaman Çizelgeleri
  - ➢ Çizelgelerin Görünürlüğü

Hangi Metodoloji Seçilmeli

✓ **Kullanıcı gereksinimleri belirgin değil ise**

Durumun Kontrolü	Metodoloji	Neden
Mükemmel	RAD – Prototipleme	• Detaylı planlamaya ihtiyaç yok • Kullanıcı ihtiyaçları süreç içerisinde toplanıyor
Mükemmel	Çevik Yaklaşımlar	• Detaylı planlamaya ihtiyaç yok • Kullanıcı katılımı yüksek
İyi	RAD – Aşamalı Geliştirme	• Detaylı planlama gerektirmesine rağmen Kullanıcı ihtiyaçları süreç içerisinde toplanıyor
İyi	Spiral Model	• Detaylı planlama gerektirmesine rağmen Kullanıcı ihtiyaçları süreç içerisinde toplanıyor
Zayıf	Yapısal Metodolojiler (Spiral Model Hariç)	• Eğer gereksinimler net değil ise yapısal modeller verimli olmaz

Hangi Metodoloji Seçilmeli

✓ **Yeni teknolojileri yatkınlık yoksa**

Durumun Kontrolü	Metodoloji	Neden
Mükemmel	RAD – Kullan-At Prototipleme	• Tasarım prototipleri ile riskin azaltılması
İyi	RAD – Aşamalı Geliştirme	• Yeni bilişim teknolojileri ilk tasarımdan önce mutlaka incelenmeli
İyi	Çevik Yaklaşımlar	• Programlama temelli yapı nedeni ile
Zayıf	RAD – Evrimsel Prototipleme	• Yeni teknolojilerin geç iterasyonlardaki prototiplere uygulanmasının zorluğu
Zayıf	Yapısal Metodolojiler	• Çok uzun süren sıralı yapı nedeni ile teknolojik adaptasyon sorunları

Hangi Metodoloji Seçilmeli

✓ **Sistem karmaşık yapıda ise**

Durumun Kontrolü	Metodoloji	Neden
Mükemmel	RAD – Kullan-At Prototipleme	• Tasarım prototipleri ile detaylı analiz ve tasarımların sağlanması
Mükemmel	Çevik Yaklaşımlar (Küçük ve orta boyutlu projeler)	• Programlama süreçlerinin nesne yönelimli iteratif yapısı nedeni ile
İyi	RAD – Aşamalı Geliştirme	• Kullanıcı katılımının erken dönemlerde sağlanması
İyi	Yapısal Metodolojiler	• Katı ve detaylı planlama ve analiz süreçlerinin kullanılması nedeni ile
Zayıf	RAD – Evrimsel Prototipleme	• Yeni teknolojilerin geç iterasyonlardaki prototiplere uygulanmasının zorluğu
Zayıf	Çevik Yaklaşımları (Büyük boyutlu projeler)	• Programlama süreçlerinin nesne yönelimli iteratif yapısı nedeni ile

Hangi Metodoloji Seçilmeli

✓ **Sistem güvenilirliği yüksek öncelik ise**

Durumun Kontrolü	Metodoloji	Neden
Mükemmel	RAD – Kullan-At Prototipleme	• Tasarım prototipleri ile riskin azaltılması
Mükemmel	Çevik Yaklaşımlar	• Test, gereksinim değerlendirilmesi üzerinde yoğun odaklanma
İyi	RAD – Aşamalı Geliştirme	• Analiz ve Tasarım süreçlerine kullanıcı katılımının erken dönemlerde sağlanması
İyi	Yapısal Metodolojiler	• Katı ve detaylı planlama ve analiz süreçlerinin kullanılması nedeni ile
Zayıf	RAD – Evrimsel Prototipleme	• İlk sistem prototipinde dikkatli analiz ve tasarım süreçlerinin yürütülmemesi nedeni ile

Hangi Metodoloji Seçilmeli

✓ **Proje zaman kısıtlı ise**

Durumun Kontrolü	Metodoloji	Neden
Mükemmel	Çevik Yaklaşımlar	• Çok kısa çevrim zamanı • Çizelge değişikliklerine adaptasyon
Mükemmel	RAD – Aşamalı Geliştirme	• Çok kısa çevrim zamanı • Çizelge değişikliklerine adaptasyon
Mükemmel	RAD – Evrimsel Prototipleme	• İlk prototip için çok kısa çevrim zamanı
İyi	RAD – Kullan-At Prototipleme	• Kullanıcı ihtiyaçları belirleme sürecinin nispeten uzun olması
Zayıf	Yapısal Metodolojiler	• Planlama sürecinin çok uzun olması • Çizelge değişimlerine uygun olmaması

Hangi Metodoloji Seçilmeli

✓ **Zaman çizelgesi görünürlüğü düşük ise**

Durumun Kontrolü	Metodoloji	Neden
Mükemmel	Çevik Yaklaşımlar	• İşbirlikçi analiz ve tasarım süreçlerinin varlığı
Mükemmel	RAD – Aşamalı Geliştirme	• Risk faktörlerinin erken tespiti
Mükemmel	RAD – Evrimsel Prototipleme	• Risk faktörlerinin erken tespiti
İyi	RAD – Kullan-At Prototipleme	• Kullanıcı ihtiyaçları belirleme sürecinin nispeten uzun olması nedeni ile proje çizelgesinin tahmini zor
Zayıf	Yapısal Metodolojiler	• Planlama ve analiz süreçlerinin uzunluğu nedeni ile proje süreci tahmini zor

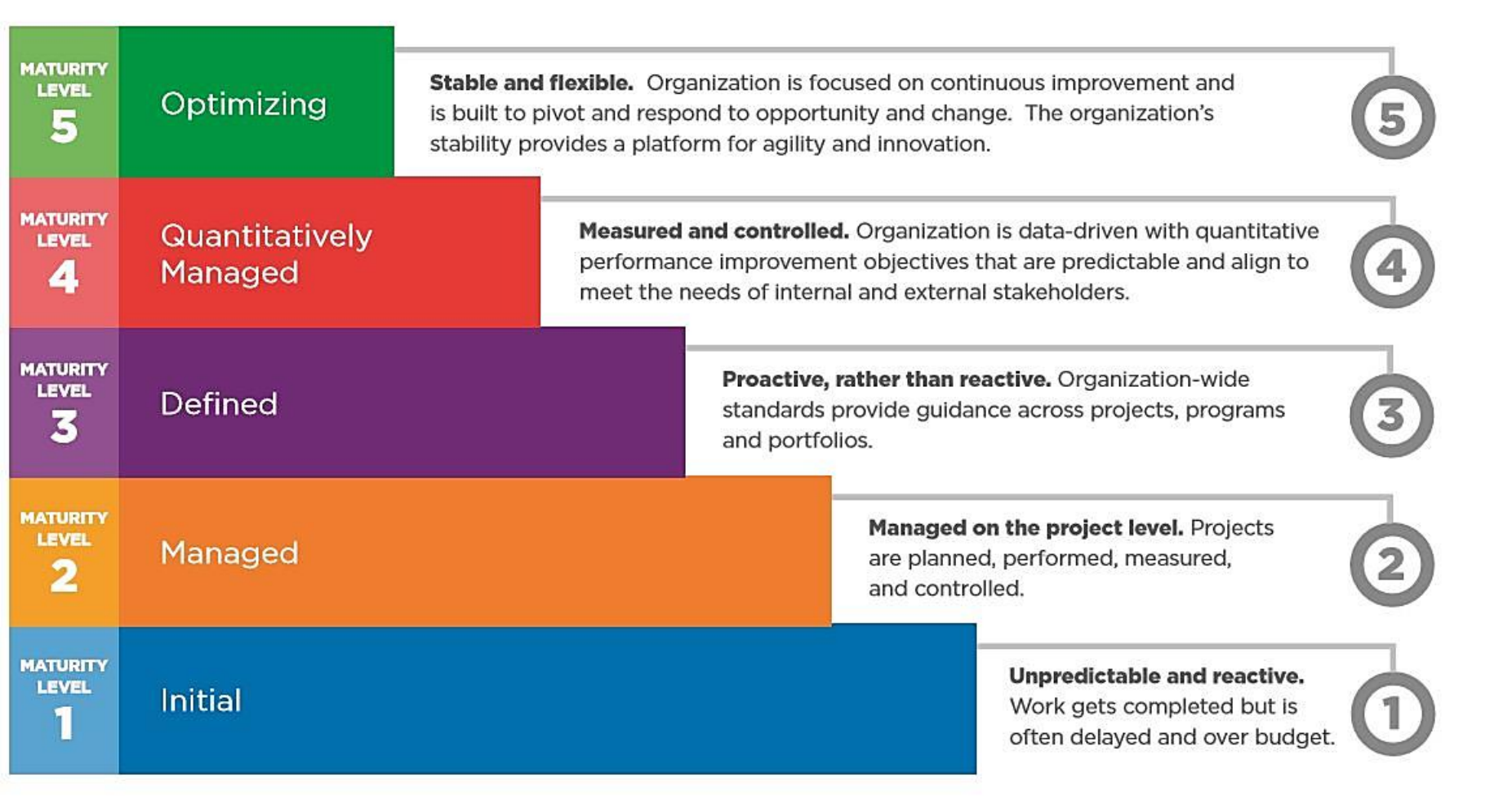
Hangi Metodoloji Seçilmeli

Sistem Geliştirme Yetkinliği	Yapısal Metotlar	Aşamalı Geliştirme	Evrimsel Prototipleme	Kullan-At Prototipleme	Çevik Yaklaşımlar
Kullanıcı gereksinimleri belirgin değil ise	Zayıf	İyi	Mükemmel	Mükemmel	Mükemmel
Yeni teknolojileri yatkınlık yoksa	Zayıf	İyi	Zayıf	Mükemmel	İyi
Sistem karmaşık yapıda ise	İyi	İyi	Zayıf	Mükemmel	İyi
Sistem güvenilirliği yüksek öncelik ise	İyi	İyi	Zayıf	Mükemmel	Mükemmel
Proje zaman kısıtlı ise	Zayıf	Mükemmel	Mükemmel	İyi	Mükemmel
Zaman çizelgesi görünürlüğü düşük ise	Zayıf	Mükemmel	Mükemmel	İyi	Mükemmel

Yetkinlik Olgunluk Modeli:

- ✓ Amerikan ordusunun yazılım firmalarını değerlendirmek amacıyla Yazılım Mühendisliği Enstitüsüne hazırlattığı bir modeldir.
- ✓ Model etkin bir yazılım geliştirme süreci ile ilgili anahtar aktiviteleri (elemanları) açıklamaktadır.
- ✓ Evrimsel bir gelişim yolu önerisinde bulunmaktadır.
- ✓ Yazılım geliştirme süreçlerinin nasıl yönetileceği ile ilgili bir kılavuz niteliğindedir.

Yetkinlik Olgunluk Modeli:



Yetkinlik Olgunluk Modeli:

- ✓ **Level 5: Optimize Edilmiş:**
  - Süreç ve Teknoloji değişim yönetimi
  - Hata önleme
- ✓ **Level 4: Yönetilmiş:**
  - Yazılım kalitesi yönetimi
  - Kantitatif süreç yönetimi
- ✓ **Level 3: Tanımlanmış:**
  - Akran değerlendirmesi
  - Gruplar arası koordinasyon
  - Yazılım son ürün mühendisliği
  - Bütünleşik yazılım yönetimi
  - Eğitim programı
  - Organizasyon süreç odağı ve tanımı
- ✓ **Level 2: Tekrarlanabilir:**
  - Yazılım konfigürasyon yönetimi
  - Yazılımı kalite güvencesi
  - Yazılım proje planlama ve takibi
  - Gereksinim yönetimi
- ✓ **Level 1: Başlangıç**

Kaynaklar

- ✓ CEBECİ, H. İ. (2019). *Sistem Analizi ve Tasarımı Ders Notları*
- ✓ Valacich, J., George, J., & Hoffer, J. (2015). *Essentials of systems analysis and design*. Prentice Hall Press.
- ✓ Valacich, J. S., & George, J. (2016). *Modern systems analysis and design*. Pearson.
- ✓ Yeates, D., & Wakefield, T. (2004). *Systems analysis and design*. Pearson Education.
- ✓ Kendall, K. E., & Kendall, J. E. (2014). *Systems analysis and design*. Prentice Hall Press.
- ✓ Rosenblatt, H. J. (2014). *Systems analysis and design*. Cengage Learning.
- ✓ Bentley, L. D., Dittman, K. C., & Whitten, J. L. (2007). *Systems analysis and design methods*. Irwin/McGraw Hill.
- ✓ Dennis, A., Wixom, B. H., & Tegarden, D. (2015). *Systems analysis and design: An object-oriented approach with UML*. John Wiley & Sons.