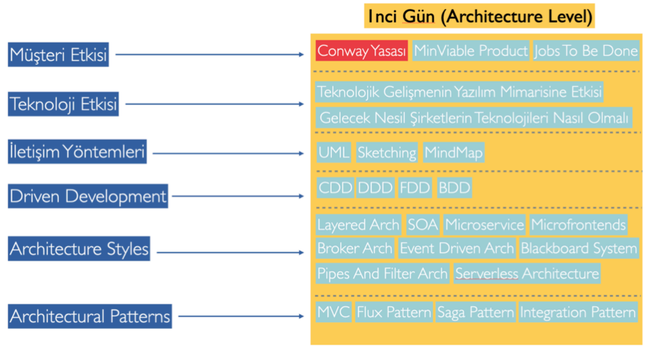
YAZILIM MİMARİSİ

https://www.onurdayibasi.com/trainings/training-software-architecture/

"Real learning is **not memorizing** knowledge. It's understanding and knowing how to use and find knowledge. Learning is what you do with knowledge, how you integrate it, how you talk to your family, friends and classmates about it. That's what learning is." (Dennis Littky)

İÇERİK

- Temel Kavramlar
- Deneyimler
- Yazılım Mimari Yol Haritası
- Bu Günün Konu Başlıkları



TANIŞMA

EĞİTİMCİYİ TANIYALIM

ÖĞRENCİLERİ TANIYALIM

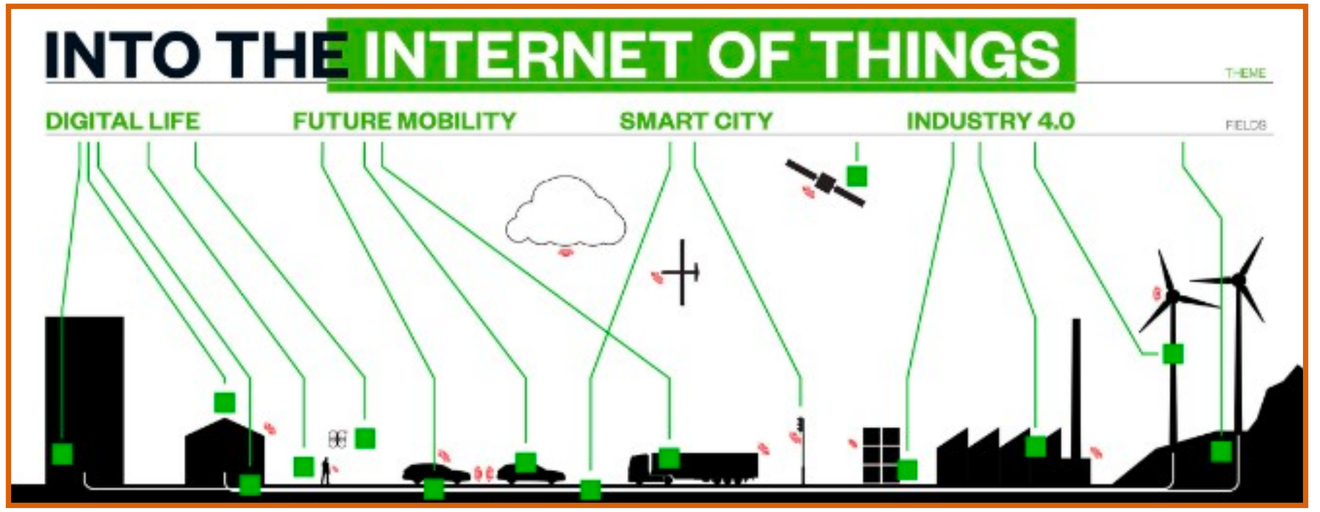
TEMEL KAVRAMLAR

YAZILIM MİMARİSİ

YAZILIM MİMARİSİ NE İŞİMİZE YARAYACAK ?

Yazılım Sistemlerinin Yaşamımızdaki Yeri

- Cep telefonlarında,
- Televizyonlarda,
- Şehirdeki trafik ışıklarında,
- Metrolar ve yaşam alanlarında,
- Savunma sistemlerinde,
- Okulda,
- Havalimanlarında,
- Uzay Mekiklerinde,



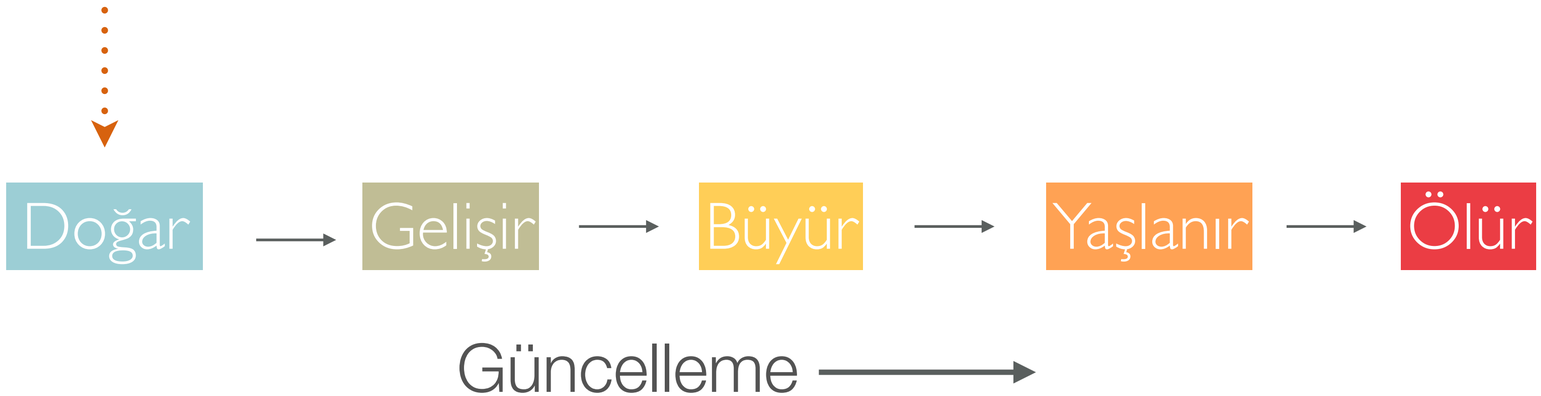
Hayatımızın **her alanında**, her işletmenin **merkezinde** yazılım yer alıyor.
Endüstri 4.0 ile birlikte giderek kapsadığı alanında genişletecek.

Yazılım Mimarisinin Önemi

Hal böyleyken birbiri ile çalışabilen büyük sistemlerin oluşturulabilmesi için, yazılımların, sistemlerin, ekiplerin belli bir düzen ve disiplinde çalışabilmesi gerekiyor.

İşte burda **Yazılım Mimarisi** devreye giriyor.

Yazılım Sistemleri Canlıdır



Şehirler, şirketler ve yazılımı yaşamlarının önemli bir parçası olarak kullanan organizasyonlar;

Gelişen teknoloji ve yeni gereksinimlere ayak uydurabilecek şekilde yazılımlarını **güncellemeli**,

- Bunu yaparken mevcut sistemi bozmadan,
- Kaliteyi koruyarak,
- Kabul edilebilir bir harcama ile bunu gerçekleştirmelidir.

Yazılım, Donanım Uyumu ve Maliyetler

Yazılım ve Donanım Yazılım Mimarisinin karakteristiklerini oluşturan temel gereksinimlerdir.

Bu derste her ne kadar yazılımdan **(Software)** bahsetsekte. Yazılım belirli bir kalitede, güvenilir , performanslı, güvenlikli, dayanıklı olabilmesi için üzerinde çalışacağı donanım ile uyumuna bağlıdır.

Burada devreye **para** girdiği için elinizde sınırsız kaynak varmış gibi düşünmemeniz gerekir. İhtiyaçlarınız doğrultusunda en **optimum sonuçları** alacak şekilde yazılım ve donanımı bir araya getirmelisiniz.

YAZILIM MİMARİSİ TANIMI

Mimari Nedir?

Architecture is the fundamental **organization** of a **system** embodied in its **components**, their **relationships** to each other, and to the **environment**, and the principles guiding its design and evolution. [IEEE 1471]

Mimari, sistemin temel organizasyonuna ait somutlaşmış bileşenlerinin birbirleriyle, çevreleri ile ilişkilerinde, tasarımına ve gelişimine **yol gösteren** prensiplerdir.

Sistem Nedir?

A system is a collection of **components** organized to accomplish a specific function or set of functions. The term system encompasses individual applications, systems in the traditional sense, subsystems, systems of systems, product lines, product families, whole enterprises, and other aggregations of interest. A system exists to fulfill one or more missions in its environment. [IEEE 1471]

Sistem, belirli **bir işlevi veya bir dizi işlevi** gerçekleştirmek için düzenlenmiş bir **bileşen grubudur** …

Bileşen Nedir?

[A component is] a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A component defines its behavior in terms of provided and required interfaces. As such, a component serves as a type, whose conformance is defined by these provided and required interfaces (encompassing both their static as well as dynamic semantics) [UML 2.0]

Sistemin oluşturan **modüler parçalarıdır.** Kendi içeriklerini tutarak belirli arayüzler üzerinden birbirleri ile iletişim kurarlar. Bu arayüzler üzerinden belli davranışlar sergilerler ve tuttukları içerikleri değiştirirler.

Çevre Nedir?

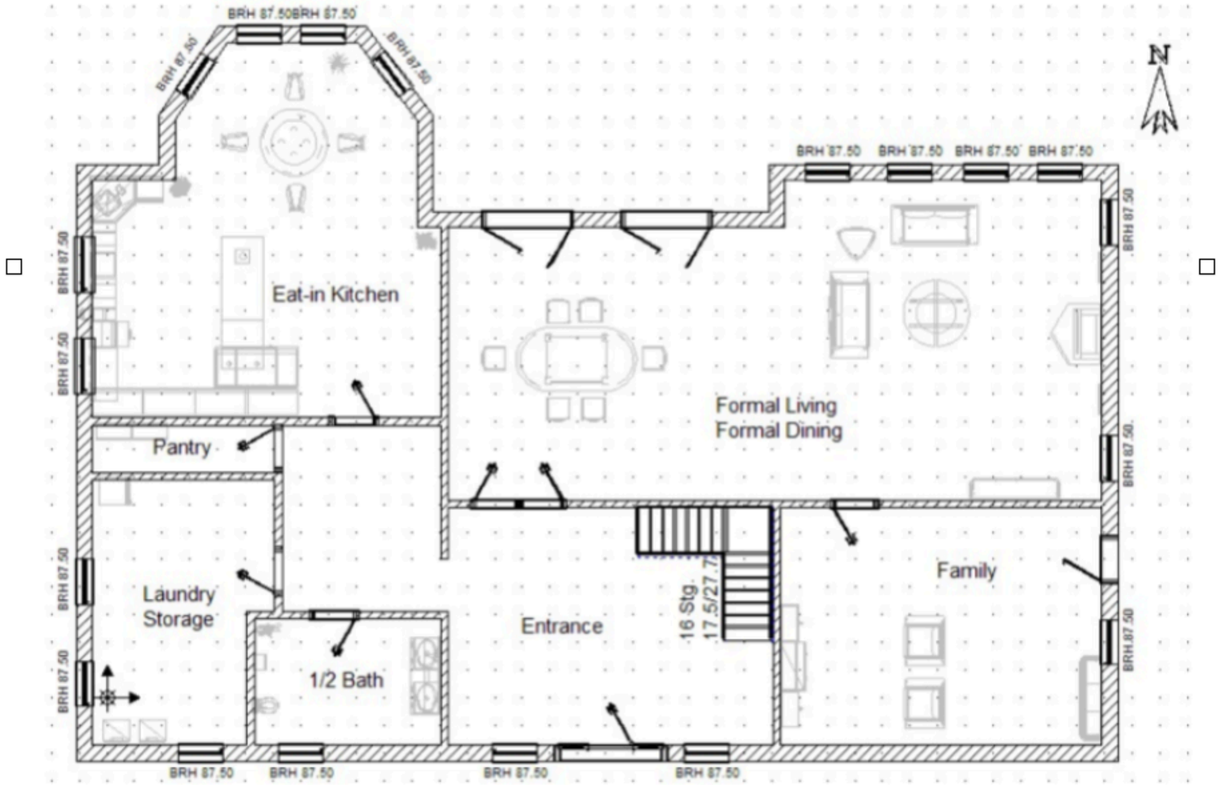
The environment, or context, determines the setting and circumstances of developmental, operational, political, and other influences upon that system. [IEEE 1471]

Çevre veya bağlam, sistem üzerindeki gelişimsel, operasyonel, politik ve diğer etkilerin **ortamını ve koşullarını** belirler.

YAZILIM MİMARİSİ YAPIYI BELİRLER (STRUCTURE)

Evinizin Mimarisi Denilince Aklınıza Ne Geliyor ?

- Odalar
- Girişler
- Kapılar
- Pencereler
- Odaların Türleri
- Büyüklükleri
- Birbirine Bağlantıları



Ev Planı

Yazılımı ile Ev Mimarisi Arasında Benzerlik Kurarsak

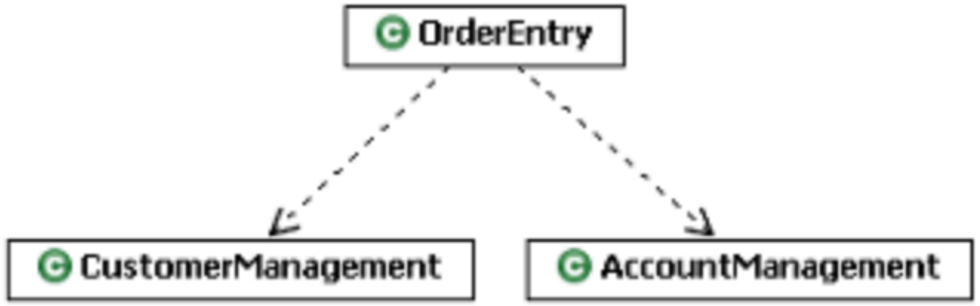
Yazılım Mimarisinde de aynısı her yazılımcının anlayacağı bazı standart çizimler ve bu çizimlerin dili mevcuttur. Buna UML yani **Unified Modeling Language** diyoruz.

UML kullanarak aşağıdaki elemanları kullanarak bu yapıyı oluşturmaya çalışır.

- subsystem, (Alt-sistem)
- a process,
- a library, (kütüphane)
- a database, (veritabanı)
- a computational node, (işlemci düğüm)
- a legacy system, (kullanılan mevcut sistem)
- off-the-shelf product, (hazır alınmış ürün)

Yazılımı Mimari Yapısı

Yapısı (Structure) sadece elemanların tek tek havada durması değil, birbirini kapsayarak daha büyük bileşenler oluşturması veya birbirleri ile belli arayüzler aracılığı ile iletişiminin olduğununda gösterilmesi içerir. Bu iletişim nasıl olacağı, **tek/çift yönlü** , **sync/async** vb.. detaylarıda içermelidir.



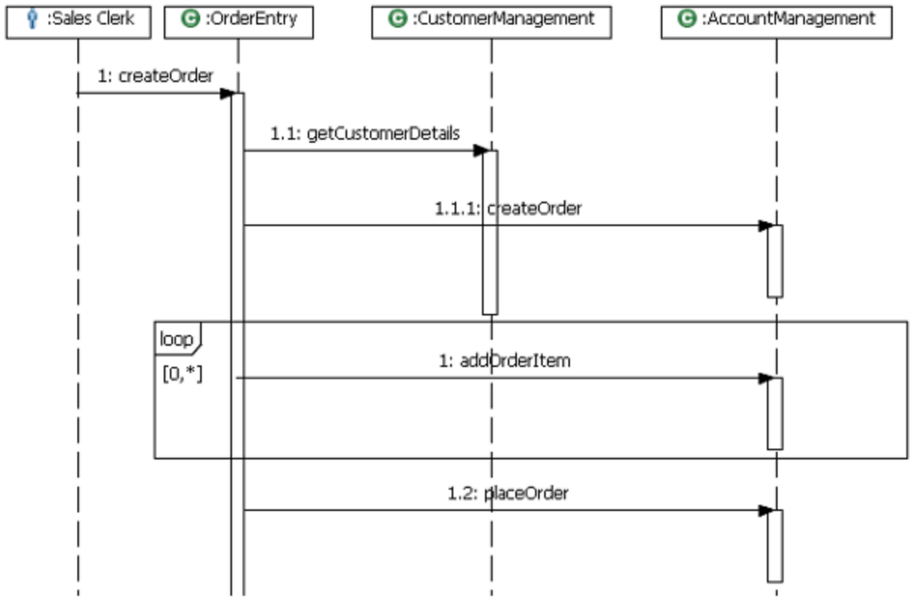
YAZILIM MİMARİSİ DAVRANIŞI (BEHAVIOR) BELİRLER

Yazılımı Mimarisi Sistemin Davranışını Belirler

Bileşenler ve çevre yazılımlar bu etkileşimi sistemin istenilen davranışı sağlayabilmesi için gerçekleştirir.

UML **Sequence Diagram**'da bileşenlerin bir işi yapabilmesi için hangi sıra ile birbirleri ile etkileşimi gireceğinin bilgisi verilir.

Mimarinin aynı zamanda üst seviye davranışı da belirlemesi gerekir.



YAZILIM MİMARİSİ ÖNEMLİ ÖĞELER ÜZERİNE ODAKLANIR

Mimariye Belli Soyutlamalar Yaparak Bakmalısınız

Yazılım Mimarisi sistemin yapısını ve davranışını tanımladığını söylemiştik. Burada kastedilen **tüm detaylı yapısı ve tüm detaylı davranışları değildir**.

Mimari demek sisteme belli bir **soyutlama yaparak bakmak** demektir. Önemli, temel elemanlar ile davranışları ortaya çıkararak detaylarda boğulmamaktır.

Detaydan Kaçının

Bileşenleri **belirli büyüklüklerde detaya girmeden** tanımlamak gerekiyor ki dışarıdan bakıldığında anlaşılsın, ve bileşenleri ilerleyen süreçlerde sürekli değiştirmek gerekmesin.

Davranışları iş akışlarını **üst seviye tanımlamalı** ki bu detaylardaki değişikliklerden etkilenmesin.

Mimaride'de tanımlanması gereken bileşenler, bunları birbirleri ile olan iletişimlerinin **kalitesi, güvenilirliği, ölçeklendirebilirliği, güvenliği, güvenilirliği, performansı** gibi önemli değerleri olmalıdır.

Mimari, Mimarın sistemin tüm detaylarından kendisini soyutlayarak **karmaşıklığı** yönetmesini sağlayan bir araçtır.

Önemli Öğeler/Değerler Kümesi Hiç Değişmez mi ?

Mimari Zaman İçerisinde **önemli öğeler kümesi** hiç değişmez mi ?

- İhtiyaçlar iyileştirilmesi
- Risklerin tanımlanması
- Çalıştırılabilir yazılımların oluşturulması
- Ve bunlardan alınan dersler sonrasında

zaman içerisinde bir takım konular daha iyi anlaşılacağı için önemli öğeler kümesi değiştirebilir.

Burada mimarinin bu **ciddi değişikliklere karşı ufak revizeler ile ilerleyebiliyor olması** iyi bir mimariye **kararlı bir yapıya** sahip olduğunun göstergesidir.

YAZILIM MİMARİSİ PAYDAŞLARIN İHTİYAÇLARINI DENGELER

PAYDAŞLAR

- Son Kullanıcı
- Sistem Yöneticisi
- Pazarlama Market
- Müşteri
- Yazılım Geliştiriciler
- Proje Yöneticisi
- Bakım Ekibi
- Mimar

Paydaşların Beklentileri

Son kullanıcı sistemin **doğru davranışı, performansı, güvenilirliği , kullanabilirliği, erişilebilirliği ve güvenliği** ile ilgilenir.

Sistem yöneticisi, sistemin davranışı, **yönetimi ve takibi için gerekli olan araçlarla** ilgilenir.

Pazarlama ve Market çalışanları diğer ürünlere göre **rekabetçi özellikler**, **zamanında markette yer almak** , diğer ürünlere göre **pozisyonu** ve maliyetle ilgilenir.

Paydaşların Beklentileri

Yazılım Geliştiriciler **açık gereksinimler, basit ve tutarlı tasarım** yaklaşımları ile ilgilenir.

Proje yöneticisi tahminlenebilir takip edilebilir bir proje , proje planı, **kaynakların üretkenliği ve bütçe** ile ilgilenir.

Bakım ile uğraşan çalışanlar **anlaşılır, tutarlı ve belgelenmiş tasarım yaklaşımı** ve kolay değişiklik yapabilmeyle ilgilenir.

Beklentilerin Çakışması ve Dengelenmesi

Bu ihtiyaçlar her zaman her paydaşın istediği gibi olamaz. Birisinin ihtiyacı henüz başka bir nedenden diğer bir paydaşın ihtiyacının tanımlanmasını **engelliyor** olabilir. Veya ihtiyaçlar birbirleri ile çelişiyor/çakışıyor olabilir.

Mimari paydaşların birbirleri ile anlaşabilmesi için **ihtiyaç dengelemesini gerçekleştirir.**

İhtiyaçlar her zaman fonksiyonel olmaz, hatta **fonksiyonel olmayan gereksinimler** tüm sistemi etkileyen kalite, sınırlandırmalar ele almak fonksiyonel gereksinimlerden çok çok daha zordur. Mimar'ın özellikle **üzerinde durması ve ayrıntılı hale getirmesi** gerekenler bu özelliklerdir.

YAZILIM MİMARİSİ MANTIĞA DAYALI KARARLAR ÜZERİNE KURULUR

Mimari Kararlar Neleri İçermeli

Mimari **kesin sonuç** anlamına gelmez. Neden bu şekilde tasarlandığına, karar alındığına dair mantıkların, gerekçeleri içeren bilgiler ve belgeler **bütünüdür**.

Bu bilgiler, özellikle sistemin **bakımını yapanlar olmak üzere birçok paydaşla** ilgilidir.

Bu bilgiler, alınan kararların **ardındaki mantığı** yeniden gözden geçirmesi ve değerlendirilmesi oldukça önemlidir.

Zaten yazılımın ilerleyen süreçlerinde mimarinin **sürekli olarak değişmemesi** bu mantıksal kararlardan kaynaklanacaktır.

YAZILIM MİMARİSİ MİMARİ STİLLERE UYGUNLUK BİRÇOK RİSKİ AZALTIR

Mimari Stiller / Örüntüler

Çoğu mimari, **benzer endişeleri** paylaşan sistemlerden türemiştir.

Bu üst düzey soyutlamalardaki benzerliklere **mimari stiller** diyebiliriz.
Mimari stil geçmiş sistem deneyimlerinden öğrenilmiş ,edinilmiş örüntülerdir.

Mimari örüntü, bu mimari stili gerçekleştirmenin bir yoludur.

Microservices

Monalitic

Layered

Component-Based

SOA

MVC Pattern

Flux Pattern

Saga Pattern

Örüntü Türlerini Bilmek Neden Önemlidir?

Bu örüntüler belli yapı ve davranışı sağlayan sistemlerin ortak uyması gereken prensipleri belirleyen yöntemleri tanımlamıştır.

Mimar bu örüntüleri bilip, geliştirdiği sistemin hangi örüntüye uyduğunu bulursa, daha önceden karşılaşılmış sorunların **nasıl çözümleneceği**,
hangi **en iyi yöntemleri kullanacağı** bu örüntüleri nasıl kullanması gerektiği bir çok kaynaktan bulabilecektir.

Bu **sayede;**

- işleri kolaylaşacak,
- hata yapma olasılığı azalacak,
- ve yanlış yönde ilerleme durumu ortadan kalkacaktır.

YAZILIM MİMARİSİ ÇEVRESİNDEN ETKİLENİR

Mimariyi Etkileyen Çevresel Faktörler

Mimari çevresinden etkilenir.

Çevresi **sistemin** sınırlarında diğer **hangi sistemler ile etkileşimde olacağını** belirlediği için mimariniz çevrenizdeki sistemler ile uyumlu çalışması durumunda anlamlı hale gelecektir.

Bu uyumluluklar mimariye bir çok ekstra iş ve zorluk çıkarabilir.

Mimariyi Etkileyen Çevresel Faktörler Örnek

Organizasyon standartlarına uyma

Harici Teknik Kısıtlamalar Örn: Harici bir arayüzle arayüz oluşturmak ihtiyacı

Sistem ve Harici düzenleyici standartlara uyma.

Peki Mimari Çevresel Faktörleri Etkiler mi?

Bunun yanında sizin geliştirmiş olduğunuz Mimari'de çevreyi etkileyebilir ve değiştirebilir.

Örneğin ortak , **tekrar kullanabilir bileşenler** sayesinde çevredeki bazı bileşenlere ihtiyaç kalmayabilir, ortamın teknolojisini daha ileriye götürmeye zorlayabilir.

Buradan da anlaşılacağı gibi ***çevre ve mimari*** birbirini etkileyen yapılardır.

Sistem Kavramını Tekrar İnceleyelim.

[A **system** is] a set of resources that provide services that are used by an enterprise to carry out a business purpose or mission. System components typically consist of hardware, software, data, and workers

Sistem bir grup kaynak tarafından işletmenin belli bir amaç ve misyonu için servis sağlamasıdır. Sistem bileşenleri de **donanım, yazılım, veri ve çalışanlardan** oluşur.

Sistem Mühendisinin Görevi

Sistem Mühendisliği alanında bu kaynakların nasıl kullanılacağı **maliyet-performans analizleri** yapılarak karşılaştırılır.

Örneğin çok zor yapılacak bir işiniz olsun, bunu Donamıma para koyarak yapabilirsiniz en pahalısı bu olacaktır belki ama en performans sağlayanıda bu olacaktır. Donanımdan feragat ederek bunu yazılım ile çözebilirsiniz, hazırını alabilirsiniz veya kendi ekibinize geliştirtebilirsiniz gibi

YAZILIM MİMARİSİ EKİP YAPISINI ETKİLER

Mimari Ekip Yapısını Etkiler

Yazılım Mimarisi **ana bileşenleri** oluştururken bunlar ile **uyumlu** birbirinden farklı ekiplerde oluşturur.

Bunların **farklı disiplinlerde, farklı yetenek setlerine , farklı son kullanıcılara** sahip olması bu ayrışmaya neden olur.

- sipariş işleme sistemi ,
- sipariş girişi,
- hesap yönetimi,
- müşteri yönetimi, harici sistemlerle entegrasyonlar,
- kalıcılık(persistency) ve güvenlik altyapı

YAZILIM MİMARİSİNDE BELGELEME ÖNEMLİDİR

Belgelemenin Önemi

Mimari resmi olarak belgelenmese veya sistem son derece basit ve örneğin tek bir unsurdan oluşsa bile, her sistemin bir mimarisi vardır.

Belgelenmiş mimariler, mimariyi kaydetme süreci **doğal olarak fikirleri değerlendirmeye yol açtığından**, daha dikkatli bir şekilde değerlendirilme yapılır.

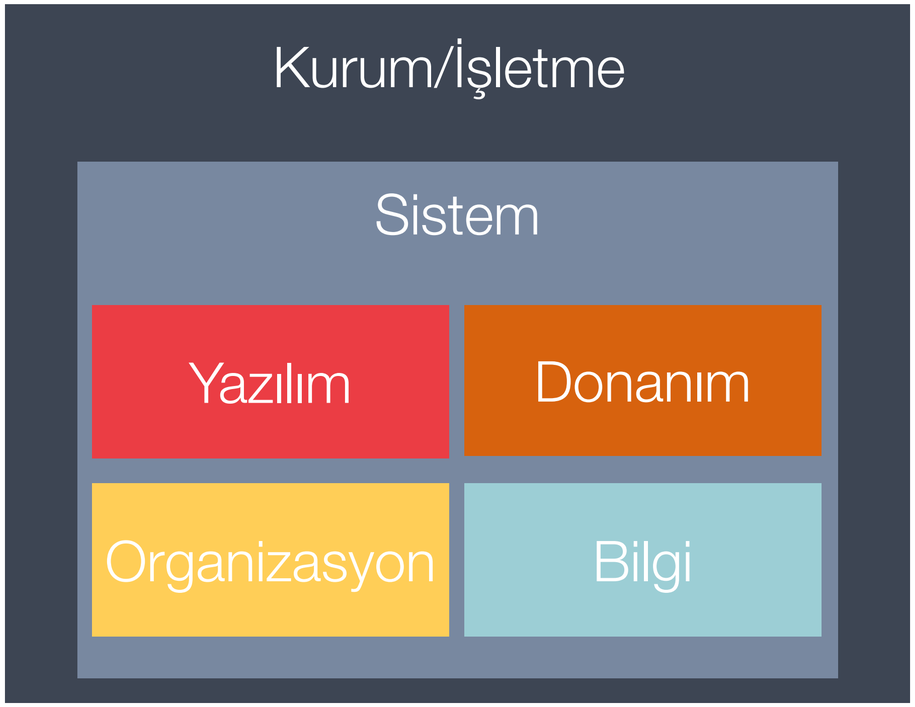
Mimari belgelenmemişse**, sürdürülebilirlik**, en iyi uygulamaların uyumlulaştırılması ve benzeri gibi nitelikleri ele alma açısından belirtilen gereksinimleri karşıladığını **kanıtlamak** zordur.

Belgelenmemiş, bugün var olan çoğunluk gibi görünen mimariler, tesadüfidir

YAZILIM MİMARİSİNİN KAPSAMI

Mimari Kapsam

Tüm bu mimariler farklı farklı kapsamlarda ve bakış açıları ile mimariyi ele alırlar.



Mimari Kapsam

Yazılım (Software) Mimarisi : Önceki Slaytlarda yazılım için bahsettiğimiz kapsamları ele alır.

Kurumsal (Enterprise) Mimari: işlemci, bellek, sabit disk, dış cihazlar , yazıcılar, ağ elemanları ve bunların birbirleri ile fiziksel bağlantılarını ele alır.

Sistem (System) Mimari: iş süreçleri , organizasyonel, roller ve sorumlulukları ve kuruluşun temel yetkinliklerini ele alır.

Organizasyon (Organization) Mimari,
Bilgi (Information) Mimarisi,
Donanım (Hardware) Mimarisi,
Uygulama (Application) Mimarisi,
Altyapı (Infrastructure) Mimarisi,

YAZILIM GELİŞTİRME

NEDEN FARKLI YAZILIM GELİŞTİRME MODELLERİ VAR ?

Neden?

Yazılım geliştirme her ne kadar kod parçası ve bunun bir uygulama olarak kullanılması gibi gözüksede aslında içerisinde **bundan fazlasını** barındırır.

Farklı farklı yazılım nedir ? Eninde sonunda hepsi **kod parçası** değil mi ? Bu konuyu iyi anlamanız/kavramanız gerekiyor ki Yazılım Geliştirme Modellerinin **nereden çıkmış olduğunu** ve **neden bir tane olmadığını** anlayabilelim.

Bu konuyu uzun yıllar çalıştığım **sektör içerisinden örnekler** vererek anlatmak istiyorum.

Komuta Kontrol Yazılım Uyuşum Projesi — (Savunma Sanayi)

2002 yılında Havelsan Şirketinde Komuta Kontrol Bilgi Sistemi Uyuşum Projesi çalışıyordum.

NATO ortak projesi olan bu proje, NATO unsurları ile bizim unsurlarımızın ortak harekat yapabilmesini sağlacak bir yazılımdı.

Yazılımların sahada komuta kontrol merkezlerinde çalışacağı, 15 yıldır NATO tarafından geliştirilen bir veritabanına entegrasyon yapmamız gereken bir projeydi.

Projenin Karakteristik Özellikleri: İletişim hızı, güvenlik, doğruluk ve sistemin çökmemesinin çok kritik olduğu bir proje geliştirilmesi.

Süresi: (Geliştirme safhası 3,4 sene iken, proje aktif olarak geliştirilmeye ve kullanılmaya devam ediyor.

Ufak Bir Oyun Projesi — (TRT1 için Oyun)

2003–2004 yılında hobi olarak oyun programlaması ile uğraşıyorum.

Arkadaşlarım İzmir’den aradı TRT1'de telefondan çocukların oynayabileceği basit bir oyun geliştirebilir miyiz şeklinde bir soruları oldu.

Basit PC üzerinde çalıştırılabilen bir uygulama geliştirmiştim.

Grafiker ve benim çok kısa zamanda müzik ve seslerini açık kaynaklardan temin ederek geliştirdiğimiz bir oyundu. Oyun basitti, kısıtlı kuralları vardı sadece PC Windows çalışıyordu. Çok kısa zamanda 2 kişi tarafından geliştirilmişti.

Müşterek EH Simülasyon Projesi — (Savunma Sanayi)

Milsoft ’daki ilk 3 yılımda Müşterek EH Simülasyon ortamı geliştirdik.

Bu Elektronik Harp ve Savunma konusunda çalışan askerlerin eğitimi için geliştirilmiş bir projeydi. Kırmızı ve Mavi odalarda Eğitmenleri eşliğinde arkerler gerçek şavaş ortamının simülasyonundaymış gibi yazılım üzerinden sistemleri/araçları kontrol ediyorlardı. Bir nevi Red Alert, Tiberian Sun, C&C oyunlarının daha gerçekçisi(hareket kabiliyeti ve yetenekleri açısından) olan bir oyun olarak düşünebilirsiniz.

Daha gerçekçi dememin sebebi altyapısında bulunan High Level Architecture(HLA) dağıtık simülasyon sistemine istediğinde gerçek bir Helikopter , füze veya gemi bilgilerinin aktarılabiliyor olmasıydı.

Burda **çok büyük ekipler 3 yıllık çalışmanın sonucunda bu uygulamayı** ortaya çıkardı. Elektronik Harp ve Savunma Domain bilmek, Yeni sistemleri/araçları sisteme tanıtmak ve bunların simülasyonunu yapıp sisteme eklemenin basit olması, Bir birine takılabilir çıkarılabilir sistemler geliştirebilmek Uçak Gemisi → Uçak, Uçak → Füze, Füze → IVR sistemi vb.

Link Sistemleri L11/L16/L22 — (Savunma Sanayi)

Milsoft sonraki 3 yılımda Link11, Link16 sistemleri sonrasında Link11 bir üst versiyonu olan Link22 Sistemi olan bir projede çalıştım.

Burada geliştirilen yazılımlar savaş gemilerinde, deniz altında, awacs vb komuta ve iletişim merkezlerinin yönettiği sistemlerde kullanılmakta.

Belki belli özellikler 1 veya 2 defa kullanılacak ama **kesin ve sorunsuz çalışmalı**.
Güvenlik çok önemli

Online Hizmet Merkezi — (Telco)

T2 firmasında Türk Telekom ile birlikte Online Hizmet Merkezleri geliştirdik.
Bu sistemlerin hem mobil hemde web uygulamalarını geliştirdik.

Bu uygulamalar altyapıda bulunan

- CRM(Customer Management System),
- Billing,
- Ordering,
- Arıza,
- Kampanya vb

Bir çok sistemin önünde bulunan Kurum/Kullanıcı'lara son arayüzü sağlayan uygulamalardı. Estetik ve işlevsel bir UI tasarımı olmalıydı. 7/24 ayakta olması. Diğer sistemlerin stabil bir şekilde çalışması ve bu ortak entegrasyon projesine servis sunması gerekiyordu. **Entegrasyon projesiydi**

Mobil Uygulamalar — (Gelir İdaresi, TOBB, EFinans)

Cybersoft firmasında Gelir İdaresi, TOBB, eFinans gibi büyük firmaların Mobil uygulamalarının geliştirilmesinde çalıştım.

Mobil uygulamalar yine arkaplanda bulunan bir çok servisin son kullanıcıya ulaştırılmasıydı.

Bir entegrasyon ve ekstradan **notification(bildirim)** kullanıcıları gerekli konulardan haberdar etmesi önemliydi. Tabi mobil projelerde devreye bu yazılımları AppStore, GooglePlayStore yayınlama izni veren Apple ve Google Politikaları ve süreçleri düzgün işletmekte önemliydi.

Karar Destek ve Raporlama Uygulaması — (Enerji Bakanlığı)

Cybersoft UNDP(Birleşmiş Milletler Kalkınma Programı) kapsamında tüm

Türkiye'nin Enerji Verisini işleyen ve bunu Karar Destek olarak ilgili üst kurumlara raporlayan yazılım projesinin analiz, tasarım ve geliştirme aşamalarında bulundum. Bu projedeki en önemli konu dinamik bir sistem tasarlamak ve bu dinamik sistem üzerinden dinamik raporları performanslı olarak alabilmekti.

Dinamiklik (Toplanan verilerin dinamik olarak değiştirilmesi ve bundan toplanan verilerin üzerinde Dinamik Sorgular AdHoc Query çalıştırabilme ve bunları Dinamik Chartlar üzerinden gösterebilmekti.)

Makine Öğrenmesi /NLP — (Tübitak Arge)

Cybersoft 1501 Tübitak kapsamında geliştirdiğimiz ARGE projesi yöneticisiydim.

Bu projede önemli olan Tübitak vadedilen projenin istenen süreçte ilerletilmesi, proje içerisinde ARGE yapılarak akademik makale yazılması ve ara raporların (teknik ve maliyet raporların) düzgün bir şekilde hazırlanıp sunulmasıydı.

Bu projede akademik bir hoca ile birlikte çalışarak **ARGE araştırmalar ve denemeler yapmak** en önemli konulardan bir tanesiydi.

iBeacon, Skorlama Motoru vb.. — (Şirket İçi Arge)

Cybersoft şirket içi ARGE kapsamında geliştirttiği iBeacon İşe Giriş/Çıkış uygulaması Bluetooth ve iBeacon teknolojini denerken,

Sigortacılıkta Risk Skorlama Motorunda Dinamik Kuralların çalıştırılarak Risk hesaplamasını performanslı yapılmasını sağlayan bir sistem,

Mobil'de Native — Web arasında geliştirilen Native-JS bride sayesinde Web ile Native birlikte çalışabilir birbiri ile mesajlaşabilir şekilde projeler

geliştirmenin amacı hepsinde **öncelikle bir Prototip geliştirmek** ve bu prototiplerin şirket içerisinde ekipler tarafından değerlendirilerek gerçek projelerin içerisinde kullanılmasıydı. Amaç teknik olarak problemli ama proje için önemli bir konuda **hızlı prototipler üretebilmek ve riskli kısımları önceden farkedip üstesinden gelip, gelemeyeceğini belirleme** çalışmasıydı.

Application Performance Management — (APM)

Thundra şirketi AWS üzerinde serverless ve container alanında uygulama geliştiren firmalara monitoring, güvenlik, debugging vb alanlarda yardımcı araçlar geliştirerek kullanıcıların daha verimli , daha sorunsuz uygulamalar geliştirmesini, canlıdaki oluşan veya oluşacak problemleri daha önceden farketmelerini sağlayan bir servistir.

Buradaki amaç bir ürünün **müşteri/pazar ihtiyaçlarına göre şekillenecek** şekilde en iyi şekilde kullanılabilecek hale getirilmesi. Bu da diğer projelerden farklı olarak ürünün üzerine daha çok odaklanılması, **ürünün çok daha kaliteli hale getirilmeye çalışıldığı** bir proje

Sidekick- Debugging without Stop

Foresight - Observability for CI Pipelines

Etkenler Nelerdir ? (Farklı Yazılım Geliştirme Modelleri)

Yukarıda çalıştığım farklı farklı projelerden bahsetmemin sebebi projelerin birbirinden ne kadar farklı olduğunu göstermek istememden kaynaklanıyor. Bu farklılıklar sizin projenizde kullanacağınız **Yazılım Geliştirme Modelini belirlemenizde etken** oluşturur.

- Projenin Büyüklüğü (Bütçesi, Uzunluğu, Adam Ay Maliyeti)
- Bu Bütçeyi Nasıl Alacağınız :
- Projeyi Finanse Eden Kişi/Kurum ve Kullanıcısı
- Projeyi Kullanacak Kişilerin Farklılığı, Kullanma Amaçları ve Sıklığı
- Projenin Farklı Disiplinleri ve Çevreyi İçeriyor Olması
- Projenin Çalıştırılacağı Ortam Koşulları

Projenin Büyüklüğü (Bütçesi, Uzunluğu, Adam Ay Maliyeti)

- Askeri Projeler çok büyük projeler iken,
- Enterprise(Telekom, TSE, Enerji Bakanlığı) ve Tübitak projeler orta ölçekli
- Mobil projeler. küçük ölçekli
- Prototip projeler tek kişinin olduğu projeler.

İşin Karşılığını (Parayı) Nasıl Alacağınız ?

- Sözleşme imzalamışsanız. Belirlenen fazlar üzerinden paranızı alırsınız. Askeri ve Birleşmiş Milletler Kalkınma Projelerinde genelde **Planlama, Analiz, Tasarım, Kodlama/Test, Entegrasyon, Kurulum ve Bakım** aşamalarının sonundaki çıktılar doğrultusunda paranızı alırsınız.

- Sözleşmenizi projenizi **belli modüllere** bölmüşseniz. Bu modüllerin gerçekleştirimi ve canlıya alındıkça paranızı alabilirsiniz.

- Tübitak kapsamında projenin ilgili adımlarını ve bunlara yaptığına dair harcamalarını **muhasebe ve teknik raporlarını tamamlayıp**, jüriye kabul ettirmen gerekiyor.

- SaaS projesinde ürününüz geliştirerek , kullanıldıkça **müşterilerinizden ay sonunda veya yıllık** para alırsınız. Spotify, Netflix modeli gibi.

Projesi Finanse Eden Kişi/Kurum ve Kullanıcısı

- Askeri projelerde projeyi finanse eden **SSM olurken** projenin kullanıcısı Askeriye'nin herhangi bir birimi olabilir.

- UNDP , Tübitak Projelerinde destek finansmanını bu **kurum ve kuruluşlar sağlarken** amacı ülkeye fayda sağlayacak yazılımların geliştirilmesi dolayısıyla kullanıcıları çok farklı olabilir

- Telekom, Gelir İdaresi, TOBB, TSE vb **kendi servisleri için finanse ettiği** yazılımlarda kendi müşterileri ve çalışanlarının kullanacağı yazılımlar olur.

- SaaS uygulamalarda ise **birebir finanse eden kullanıcının ürünü kullandığı** ürünlerdir.

Projesi Kullanan Kişilerin Farklılıkları, Kullanma Amaçları ve Sıklıkları

- Son kullanıcı , alan uzmanı, vb..
- Kullanıcı bir Asker ise sadece savaş/tatbikat sırasında kullanacağı ama kullanıldığı zaman hata çıkmaması gereken bir yazılım.
- Gelir İdaresinde Motorlu Taşıtlar vergisi için bir çok kullanıcının yılda 2 defa kullanması
- Spotify, Netflix, Word gibi her gün kullanılabilecek uygulamalar.
- Prototip gibi üretilip sonradan kullanılmadan atılabilecek uygulamalar.

Projenin Farklı Disiplinler ve Limitlerinin Olması

- Thundra dillere çok bağımlı. Dillere, kütüphanelere ve teknolojik gelişmelere yakından takip etmeli. Bir çok rakibi var, rakiplerinin ürettiklerini takip etmeli.

- Komuta Kontrol Sistemleri kesin doğruluklar ile çalışmalı

- GiB Mobil , Telekom Online Hizmet Merkezi gibi sistemler 7/24 çalışmalı ve diğer alt sistemler ile entegre bir şekilde çalışmalı.

Projenin Çalıştırılacağı Ortam Koşulları

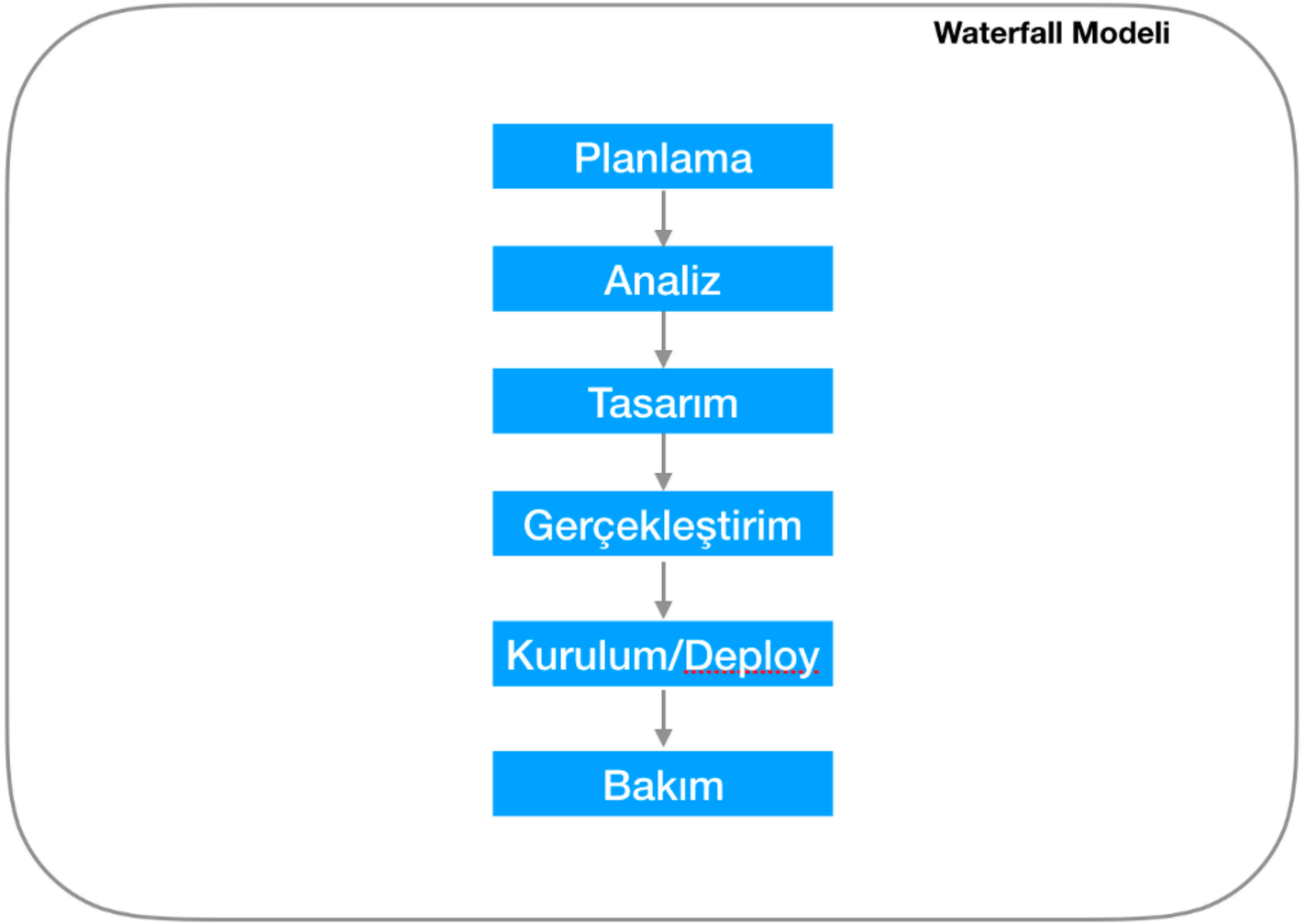
- Projenin Çalıştırılacağı Ortamlar (PC, Web, Mobile Cihaz, Gemi, Uçak vb.)
- Projenin Güvenliği, Ağ ve Network Hızı vb…

Ve bunun gibi burda belirtmediğim birçok farklılıktan dolayı birbirinden farklı **Yazım Geliştirme Modeli** bulunur.

Siz projenizin özelliklerini ve süreçlerinize göre aşağıdaki fazları hangi uzunlukta, hangi öncelikte, hangi tekrarla yapacağınıza karar verip hangi model size uygunsa o modeli projenizde kullanabilirsiniz.

YAZILIM GELİŞTİRME FAZLARI NELERDİR?

Waterfall (Şelale Modelinde) Yazılımın Fazları



Yazılım Geliştirme Fazları

Hangi yazılım geliştirme yaşam döngüsünü kullanırsak kullanalım bu temel fazların üzerinden **1 veya 1 den fazla bu fazların üzerinden geçecek** şekilde proje geliştirmeyi sürdürürüz. Aşağıda bu fazların ne anlama geldiğinden kısaca bahsedeceğim.

Yaşam döngüsü üzerindeki fazlar **Planlama, Analiz, Tasarım, Gerçekleştirme, Kullanım için Yükleme ve Bakım** olarak sınıflandırabiliriz.

Her faz kendi içerisinde **birçok farklı disiplin** içerir. Her disiplin üzerine derinlemesine akademik araştırmalar ve birçok yazılmış kitap bulunuyor. Bunu şundan dolayı belirtiyorum. Bura da her bir fazı kısaca anlatıyoruz ama her bir faz kendi içerisinde devasa deneyim barındırıyor.

Planlama Fazı

Projeye ait aşağıdaki konuların **planlama çalışmalarını** içerir.

- maliyeti,
- risk analizi,
- iş kırılımları,
- bu iş kırılımlarının hangi sırada,
- ne kadar sürede,
- hangi kalitede yapılacağı,
- bu işlerde hangi yeteneklerde,
- kimlerin çalışacağı,
- müşteri ile iletişimi, eğitimi,
- uygulamanın kullanılması ve
- çıkan hataların giderilmesi ,
- yeni özelliklerin nasıl ekleneceği vb…

Örnek Planlama

İş Paketleri ve Alt Kırılımlarının Belirlenmesi

Projenin **ana iş paketlerini ve alt kırılımlarının belirlenmesi** gerçekleştirilir. Her birisine kod verilirki ana planlama dokumanında bu kodlar üzerinden çalışılabilsin.

- A Proje Yönetimi
  A01 Scrum Yaşam Döngüsünün Uygulanması
  A02 Gereksinim dokümanının Hazırlanması
  A03 Test dokümanının Hazırlanması

- B Arayüz Modülünün Gerçekleştirimi
  B01 Kullanıcı Giriş Ekranlarının Gerçekleştirimi
  B02 Şifre Unuttum Ekranlarının Gerçekleştirimi

Ekibinizde Kaç Kişi Olacak, Rolleri Ne Olacak ?

Ekibinizin **hangi yetenekteki kaç kişiden oluşacağını** tahminen bilmeniz gerekiyor. Proje bütçesi, bitirmeniz gereken süreye, iş kalemlerinin büyüklüğü ve iş konusundaki deneyiminiz tüm bunlar **size ekibinizi nasıl kurmanız gerektiğini** anlatacaktır.

Onur D. → Proje Yöneticisi
Ahmet C →Yazılım Mühendisi
Mehmet A → Test Uzmanı

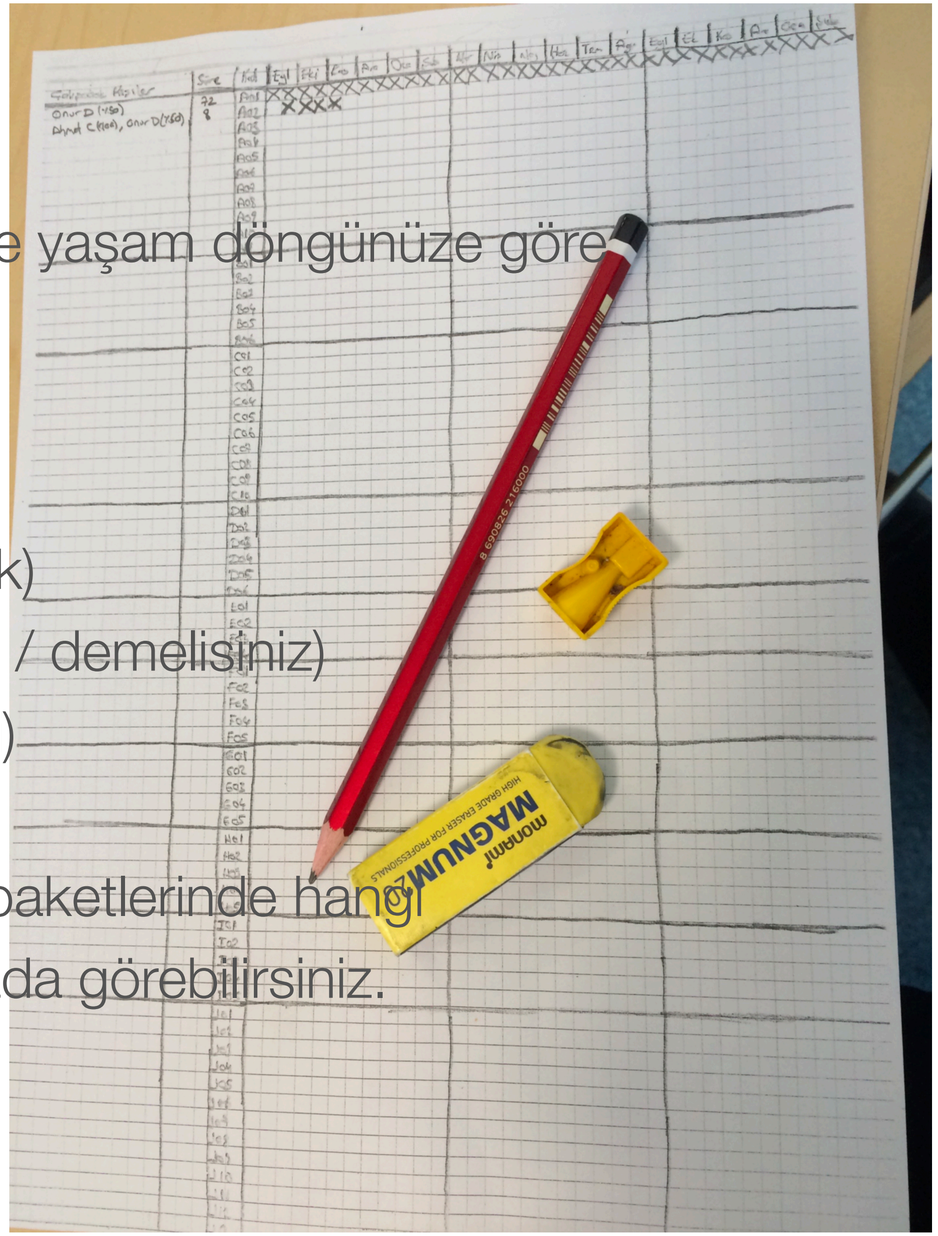
Proje Planlama Şablonu

Proje planlama şablonunu sizin ihtiyaçlarınız ve geliştirme yaşam döngünüze göre hazırlayabilirsiniz.

Şablonumuz şu başlıklardan oluşuyor.

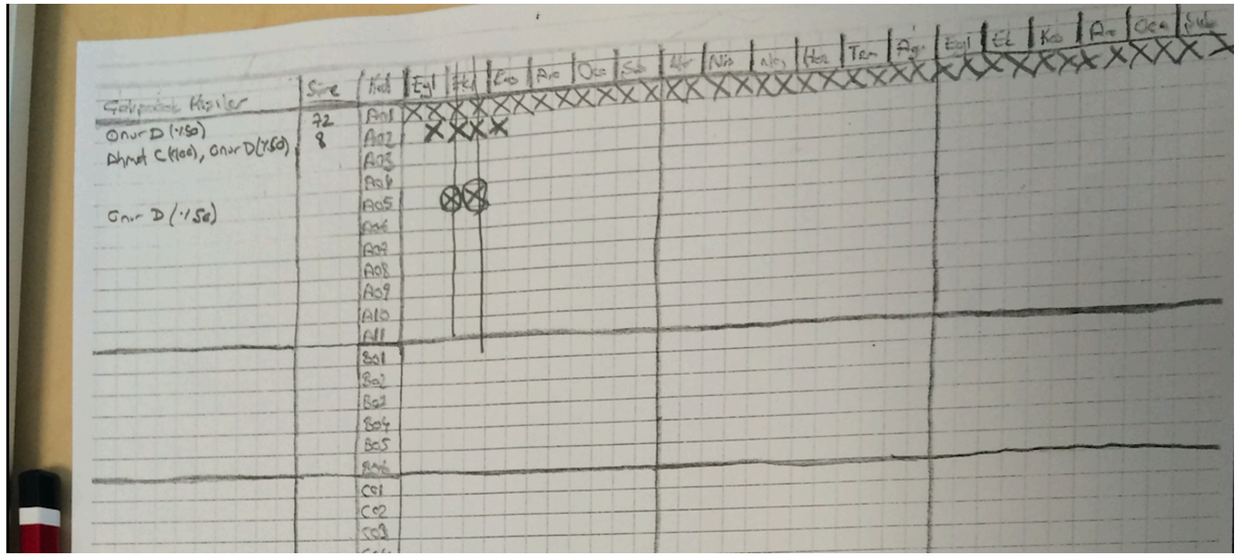
- Kod (İş Paketindeki iş kaleminin kodu, üstte vermiştik)
- Aylar (Her X işareti 2 haftayı işaret ediyor. 1 hafta için / demelisiniz)
- Süre (Hafta cinsinden iş kaleminin ne kadar süreceği)

İşte hepsi bu şablonu kalem ve silgi ile kişilerin hangi iş paketlerinde hangi zamanlarda çalışacağını işaretleyin. Hepsini tek bir sayfada görebilirsiniz.



Projenizdeki Çalışma Zamanları Çakışıyor mu diye kontrol edin

Bunun için yukarıdan aşağıya doğru çizgiler çizin ve kişiler için harcanan efor %100 geçiyor ise bu adamı o hafta için fazla atamışsınız demektir. Silgi ile düzeltin



Projenizdeki MS Project Aktarın

Çoğu yer, yönetici proje yönetim planını MS Project ortamında ister. Bunun için yapmanız gereken yukarıdaki bilgileri tool’a araca girmeniz.

Tasarım Fazı

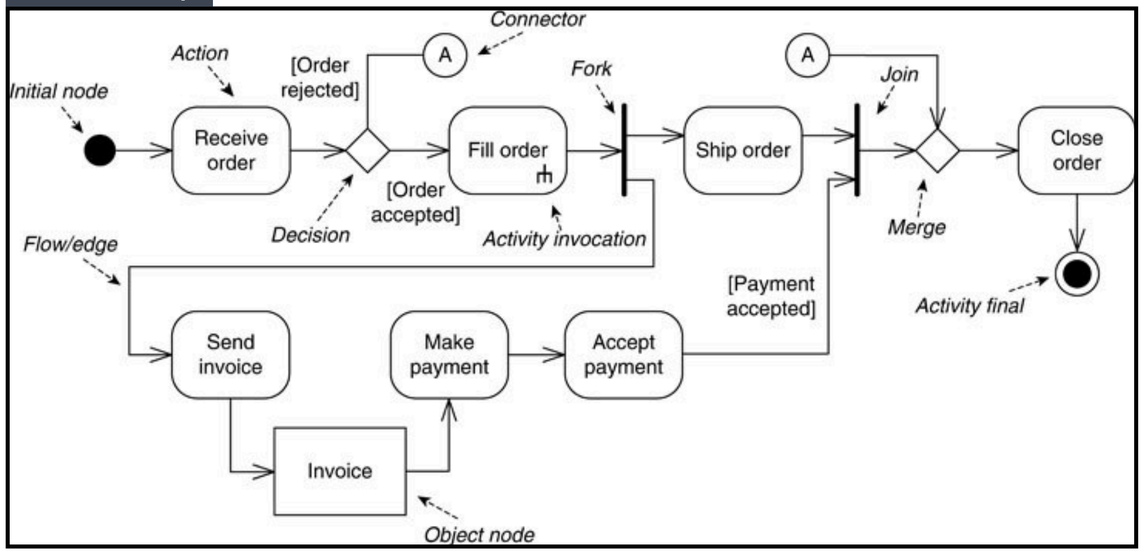
Ne yapacağınızı anladıktan sonra **nasıl yapacağınız** konusunda tasarım yapmanız gerekir. Bu aşamada **analizleri görselleştirmekten** biraz bahsetmek istiyorum.

Yazılım Mühendisliğinde Görselleştirme için UML (Unified Modeling Language) kullanarak;

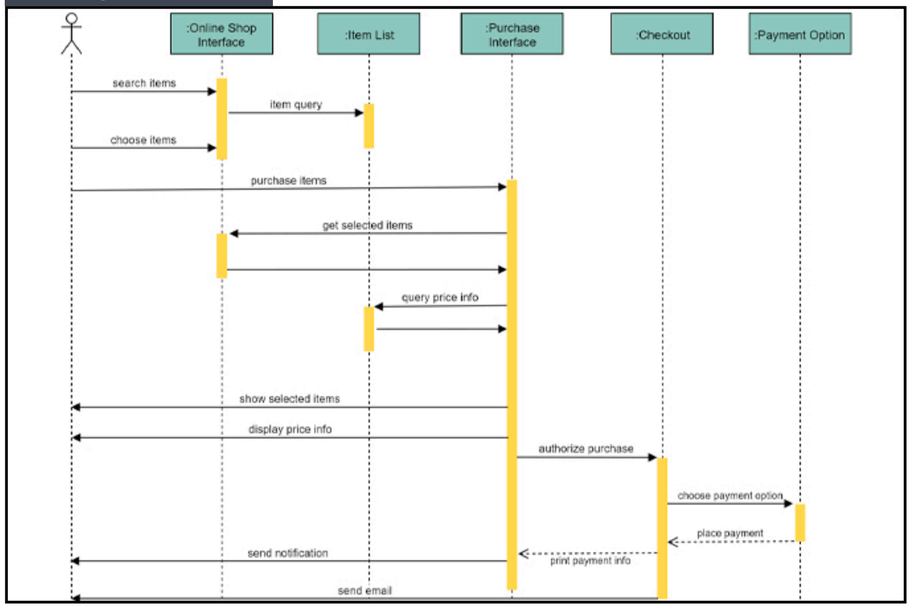
Behavioral Diagram ile Davranışı Modellemesi Yapılır.

- Activity,
- State Machine,
- Use Case,
- Timing, Sequence,
- Communication,
- Interaction Overview Diagrams

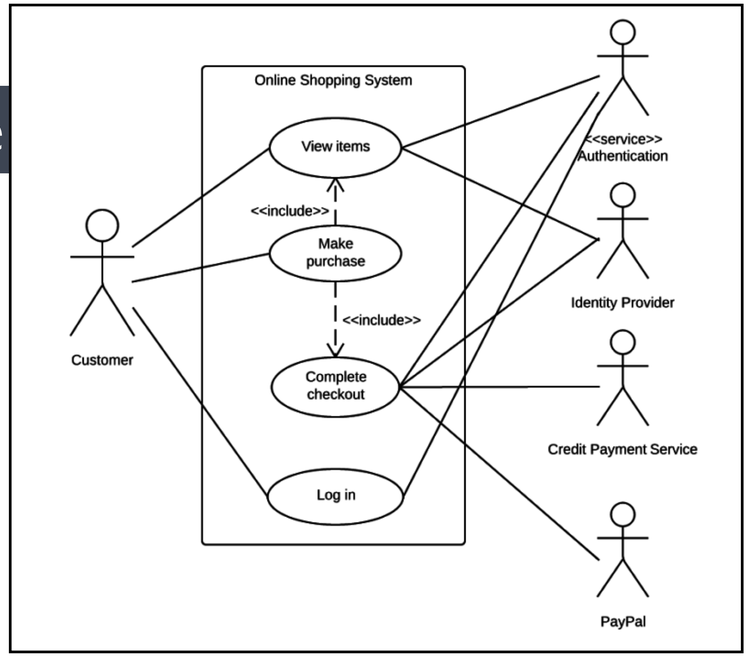
Activity



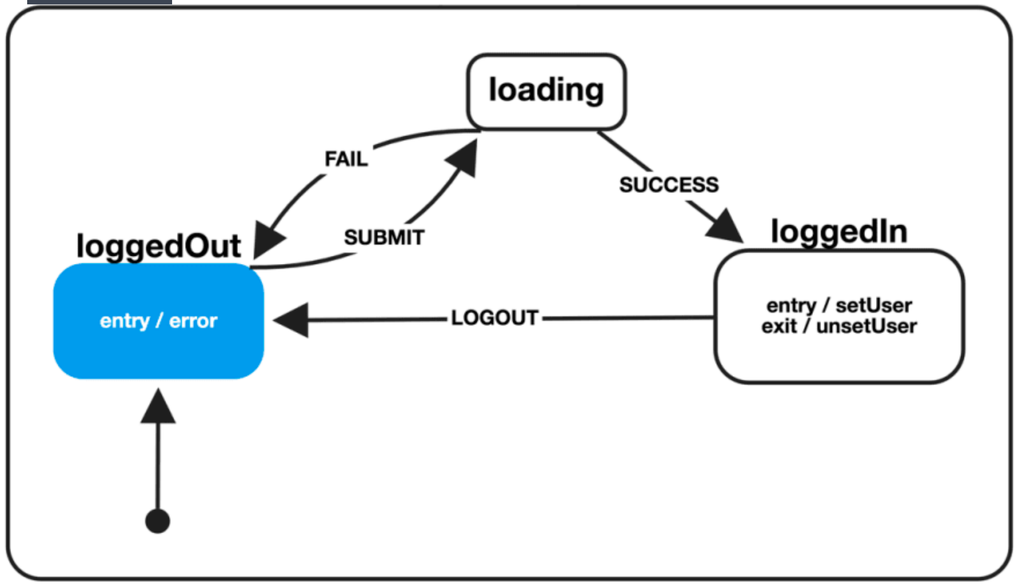
Sequence



UseCase



State



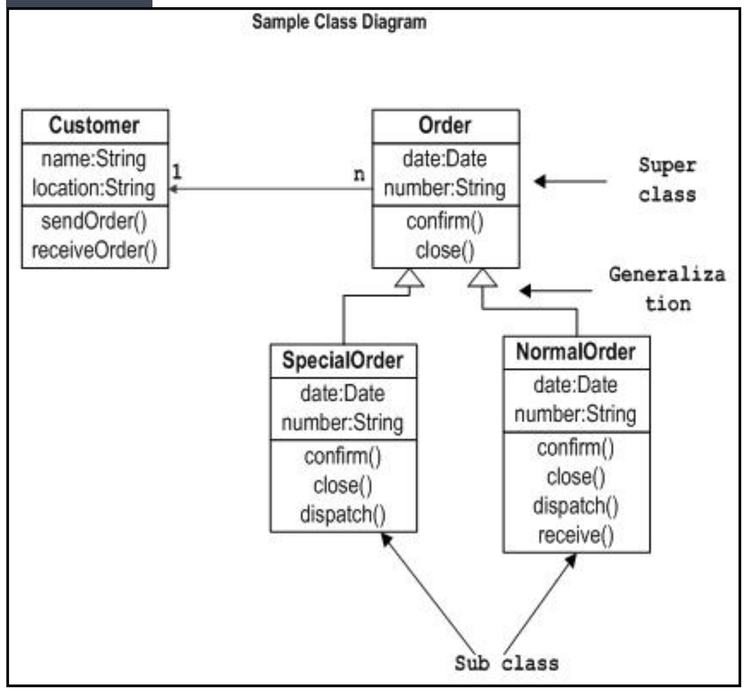
Tasarım Fazı

Yazılım Mühendisliğinde Görselleştirme için UML (Unified Modeling Language) kullanarak;

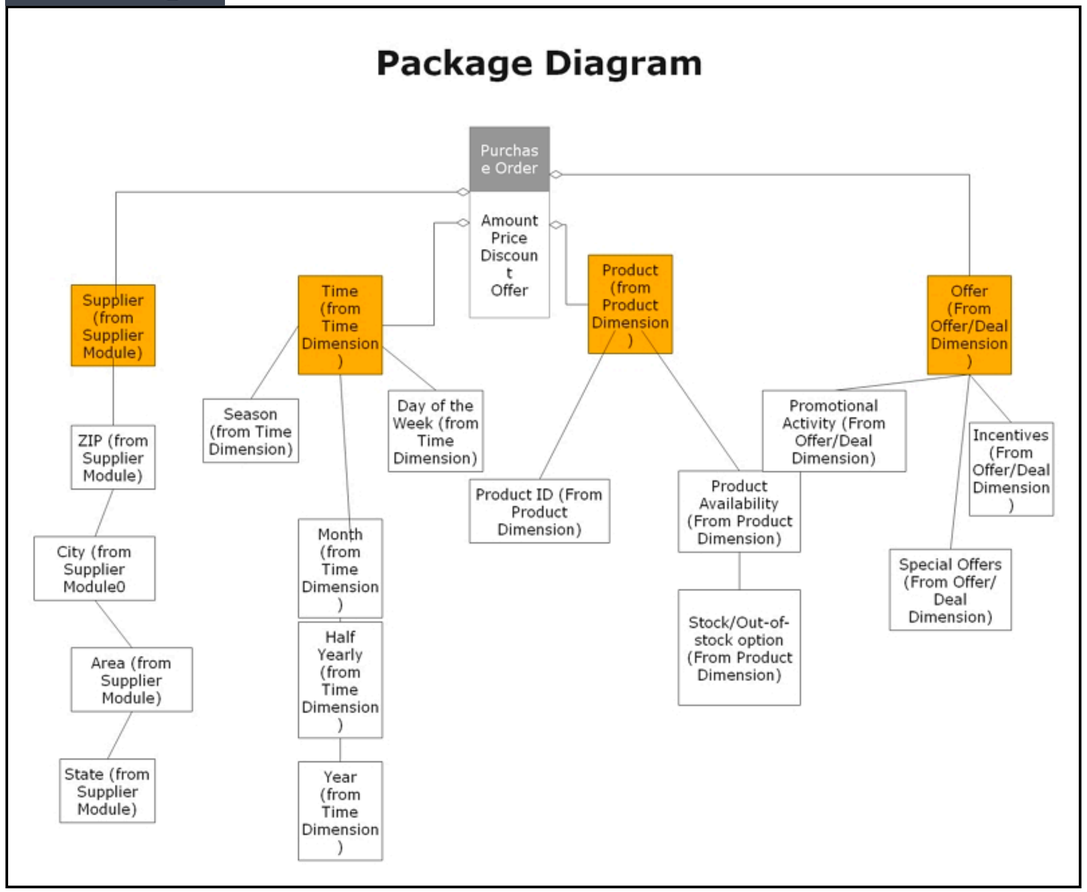
Structural Diagram ile Yapısal Modellemesi Yapılır ?

- Class,
- Composite Structure,
- Component,
- Deployment,
- Object,
- Package,
- Profile Diagrams

Class



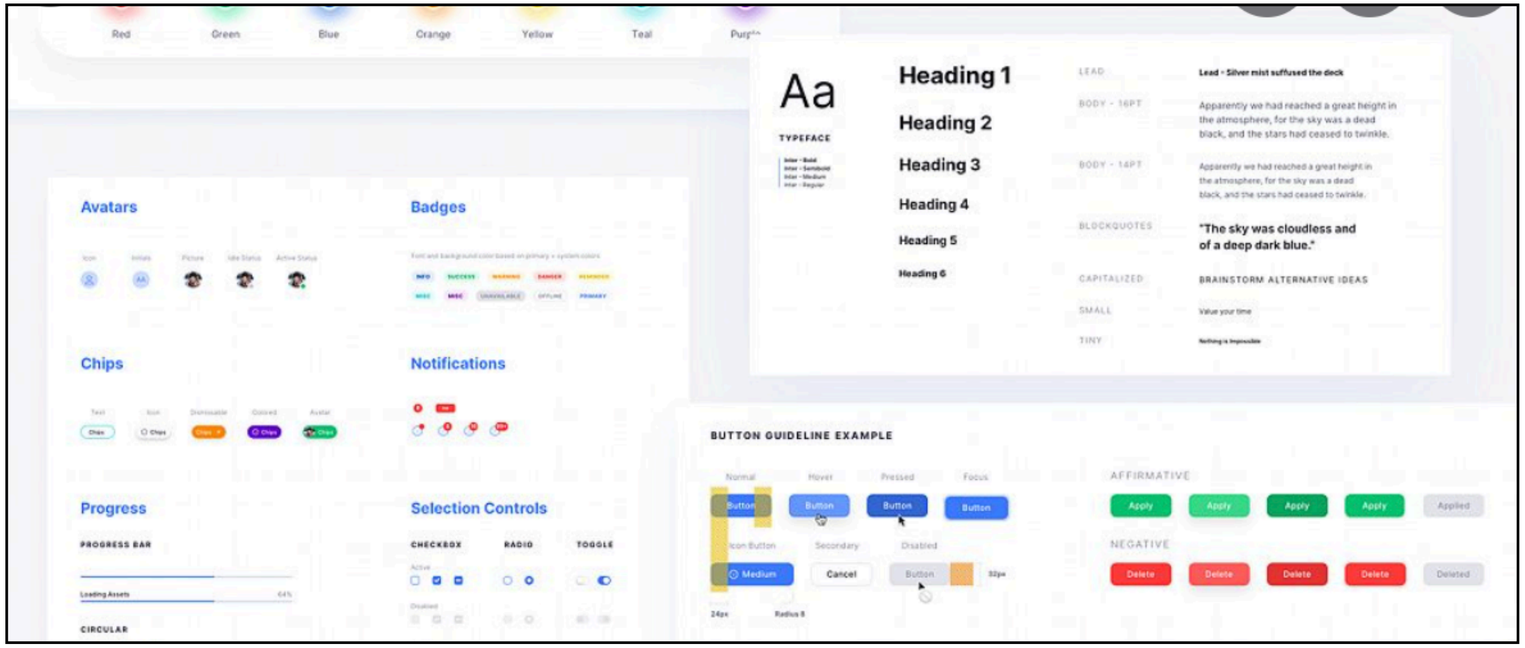
Package



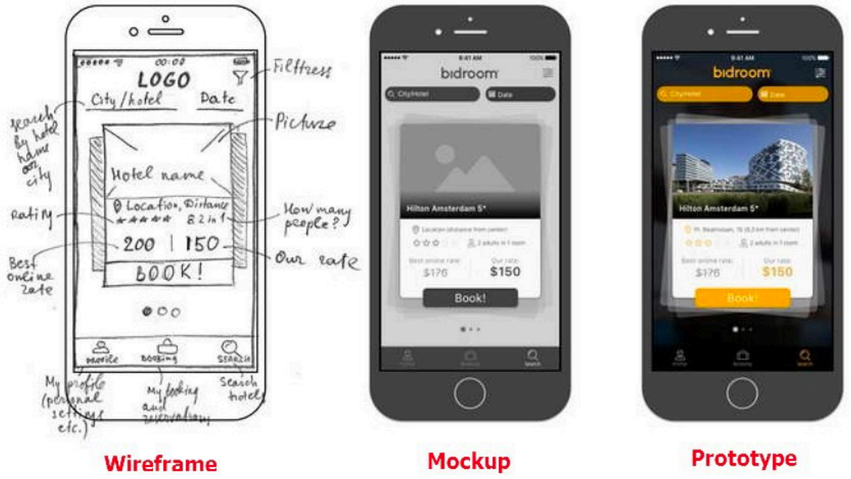
Tasarım Fazı

Ürün UI Görselleştirme ve Kullanabilirliğin Modellenmesi açısından;

- Design System , Pattern Library, Style Guiding
- Sketch, Wireframe, Mockup, Prototype



Tasarım Fazı



Sketch → Wireframe → Mockup → Prototype

Sketch: Projenin en başlarında fikrinizi hızlı bir şekilde anlatmanızı sağlayan karalamalar, çizimler. (MindMap ’te kullanabilirsiniz)

Wireframe: Projeniz ilerlemiş fonksiyonel akışlar belirlenmiş bu yönde projenizi anlatmanızı ve tasarım seviyesine geçmenizi sağlar. Ekranlar yinede değişiklik gösterdiği için bu aşamada tasarım ile uğraşmamak daha anlamlıdır.

Mockup: Projenizde yer alan sayfaların görsel tasarımına yönelik , renk paletleri , bileşen görsellikleri üzerine yapılan çalışmadır.

Prototype: Bileşenler ve sayfalar ile kullanıcının etkileşimi test edebilmek için yapılan çalışmadır.

Gerçekleştirme Fazı

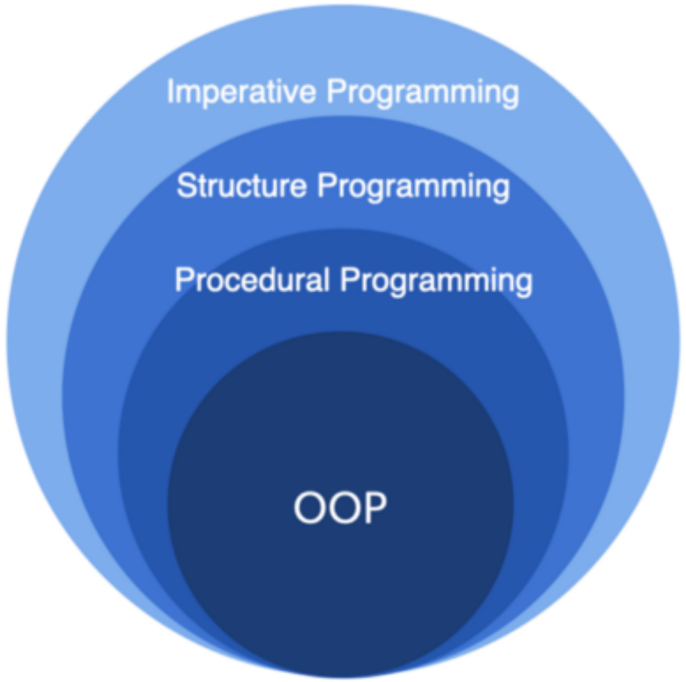
Ürünün **kodlanması**, **testlerinin yazılması**. Bu aşama oldukça farklı yöntemler ve disiplinler içerir.

- Procedural, Structural, Imperative, Declarative
- Functional Programming, Object Oriented Programming
- SOLID, KISS, DRY, YAGNI
- Design Patterns, Architectural Patterns, Integration Patterns
- Unit, Integration, Regression, Smoke, Beta, System, Stress, Performance Testing

Gerçekleştirme Fazı

Programlama Paradigmaları

- **Imperative Programlama**
  - Procedural Programlama
  - Object Oriented Programlama
  - Paralel İşleme Yaklaşımları
- **Declarative Programlama**
  - Mantıksal Programlama
  - Fonksiyonel Programlama
  - DB İşleme Yaklaşımları



Deployment Fazı

Bu faz projeden projeye çok farklılık içeren bir durum.

Askeri uygulamada bir Gemiye, Uçağa deploy ediyorsanız bu aracında görev zamanına uygun bir zamanda deploy etmeniz

Mobil uygulamayı AppStore ve GooglePlay Store atma sürecinden geçmeniz gerekiyor bu ilk uygulama atma sırasında bazen 1 haftayı bulabiliyor ama güncelleme daha kısa sürüyor genelde.

Web uygulamalarını canlı ortama deploy sunucular ekibin kontrolünde olduğu CI (Continuous integration) ve CD (Continuous Delivery). canlı ortama hızlı bir şekilde gönderilebilir. Ama farklı şekilde deployment yöntemleri geliştirilmiştir. (Bing Bang, Rolling, Blue-Green, RedBlack, A/B Canary Deployment)

Bakım Fazı

Ürün canlıya alındıktan sonra

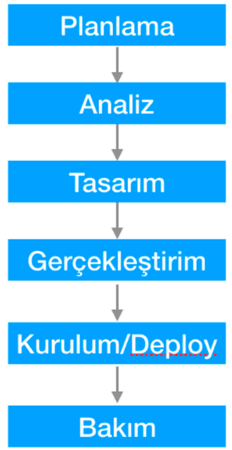
- çıkan hatalar ,
- müşterinin yeni istekleri,
- regülasyon dan dolayı değişiklikler,
- entegrasyon api değişiklikler,
- kullanılan kütüphanelerin yeni versiyonları ,
- güvenlik açıklarının kapatılması
- vb nedenlerle

canlıdaki ürün üzerinde geliştirmeler yapılıp, canlıya bu geliştirilmelerin gönderilmesi çalışmasıdır.

YAZILIM GELİŞTİRME MODELLERİ

Yazılım Geliştirme Modelleri Nasıl Ortaya Çıkıyor ?

Bu fazların sırası, **tekrar edilme sayısı** , **uzunlukları** ve **ne kadar üstüne düşüldüklerine** göre bu modeller değişir.



Daha önceden de bahsettiğim gibi
Askeri yazılım,
Enterprise yazılım,
Mobil yazılım,
SaaS yazılım,

geliştirme arasında oluşacak farklardan dolayı seçeceğimiz modellerde farklı olacaktır.

Yazılım Geliştirme Modellerinden Örnekler

Code & Fix

Waterfall

Waterfall Iterative

V Model

Iterative

Incremental

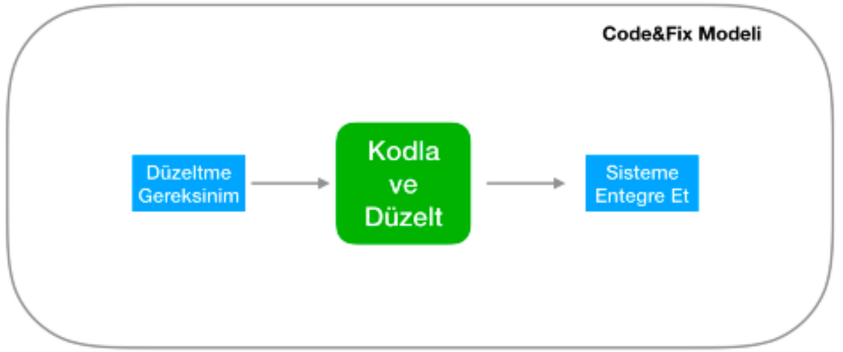
Evolutionary

Spiral

Rational Unified Process

Agile (XP, Kanban, Scrum)

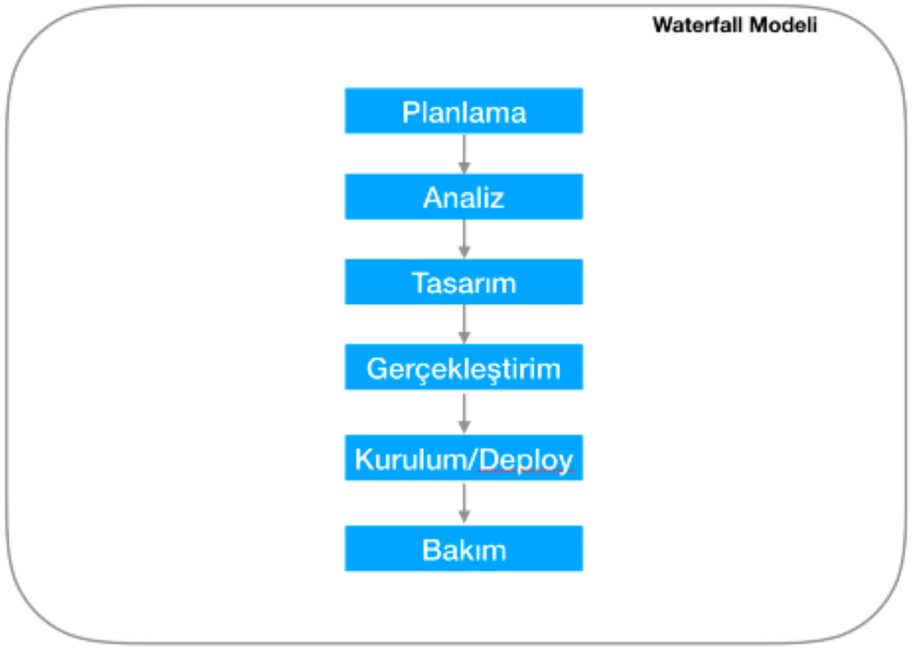
Code & Fix Modeli



Genelde bir konuda araştırma yapıyorsan, veya bir konuyu yeni öğreniyorsan veya belli bir problemi çözmeye çalışıyorsan kullanacağın en basit yöntem, kodla ve düzelt modelini kullanmak olacaktır.

En basit ürün geliştirme**(Cowboy Coding)** olarak adlandırılan bu modelde en kısa sürede prototipleyerek sonuca gitme hedeflenir. Planlama, Analiz yapma vs kısımlar ile çok vakit kaybetmeden hemen ana problem üzerine odaklanılarak sonuca gitme hedeflenir.

Waterfall Modeli



Analiz, Tasarım, Gerçekleştirim, Kodlama, Test, Entegrasyon … adımların **sıra ile işletilerek bitirilmesi** sürecidir.

Geri dönülmemesi için her fazın hakkını vererek doğru bir şekilde yapılması gerekir. Eski askeri yazılımlar ve projelerde bu model oldukça sık kullanılmıştır.

Eski ile Yeni Arasındaki Temel Fark Deneme Yanılma İmkanı

Ama sonradan bu modelin **işletiminin çok da basit olmadığı**, analizin süreç içerisinde değişebildiği, kodlama sırasında tasarımın değiştiği farkedilmiş bunun yerine daha uygulanabilir modeller aranmaya başlanmıştır.

Bunun yöntemide bu safhalara **yeri geldikçe geri dönebildikleri fazların tekrar tekrar üzerinden geçebildikleri sistemler** oluşturabilmekti.

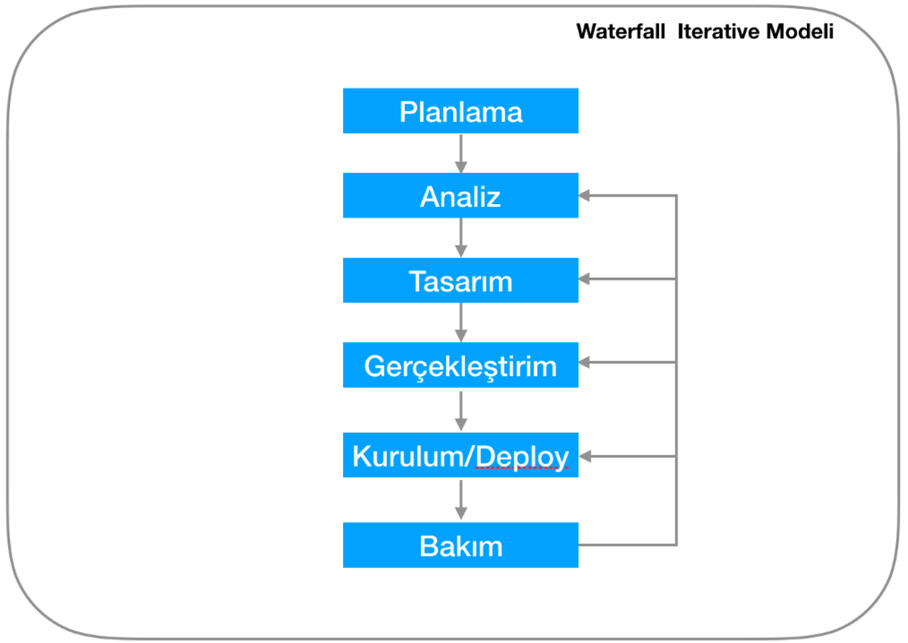
Eski ile Yeni Arasındaki Temel Fark Deneme Yanılma İmkanı

Tabi bundaki bir diğer etkende eskiden **programların yazılma, işletilme maliyetleri** çok yüksekti.

Yani çalıştığı zaman sorunsuz çalışacak sistemler üretmek çok kritikti. Bunun için baştan analizlerin çok çok iyi yapılması gerekiyorki Uzay/Uydu aracında kullanılacak program, Askeri Cihaz/Füze/Teçhizat kullanılacak program vb için böyle tekrar safhalı yazılım geliştirme mümkün değildi.

İlerleyen süreçte, teknolojiyle yazılımın hayatımızın her alanında kullanılmasıyla, değişen ihtiyaçlar ve koşullar sayesinde farklı farklı yazılım geliştirme modelleride ortaya çıkmıştır.

Waterfall İterative Modeli

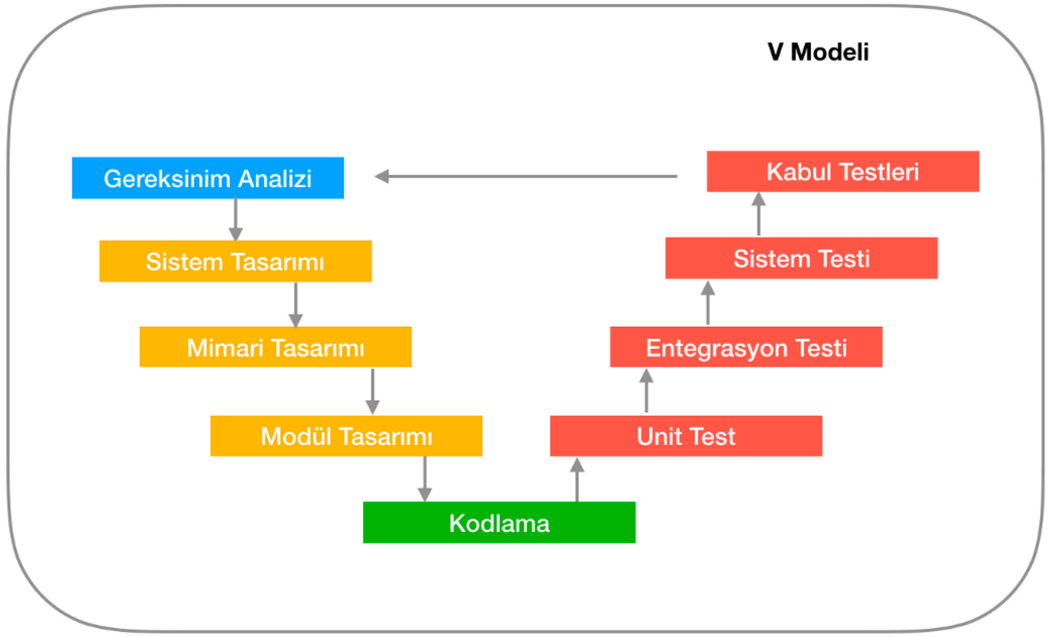


Waterfall Modelinde tüm fazlar tamamlandıktan sonra sistemin düzgün çalışmadığını farkettiğimiz zaman **sorunun nerden kaynaklandığını anlayıp o faza dönüp sonraki fazların ardı sıra işletildiği** modeldir.

Örneğin kod geliştirme aşamasında bir hata yapıldıysa bu faza dönüp, bu fazdaki hata düzeltilip diğer fazlar işletilir.

En kötü durum Analiz'in yanlış yapılması olacaktır.

V Modeli



Waterfall Modeline **Verification(Doğrulama) ve Validation(Onaylama)** mekanizmasının eklenmiş halidir.

V Model, analiz sonucunda sistemi tepeden aşağıya doğru parçaladığımız ve tasarladığımız sonrasında da gerçekleştirme sırasında sistemi toparladığımız bir yapı olduğunu farketmiş ve bu toparlama sırasının **her aşamanın karşısına onun ile ilgili doğrulama ve onaylama mekanizmasını** koymuştur.

Waterfall Modeline Verification & Validation Eklenmesi

Unit Test geçen Kodlarınız başarılı çalışıyor demektir.

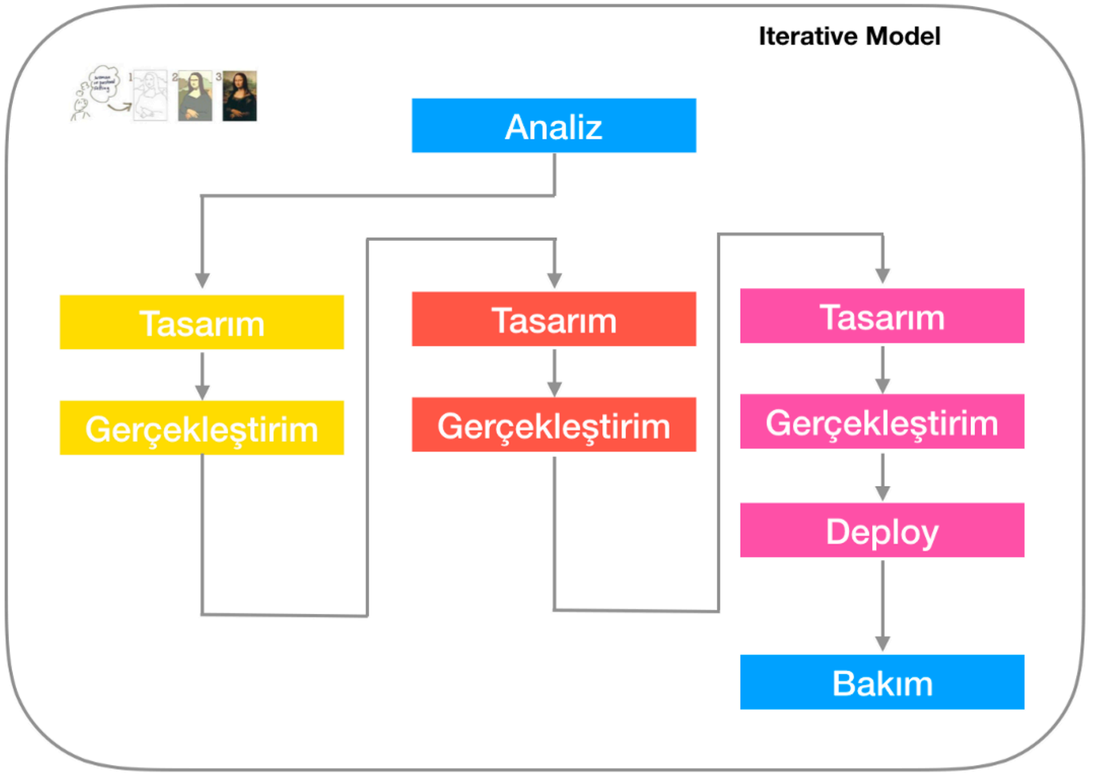
Entegrasyon Testi başarılıysa Modül ve Mimari Tasarımınız doğrudur.

Sistem Testlerinden geçiyorsanız Sistem tasarımınız doğrudur.

En son olarak **müşteri kabul testlerinden** geçiyorsanız Gereksinim analizini doğru yaptınız anlamına gelir.

Tests => Unit, Integration, Regression, Smoke, Beta, System, Stress, Performance Testing

Iterative Model



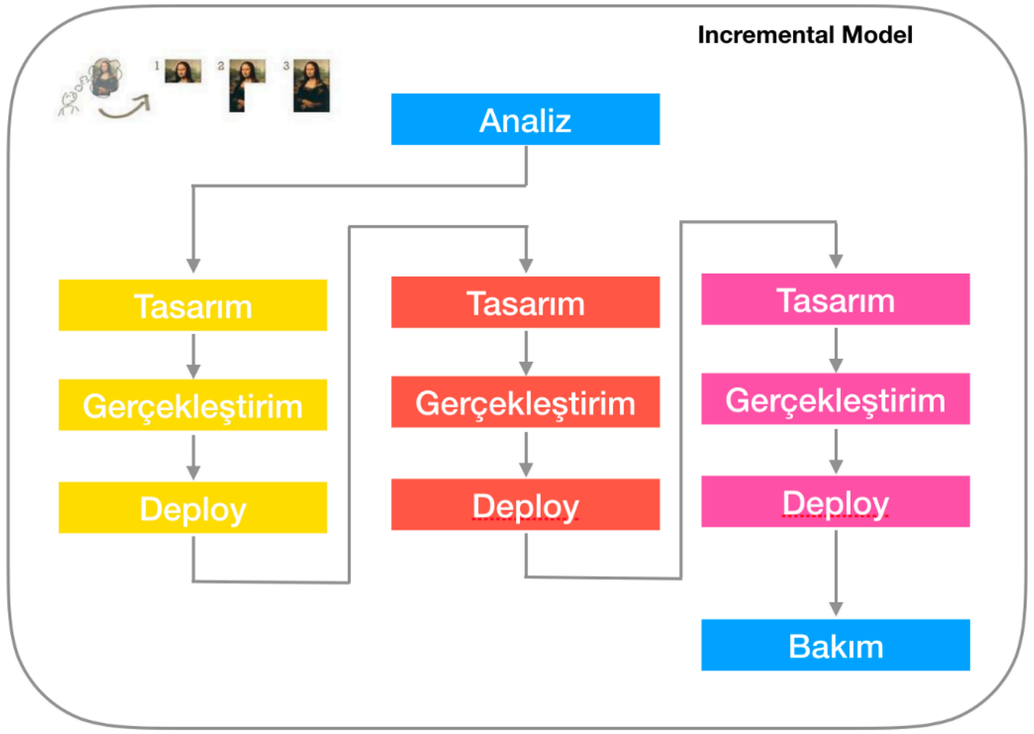
Projenin öncesinde **iş analistleri ve ürün yöneticileri** gerekli analizleri, toplantıları vb yaparak gereksinimleri çıkarır. Bu tamamlandıktan sonra Proje yöneticisi tarafından iş **belli önceliklere göre birden fazla cycle/döngü** halinde geliştirilir, en son tüm gereksinimlerin gerçekleştirilmesi tamamlandıktan **sonra ürün müşterinin önüne** çıkarılır.

İterative Model de İş Önceliklerinin Belirlenmesi

Bundan dolayı her döngü sonrasında çalışabilir bir ürün çıktısı olmadığı için bu döngülerde önce aşağıdaki kısımlar geliştirilebilir

- **Riskli bölgeler** geliştirilebilir
- **Altyapı çalışmaları** yapılıp (güvenlik, yetklilendirme, doğrulama, ui altyapısı, backend mimarisi, frontend-backend iletişimi ,CI/CD, testing)
- Tüm kodlar tarafından **ortak kullanılabilecek kütüphanelerin** geliştirimi
- Daha sonrada bunlar üzerinde **müşterinin istediği gereksinimler** doğrultusunda uygulama yetenekleri eklenebilir.

Incremental Model



(+)
Deploy Edilmesi

Iterative Modelde ki gibi iş Analistleri ve ürün Yöneticisi daha önceden gereksinimleri hazırlar ve yine belli döngüler ile projeyi geliştirmeniz gerekir ama burda bir fark bu döngülerin **sonunda çalışan çıktılarının olması** ve bunun **müşteri tarafından kullanılabilmesi** gerekir.

Proje Yöneticinin İşi Çalışabilen Parçalara Bölmesi Gerekir

Proje yöneticisine **işi çalışabilen parçalara/modüllere** ayırabilme zorunluluğu getirir. Bu açıdan Bu gereksinimi karşılayan projeyi **incremental model** ile geliştirebilirsiniz.

Bu sizin **tüm sistemin tasarımını yapmanız gerekmeden** kodlamaya, teste ve canlı ortama kurmaya götürecek ve müşteri ile daha kısa zaman içerisi iletişim kurup **geri dönüşler** almanızı sağlayacaktır.

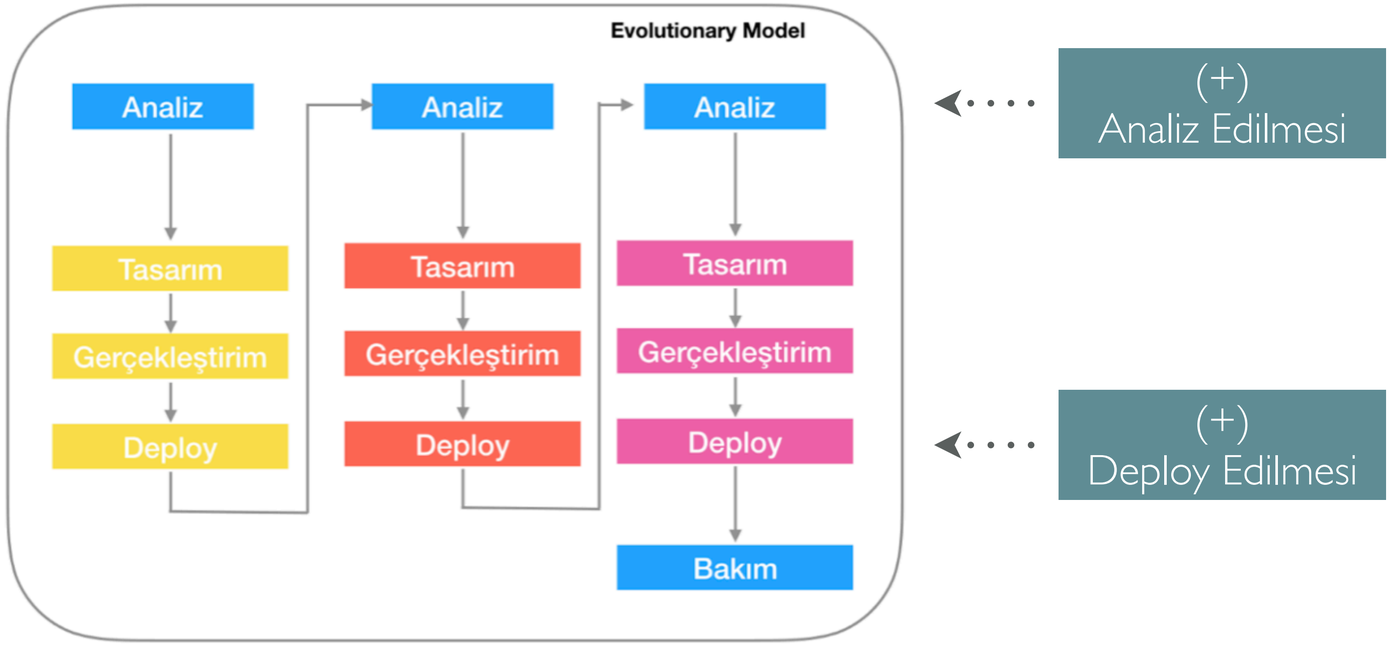
Incremental Modelin Başarısı

Bu modelin başarısı iterasyonlarda bölünen parçaların birbirinden **olabildiğince ayrı/soyut olması** ve birbirleri ile **entegrasyon ihtiyaçlarının az olması** sayesinde olur.

Başarızlığı Durumunda

Aksi durum yani **parçaların birbiri ile çok ilişkili olması** sonraki iterasyonlardaki tasarımların bir önceki iterasyondaki tasarımı etkilemesi durumunda bu model waterfall modelinden **daha maliyetli** bir hale gelmiş olur.

Evalutionary Model



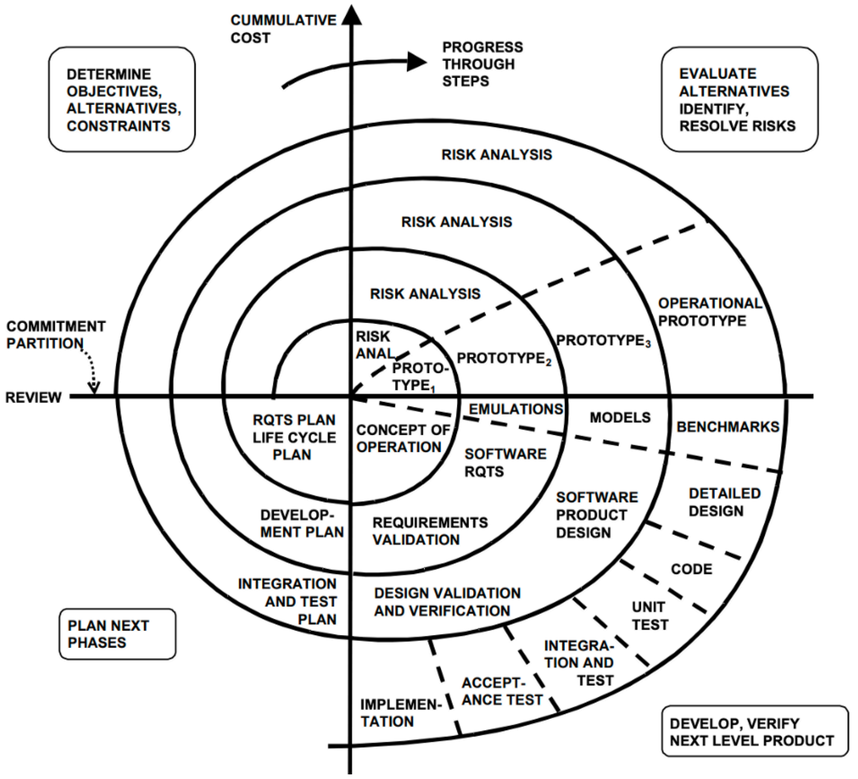
Evolutionary model analiz aşamasının **hazır halde geliştirici ekibe gelmesi** bazı durumlarda ürün kalitesini olumsuz şekilde etkiler. Analizi bu süreçlerin bir parçası olarak geliştirme ekibi ile yapılması,

- gereksinimlerin geliştirici ekip tarafından daha iyi anlaşılması
- geliştirmenin çok zor olacağı gereksinimler yerine daha ayağı yere basan gereksinimlerin ortaya çıkmasını sağlar. (Burda **amaç** gereksinimi yapmaktan kaçmak değil. Ürün sahibi, geliştirici ve iş analistlerinin **ortak bir şekilde analizde yer almasıdır.**)

Spiral Model

Büyük projelerde planlama ve gereksinim analizini düzgün bir şekilde yapabilmeniz çok zordur çünkü karşınıza **bilmediğiniz bir çok risk** çıkabilir.

İşte spiral model **bu riskleri baz alarak** bir model geliştirmiştir. Amacı riski seviye seviye azaltarak projenin başarılı bir şekilde tamamlanmasını sağlamaktır.



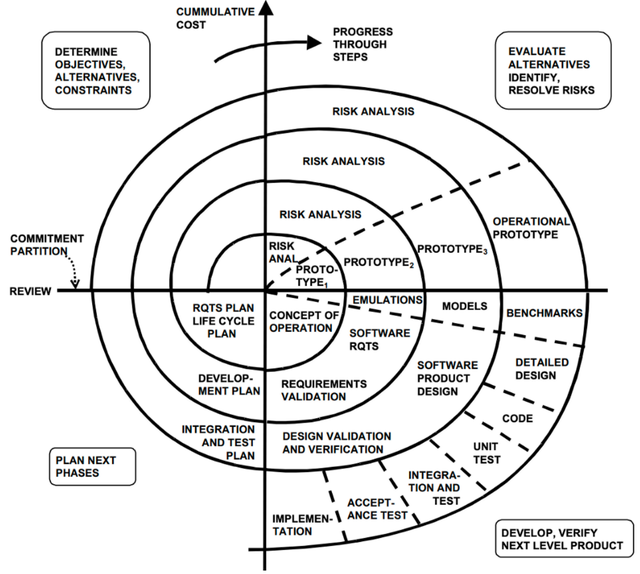
Neden Risk Analizine İhtiyacımız Olsun ki ?

Risk Analizi ve Yönetimi önceden bahsettiğim Waterfall modelleri içerisinde bu kadar detaylı ele alınıp incelenmemişti. Ama diyelim ki;

- projenin spesifikasyonları çok belli değil.
- gereksinimler tam olarak ortaya çıkarılamamış
- entegre olunacak sistemler yeni geliştiriliyor veya belirsiz bir sürü konu var
- vb..

Risk Analizi Fazları

Spiral Modelde aynı döngü **risklerin azaltılarak** projenin geliştirimesi hedeflenir. Buradaki fazlar;



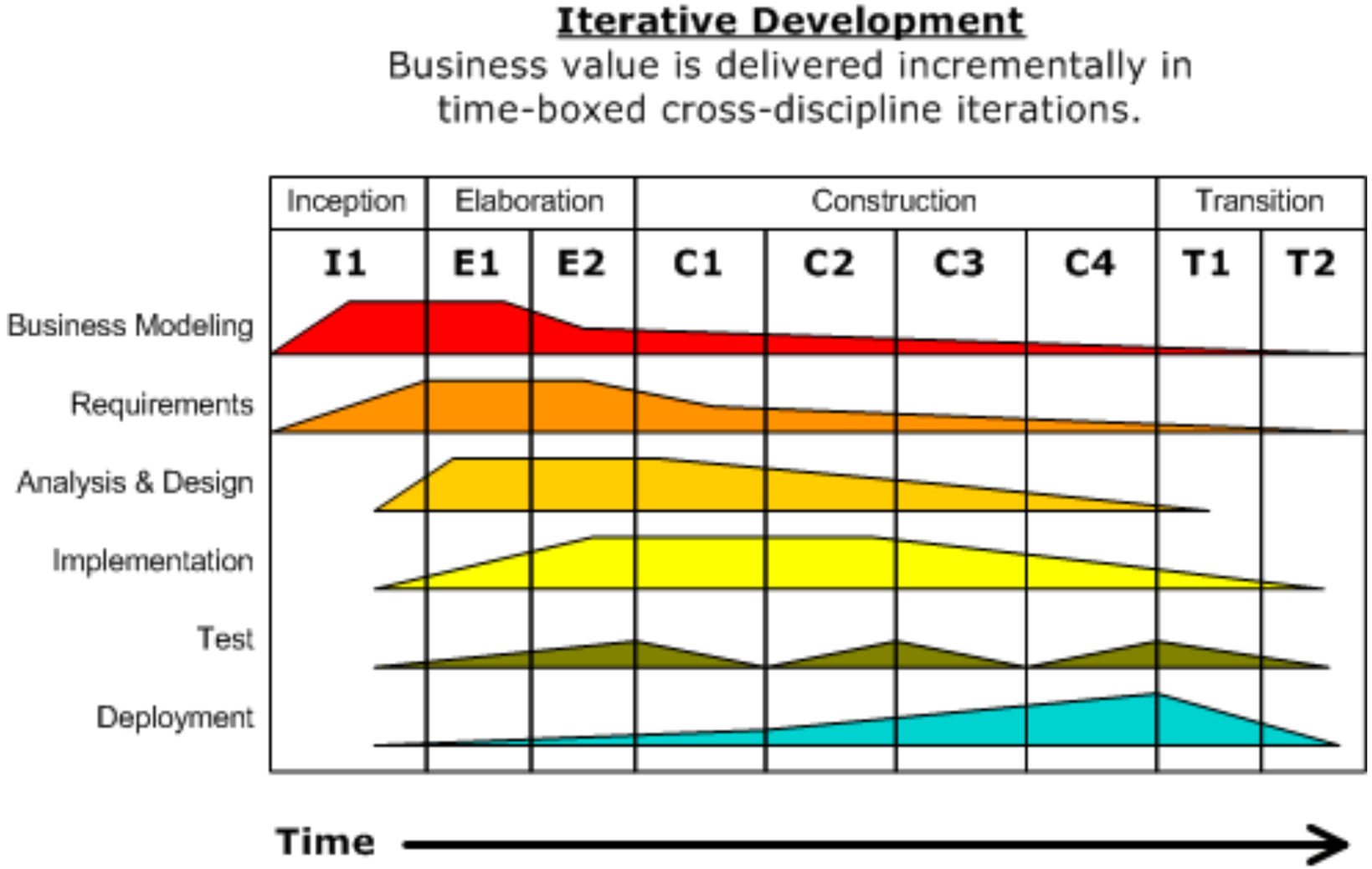
Hedeflerin Belirlenmesi ve Alternatif Çözüm Yollarının Bulunması: Bu aşamada müşteriden elde edilen gereksinimler sonucunda hedefler belirlenir ve bu hedeflere göre çözüm önerileri üretilir.

Çözüm Yolu Seçilir ve Risk Ortadan Kaldırılır: Bu aşamada çözüm yolları değerlendirilir ve en iyi çözüm yolu belirlenip bu konuda prototip geliştirilir.

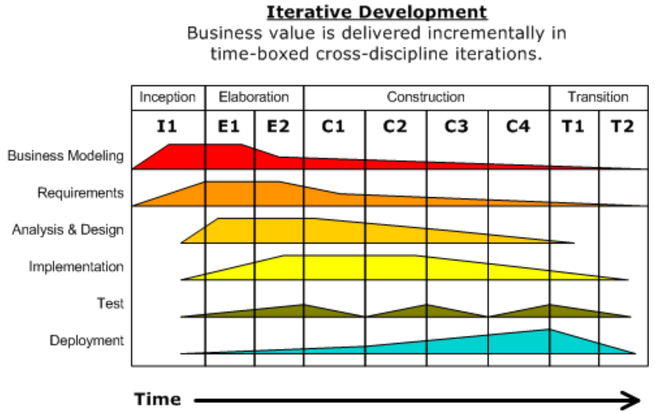
Ürünün Bir Sonraki Aşaması için Geliştirme Yapılır: Bu aşamada bu bulunan çözüm yoluna göre geliştirme ve testleri yapılır ve ürün bir sonraki aşamaya hazırlanır.

Bir Sonraki Faz Planlanır: Bu aşamada bir sonraki fazın planlaması yapılır. Operasyonel planlamalar, geliştirme planlamaları, entegrasyon ve test planlaması gibi.

RUP (Rational Unified Model)



Fazlar Aslında Birbirinden Kopuk Değildir.



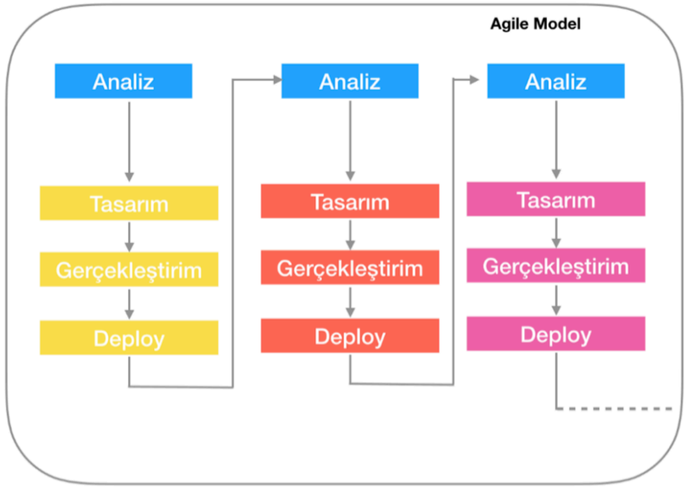
Inception(Başlangıç) I1: Projenin başlangıcında en fazla iş modelinin ortaya çıkarılması ve gereksinimlerin belirlenmesidir. Bu aşamada Analiz & Tasarım ve diğer Kodlama, Test ve Deploy işleri ile hiç uğraşılmaz.

Eloboration (Detaylandırma) E1,E2: Eldeki iş modelini , analiz ve tasarımınızı detaylandırdığınız zaman aralığıdır. Bu aşamada kodlama azda olsa başlar diğer işler çok çok az üzerinde durulur.

Construction (İnşaa etmek, Oluşturmak) C1,C2,C3 ve C4: Üründe en fazla kodlama, test ve deploy işlemlerinin arttığı kısımdır. Az da olsa işlerin ters gittiği kısımlarda tasarım ve analize geriye dönülerek düzeltmeler yapılabilir.

Transition (Geçiş) T1, T2: Ürünün müşteriye aktarılması aşamasıdır. Beta Tesleri ile sistem müşterinin isteklerine uyup uymadığı test edilir. Aynı zamanda kullanıcı, bakım ekiplerine eğitimlerin verilmesini içerir.

Çevik Model (Agile)



Neden çevik demişler ? Çeviklikten ne anlaşılması gerekiyor ?

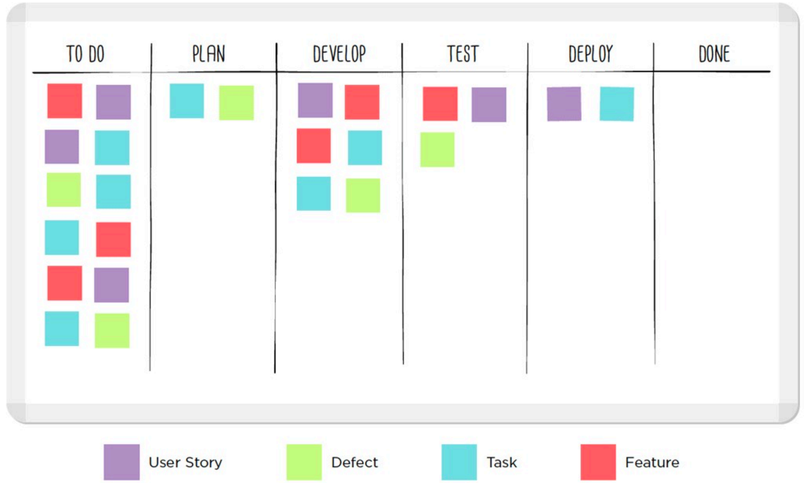
Müşterinin karşısına **en hızlı sürede ürünü çıkarmak**, üründe bir değişiklik ihtiyacı olduğunda veya hata çıktığında bunu **en kısa sürede en kalite şekilde çözebilme** modelidir.

Yukarıda bahsettiğim yöntemlerden **Evolutionary bu yapı için uygundur**. Çünkü analiz çalışmasına müşteri, iş uzmanı, ürün sahibi, ve geliştirici gibi ekibin topluca bu aşamaya katılması uygulamadan istenenleri gereksinim olarak doğru tariflemeyi sağlar.

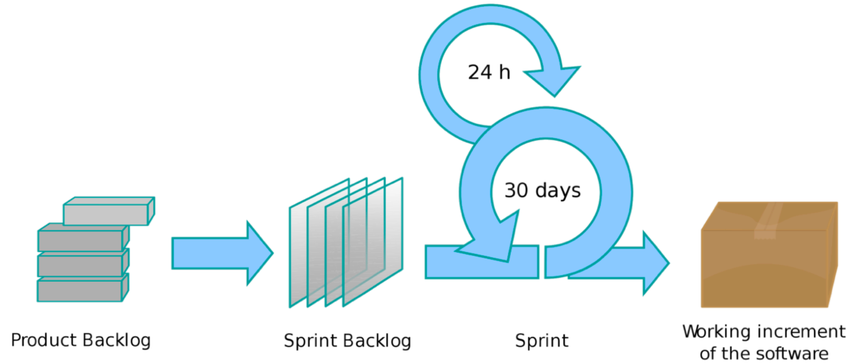
(AGILE) ÇEVİK YAZILIM GELİŞTİRME DETAY

Çevik Yazılım Türleri

KANBAN

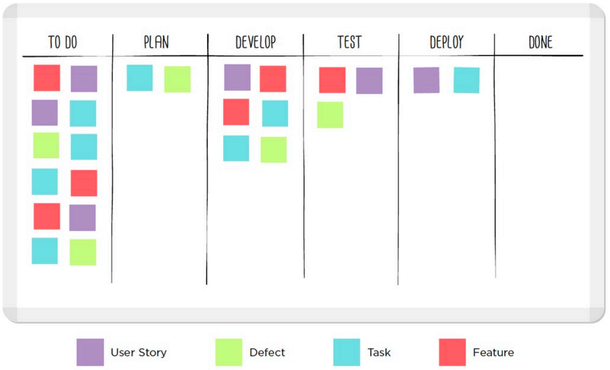


Scrum



XP

Çevik Yazılım Çıkış Felsefeleri



KANBAN

1940 sonrasında Toyota mühendisi Taiichi Ohno tarafından geliştirildi.

Süpermarket raflarda tükenen malların stoklanması gözlemleyerek, bunları kendi tedarik sistemine uyarladı.

Ana amaç **fazlalık üretmekten kaçma ve kaynaklarını olabildiğince verimli kullanmak** üzerinedir.

Bunun için birincil öncelik **üretim bandında ilerleyen kaynakları görselleştirebilmektir**. Bu sayede herkes süreç hakkında bilgi sahibi olup fazlalığı ve azlığı gerçek zamanlı yönetebilmesine olanak sağlar.

Çevik Yazılım Çıkış Felsefeleri

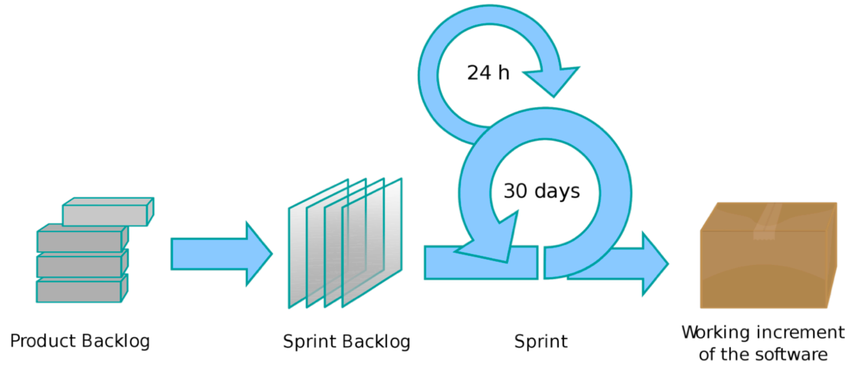
XP

Kent Beck **(Agile Manifesto, JUnit, XP, Design Patterns)** oluşumu içerisinde yer alan birisi olarak Chrysler ‘da çalışırken yazılım geliştirme içerisinde güzel uygulamalarını alıp XP(Extreme Programming) içerisine yerleştirmiştir.

Diğer yöntemler geliştirme süreçlerine ve endüstriyel yaklaşım iken bu yöntem direk **yazılımın geliştirme kısmına** odaklanmıştır.

Çevik Yazılım Çıkış Felsefeleri

SCRUM



Geliştiricilerin kendi planlarını yapabildikleri, güncelleyebildikleri, sistem işleyişini retrospective değerlendirebildikleri,

Sistem düzgün işlemesinden **Scrum Master** sorumlu olduğu,

Müşterinin isteklerini ürüne doğru ve öncelikli aktarılmasından **Product Manager** sorumlu olduğu,

Müşterinin **istekleri doğrulrultusunda önceliklendirilmiş ürün odaklı** çevik geliştirme yöntemidir.

Çevik Yazılım Geliştirme Manifestosu

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 bulmaktayız.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

Çevik Yazılımın Prensipleri - 1

En önemli önceliğimiz değerli yazılımın erken ve devamlı teslimini sağlayarak müşterileri memnun etmektir.

Çevik (Agile) bildirinin merkezinde müşterinin yer aldığını görebilirsiniz.

- Değerli yazılım derken temelde para vb.. somut kazançlar sağlayacak ürünün/ yazılımın erken ve sürekli müşteriye ulaştırması,
- Özünde müşterilerinizi mutlu etme,
- Problemleri en erken safhalarda tespit etme,
- Geribildirimler ile ürünü doğru yola sokma çabasıdır.

Çevik Yazılımın Prensipleri -2

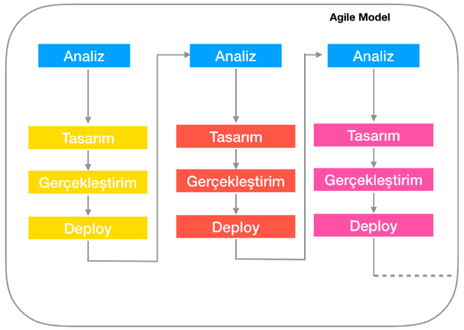
Değişen gereksinimler yazılım sürecinin son aşamalarında bile kabul edilmelidir.Çevik süreçler değişimi müşterinin rekabet avantajı için kullanır.

Bu ilke de yine **müşteriyi memnun etme** çabası vardır. Çünkü müşteri istediği ürünü en başta tarif ederken tam olarak ne istediğini bilmiyor olabilir.

Yazılım şekillendikçe ve ortaya çıktıkça, müşterilerin ürün hakkında daha çok ve kaliteli yorum yaptığını görebilirsiniz.

Bu değişimleri teknolojik gelişmelerden dolayı veya siz projeyi geliştirirken rakiplerinizin, pazarın durumu, ve bir çok konu gereksinimlerinizi değiştirme yolunda tetikleyebilir.
Bu değişimi yazılımın **her aşamasında karşılamayı ve kabul etmeye** hazır olunmalıdır.

Çevik Yazılımın Prensipleri -3



Çalışan yazılım, tercihen kısa zaman aralıkları belirlenerek birkaç haftada ya da birkaç ayda bir düzenli olarak müşteriye sunulmalıdır.

Agile(Çevik) Geliştime modelinde kısa zaman aralıklarında düzenli olarak müşterinin önüne çalışabilir uygulamalar sunarak, müşteriden geribildirimler ile Analiz → Tasarım → Gerçekleştirim kısmını tekrar ederek tekrar müşterinin karşısına çıkmayı hedefler.

Not: Çevik geliştirme modeli işletim sistemlerinin geliştirilmesinden, askeri yazılımlara çok geniş alanda yer bulmaktadır. Yazılım Güncellemesi, İşletim Sistemi Güncellemesi gibi çok sık aralıklar ile güncellemeler yapılıyor ise altyapıda Agile Geliştirme Modeli olduğunu düşünebilirsiniz.

Çevik Yazılımın Prensipleri -4

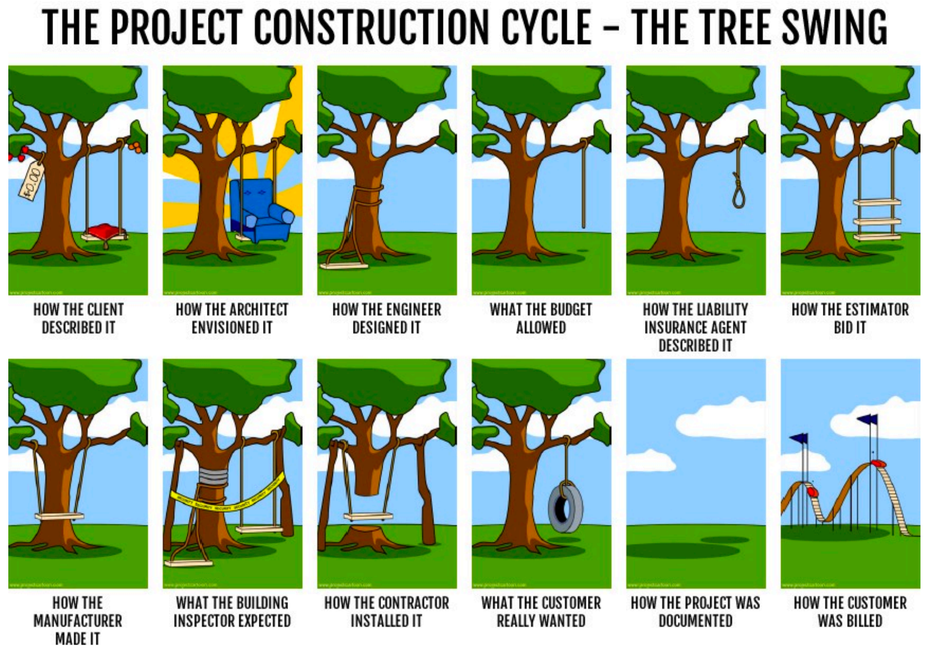
İş süreçlerinin sahipleri ve yazılımcılar proje boyunca her gün birlikte çalışmalıdırlar.

Yazılım geliştirme bir **ekip** işidir. Geliştiren ekip, bakım ekibi ve aşağıda bahsettiğim grupların birbirleri ile **uyumlu ve senkron** çalışması ürünün daha kaliteli ve müşterinin istediği gibi ortaya çıkması sağlanır. Son kullanıcı ürünü kullanarak bir nevi gereksinim ve diğer testleri yaparak geliştirme ekibine geribildirimlerde bulunur.

Çevik Yazılımın Prensipleri -4 Devam

Not: Çevik olmayan geliştirme modellerinde de ekipler birlikteydi ama bu ekipler de daha katman katman yapılar ve bir **hiyerarşi** söz konusu idi, çevik yöntem ise bu grupları birbirlerine harmanlayarak, hiyerarşiyi ortadan kaldırarak daha etkileşimli çalışmalarını önerir. Resmi bir çoğunuz hatırlıyorsunuz belki 😀

İşte buna benzer bir sonuçla karşılaşmamak için yukarıda bahsettiğim yazılım paydaşlarının her bir resim çerçevesinde bir arada olmaları çok çok önemlidir.



Çevik Yazılımın Prensipleri -5

Projelerin temelinde motive olmuş bireyler yer almalıdır. Onlara ihtiyaçları olan ortam ve destek sağlanmalı, işi başaracakları konusunda güven duyulmalıdır.

Burada kastedilen **donanım ve fiziksel araç gereç değil**.

Zaten ekibinize bütçeniz doğrultusunda işleri yapabilecek en iyi fiziksel ortamı, araç gereçleri sağlamanız hem kişinin kendisini **değerli hissetmesine** sebep olacak, hem de işleri **daha hızlı sorunsuz** hale getirecektir.

Bu ilkede esas işi verdiğiniz kişilere duyduğunuz **GÜVEN'den** ve o kişilere bu güveni hissetirmenizden bahsediyor.

Çevik Yazılımın Prensipleri -5 Devam

- Ekip veya ekipteki kişiler birbirine şüphe ile mi bakıyor ?
- Yöneticiler verdiğiniz tahminlemelere inanmıyor mu ?
- Size gerekli yetkileri veya bu yetkileri nasıl kullanacağınız konusunda bilgi verilmiyor mu?
- Bir problemle karşılaştığınızda olaya bireysel mi yaklaşılıyor? Şuçlama ile mi yaklaşılıyor ?
- Bireyler ile konuşmadan arka planda bilgi birikimini yedeklemeye mi çalışıyorlar ?

Yani yukarıda yazılanlara benzer durumları ekip bir arada açık bir şekilde konuşmuyor ve arkaplanda bunlar yaşanıyor ise, ekipteki bireylerin motivasyonun uzun süreli olması beklenemez.

Çevik Yazılımın Prensipleri -6

Bir yazılım takımında bilgi alışverişinin en verimli ve etkin yöntemi yüz yüze iletişimdir.

Özellikle remote çalışma ile birlikte, kurumsal şirketlerde bir takım dokümanlar veya araçlar üzerinden asenkron iletişim yöntemleri sıkça kullanılmaya başlandı. Slack, ePosta vb..
Yönetim araçları online sistemler üzerinden **kişiler bir birini görmeden** yapılmaya çalışılıyor. Bu şekilde geliştirme yöntemi çevik yöntemde önerilmez.

Ekibin bir arada olması, **aynı anda yüz yüze konuşarak** ortamda fikirlerini ve geribildirimlerini almaları, bu kısımlarda anlaşılmayan konuların üzerinde konuşularak ve tartışılarak daha net hale getirilmesi amaçlanmaktadır.

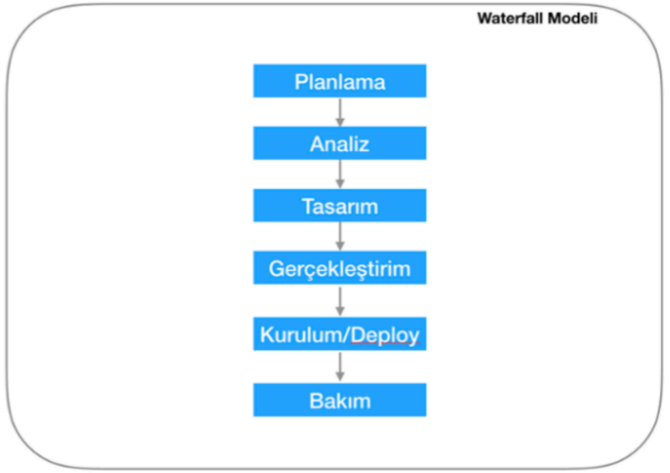
Çevik Yazılımın Prensipleri -6 Devam

Buradan da şu sonuç çıkmasın bir takım araçların kullanılması Örneğin Git, ePosta, Wireframe/Sketch vb.. araçların kullanılması reddetmiyor. Sadece bunun **dengesini iyi ayarlamamızı** söylüyor.

Örneğin bir **Wireframe/Sketch tasarımları** yapıldı Invision, Figma vb araçlar ile bu araçlar üzerinden herkes bu tasarımları görebiliyor, burdan yorumları ve bu yorumlara cevaplarıda yazabiliyoruz. Fakat bunu bir dengede tutmak önemli. Yeri geldiğinde **ara ara yüz yüze toplantılar yapılıp**, tasarımın ve yorumların ne olduğunu yüz yüze konuşmanın öneminden bahsediyor.

Yazılım geliştirdiniz ve yazılıma PR açtınız, bu PR'a bir takım yorumlar veriliyor bu yorumları sürekli asenkron çevaplayıp çözmek yerine belki **birlikte pair yaparak yüz yüze çözmek** bir çok konunun veye yorumun daha net anlaşılmasını sağlayacaktır.

Çevik Yazılımın Prensipleri -7



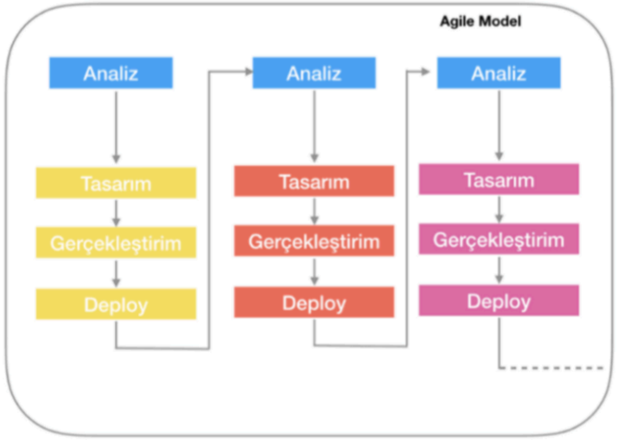
Çalışan yazılım ilerlemenin birincil ölçüsüdür.

Çevik Model çalışan ürünü hem geribildirim almak açısından hem de geliştirme ekibinin motivasyonu açısından **ilerlemenin birincil ölçüsü** olarak ele alır.

2 farklı modele bakalım. **Waterfall Modelinde** her bir fazın daha uzun sürede, belli işlere odaklanmış kişilerce bir fazda yapıldığını ve ürünün son aşamada müşterinin karşısına çıktığını görebilirsiniz.

Farklı alanda özelleşmiş bireyler (Örneğin Planlama Grubu) bir arada çalışırken , çıktılarını diğer gruba aktaran bir modeldir, **Riskleri fazladır.**

Çevik Yazılımın Prensipleri -7 Devam



Çalışan yazılım ilerlemenin birincil ölçüsüdür.

Agile yönteme baktığımızda farklı farklı alanlardan ve yetenekteki kişilerin bir araya gelerek ortak geliştirimleri yaptığı, **hep çalışan bir ürünü ayakta tutmaya** çalıştıklarını görebilirsiniz.

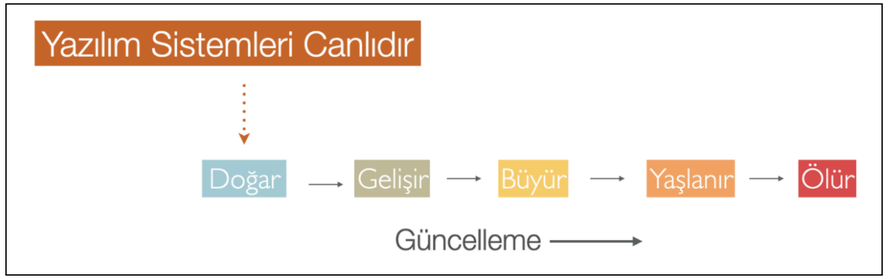
Yeni yazılım mimarileri de Örneğin microservice ve microfrontends yaklaşımları hep bu yöndedir. Hatta projeyi modüller halinde parçalayarak parallel geliştirme ve canlıya çıkma süreçleri (CI/CD) çevik yöntemler ile inanılmaz bir hıza ulaşmıştır. Bu da en ufak bir değişimin hemen müşteriye yansıtılıp geri bildirim alabilmemizi sağlar. (Örneğin A/B Testing vb..)

Özetle dokümanlar değil **çalışan yazılıma öncelik** verilmelidir.

Çevik Yazılımın Prensipleri -8

Çevik süreçler sürdürülebilir geliştirmeyi teşvik etmektedir. Sponsorlar, yazılımcılar ve kullanıcılar sabit tempoyu sürekli devam ettirebilmelidir.

Sürdürebilirlik çok çok önemli bir konu.



Çünkü yazılım geliştirme, ürünün hayatta kalması sürekli kullanılması, güncellenmesi ve değer yaratması ile ilgilidir.

Yazılım sistemleri canlıdır. Yazılımcılar ve kullanıcılar aslında bu yazılımı hayatta tutmak için uzun bir maraton koşmaktadır. Bu maratonda tükenmemek için sabit tempoyu uzun süre devam ettirebilmeli, motivasyonu ve ürün geliştirmeye devam edilmelidir. Aksi taktirde kullanılmayan kısımlar zaman içerisinde ölecektir.

Çevik Yazılımın Prensipleri -9

Teknik mükemmeliyet ve iyi tasarım konusundaki sürekli özen çevikliği artırır.

Yazılım konusunda teknik mükemmeliyet ve iyi tasarım konusunda sürekli özen göstermek çevikliği arttırır. Peki bunu nasıl yapacağız ? Bu **eğitim içerikleri** size bunu sağlıyor olacak.

Çevik Yazılımın Prensipleri -10

Sadelik, yapılmasına gerek olmayan işlerin mümkün olduğunca arttırılması sanatı, olmazsa olmazlardandır.

Çevik olabilmeniz için olabildiğince üzerinizde az yük olması gerekir. Sade bir şekilde işleri yapıyor ekstra yazılımı hiç bir şey katmıyor olmak çok önemli;

Süreçler, Ekip, Yönetim, Gereksinimler, Mimari ve Kod Tasarımlarınız, UI Tasarımlarınız Kod, Dokümantasyon, Test, vb

yazılımın parçalarını oluşturan her kısmın **sade ve yalın olması** gerekir.

Çevik Yazılımın Prensipleri -10

Sadelik için Yapılabilekler

- MVP (Minimum Viable Product)
- Overengineering (Bunu ilerde AntiPatterns kısmında bahsediyor olacağım)
- KISS, YAGNI.

Çevik Yazılımın Prensipleri -11

En iyi mimariler, gereksinimler ve tasarımlar kendi kendini örgütleyen takımlardan ortaya çıkar.

Bir takım **kendi kendini örgütleyip**, gereksinim yönetimi, tasarım, gerçekleştirim, test, deployment vb… kısımları kendi kararlar alarak ortaya çıkartırsa, bir o kadar içselleştireceği anlamına gelir.

Bu nedenle

birbiri ile koordineli bir ekip,
kendi kararlarını,
kendi süreçlerini,
dinamiklerini,
geleneklerini ortaya koymuş ekiplerden en iyi mimariler ve ürünler çıkar. Bu tarz ekipler **self-organized dır.**

Çevik Yazılımın Prensipleri -12

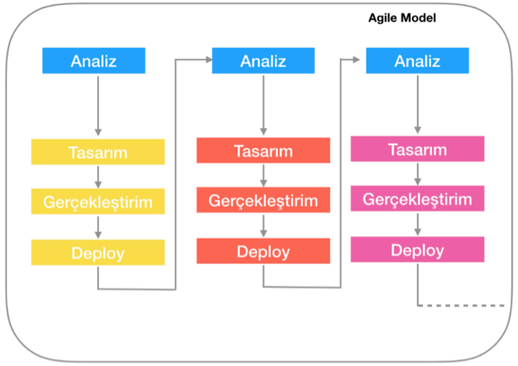
Takım, düzenli aralıklarla nasıl daha etkili ve verimli olabileceğinin üzerinde düşünür ve davranışlarını buna göre ayarlar ve düzenler.

Bu ilkeyi önceki ilkeler ile birlikte düşününce, zaten self-organized(kendini örgütleyen bir ekip ise), aynı zamanda işleri

- daha hızlı
- daha kaliteli
- daha güvenli
- daha eğlenceli
- daha müşterinin istediği

hale getirmek için ve **verimliliğini arttırmak için sürekli bir çaba** içerisinde olacaklardır.

Çevik Model Gerekleri



Standup meetings(Günlük toplantılar): Ekibin en iyi şekilde senkron olmasını, birbirini bekleyen veya problem olan bir kısmın olup olmadığını her gün kontrol etme imkanı sunar.

İterasyon kavramı : Ürünü parçalara bölüp bu parçaların müşteri önüne çıkması, bu aşama da iterasyon planlaması, gözden geçirilmesi ve işleyişinin(restrospective) değerlendirilmesi çok önemlidir. Bu işlerin ardı ardına hızlı bir şekilde işletilebilmesi

Sürekli Entegrasyon ve Canlıya Gönderebilme: Geliştirilen ürün parçalarının sürekli otomatik build/compile edilmesi, ünit testlerinin çalıştırılması, entegre edilmesi, entegrasyon testlerinin çalışması ve ardından hızlı bir şekilde canlıya müşterinin karşısına çıkarabilme yeteneğidir.

Çevik Model Gerekleri

Kanban Board: İşleri hangi safhada olduğunu takip edebildiğiniz tahta.

Refactoring, Pair Programming, Unit Test, Code Analyser: Kodunuzun kalitesini arttırmak için kötü gördüğünüz kısımlarda iyileştirmeler(refactoring) çalışmaları yapabilirsiniz.

Pair programming ile kod bir başka kişinin gözünden de yazılırken kontrolünü sağlatıp, unit test ve kod analizi ile kodun doğru işler yapıp yapmadığını test edebilirsiniz.

Kaynak kod yönetimi ve sürüm yönetimi: Kaynak kod dağıtımı ve sürüm yönetimini doğru şekilde gerçekleştirebilirseniz aynı kod yapısı üzerinde birbiriniz ile paralel kod geliştirebilme imkanı , ayrı dallar üzerinden hata giderme veya yeni feature özellik geliştirme işlemlerini gerçekleştirebilirsiniz.

DENEYİMLER

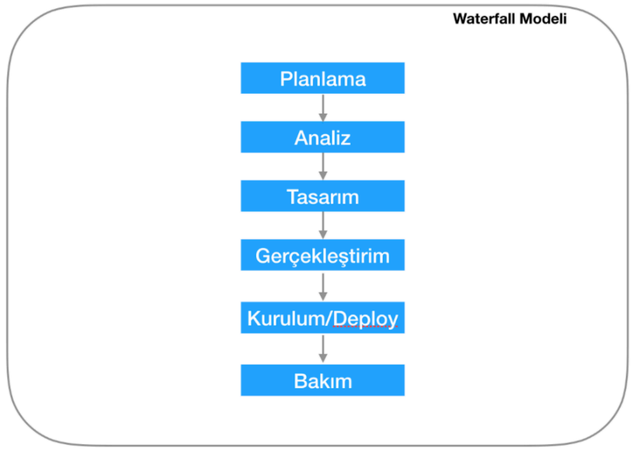
İŞ HAYATINDAN ÖRNEKLER

2002—Komuta Kontrol Yazılım Uyuşum Projesi

1. Waterfall → Incremental / Iterative -> Rational Unified Process

Üniversite'den mezun olduktan sonra 2002 yılında Havelsan' da işe başladım. İlk projemizin ismi Komuta Kontrol Bilgi Sistemleri Yazılım Uyuşum projesiydi. Proje 2 yıllık bir projeydi ve ben şirkete ilk geldiğimide ***Waterfall*** modeli kötüdür, bunun yerine ***Iterative*** Model mi kullanalım yoksa ***Incremental*** diye yöneticilerimiz bu süreç yöntemlerini değerlendiriyordu.

Tabi biz yeni mezun olduğumuz için o zaman ne konuştuklarını bu konuların önemini anlamıyorduk. Veritabanı NATO üzerinde 15 yıldır çalıştığı spesifikasyonları belli olan bir projeydi, veritabanı üzerine geliştiricek Adaptör Katmanı uygulaması Waterfall modeli ile yapılması uygundu



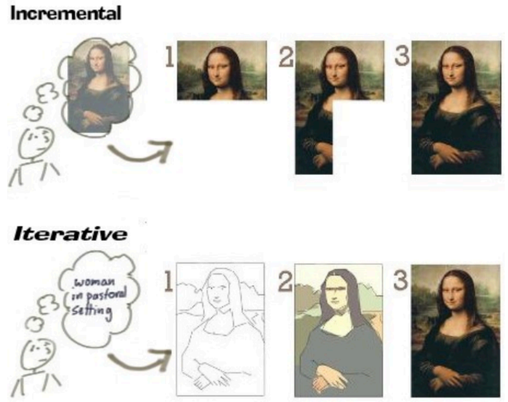
Waterfall Modeli

ama bunun üzerine geliştirilecek Muhabere ve EH uygulamasının gereksinimlerinin net olarak belli değildi. Bu durumda geliştirilecek 2 uygulamanın farklı modeller kullanmasını daha mantıklı olacaktı.

Bu nedenle yöneticilerimiz 2nci uygulama için ***Iterative*** Modeli mi seçelim yoksa ***Incremental*** Modeli mi seçelim mi diye düşündüler.

Incremental modelde işi parçalara bölüyoruz. İlk önce birinci kısmını yapıyoruz. Sonra diğer parçaları adım adım yaparak geliştiriyoruz.

Iterative modelde ise işi yavaş yavaş olgunlaştırarak daha net bir hale getiriyoruz

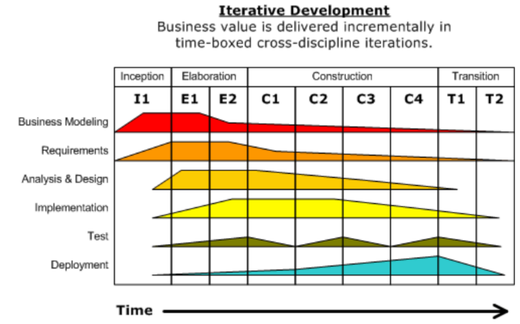


Iterative vs Incremental Model

Proje planına her 2 çözüm yönteminide kullanacaklarını belirttiler. İlk önce projeyi 3 parçaya böldüler. Bu bölümler ***Incremental*** geliştirilecekti. Bu büyük modüller kendi içerisinde ***Iterative model*** ile geliştirilecekti.

Proje planına her 2 çözüm yönteminide kullanacaklarını belirttiler. İlk önce projeyi 3 parçaya böldüler. Bu bölümler ***Incremental*** geliştirilecekti. Bu büyük modüller kendi içerisinde ***Iterative model*** ile geliştirilecekti.

Bütün bunlar hazırlandıktan sonra bunların araçlar ile desteklenmesi gerektiğinin farkına vardık. O zamanlar Rational ve bu şirketin geliştirmiş olduğu RUP(Rational Unified Process) aracı çok popülerdi. Rational sonra IBM tarafından satın alındı. Kullandığımız araçlar



Rational Unified Process

- Versiyon Kontrolü için ***ClearCase***
- Gereksinim Yönetimi ***Rational RequiresePro***
- İş Analizi ve Tasarım için UML kullanarak tasarım yapmayı sağlayan ***Rational Rose/Modeller***
- Yazılım geliştirmek için ***Borland JBuilder*** aracını kullanıyorduk.

Tasarladığımız modellerden kod üretebiliyor veya tam tersi, yazdığımız kodları reverse ederek UML modelleri üretebiliyorduk. Bu sayede tasarım ve kod senkronizasyonunu yapabiliyorduk.

- Gereksinim → İzlenebilirlik Matrisleri → Tasarım senkronizasyonu
- Tasarım → Code Üretimi → Kod senkronizasyonunu

Müşterek EH Simülatörü

2. Object Oriented Design,Test Driven Development, MVP, Design Patterns, CMMI, Agile, Interface, Dependeny Injection, Aspect Oriented Programming

2005 yılı Asker dönüşü Milsoft şirkeinde MEHSİM(Müşterek Elektronik Harp Simulatörü) projesinde işe başladığımda konuşulan herşey bana farklı geliyordu, birçok konuyu ve terimi anlamıyordum çünkü şirket ***CMMI5(***Capability Maturity Model Integrated***)*** ti. Süreçler ve süreçlerin içerisindeki herşeyin, her dokümanın adı ve kısaltması vardı ve çalışanların bu kavramları , bu kısaltmaları anlamaları bekleniyordu, bunun için şirket için uzun eğitimler veriliyordu. 1 yılı geçen uzun eğitimlerden sonra bende bu kavramları konuşanlardan birisi olmuştum :)

Mühendislik üzerine daha fazla düşenen, tartışan ve konuşan bir ekibin içerisinde yer alıyordum. Nesne tabanlı programlamada geliştiriciler sınıfları direk birbirlerine bağlamıyorlar, arasına interface, abstract sınıflar koyuyorlar hatta tasarım örüntülerine(design patternlere) göre geliştirme yapıyorlardı.

Ekip arkadaşlarımı dinledikten sonra okulda aldığım Nesne-Tabanlı(Object Oriented) ve Tasarım Örüntüleri(Design Pattern) derslerinden hiç bir sey öğrenmediğimi farkettim.

- Model View Presenter ve Model View Controller ***(Architectural Patern)***
- Mediator, Singleton, Decorator, Facede, Observer, Abstract Factory Pattern, Adapter, Iterator, State ***(Design Patterns)***

Sonrasında ilk ***Refactoring ve Architecture Clearance*** kelimelerini bu yıllarda duydum. Refactoring nasıl yapılması gerektiğini bu projedeyken Martin Fowler kitabından okuyarak öğrendim. Tabi sağolsun sonra Eclipse bu refactoring araçların hepsini editörün içerisine gömdü ve refactoring işlerini oldukça basit hale getirdi.

Bu sırada bir çok arayüz bileşenlerimiz ve bunları dinleyen listenerlarımız oldu. Event Listener ve Model Listener . Hatta bu bileşenlerimizin state'leri vardı. Yani kendi çapımızda Assembler, Dependeny Injection ve UI Componentlar oluşturmuştuk.

Tüm bunlar olurken ekipten bazı arkadaşların yurt dışında Test Driven Development eğitimi almıştı. Tüm ekibe ***Object Oriented Design → Test Driven Development*** yönlendiriyorlardı. Herkese zorunlu test yazdırmalarıda bir yandan bizi inanılmaz zorlasa ve yorsa da ***Unit Test ve Automated Testlerin*** ileride bize faydasını çok net görecektik. Bizim için hayat kurtarıcı olmuştu.

Tüm bu geliştirmeler sırasında ***Agile(Çevik)*** yöntemleri ***XP(Extreme Programming)*** yöntemini kullandık. Bu dönem en çok eğlenerek çalıştığım dönemdi.

Burada UI(Arayüz) Bileşenlerini geliştirirken Nesne-Tabanlı (Object Oriented) geliştirmeye ters olan iki durum ile karşılaştım.

- Birincisi Authorization ve Roles
- İkinciside Application Mode (Running Mode ve Replay Mode) .

Bu iki durumda tüm bileşenleri ***Cross Cut*** eden disabled, non editable, editable görünmesine sebep olan durumlardı. İşte ilk olarak ***Aspect Oriented Programming*** ile karşılaştığım kısım burası oldu.Bedir Tekinerdoğan Hocanın Aspect programla dersi verdiği 2007 yılında Aspect ile yazdığımız kodlar çok yavaş çalıştığı için biz yine ***Aspect yaklaşımları Object Oriented yöntemler ve Design Patternler*** ile çözdük.

- Gereksinim yönetimi için ***Doors*** kullanıyorduk.
- Source kontrol olarak ***Clear Case***,
- Designer olarak ***Rational Modeller*** kullanıyorduk. Sürekli kodları Reverse ederek tasarım dökümanlarını oluşturuyorduk.

Link Sistemleri 11/16/22

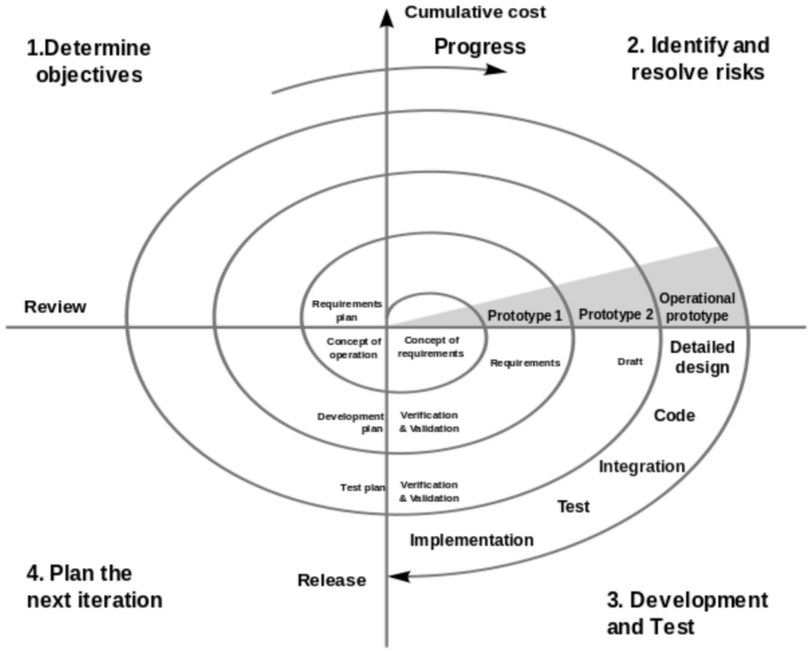
3. Enterprise Architect, Spiral, Maven, Spring, MVP, Automated Test, Automated Help, Dynamic Layout

2008 yılında Milsoft'da GVLS'nin LinkMMI (Genesis Veri Link Sistemleri Arayüz) kısmına geçtim. Burada MEHSİM'den edindiğim bilgi ile daha iyi bir tasarım oluşturduğumuzu düşünüyorum.

MVP(Model View Presenter) daha basit bir hale getirdik. Delegation üzerine Delegation yöntemleri basitleştirdik. Eclipse'den plugin ile DataChannel ve Model tipi vererek otomatik proje oluşturma templatelerini oluşturmasını sağladık. Ekran Unit Testleri için Model ve Channel Datalarını otomatik dolduran ve reflection ile karşılaştıran araçlar yazdık. Automated Help oluşturması için Word -> HTML parçalara dönüştüren ve bunları birine linklereyek Help oluşturan araçlar yazdık.Link11, Link16, Link22 göre ekranların otomatik değişebilmesi için XML üzerinden çalışabilecek ***Dimamik Layoutlar*** geliştirdik.

Özetle LinkMMI projesi Swing UI Bileşenleri açısından Maven, Spring, Component yapılarını daha düzgün ve gelişmiş bir proje olmuştur.

Link MMI projesinde ***Spiral Metodu*** kullandık. Sürekli prototipler çıkararak entegrasyon + testler gerçekleştirdik. İlk defa farklı ortamlarda bulunan kişilerin geliştirdiği yazılımlar bir araya getirilip, entegre edilip ayrı bir deployment ortamında deneniyordu. İlk defa ***deployment*** kavramı bu projede anlayabildim.



Spiral Model

Türk Telekom Online Hizmet Merkezi

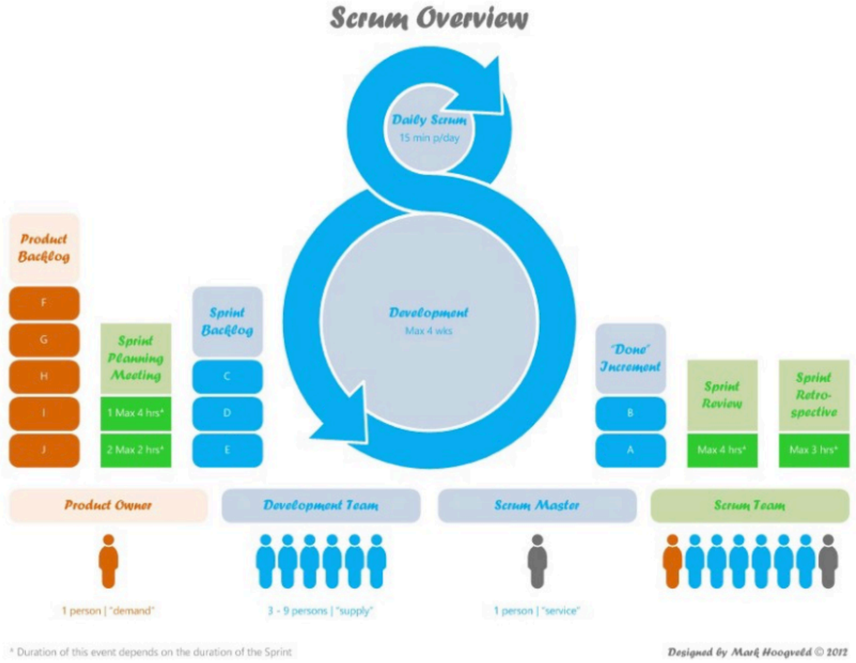
4. Scrum, SOAP, WSDL, JSP, SQL, Oracle, WebLogic, Maven

2011 yılında Savunma sektöründen çıkıp T2 firmasına geçtiğimde Türk Telekom iş yapmaya başladık.

- Mobile Web Online Hizmet Merkezi,
- iPhone uygulaması,
- Kurumsal Online Hizmet merkezi projelerinde çalıştım.

İlk günlerimde gerçekten projelere yabancılık hissettim. Savunma sanayinde en az ***3 yılda biten projeleri burada 3–6 ay*** arasında bitmesini istiyorlardı.

Başladığımda ne ***SpringMVC, ne SpringSecurity*** konularında deneyimliydim. ***WSDL, SOAP, Weblogic, Oracle, SQL vb..*** konularda sürekli birşeyler isteniyor. Çok kısa sürede bunlara adapte olup projeleri gerçekleştirmemiz bekleniyordu. Neyse ki bir şekilde ortama adapte olup ***Maven, Weblogic, JS, JSP, ObjectiveC*** konseptlerini kullandığımız uygulamalar geliştirdik. Yöntem olarak ta o zamanlar Barış Bal arkadaşımızın ısrarla üzerinde durduğu Scrum Modelini kullanmaya çalışıyorduk. Barış sürekli gelip ***Scrum*** şöyle iyi, böyle iyi şeklinde öğütler verip bize Scrum kullandırtmaya çalışıyordu. Bence Scrum hiç kullanamadık. Çünkü kimse bu yöntemi kabullenememiş. Herkes bodoslamasına iş yapmayı seviyordu. İş yapıyorduk ama kalite iş çıkaramıyorduk.(Source CheckStyle, UnitTest, Dokumentation) yoktu. Direk kişilere bağımlı bir geliştirme modelini kullanıyorduk.



Agile — Scrum Framework

5. Kanban , TMForum, Hibernate, Confluence, JIRA, Bamboo

T2'de TSE projesine geçtiğimiz zaman İbrahim Güntaş Scrum'ın bizim için çok anlamlı olmadığını Kanban yönteminin bizim geliştirme süreçlerimiz için daha uygun olduğuna inanmıştı. ***Analizin ve Planlamanın*** gücünü bu projede gördüm diyebilirim. Herşey TMForum tasarımlarından alınarak bizim projeye uyarlanmaya çalışılıyor ve hepsi birden ***Confluence*** madde madde giriliyordu. Kanban yöntemiyle oldukça basit şekilde JIRA görevler atanıyor ve ekibin bu görevleri tamamlaması bekleniyordu.

Uzun zamandan sonra ilk kez Continuous Integration ***Bamboo*** ile burada gerçekleştirdik diyebilirim.

- Gereksinim ve Dökümantasyon için ***Confluence***
- İş Takibi ve Planlama için ***JIRA***
- Continuous Integration ***Bamboo***

Kanban Board

Mobil Uygulamalar

6. Stash, Jira, Bamboo, Integration Patterns, AWS(Amazon Web Services), Redis, ActiveMQ

2014 yılında Cybersoft şirketine geçtim. Mobil Uygulamalar Grup yöneticisi bu firmaya geçtim ve bu şirkette mobil ekip oluşturduk. Mobil uygulamaları geliştirirken Scrum çevik modelini uygulamaya çalışıyorduk , En çok kullandığımız özelliği ***Backlog*** oluşturma, ***SprintLog*** belirleme ve ***Günlük Toplantılar.*** temel olarak iyi yaptığımız işlerdendi.

- Kod versiyon yönetimi için ***Stash***
- Proje yönetimi ve görev ataması için ***Basecamp***
- Sürekli Entegrasyon(Continuous Integration) için ***Bamboo***
- Hata takibi, müşteriden gelen istekler için ***Jira***
- Mobil uygulamaların Alfa/Beta Testleri için ***Testflight***
- Mobil uygulamalarda oluşan hataları yakalamak için ***Fabric***
- Mobil uygulamaları Notification(Bildirim) için ***AWS SNS/Firebase***
- Mobil uygulamaları sanat test cihzlarında test edebilmek için AWS ***DeviceFarm*** kullanıyorduk.

Mobil kısım dışında sunucu tarafındaki Enterprise servisleri Mobil uygulamalarda kullanamadığımız için kendi mockup servislerimizi AWS EC2 makinelerine yükleyip bu ortamda infrastructure oluşturmaya çalışıyorduk. İlk mimari yapılarımız

- AWS EC2, ElasticBalancer, EBS, RDS, ActiveMQ, SNS

Daha sonra Arge Projeler için iBeacon ile şirkete giriş/çıkış kimlik kontrolü ve çalışma süresi hesaplayan uygulama için

- AWS ECS ve Docker ve AWS RDS altyapılarını kullandık.

Sigortacılık Skorlama Motoru ve Editorünün çalıştırılması, ve Ölçeklenebilirliğin hızlı bir şekilde yapılabilmesi için

- AWS Elasticbeanstalk, RDS, AWS S3 altyapılarını kullandık.

İlerleyen süreçte geliştirdiğimiz Tubitak Projesi kapsamında maliyetler açısından tüm yapımızı Serverless bir mimariye dönüştürmüştük.

- AWS Lambda, ApiGateway, SNS, SQS, S3, SES, DynamoDB vb AWS Managed Servisleri kullandık.

İlk zamanlar ***Scrum*** olarak düzgün bir şekilde ilerdiğimiz projeler sonraki dönemlerde benim kısa sürelerde işler geliştirmemiz, işin ARGE kısmının çok olması, benim teknik altyapıyı ilerlettiğim, altyapısını geliştirdiğim bundan sonra bunu ekibe aktardığım bir döneme dönüştü.

- Altyapısal denemeler/testlerin yapılması
- Atyapının hazırlanması
- Ekiplere bu bilgilerin aktarılması
- Ekibin işleri geliştirmesi
- Test Edilerek Canlıya Alınması

Bu sürecin sıkıntısı, geliştirmeler tek kişiye bağımlı, onun sözüne göre ilerliyor, ekibin buna dur demesi, yanlış yapıyorsun deme imkanı yok, ekip tam ekip olamıyor, sadece iş verdiğin ve işleri gerçekleştiren kişiler olmuş oluyor ve projenin bir parçası gibi hissetmiyorlar.

Bu sistemin bir süre iyi işlediği yer ise aşağıdaki gibi durumlardır;

- ***ekip dengeli kurulmamış*** (çok deneyimli bir kişi ve diğer ekip üyeleri yeni mezun olduğu durumlarda)
- ***projenin geliştirme zamanı çok çok kısa*** (projenin analizi, tasarımı , geliştirimi gibi süreçleri işletebilecek zaman yok 1,2 ayda çıktı vermeniz gerekiyorsa)
- **projenin teknik olarak işlerinin çok çok zor olması** (piyasada önceden çokça yapılan bir uygulama değilde , belli özellik ve deneyimlere sahip kişişerin geliştirebileceği bir yazılım ise)

Burda ekibin başı, işleri asıl sürükleyen kişinin kendi altyapısını kurduğu sistemi olabildiğince diğer geliştiricilerden soyutlayarak bir motor/engine gibi bu kısımları yazması ve geliştirmesi diğer geliştiricilere bu sistemin üzerine nasıl kod geliştirebilecekleri konusunda yardım etmesi gerekir.

Startup - SaaS

7. Jira, Scrum,Shapeup, Git, Github, Jenkins,

2018 sonunda Thundra firmasına Senior Frontend Developer yani uygulamanın önyüzünü geliştirmek için katıldım. İlk geldiğim zamanlarda Scrum uyguluyorduk. Burda uygulamanın gereksinimlerini ve ihtiyaçlarını belirlerken aşağıdaki mekanizmalardan faydalanıyorduk.

- ***Jobs to Be Done*** (Müşteri uygulamayı neden kullansın, hangi işini yapıyorsun)
- ***Competitors Tracking And Analysis*** (Rakipleri Takip ve Analiz)
- ***MVP*** (Min-Value-Product)

Tüm kaynak kod kontrolümüzü ***Git*** üzerinden yapıyor, CI/CD ***Jenkins*** üzerinden yapılıyor ve bu komutları ***Slack Bot*** olarak yazılmış bir uygulamadan kontrol edebiliyorduk.

Bu firmada diğer hoşuma giden özellik ise, geliştirme ortamlarının birbirinden çok net olarak ayrılması ve bu ortamlardaki belli hesaplarda bir sürü örnek projenin bulunmasıydı.

- Local, Lab, Staging, Canlı, Demo

Farklı env bulunması, uygulamanın canlıya giderken bir çok ortamda test edilmesi ve en doğru , en kaliteli şekilde canlıya atılmasını sağlıyor. UI anlamında bir çok bileşenin bir çok farklı testi gerekiyor. Bunun en iyi yöntemi elinizin altında her durumu deneyebileceğiniz bir çok örneğin bulunması. Bizim demo kullanıcılarımız içerisindeki örnekler bu testleri yapmamızı sağlayacak bir çok örnek proje barındırması bizim işlerimizi hem kolaylaştırıyor hemde daha iyi test yapabilmemize imkan sunuyor.

Gereksinimlerin analizi konusunda bazen sorunlar olabiliyor tam analiz etmeden bazen özelliği geliştirmeye başlıyorduk. Bazen doğru özellikleri geliştirip geliştirmediğimiz konusunda problemler yaşayabiliyoruz. Bu konuda Basecamp firmasının geliştirmiş olduğu Shape-Up deneyeceğiz.

Özet

Dönüp geriye baktığımda **Ekip, Bilgi ve Deneyim** var ise *Süreç, Geliştirme Modeli ve Kalite* kendiliğinden geliyor. Her zaman bir model yok hibrit modelleri uygulayabiliyor veya bu modelleri kendilerine göre değiştirip kullanıyor ekipler.

Önemli konu

- Ekibin birbirini sevmesi sayması,
- Ekibin birbirine güvenerek en iyisini yaptığını düşünmesi
- Ekibin birbirine yardım edebilmesi.
- Anlaması, Empati kurabilmesi.
- Ekibin öğrenmeye aç olması, bir şeyler öğrenmekten kaçmaması

Süreç zaten sonrasından geliyor. ister ***Agile***, ister ***Spiral***, ister ***RUP***, ne isterseniz kullanabilirsiniz. Esas olay ekibin durum karşısında hangi süreç ve araçları kullanmak istemesine kalır. Yeterki ekip anlaşabilsin ve uyum içinde çalışabilsin.

Yani süreç ekibi bir şeylere dönüştüremez. Daha iyi bir hale getiremez. Ekip gerçekten varsa ve anlaşıyorsa süreçler olur ve anlam kazanır.

YAZILIM MİMARININ 5 HATASI

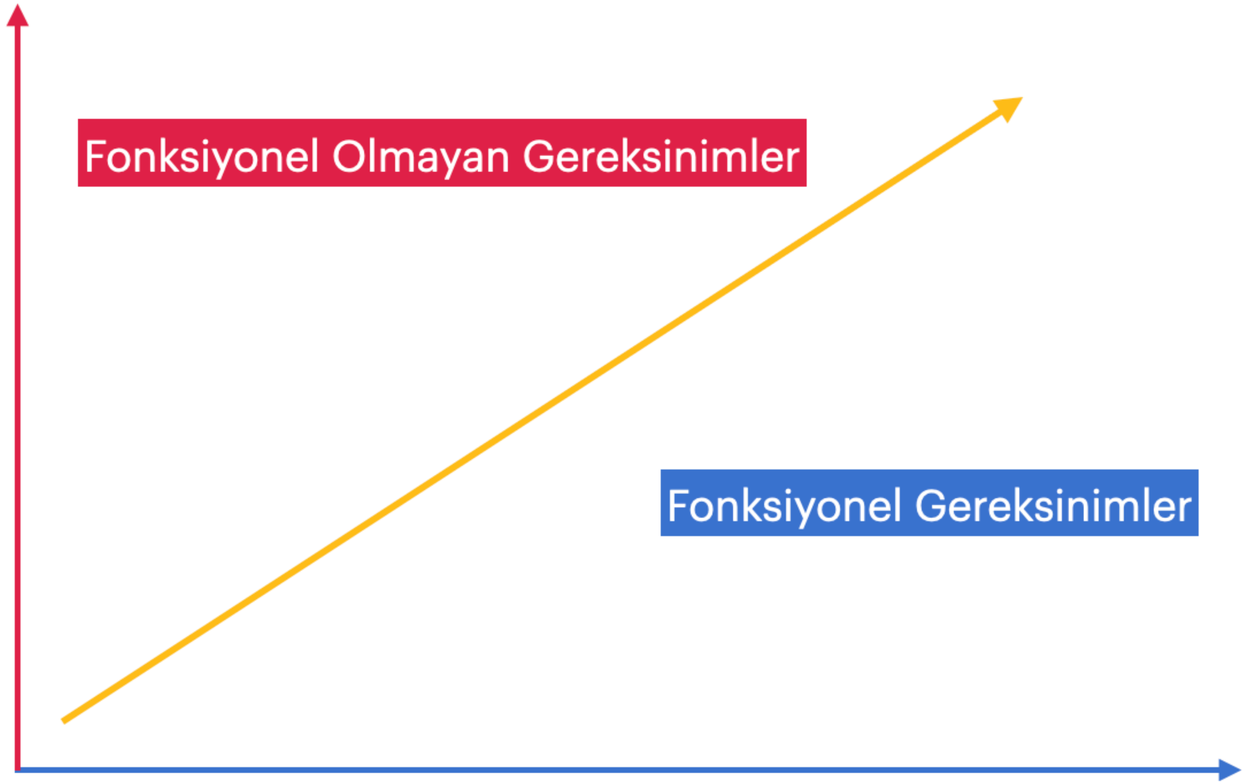
Yazılım Mimarının Kurması Gereken Denge

Yazılım Mimarisinin temeli dengedir. Yazılım Mimarının kurması gereken denge. Aşağıda bahsedeceğim konularda bu denge kavramının çok önemli olduğunu sizlerde göreceksiniz;

- Fonksiyonel ve Fonksiyonel Olmayan Gereksinimlerin Gerçekleştirimi,
- Gereksinimleri ve Sorunları Doğru Teknik Alanlarda Çözmek,
- Değişime ve Yeni İsteklere Karşı Dirençli Olma Konusunu, Doğru Anlama,
- Her Türlü Problemi Çözebilecek Mimari Oluşturma Çabasının Sınırlarını ve Maliyetlerini Anlama,
- Mimarın yaptığı geliştirmeleri Ekibe AktarMaması.

1. Fonksiyonel ve Fonksiyonel Olmayan Gereksinimleri Doğru Şekilde **İlerletmemek**

Fonksiyonel gereksinimler bir sistemin nasıl davranacağı, iş kurallarını, akışı için gereksinimleri tanımlarken,



Fonksiyonel ve Fonksiyonel Olmayan Gereksinimlere Örnek

Örneğin: Kullanıcı sisteme kullanıcı adı ve şifresini girerek giriş yapabilecektir. Fonksiyonel olmayan gereksinimler ise sistemin bu davranışları nasıl gerçekleştirdiği ile ilgilenir.

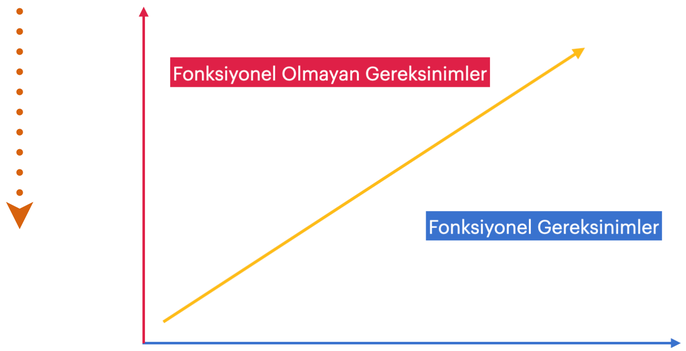
Kullanıcı şifresi her ortamda **encrpyt ederek** taşıyacak ve saklayacaktır. (Güvenlik Gereksinimi)

Özetle Fonksiyonel olmayan gereksinimler, sistemin davranışının,yeteneklerinin,genel karakteristiklerinin **kullanıcı deneyimi üzerindeki etkisidir.**

Fonksiyonel ve Fonksiyonel Olmayan Gereksinimlere Örnek

Örneğin bir denetim uygulaması geliştiriyorsunuz. Bu denetim uygulaması belli ofislere gidip denetleme yapacak mobil bir uygulama olsun. Eğer ki bu denetlemeyi yapacak personelin yaşı büyük olduğu ve gözlerinin çok iyi görmediği ve düğmelere iyi **basamadığını** analiz etmiş iseniz. Fontların ve Düğmelerin Boyutları her ekran için etkilenecektir. Gördüğünüz gibi fonksiyonel olmayan gereksinimleri ele almak oldukça zordur.

Fonksiyonel ve Fonksiyonel Olmayan Gereksinimlere Örnek



- Bu durumda yukarıdaki görseldekine benzer bir ilerleme en doğrusu olacaktır. Fonksiyonel gereksinimlerde biraz ilerlemek çünkü FG biraz ilerlemeden FOG için gereksinimleri tam olarak anlayamazssınız. Öncelikle **biraz sistem davranışını anlıyor** olmanız lazım.

- Sürekli fonksiyonel gereksinimleri geliştirirseniz. İşin sonuna geldiğinizde **riskli olan ve sistem davranışlarının tüm yapısına etkileyecek** olan FOG uygulama ve denemede geç kalmış olursunuz. FOG düşündüğünüz şekilde yolunda gitmeyen durumların etkisini tüm önceden geliştirdiğiniz fonksiyonel gereksinimlere uygulamanız gerekir.

2. Gereksimimleri ve Sorunları Doğru Alanlarda **Çözmemek**

Yazılım mimari sorunları deneyiminin fazla olduğu alanlarda çözmeye çalışır. Çünkü bu yöntem ile kendisini daha güvende hisseder. Sorunu hangi alanda, hangi araçlara göre çözeceğiz ?

2. Gereksimimleri ve Sorunları Doğru Alanlarda **Çözmemek**

- Yazılım mimarı geçmişte **veritabanı uzmanlığı** ile geçmiş ise, sorunları veritabanı sorguları ve ara tabloları, tetikleme mekanizmaları, SQL stored procedures ile çözmeye çalışır.

- Yazılım mimarının geçmişte **frontend uzmanı** ise, olabildiğince backend ve diğer yapıları çok az kullanarak tüm veriyi client(istemciye) aktarıp sorunlarını tarayıcı ve UI üzerinden çözmeye çalışır.

- Yazılım mimarının geçmişte **backend uzmanı** ise, olabildiğince backend düşüncesiyle hareket eder.

- Yazılım mimarının geçmişte sistem ve **DevOps ile uğraşmış** ise bu yapıyı sistemler üzerinden dağıtık bir şekilde nasıl yapacağını düşünür ?

- Proje analizi, gereksinim analizi, **proje yönetimi ile uğraşmış** ise müşteri üzerinden sorunları veya yapıları tanımlaya çalışır.

2. Gereksimimleri ve Sorunları Doğru Alanlarda **Çözmemek**

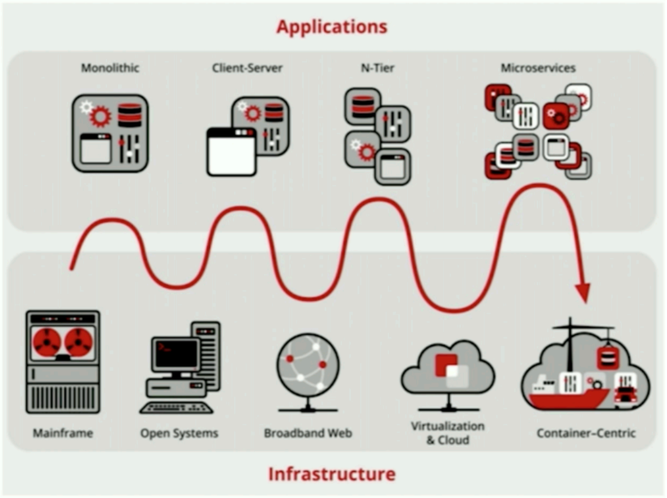
Burada önemli olan yazılım mimarı dediğimiz kişi veya kişilerin geçmişte bu alanlarda deneyimi olup, sistemin **gereklerini doğru alanlara eşleştirmek** olmalıdır.

Müşteri, Frontend, Backend, DevOps, DB, Cloud ile donanım, yazılım ve bunların iletişimlerini doğru bir şekilde parçalayabilme kabiliyetine sahip olmalıdır.

3.Değişime ve Yeni İsteklere Karşı Mimariniz, Ufak Yamalar ile Projenizi Ne Kadar Ayakta Tutabilirsiniz ?

Projelerin büyük bir çoğunluğunda baştan düşünülen, kağıt üzerinde yazılan ve tasarlanan ile sonuçta ortaya çıkan ürün arasında fark olur. Bunun farklı farklı nedenleri olabilir,

- Teknoloji değişmiştir,
- Müşteri konuya o anda çok hakim değildir,
- Sistem gereksinimlerini belirleyen yeni kişiler konuya dahil olmuştur,
- vb..



Hangi nedenden dolayı kaynaklanırsa kaynaklansın, mimari yapı ve bunun üzerinde tutulan **veri model yapısı oturmaya başladıktan sonra** sistem üzerindeki değişiklik yapmak zorlaşacaktır.

3.Değişime ve Yeni İsteklere Karşı Mimariniz, Ufak Yamalar ile Projenizi Ne Kadar Ayakta Tutabilirsiniz ?

Burada mimarinizi bozmayacak **yamalar veya ufak değişiklikler** ile ilerlemeye ve bu istekleri sisteminize katmaya çalışırsınız. Ama belli bir aşamadan sonra katılamayacak durumada gelebilir.

Eğer bu kısımda **açık olmazsanız**. Bu mimari bu istekleri karşılamaz şeklinde geribildirimlerde bulunmazsanız, müşteri ve ürün yöneticisi **daha fazla istemeye** devam edecek ve zaman içerisinde mimari kalmayacaktır. Daha sonrasında sistem her tarafından yavaş yavaş patlaklar ve hatalar verir.

Genelde son aşamaya kadar gerekli uyarıları vermeyen mimar rolündeki kişiler, gereksinim baskısı ve sistemin dağılması kısmını yönetemedikleri için **istifa yöntemine** veya o ortamdan çıkış yolu bulma yoluna gideceklerdir.

3.Değişime ve Yeni İsteklere Karşı Mimariniz, Ufak Yamalar ile Projenizi Ne Kadar Ayakta Tutabilirsiniz ?

Diğer bir alternatif ise gerekli yerde **yeni isteklere dur diyebilmektir**. Bu mimarinin bu istekleri kaldırmayacağını belirterek, mimariyi baştan oluşturma veya yeniden kurgulama maliyetini dile getirmek ve yönetimi bu konuda ikna etmek gerekir.

Not: Bahsettiğim durum mimari bir takım ciddi değişiklikler de bile sistemin yeni gereksinimleri karşılamadığı durumdur.

4.Her Türlü Yeni Gereksinim İhtiyacınıza Karşı Mimarinizi Generic (Dinamik) Tasarlamanın Maliyeti

Bazı yazılımlar ihtiyaçtan dolayı dinamik ve genel tasarlanması gerekir.

Oyun Level Tasarımı : Normalda bir oyun motoru yazılır ve bu oyun motorunun kendisine verilen level (bölüm) içerisindeki karakterleri oluşturmak için oyun editörleri bulunur. Bu oyun editörlerine bu eşyaları ve karakterleri koyduktan sonra bir senaryo koşulmasını ve bu karakterlerin davranışlarının tanımlanması gerekir. İşte burada bir Scripting yazılması gerekir. Oyun geliştirmede genellikle Lua ve Python scripting dillerden faydalanır.

4.Her Türlü Yeni Gereksinim İhtiyacınıza Karşı Mimarinizi Generic (Dinamik) Tasarlamanın Maliyeti

Sigorta ve Bankacılık Skorlama ve Risk Sermaye Koşulları: Günümüz o kadar hızlı değişiyor ki sigorta ve bankacılık için risk ve sermaye koşulları sürekli değişir. Ekonomi, Afet, Deprem gibi konulardan etkilenir. Bu değişikliklerin skorlama ve risk durumlarına alan uzmanları tarafından yansıtılması gerekir. İşte burada kredilendirme ve skorlama araçları ve yapının üzerinde çalıştığı kural motorlarından oluşmuş altyapılar devreye giriyor. Bu sayede arka planda bu yapılara göre dinamik kod mekanizmaları çalıştırılıyor.. (Kampanya Sistemleride bu duruma benzer yapılardır)

4.Her Türlü Yeni Gereksinim İhtiyacınıza Karşı Mimarinizi Generic (Dinamik) Tasarlamanın Maliyeti

AdHoc Araçları, Business Intelligence , Karar Destek Sistemleri : Şirket yöneticilerinin işletme hakkında sağlıklı kararlar alabilmesi için raporları dinamik oluşturabilmeliler. Özelleşmiş dinamik dashboard bu grafikleri ekleyerek bunlar üzerinden gösterim yapabilmeliler. Bunun için, verinin doğru şekilde çekilmesinden , birleştirilmesinden ve doğru görsel ve çizelgeler ile dashboard gösterilmesi ile yapılabilir. Bu tip mekanizmalar arka planda bir takım dinamik altyapısı ile mümkündür.

4.Her Türlü Yeni Gereksinim İhtiyacınıza Karşı Mimarinizi Generic (Dinamik) Tasarlamanın Maliyeti

Uygulama Geliştirmek için No-Code, Low-Code Araçları : Bu tip araçlar sizin uygulama bileşenleri ve bunlar arasındaki ilişki ve bağlantılarını tanımlayarak dinamik bir takım scriptler yazarak bir uygulama oluşturmanızı sağlar.

BDD için Test Araçları : Bazı uygulamaların End-to-End testlerini statik kodlamak sürekli kırılgan olmasına ve sürekli bakıma neden olabilir. Birde Testi ilgili İş alanının uzmanlarının bu testleri yazması daha doğru olacaktır. Bu tarz yapıların kurulması için Cucumber vb… test araçları bulunur ve bu araçlar arka planda dinamik olarak test kodlarının çalıştırılmasını sağlar..

4.Her Türlü Yeni Gereksinim İhtiyacınıza Karşı Mimarinizi Generic (Dinamik) Tasarlamanın Maliyeti

Ama bu tarz mimarilerde uygulamanın sınırları çok iyi tanımlanmalıdır. Dinamiklik içerisinde bile sınırlar bulunur.
Bu dinamik sistemleri yönetmesi, başkalarına anlatarak eğitmesi, sistemi ayakta tutması(maintain) etmesi diğer projelere göre oldukça zor olacaktır. mimar rolündeki kişi bunları bilip aşağıdaki konulara çok fazla değer ve yatırım yapması gerektiğini biliyor olmalı

- Monitoring (Sistemi Takip Etme)
- Debugging (Kuvvetli Debug Mekanizmaları Kurma)
- Maintainance (Bakım ve Veri Model versiyonları arasında Geçişi Kontrol)
- Education (Bu sistem üzerinde geliştirme yapacak kişileri eğitim)

5.Mimariyi Tüm Proje Ekibine Yaymazsanız Limitlerini ve Kullanımını Anlatmazsanız Sonuçları Nasıl Olur?

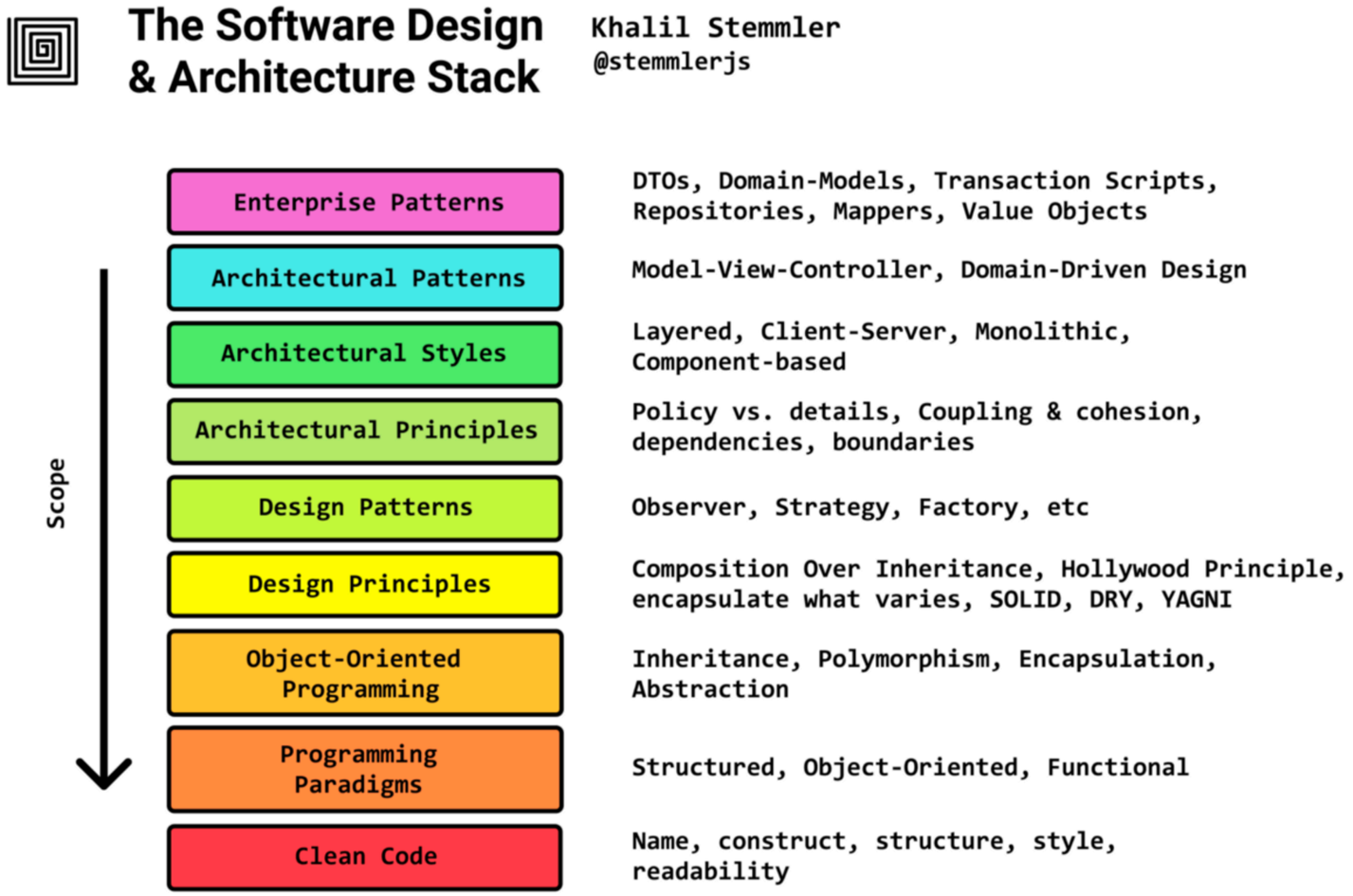
Son olarak mimar rolündeki kişi veya kişiler ortaya çıkardıkları mimariyi sürekli proje ekibine anlatmalı, onların sorularını, g**eribildirimlerini dikkate alarak bunları cevaplamalıdır**. Tüm paydaşlar bu sistem üzerinde nasıl yazılım geliştirileceğini biliyor, sınırlarını ve gücünü anlıyor olmalıki bu mimariyi doğru şekilde kullansınlar.

Aynı zamanda bu sistem üzerinde geliştirmeler yapılırken bu geliştirmeler üst seviye doğru mu, iletişim yapıları ve yöntemleri **mimariye uygun mu şeklinde sürekli bir kontrol mekanizması** kurup, bu yapının dışına çıkan işleri engellemeli veya mimariyi bu yeni istekler doğrultusunda güncelleyerek, tüm sistemin mimariye uygun işlediğinden emin olmalıdır.

YOL HARİTASI

NEDEN YOL HARİTASI LAZIM ?

Yazılım Mimarisi Stack

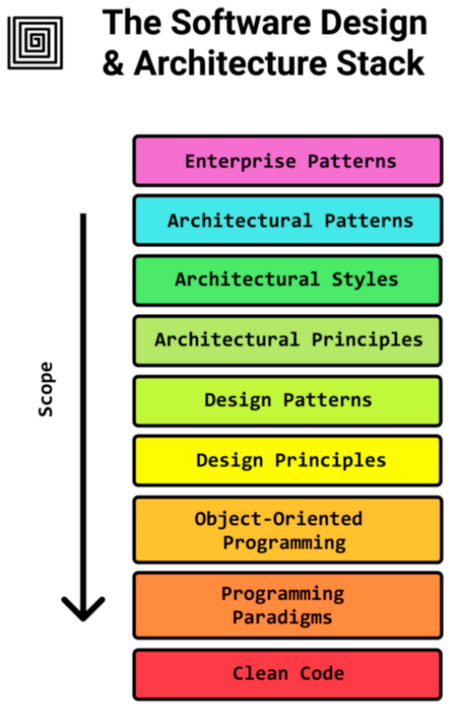


Clean Code Katmanı

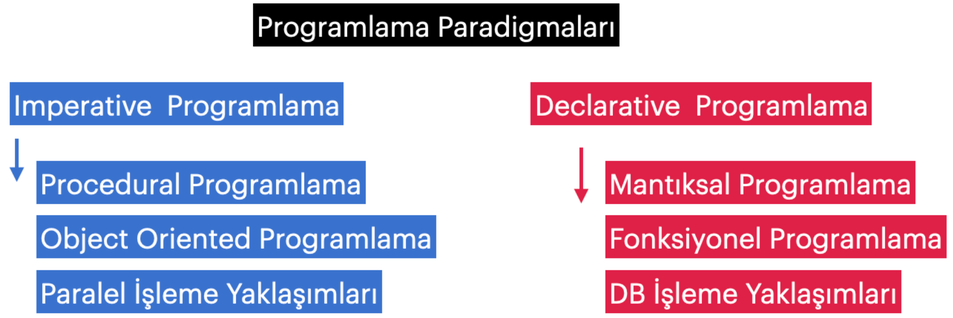
Kodun temiz olması. Kodun temiz olması isimlendirme, kodun yapısı, stili, okunabilirliği, test edilebilirliği gibi en temel yapı taşını en temel kısmı oluşturuyor.

Bu konuda **en iyi kaynak olarak aşağıdaki kitapları** okuyabilirsiniz.

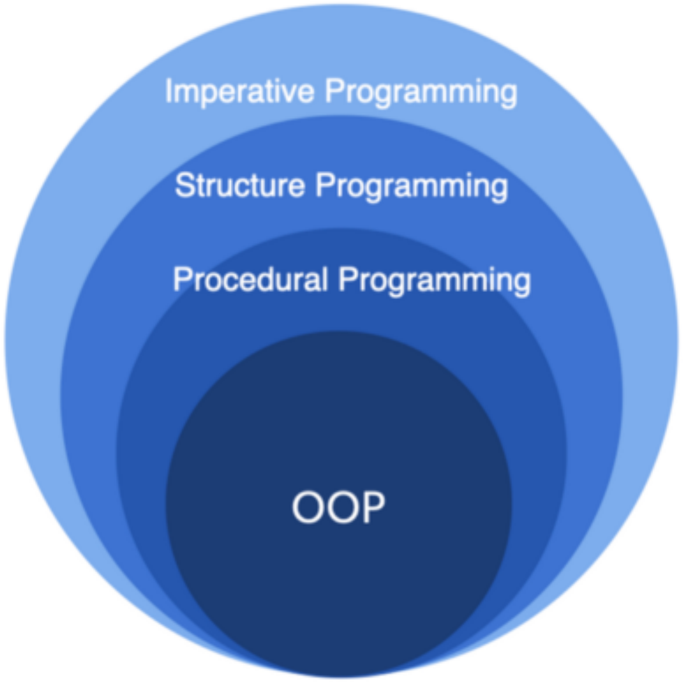
- Clean Code, by Robert C. Martin
- Refactoring, by Martin Fowler (2nd edition)
- The Pragmatic Programmer, by Andy Hunt and Dave Thomas
- The Design of Everyday Things, by Don Norman
- Code Complete, Steve McConnell

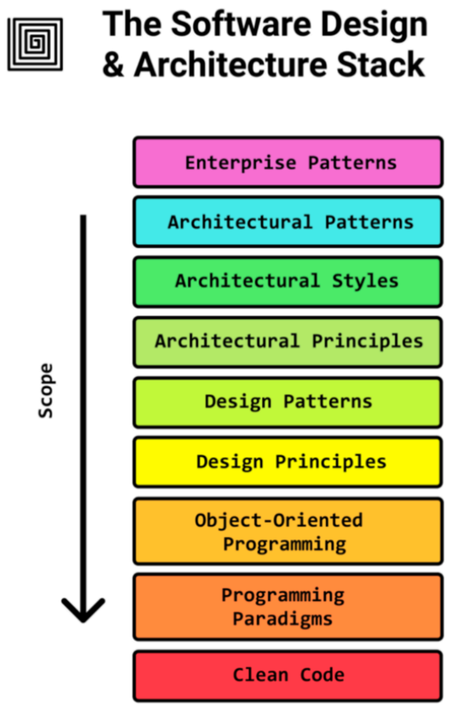


Programlama Paradigmaları Katmanı



Programlama Paradigmalarını öğrenmek için Imperative/Declarative Programlama konusunu anlıyor olmamız lazım.





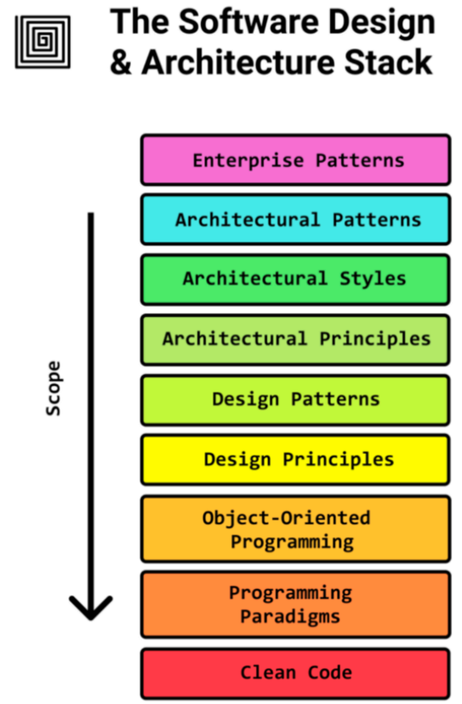
Object-Oriented Programlama Katmanı

Object-Oriented Programlama katmanı Inheritance, Polymorphism, Encapsulation ve Abstraction ve Messaging konularını içeriyor.

Bir başka deyişle Alan Kay göre Object-Oriented Programlamanın temel bileşenleri;

- Message Passing (Mesaj ile Veri aktarımı)
- Encapsulation
- Dynamic Binding

Özetle sektörün üzerinde çok durduğu ve ilk öğretilmeye başlatılan ***inheritance, subclass, polymorphism*** Alan Kay'a göre OOP temel bileşenleri değil.



Object-Oriented Programlama Katmanı - Devam

Bu kısımda farklı Inheritance yapıları bulunuyor. Ben yazılarımda JavaScript kapsamında prototype üzerinden kalıtım'dan bahsettim.

JS'te Kalıtım(Inheritance) Nasıl Gerçekleşir ?

JavaScript' te prototype, __proto__ Kavramlarını Anlamak

OOP inheritance ve polymorphism için Java ve C# dilleri nasıl ele aldığı anlaşılması için TypeScript üzerinden yazılar yazmıştım. Bu yazılarda Class,Inheritance, Interface, Abstract, ConcreteClass, Generics, Reflection, Annotations gibi konulara değinmiştim.

Java/C# Geliştiricilerine TypeScript-1

Java/C# Geliştiricilerine TypeScript-2

Object-Oriented Programlama Katmanı - Devam

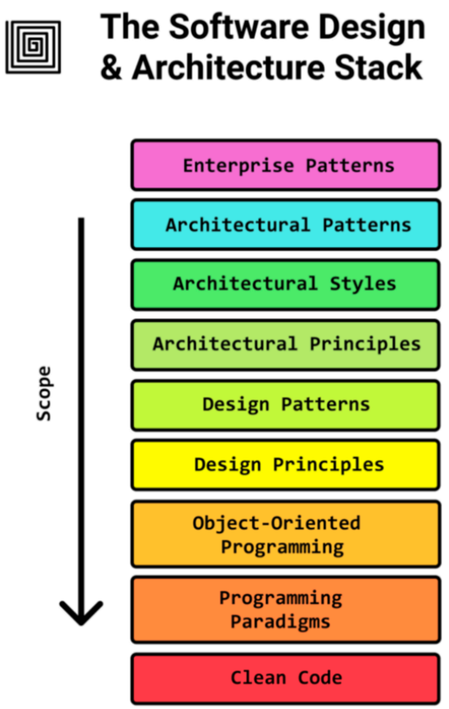
Object Oriented Tasarımın bir üst seviyesi daha model ve nesneler üzeri anlamlı grup kapsamında tanımlamalar ile geliştirme yapmak. Model Driven Development geniş bir konu ama bana Ruby on Rails, Blitz.JS entity'ler üzerinden fullstack geliştirmeyi hatırlatıyor.

BlitzJS ve NextJS

Tasarım Prensipleri Katmanı

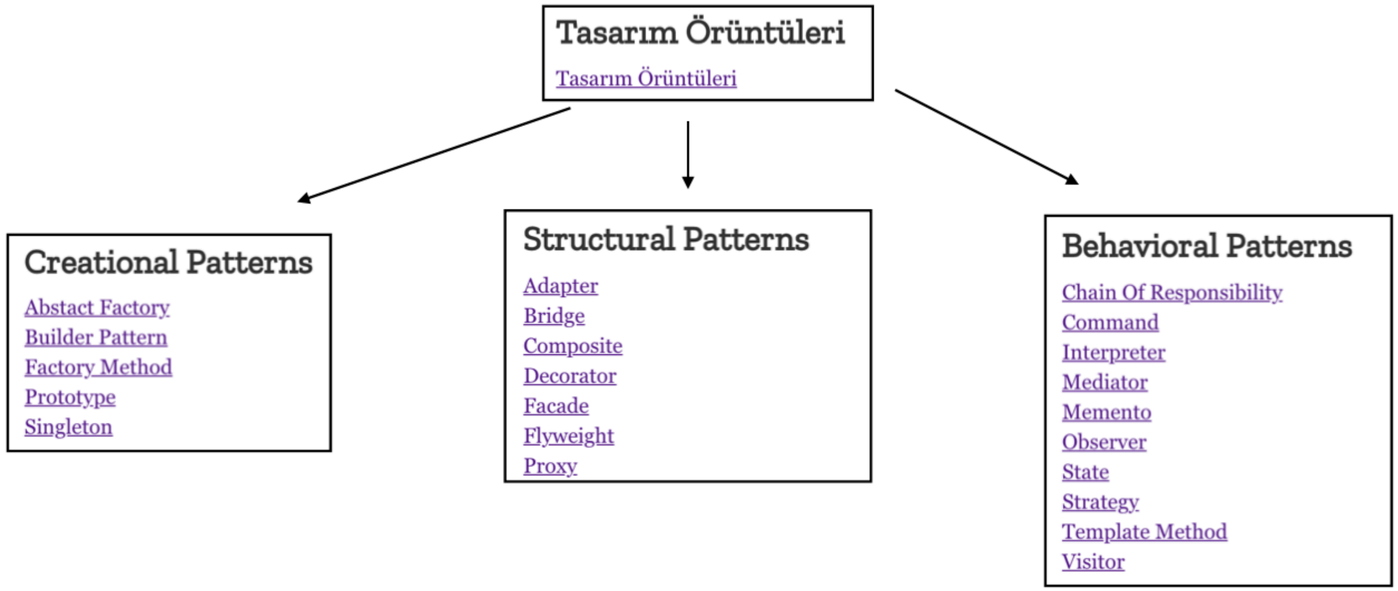
Kodu test edilebilir ve geliştirme isteklerine için esnek tutmak istiyorsanız. Ne tarz yazılım prensiplere dikkat etmeniz gerekir konusunda aşağıdaki konulara değiniyor olacağız.

Composition over Inheritance (Bunu yeni yazıyorum onun için linki henüz yok)
Yazılım Geliştirme Prensipleri
S.O.L.I.D Prensipleri
TDD(Test) vs BDD(Behaviour)
Reactive Manifesto
12 Factor App Uygun Geliştirme Nedir ?



Tasarım Örüntü Katmanı

Yazılım geliştirme sırasında uygulamanın farklı kısımlarında her geliştiricinin ortak karşılaştığı **problem-çözüm ilişkileri** bulunur. İşte tasarım örüntüleri bu problem çözüm ilişkilerini inceler.



Mimari Prensipler Katmanı

Bileşenlerin birbirleri ile olan **iletişim kuralları, birbirlerine olan bağımlılıklarının yönetimi , sınırlar , coupling ve cohesion** gibi kavramların ele alınması.

OOP Unutulmuş Tarihi-3

Mimari Stiller Katmanı

Mimari stiller konusunda aşağıdaki konulara değineceğiz.

- Bileşen Odaklı UI(Kullanıcı Arayüzü) Geliştirme
- Microservices Mimarisi
- Microfrontends Mimarisi
- SOA Mimarisi
- Serverless Architecture (FaaS)
- Yazılım Mimarisinde Brokerlik
- Event-Driven Architecture
- Blackboard System
- Pipe and filter architecture
- Katmanlı Uygulama Mimarisi

Mimari Örüntüler Katmanı

Mimari örüntüler konusunda aşağıdaki konuları analiz edeceğiz.

- Flux Patten (Redux, vb…)
- Saga Pattern
- MVC (Model View Controller)

Enterprise(Kurumsal) Örüntüler Katmanı

Bu konuda Entegrasyon Örüntüler konusundan bahsedeceğim.

Bunun konuda aşağıdaki kaynaklardan faydalanabilirsiniz.

- Patterns of Enterprise Application Architecture, by Martin Fowler
- Enterprise Integration Patterns, by Gregor Hohpe
- Domain Driven Design, by Eric Evans
- Implementing Domain-Driven Design, by Vaughn Vernon

EĞİTİM YOL HARİTASI KAPSAMINDA 2 GÜN NASIL İLERLEYECEĞİZ ?

1nci Gün (Architecture Level)

Conway Yasası
MinViable Product
Jobs To Be Done

Teknolojik Gelişmenin Yazılım Mimarisine Etkisi
Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı

UML
Sketching
MindMap

CDD
DDD
FDD
BDD

Layered Arch
SOA
Microservice
Microfrontends
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture

MVC
Flux Pattern
Saga Pattern
Integration Pattern

2nci Gün (Code Level)

Kod Kalitesini Neler Kötü Etkiler
Clean Code

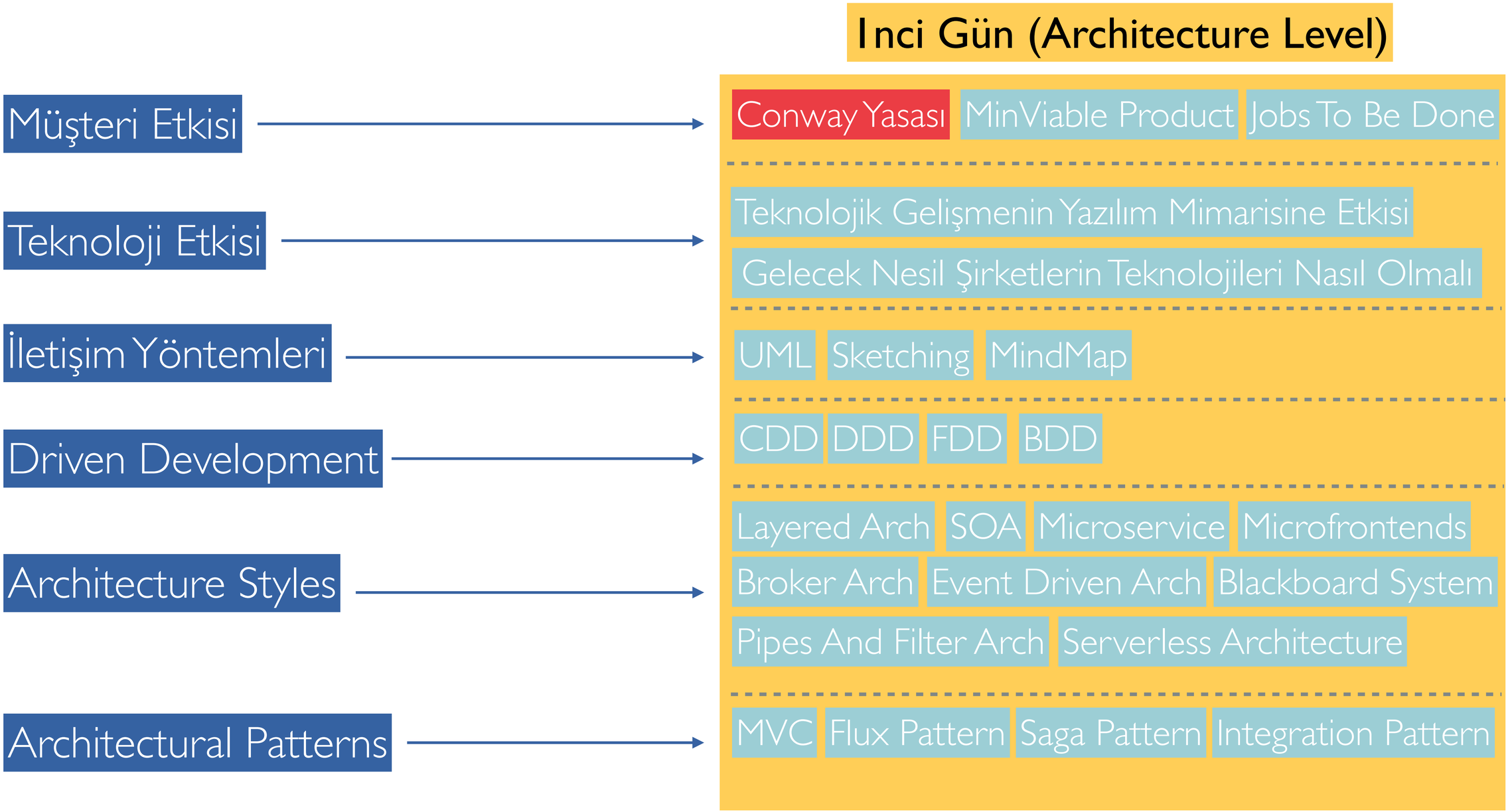
Teknik Borç
Refactoring
Architecture Clearance

AOP
TDD
SOLID
KISS
DRY
YAGNI

12 Factor App
Reactive Programming
Chaos Eng

Design Patterns
Creational Patterns
Structural Patterns
Behavioral Patterns

Anti Patterns
Spaghetti Code
Lava Flow Programming
Golden Hammer
Reinvent The Wheel


Inci Gün (Architecture Level)
Müşteri Etkisi
Conway Yasası
MinViable Product
Jobs To Be Done
Teknoloji Etkisi
Teknolojik Gelişmenin Yazılım Mimarisine Etkisi
Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı
İletişim Yöntemleri
UML
Sketching
MindMap
Driven Development
CDD
DDD
FDD
BDD
Architecture Styles
Layered Arch
SOA
Microservice
Microfrontends
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture
Architectural Patterns
MVC
Flux Pattern
Saga Pattern
Integration Pattern

Conway Yasası
Organizasyon Yapısı, Başarılı Bir Yazılım Mimarisi için Önemli mi ?

Melvin E. Conway 1967 yıllarında şu konuyu gözlemlemiştir; kuruluşlar ve işletmelerin kendi iletişim sistemlerini yansıtan yazılımlar geliştiriyorlar. Buradan da aşağıdaki Conway Yasası ortaya çıkmıştır.

> Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.[2][3]
> — Melvin E. Conway

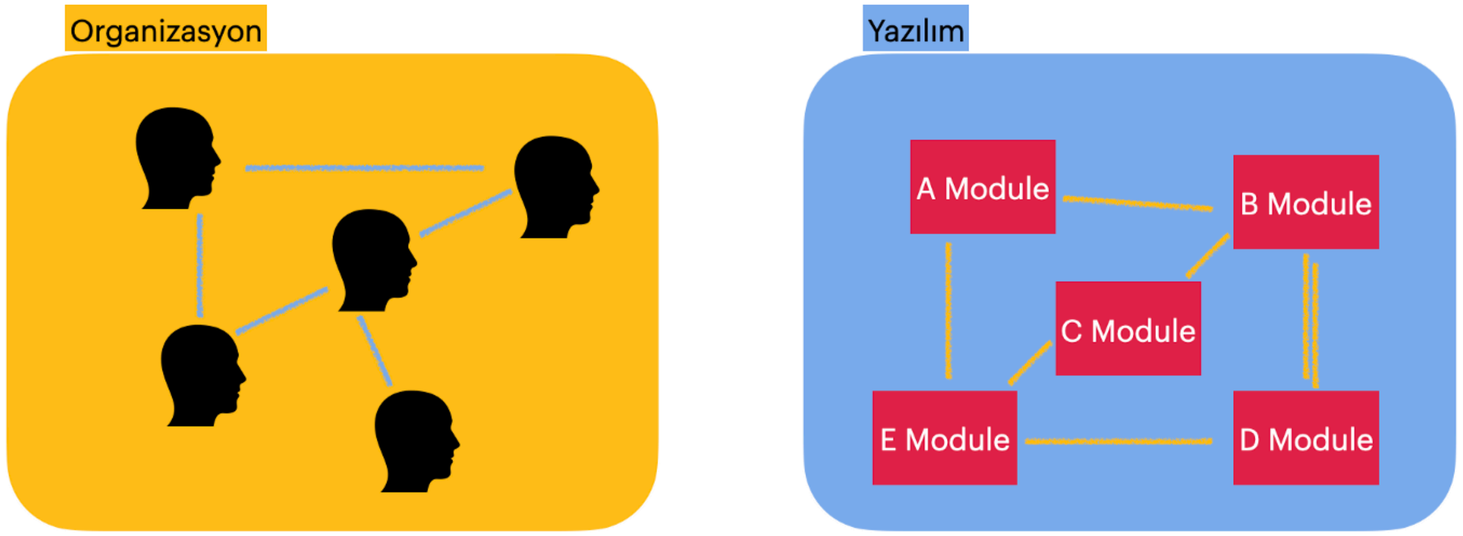
Bu yasa temelinde yazılım modülleri arasındaki ilişkiye benzer bir yapının, organizasyon yapısında olduğu ve buna parallellik gösterdiği üzerinedir. Sistem yapısındaki yazılım arayüzleri, organizasyonlar da bulunan sosyal sınırlar ile benzer şekilde çalışır ve iletişim yöntemleri içerir. Conway sosyolojik gözlemle bu yasayı ortaya koymuş. 1968 National Symposium bu Conway Yasası olarak adlandırılmaya başlanmıştır.

Bu Konu Neden Çok Önemli ?

Uzunca bir süre Askeri, Telekom, Devlet Kurumları, Banka/Finans Şirketleri vb için yazılım geliştirme ekiplerinde çalıştım. Ekip yapıları bu organizasyon yapısına göre kurgulandığında, veya bu birimlerle çok iç içe tasarlandığında daha başarılı yazılımların çıktığını farkettim.
Bu başarının altında bence 2 neden yatıyor.

- Birincisi, geliştirici ekipler organizasyon **yapısı içerisinde doğal olarak dağılıyor** ve bu şekilde modülleri ve modül arayüzleri Conway yasasının doğallığı içerisinde oluşturmuş oluyorlar.

- İkincisi, geliştirici ekip organizasyondaki domain(alan) uzmanları ile birlikte çalıştıklarında gereksinimleri **daha iyi analiz edip ihtiyaçları yerinde tespitten** kaynaklanıyor.



Not: Burada başarılıdan kastettiğim **müşterinin isteklerine uyumluluk**. Yoksa bu yazılım tekrar tekrar kullanılması anlamında söylemiyorum.

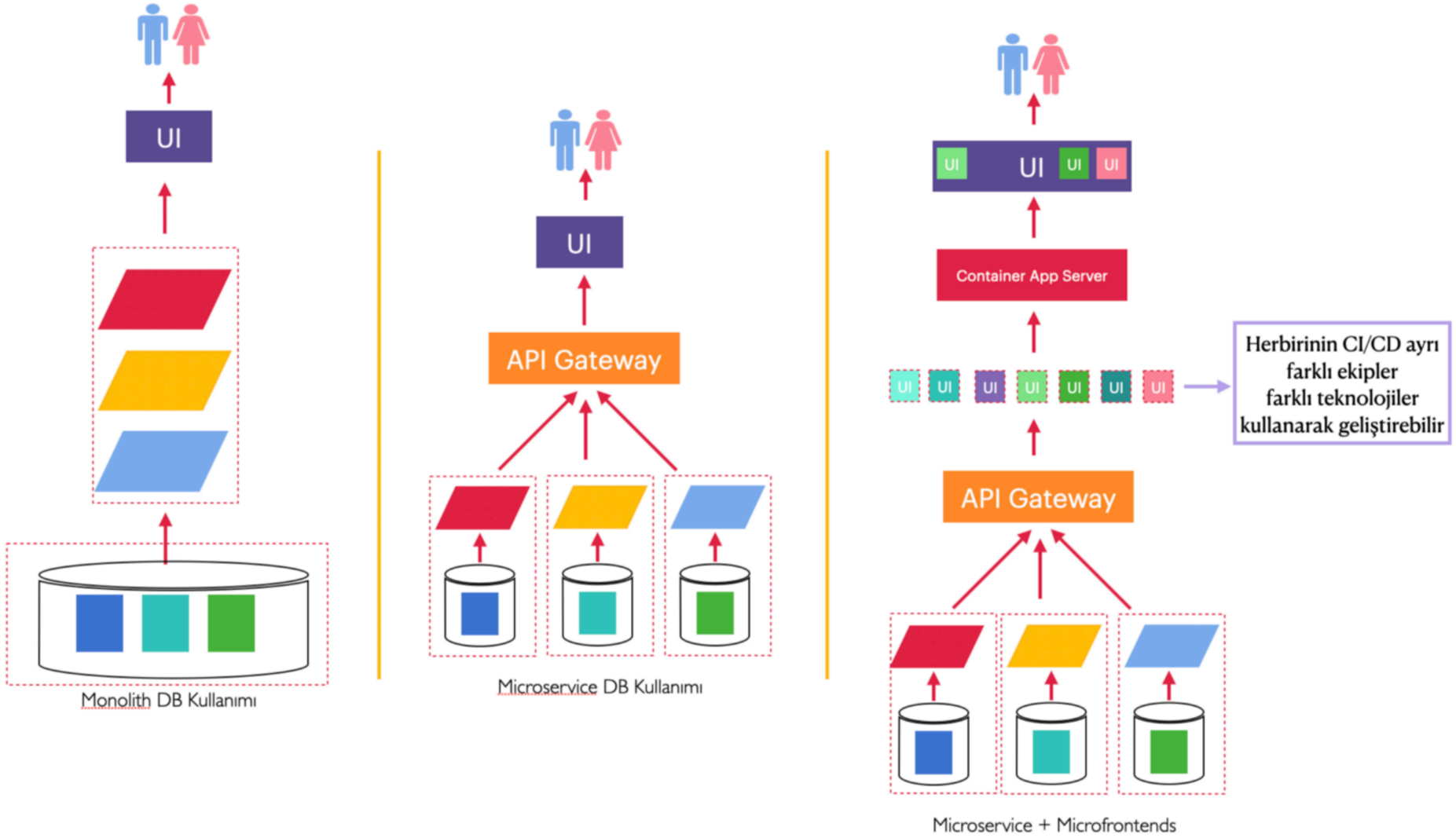
Çünkü iş yaptığınız organizasyona olan bağımlılık giderek artacağı için terzinin bir kişiye yaptığı özellikte bir ürün ortaya çıkacak, **bu ürün o kişi/organizasyon için çok uygun olurken**, başka müşterilere veya organizasyonlara kolay bir şekilde uyarlanamaz halde olacaktır.

Yeni Mimari Yaklaşımların Organizasyon Yapısı ile (Micro-service ve Micro-Frontends) Bir İlgisi Var mı ?

Evet var. **Monalith** yapıdan **Microservis** yapısına, hatta **Microfrontend** yapısına geçişte, yazılımdaki modülerlik direk olarak ekipleri ve bu ekipler arası iletişimide etkilemektedir. Geliştirici ekiplerin micro seviyede yazılımlar ile eşleştirilebildikleri için daha uyumlu ve başarılı yazılımlar ortaya çıkmaktadır. Örneğin bu sayede aşağıdaki bölümleme çok daha basit bir şekilde yapılabilmektedir.

Fatura Servisleri,
Reklam ve Kampanya Servisleri
Ürün Servisi
Satış Servisi
Müşteri Hizmetleri
CRM vb…

Not: Bu konuda yazılım geliştirme yöntem ve araçlarının giderek **Conway yasasına uygun olarak micro yapılara dönüştüğünü** yukarıdaki resimden de görebilirsiniz.



Büyük Yazılım Firmaları için Durum Nasıl ?

Not: Organizasyon yapılarındaki parçalanma ve iletişimin şirketlerin yazılımlarına direk yansıdığını görebilirsiniz.

Apple örneğin ortadaki kırmızı (Steve Jobs ve Tasarım Ekibi) tüm ürünler bu yapının çevresinde şekillenmiş bundan dolayı bir uyum söz konusu

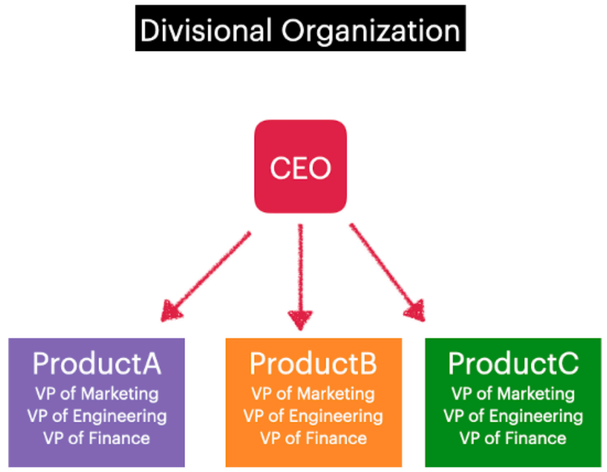
Amazon da ise Scalebility (Ölçeklenebilirlik) için ekipler belli limitleri geçmeyen hiyerarşik yapıda birbirlerine bağlı

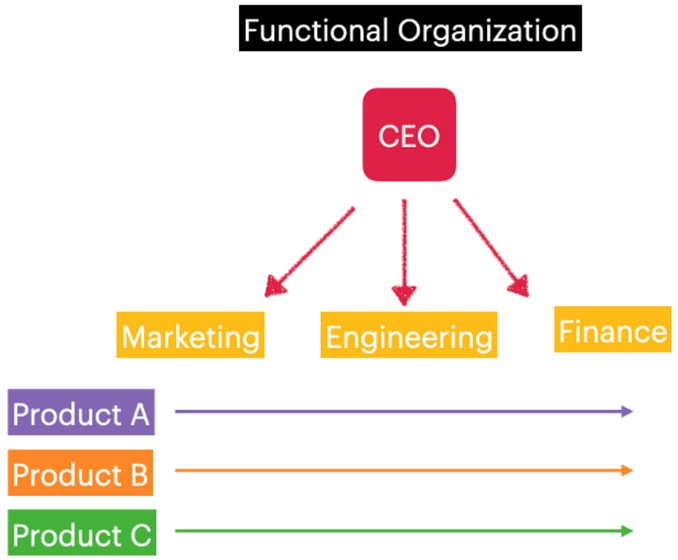
Yukarıdaki resimden **Google, Microsoft, Facebook Oracle** ve diğer şirketlerin durumunu çıkarabilirsiniz.

Organizasyon Yapıları

Divisional Organization yapısında ürünler, onun altında kendi VP of Marketting, Engineering ve Finance olarak dallara ayrılırken bazı insan kaynakları, kanun vb.. konular direk CEO bağlı ekipler tarafından yapılabilir.

Functional Organization yapısında ise yeteneklerin tüm ürünler için cross-cut matrix şeklinde kullanılması anlamına geliyor ki buda CEO ve altındaki yönetim ekibin tüm ürünlerin yönetimi ve koordinasyonu konusundaki önemini gösteriyor.





Amazon 2 PT (Two Pizza Teams)

Organizasyon yapısı ile yazılım arasındaki konuya belkide en iyi örnek Amazon'un yazılım ekiplerinde uyguladığı **(Two Pizza Teams) tekniğine** göre ekipleri oluşturma konseptinden geçiyor.

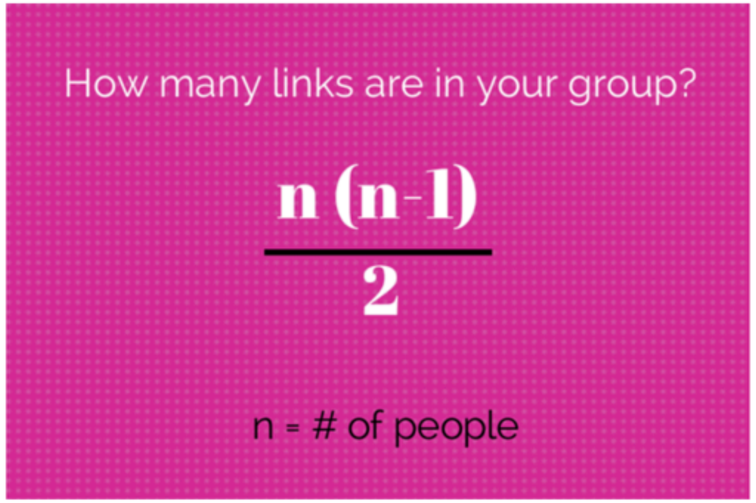
Amazon bildiğiniz gibi eTicaret ve Bulut Teknolojileri konusunda dünya lideri konumunda. Bunu gerçekleştirmek içinde sürekli innovasyon yaparak yeni teknolojiler üretmeli ve bunu müşterilerine servis olarak vermeli.

Amazon Ekip Yapıları

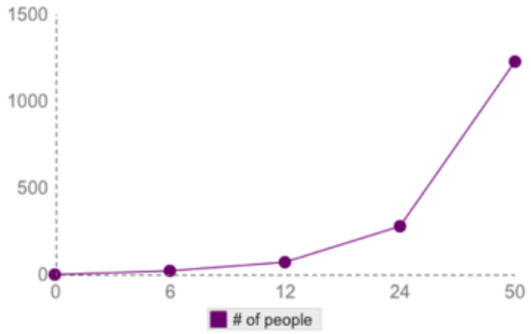
Amazon bu ölçeklenme problemini bir anda teknolojik olarak gerçekleştirmedi. İlk önce organizasyon yapısını , takımların büyüklüğünü ve birbirleri arasındaki iletişimin kurallarını belirledi. Daha sonrasında bu kaliteli servisler ortaya çıktı. Bu yazıda Amazon'un bu stratejilerinden bahsetmek istiyorum.

Takımların **büyüklüğü 2 pizzanın besleyeceğinden büyük olmamalı**. Bunun arkasında yatan felsefe çok fazla iletişim, iletişim problemlerini çözmez. Örneğin bir yemeğe, düğüne, doğum günü partisine gittiniz. Genelde çok fazla insan olsada insanların 5erli, 6şarlı gruplar halinde toplanıp konuştuğunu görürsünüz. Buda iletişim açısından takım sınırlarımızı göstermektedir.

Amazon Ekip Yapılarını Bu Şekilde Şekillendirme Nedenleri



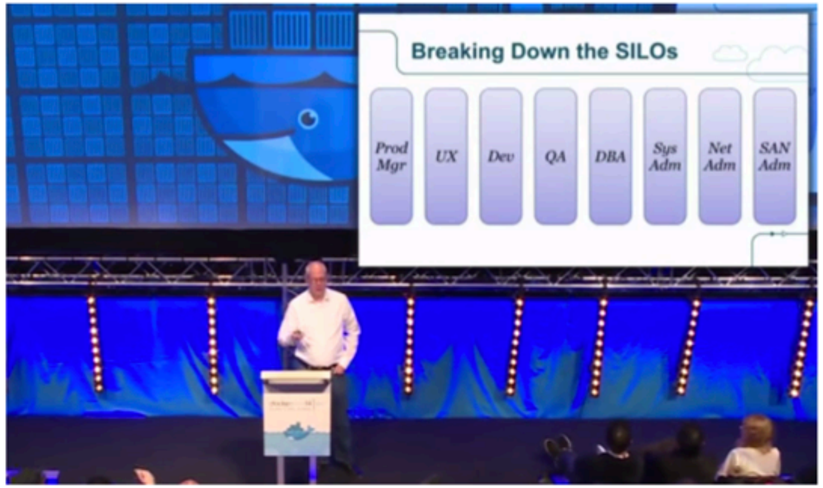
Bu karmaşıklığı aşağıdaki grafikte daha net olarak görebiliriz.



Peki 2 PT Kaç Kişiden oluşmalı ve Nasıl Bir Rol Dağılımı Olmalı

- 1 Takım Lideri
- 1 Tasarımcı/UX uzmanı
- 1 veya 2 UI Geliştiricisi
- 1 veya 2 Backend Geliştiricisi
- 1 QA uzmanı
- 1 Ürün sahibi(product owner) /iş analisti

Yukarıdaki bölümleme bizim eskiden beri gelen klasik IT bölümlememizin dışında olan bir yönelim. Ama bu yönelimler DevOps'un giderek gelişmesi, ekip elemanlarının farklı sorumlulukları alabilmesi zamanla kırılacak ve aşağıdaki resimden yavaş yavaş 2PT 'ye evrilecektir.



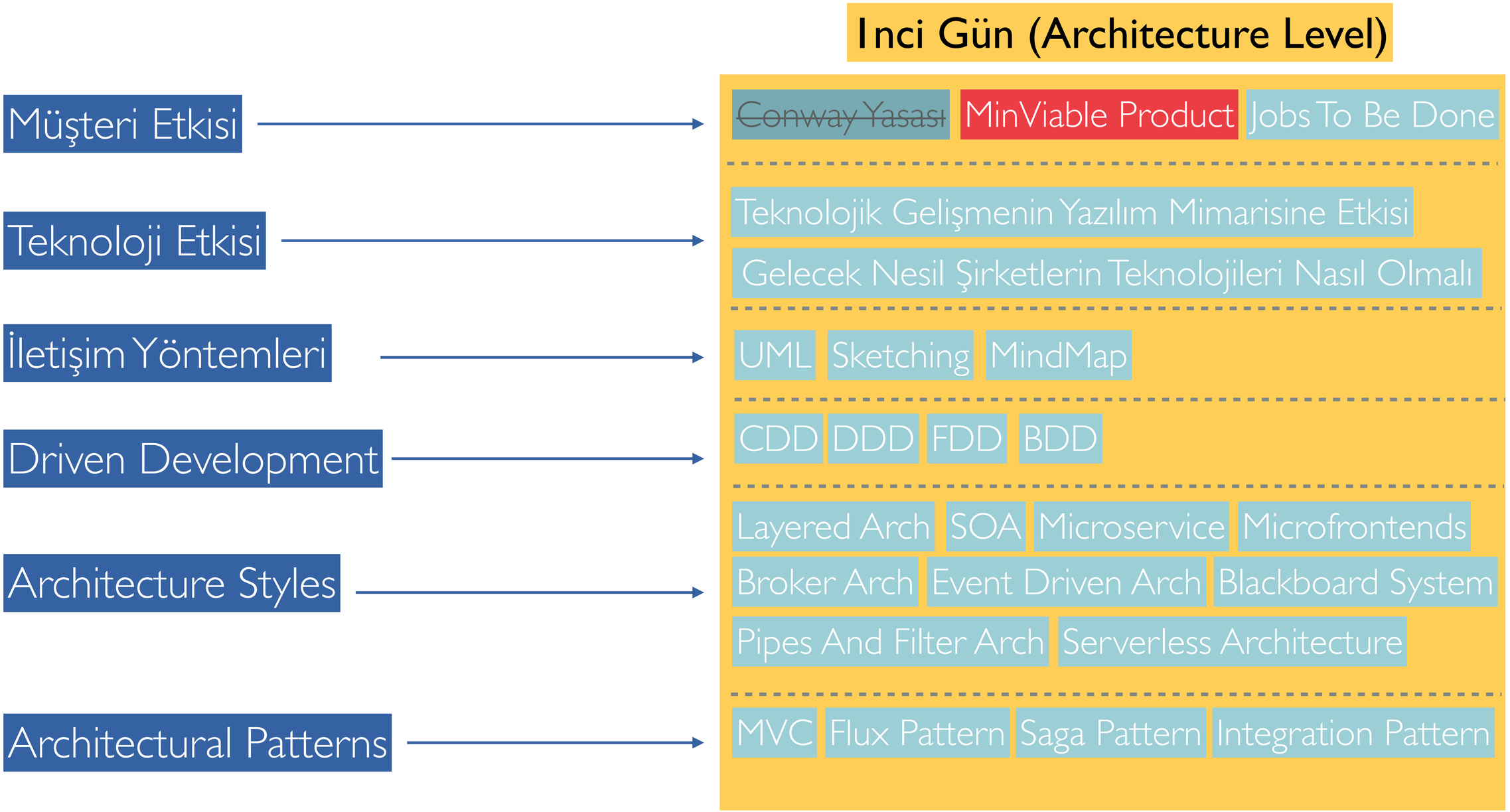
Eski Yönelim

Amazon Ekip Yapıları Arasında İletişim Kuralları

Diğer bir konuda İletişim Yöntemi. Bu kısımda belirledikleri bir takım iletişim kurallarının Amazon ne kadar fayda sağladığını sizde mevcut AWS (Amazon Web Servislerinin) oluşmasında bu kuralların ne kadar etkili olduğunu görebilirsiniz.

Bu kurallar ;

- Tüm takımlar verilerini ve fonksiyonalitelerini birbirlerini servis arayüzleri ile sağlayacaktır.
- Takımlar sadece bu servis arayüzleri üzerinden birbirleri ile iletişim kurmalıdır.
- 2nci madde dışında hiç bir iletişim şekli olmamalıdır. Ne başka bir takım diğer takımın veri modelini okuyabilmeli, ne ortak bir bellek modeli kullanılmalı. Tüm bunlar yasak. Tek iletişim birbirlerine sağlanan servisler üzerinden gerçekleştirilmeli.
- Bu iletişim sırasında hangi teknolojinin kullanıldığının bir önemi yok. HTTP, Corba, Pubsub, özelleşmiş bir çok protokol kullanılabilir.
- İstinasız tüm servisler dışarıya açılacak şekilde tasarlanmalı ve geliştirilmelidir.


Inci Gün (Architecture Level)
Müşteri Etkisi
Teknoloji Etkisi
İletişim Yöntemleri
Driven Development
Architecture Styles
Architectural Patterns
~~Conway Yasası~~
MinViable Product
Jobs To Be Done
Teknolojik Gelişmenin Yazılım Mimarisine Etkisi
Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı
UML
Sketching
MindMap
CDD
DDD
FDD
BDD
Layered Arch
SOA
Microservice
Microfrontends
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture
MVC
Flux Pattern
Saga Pattern
Integration Pattern

Min Viable Product

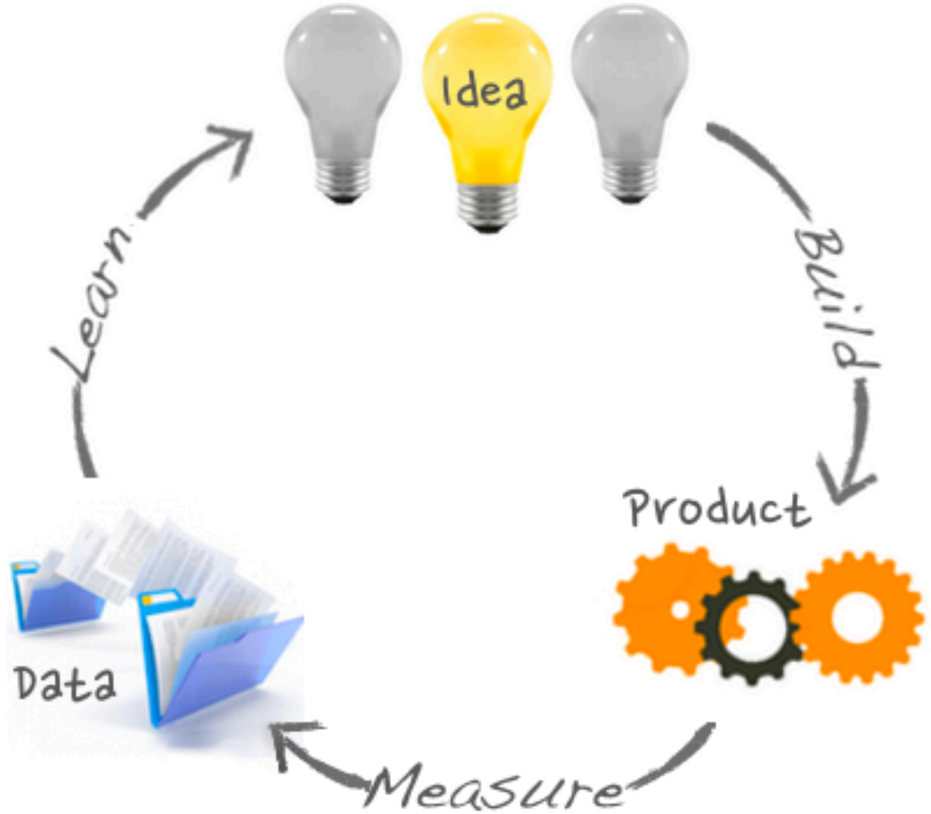
Startup’lar için geçerli bir kavram. Aklınızda bir fikir var ve bu fikri ürüne dönüştürmek istiyorsunuz. Bunun için çok karmaşık düşünerek , bütün özelliklerini tam tamına yapayım şeklinde düşünmek yerine esas müşteri tarafında en değerli yeteneğin hangisi olduğuna odaklanarak ürün geliştirmektir.

Bunu anlamanın yolu’da aşağıdaki ürün geliştirmenizin aşağıdaki şekilde dönmesini sağlamaktır.

- Müşteri için **en değerli yetenekleri** belirlemek
- **Minimum yetenekteki** bu ürünü geliştirmek
- Bu ürünü **müşterilerinizin test etmesini sağlayarak** ürününüzün kullanılıp/kullanılmayacağının test edilmesi
- Elde ettiğiniz **geribildirimlerle bu döngüyü devam ettirmeniz**.

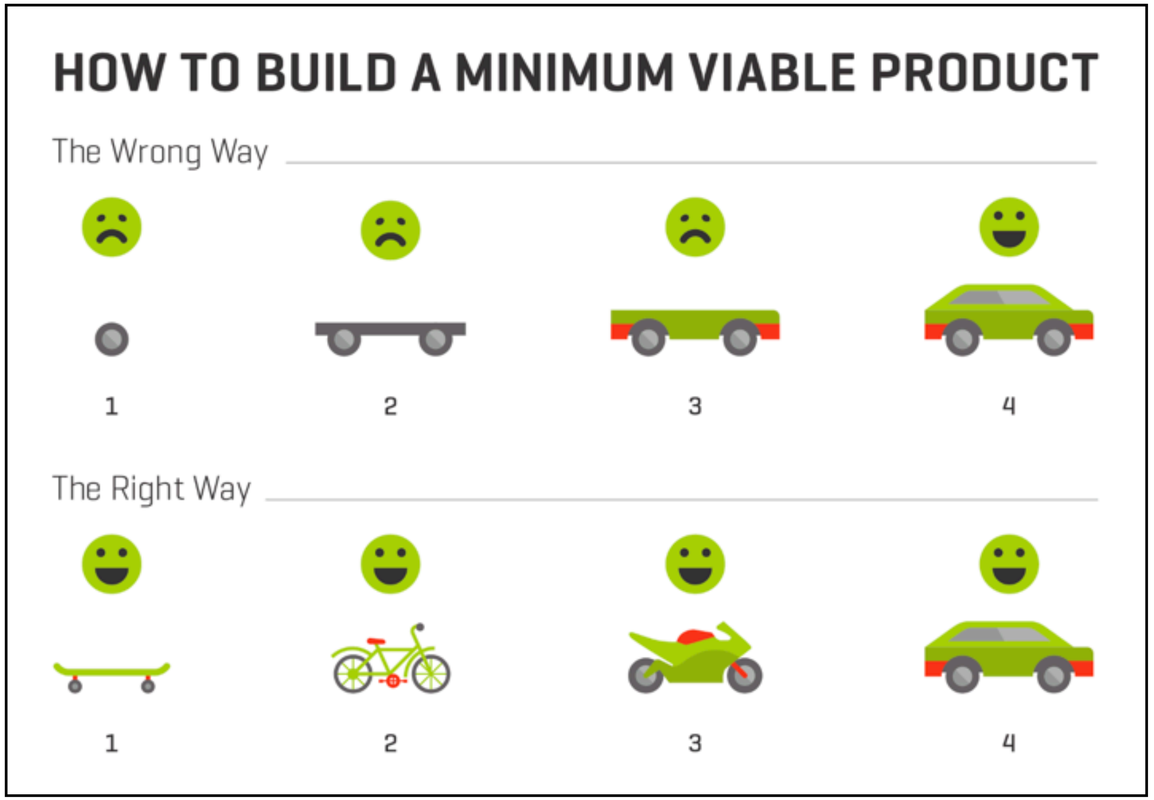
MVP Döngüsü

Minimum
viable
Product



MVP Döngüsünde Ürün Nasıl Gelişir

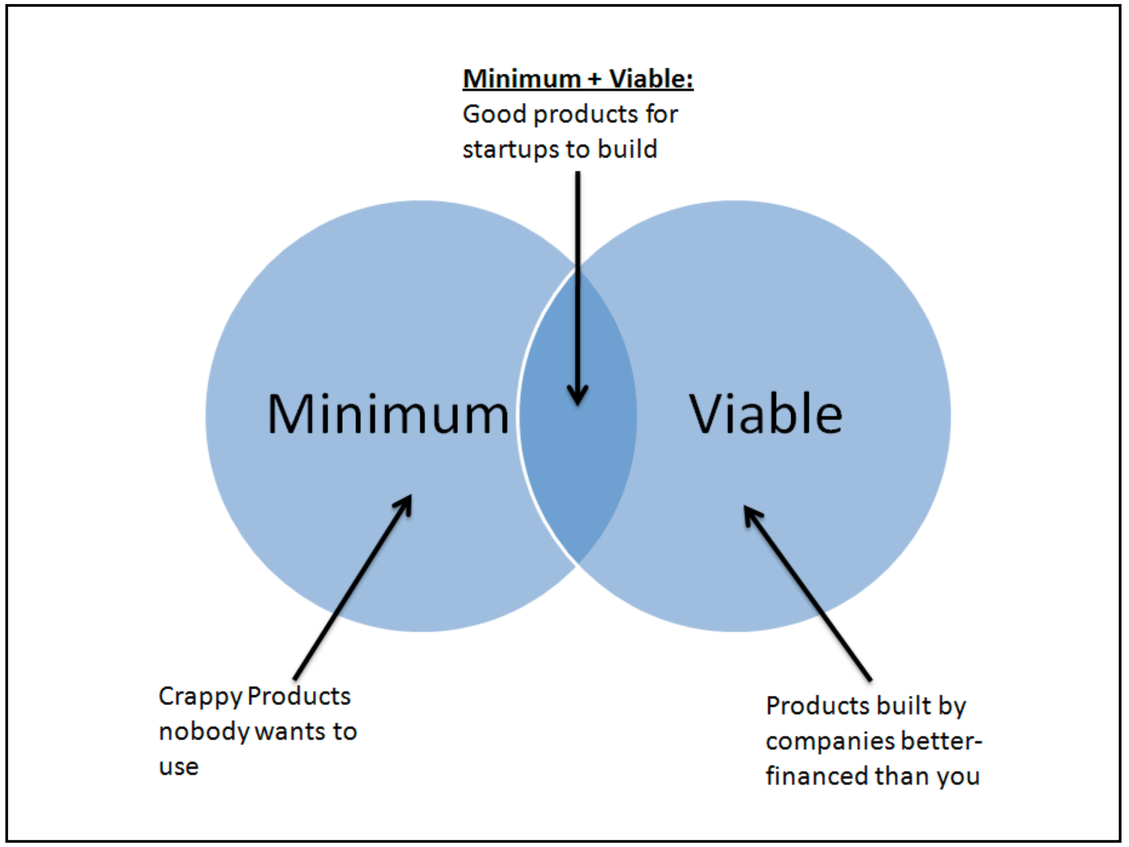
MinViableProduct mantığında **Çevik Geliştirmenin** de de yer aldığını görebilirsiniz. Her zaman müşterinin karşısına çalışan bir ürünle gitmek, müşterinizi tüm geliştirme süresince projenin içerisinde yer almasını sağlatmak…

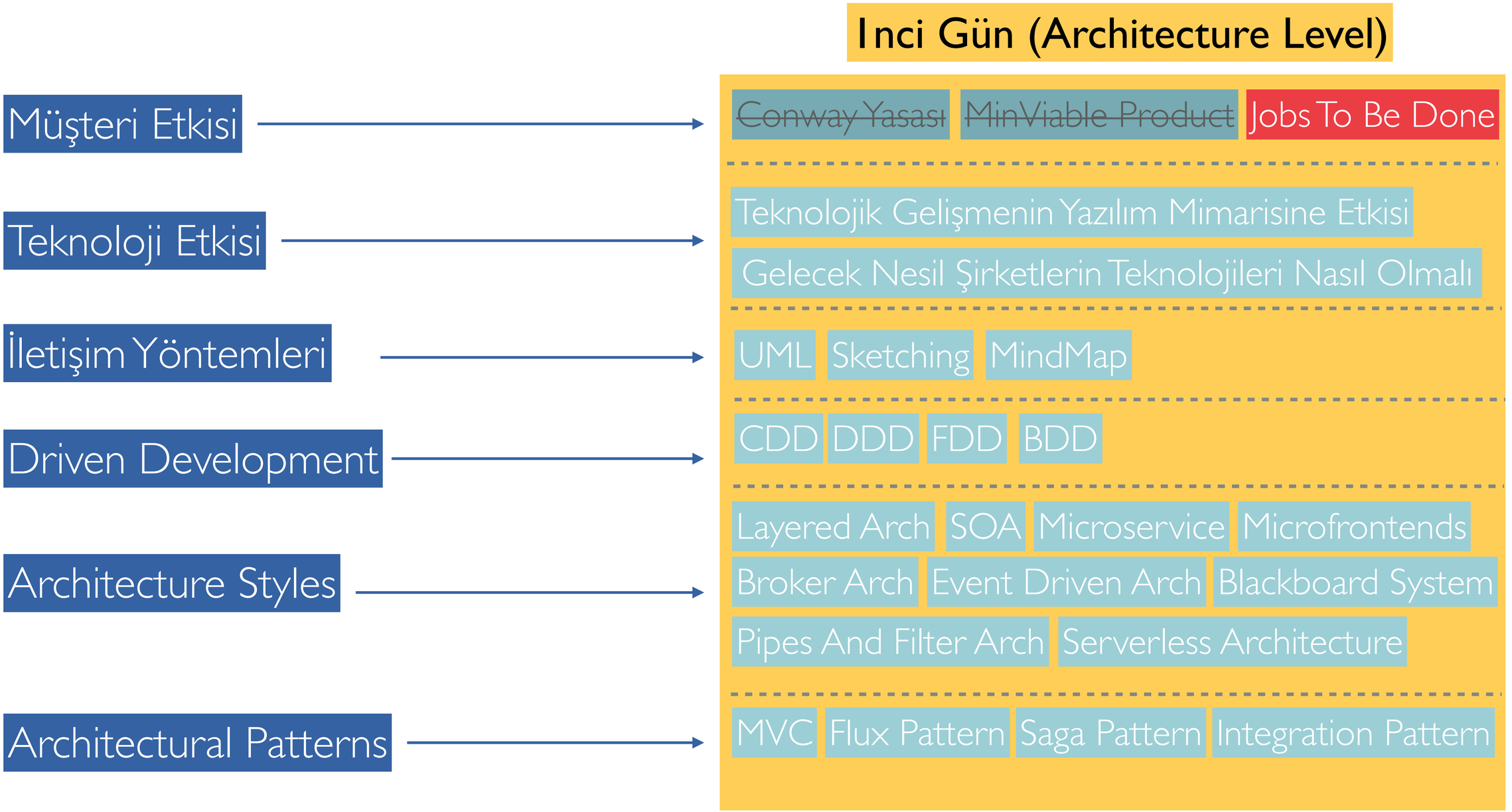


MVP Geliştirmek İstediğimiz Ürün..

Bu konu ile ilgili rastladığım güzel bir resimde Startup için geliştirilmesi anlamlı olan ürünün;

Kullanılacak yetenekler/ürünler ile bu işe sizden daha iyi yatırım yapamayacak şirketlerin ürünlerinin olmadığı kısımda ürünler geliştirmeniz…



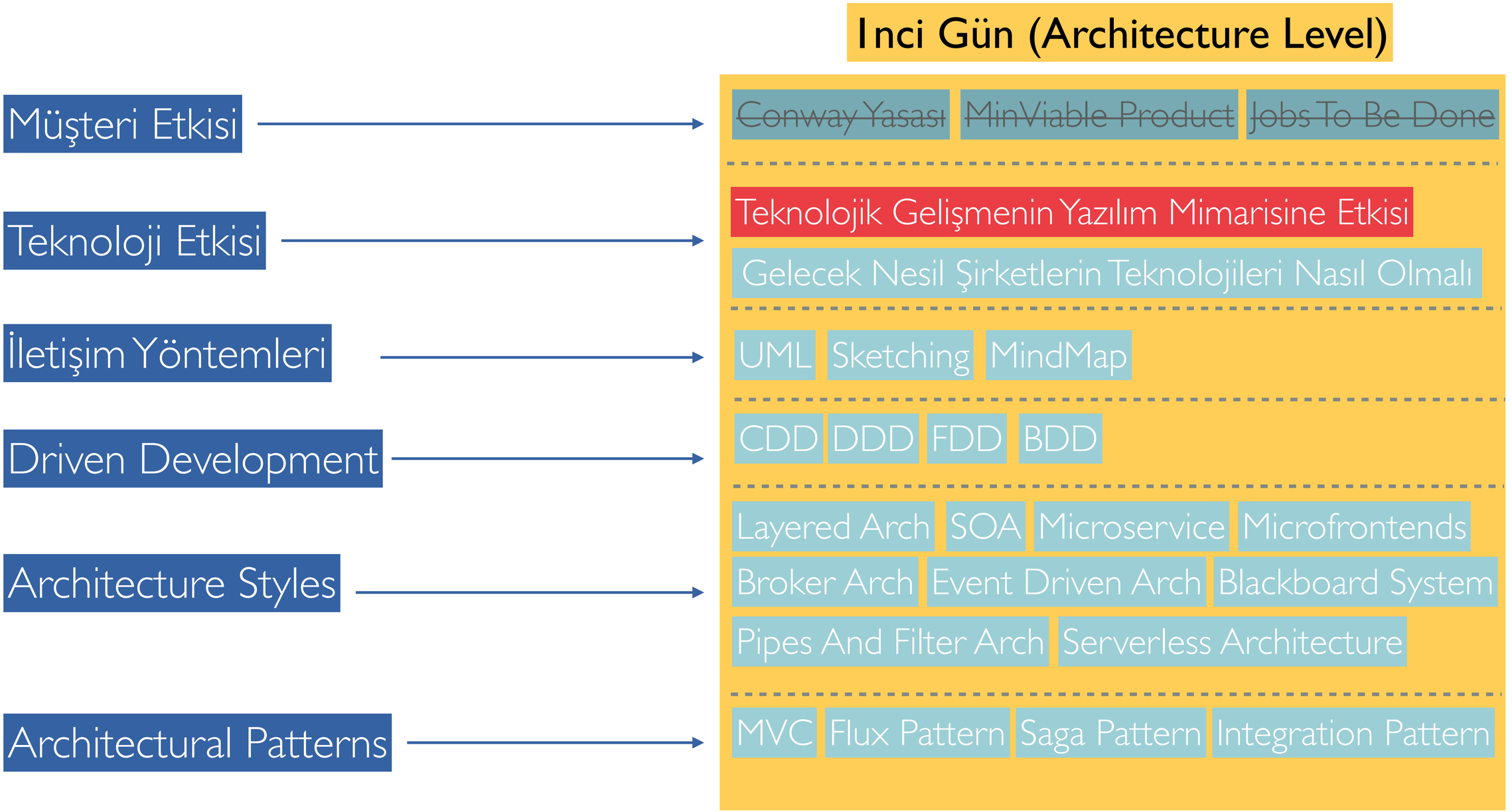

Inci Gün (Architecture Level)
Müşteri Etkisi
Teknoloji Etkisi
İletişim Yöntemleri
Driven Development
Architecture Styles
Architectural Patterns
~~Conway Yasası~~
~~MinViable Product~~
Jobs To Be Done
Teknolojik Gelişmenin Yazılım Mimarisine Etkisi
Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı
UML
Sketching
MindMap
CDD
DDD
FDD
BDD
Layered Arch
SOA
Microservice
Microfrontends
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture
MVC
Flux Pattern
Saga Pattern
Integration Pattern

Jobs To Be Done

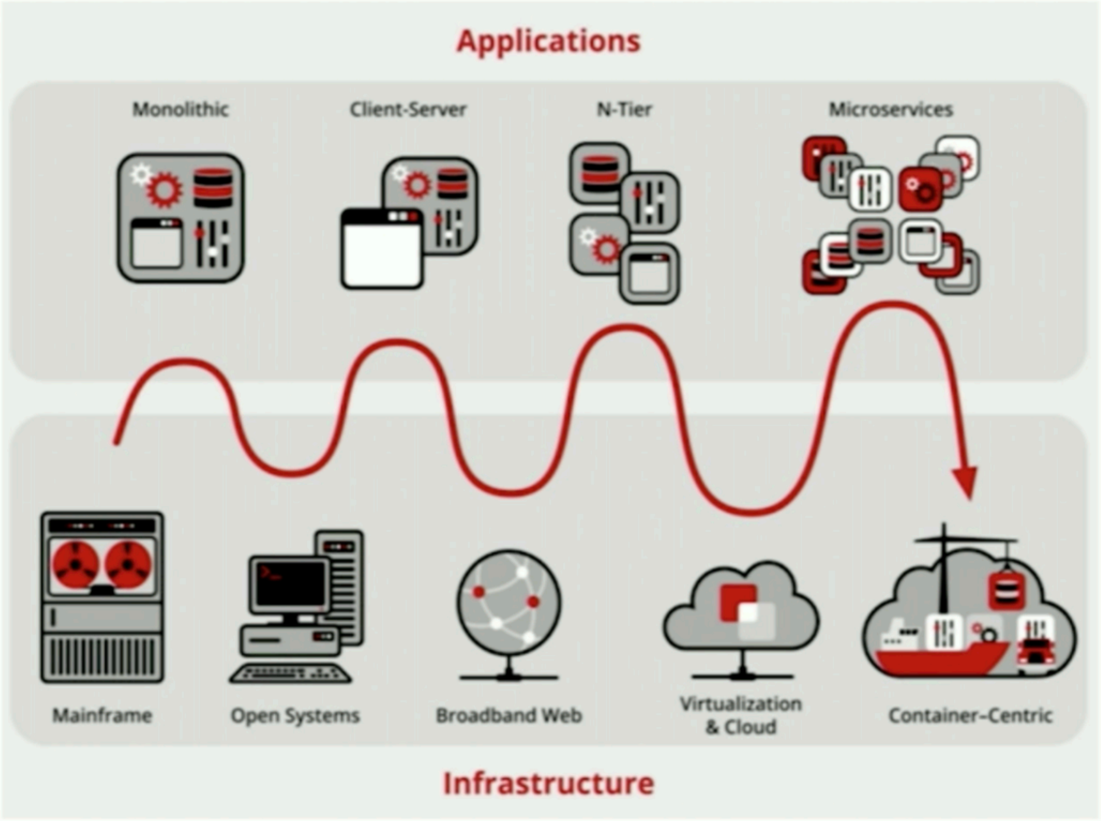
Temel felsefesi müşteriden aldığınız ürünü şöyle yaparsanız, şu özellikleri eklerseniz , fiyatını şu şekilde, limitlerini bu şekilde yaparsanız alırız söylemlerinin gerçeği yansıtmaması, özetle MVP, Müşteri Geliştirme yöntemlerinden elde ettiğiniz verilerin sonucunda geliştirdiğiniz ürün yine satmıyorsa bunun nedeni Müşteriler **belki ne istediklerini veya neye ihtiyaçları olduklarını bilmemelerinden** kaynaklanıyor olabilir

Customers don’t know what they want. (Steve Jobs)

Genelde yukarıda bahsettiğimiz nedenlerden ötürü ürünler başarısız oluyor, **ürün / market uygunluğunu** yakalaması genelde şans faktörüne bağlı oluyor. Jobs To Be Done müşterinin anlatamadığı nedenselliği nasıl bulacağımız konusunda bize bir önermede bulunuyor


Inci Gün (Architecture Level)
Müşteri Etkisi
Teknoloji Etkisi
İletişim Yöntemleri
Driven Development
Architecture Styles
Architectural Patterns
~~Conway Yasası~~
~~MinViable Product~~
~~Jobs To Be Done~~
Teknolojik Gelişmenin Yazılım Mimarisine Etkisi
Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı
UML
Sketching
MindMap
CDD
DDD
FDD
BDD
Layered Arch
SOA
Microservice
Microfrontends
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture
MVC
Flux Pattern
Saga Pattern
Integration Pattern

Teknolojik Gelişmenin Yazılım Mimarisine Etkisi

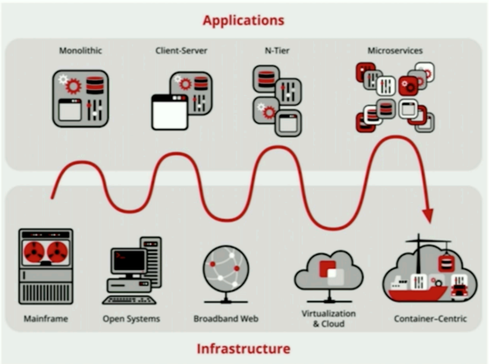


Teknolojik Gelişmenin Yazılım Mimarisine Etkisi

Uygulamada Kod Yapısı: Monolith → Client/Server → N Tier → Microservice → FAAS (Function As A Service)

Infrastructure(Computing): MainFrame → PC → onPrem Sunucu → VirtualOS(EC2) → Paas (ElasticBeanstalk) → Fargate(Docker) → Lambda (Serverless)

Yazılım Geliştirme Süreçlerinde: Waterfall → Agile



Dillerde: Assembly → C/CPP → Java, .NET → Scale, Go, Python, Javascript

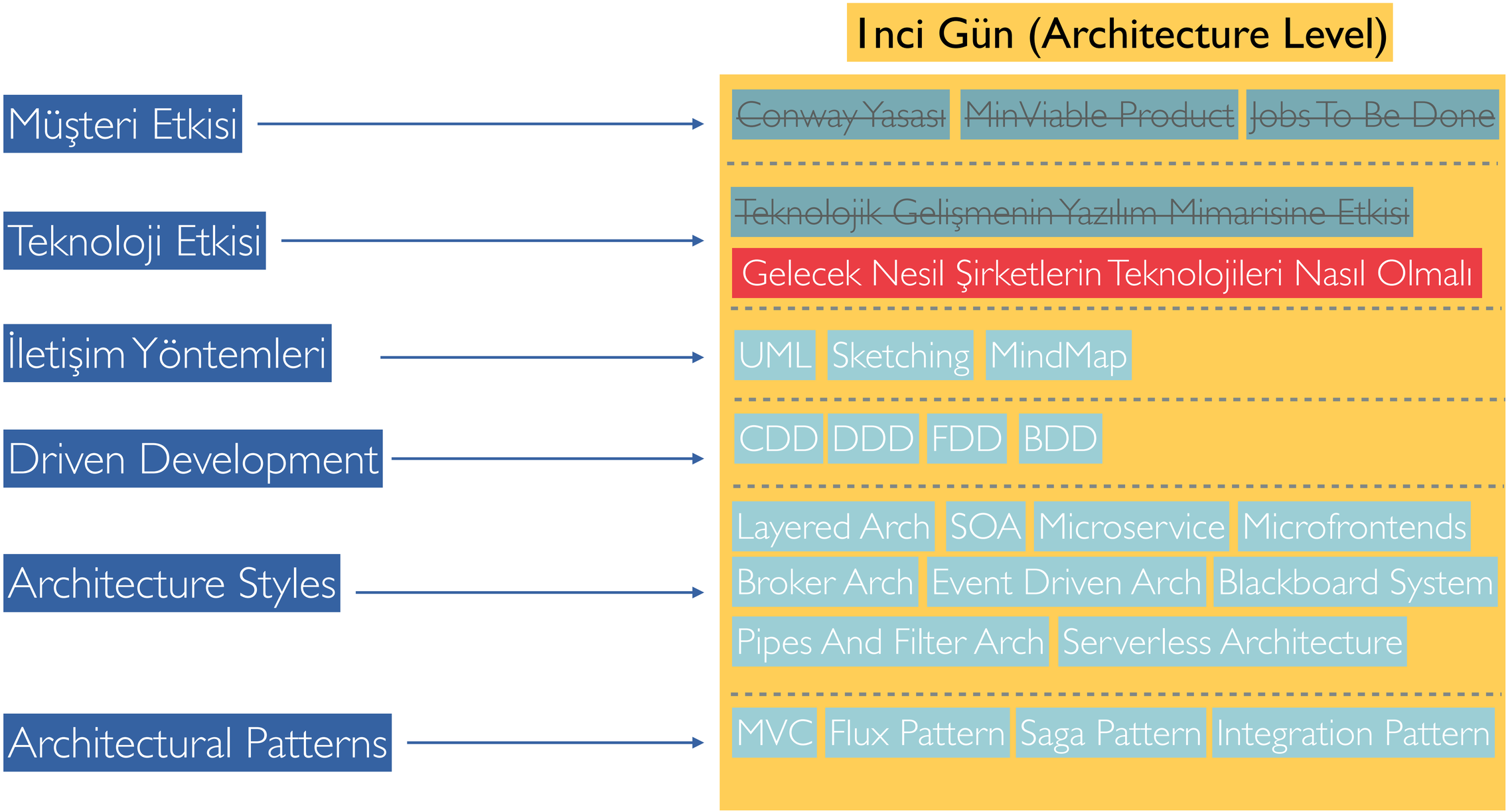
Konseptlerde: Multi Thread , Sync, → Single Thread, Non I/O Block, Async

Ekip Yapıları: Analiz, Tasarım, Geliştirme, Test → Self Managed Team (DevOps) kavramları çıktı

Canlıya Çıkma Süresi : Manual Deployment → CI/CD Otomatikleşmesi, ve sürelerin gün içerisinde yeni versiyonların deploy edilmesi

Infrastructure Kullanabilme: Manual → Infrastructure As A Code

Veri ve İşleme Gücü Üzerine Yeni Uygulama Alanları: Big Data, BlockChain , Machine/Deep Learning, Virtual Reality, Otonom Sistemler, iOT vb


Inci Gün (Architecture Level)
Müşteri Etkisi
Teknoloji Etkisi
İletişim Yöntemleri
Driven Development
Architecture Styles
Architectural Patterns
~~Conway Yasası~~
~~MinViable Product~~
~~Jobs To Be Done~~
~~Teknolojik Gelişmenin Yazılım Mimarisine Etkisi~~
Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı
UML
Sketching
MindMap
CDD
DDD
FDD
BDD
Layered Arch
SOA
Microservice
Microfrontends
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture
MVC
Flux Pattern
Saga Pattern
Integration Pattern

Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı

Pandemi yaşamı çok derinden etkiledi. Şirketler yaşamlarına teknoloji sayesinde sürdürebildi. Bazıları çok büyüdü, bazıları ise yok oldu.

Teknolojinin, şirketler için önemini her şirket sahibi bu dönemde daha çok anladı. Zamanın uzun sürede yaptığı bu ayrışımı, pandemi bir katalizör görevi hızlandırdı. Bu nedenle günüzümüzde ayakta kalmak için Modern Teknolojileri kullanan İşletmeler kurmanız gerekiyor.

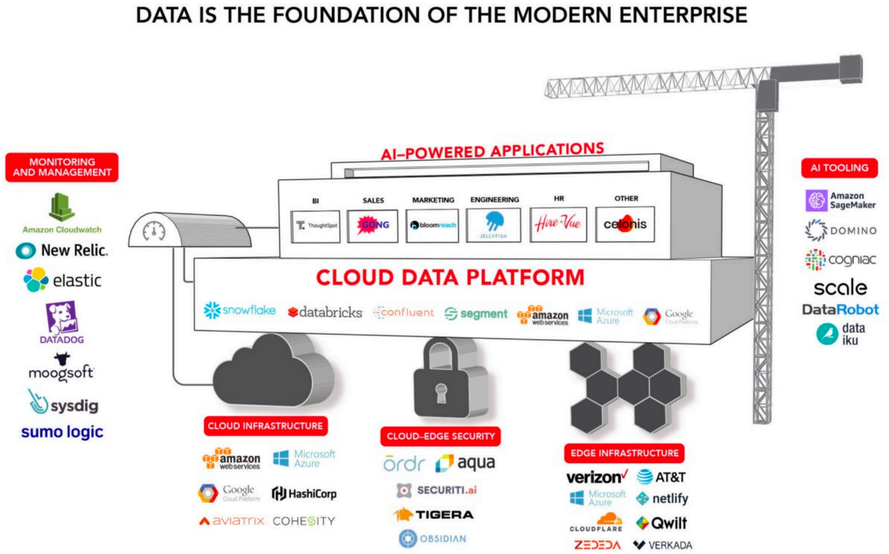
Peki soru şu Geleceğin Modern Enterprise (Kurumsal) nasıl olacak ?

- Veri Üzerine İnşaa Edecek,
- AI (Yapay Zekadan) Faydalanacak,
- Çevik Çalışma Alanlarına Sahip olacak

A. Veri Üzerine Şirketinizi İnşaa Edin.

Kurumsal şirketin departman yapısı genelde aşağıdaki şekilde ayrılır;

- BI(İş Zekası),
- HR(İnsan Kaynakları),
- Sales(Satış),
- Marketing(Marketing),
- Engineering (Mühendislik)



Bu departmanların **AI-Powered (Yapay Zeka Destekli)** uygulamalardan faydalanmaları gerekiyor.

Yapay zeka destekli uygulamalar için **AI-Tooling (Yapay Zeka)** araçlarından faydalanabilirsiniz.

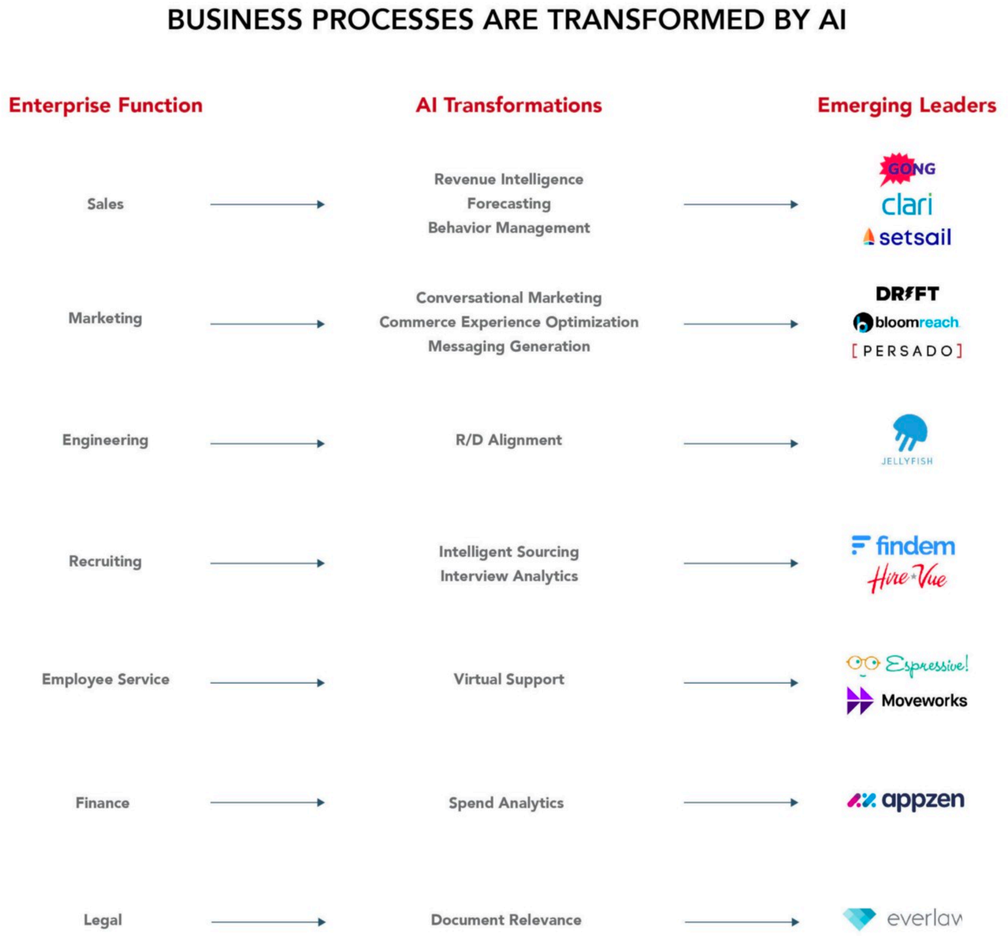
AI Modellemesi için veriye ihtiyaç duyacaksınız. Veriyi **Cloud Data Platform Servislerinde** tutabilirsiniz.

Bu veriyi depolamak ve erişmek için daha alt kısımda bir **Infrastructure Layer** ihtiyacınız bulunur.

Sistemi işletebilmek için **Monitoring & Management (Görüntüleme ve Yönetim araçlarına)** ihtiyaç duyarsınız.

B. AI-Powered (Yapay Zeka Destekli) Uygulamalar

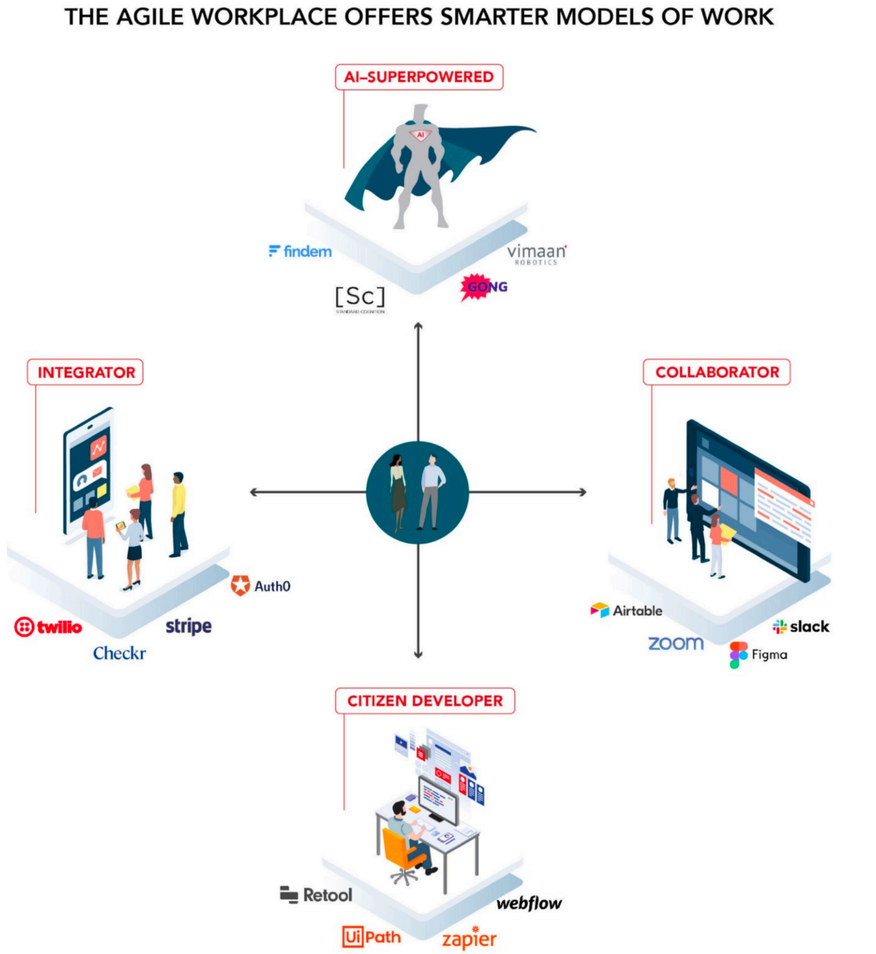
Şirketler de Kurumsal Fonksiyonlarıda **manuel yöntemler ve uygulamalar yerine yapay zeka destekli** uygulamalara dönüştürmeya çalışın.

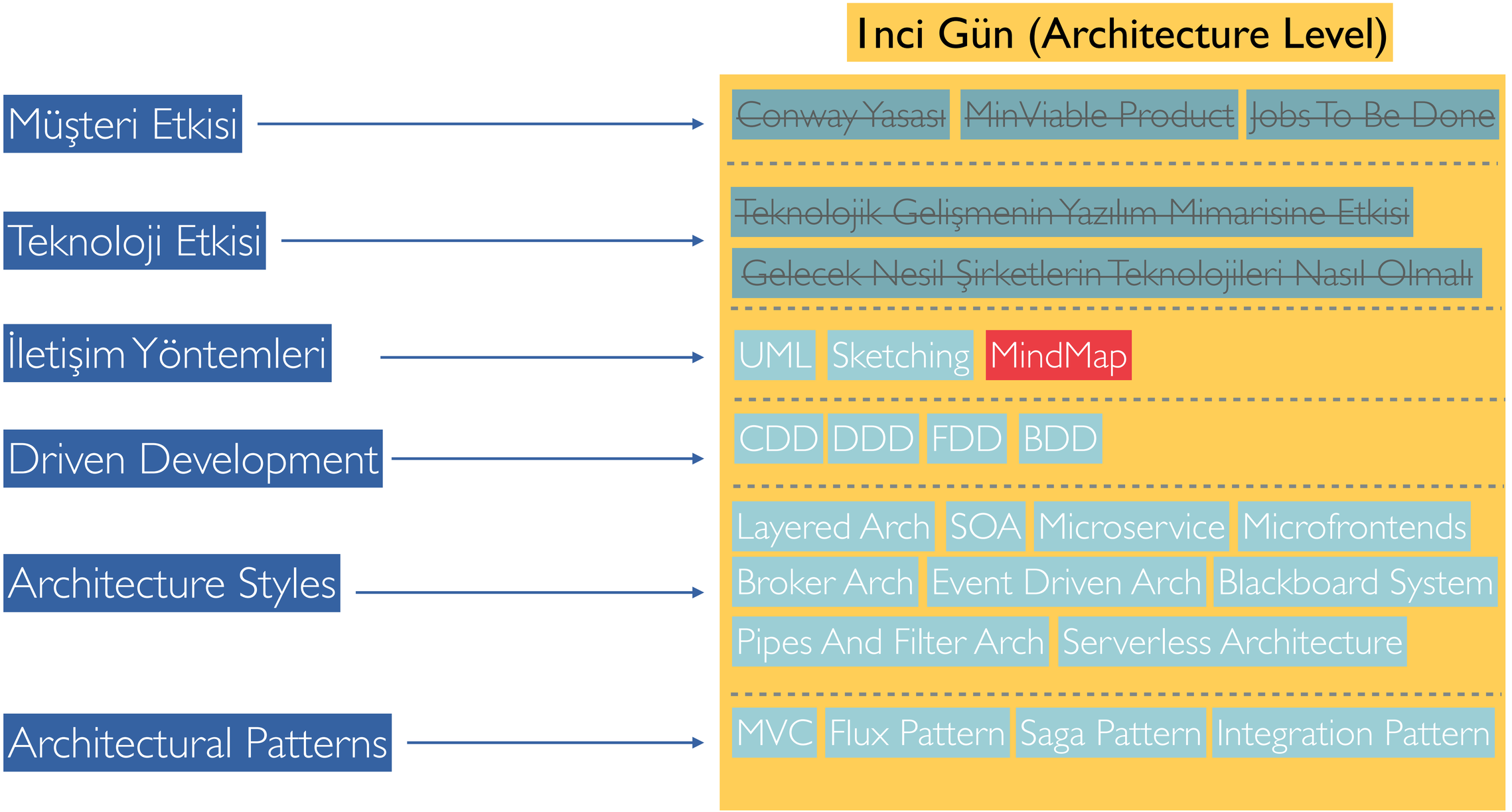


C. AGILE WORKSPACE (ÇEVİK ÇALIŞMA ALANI)

Çevik, akıllı çevik çalışma alanı modelini şirketketine getirmeye çalışın;

- AI- SuperPowered (Yapay Zeka — Super Güçlü)
- Integrator (Entegrator)
- Collaborator (Ortak Çalışan)
- Citizen Developer (Geliştirici)

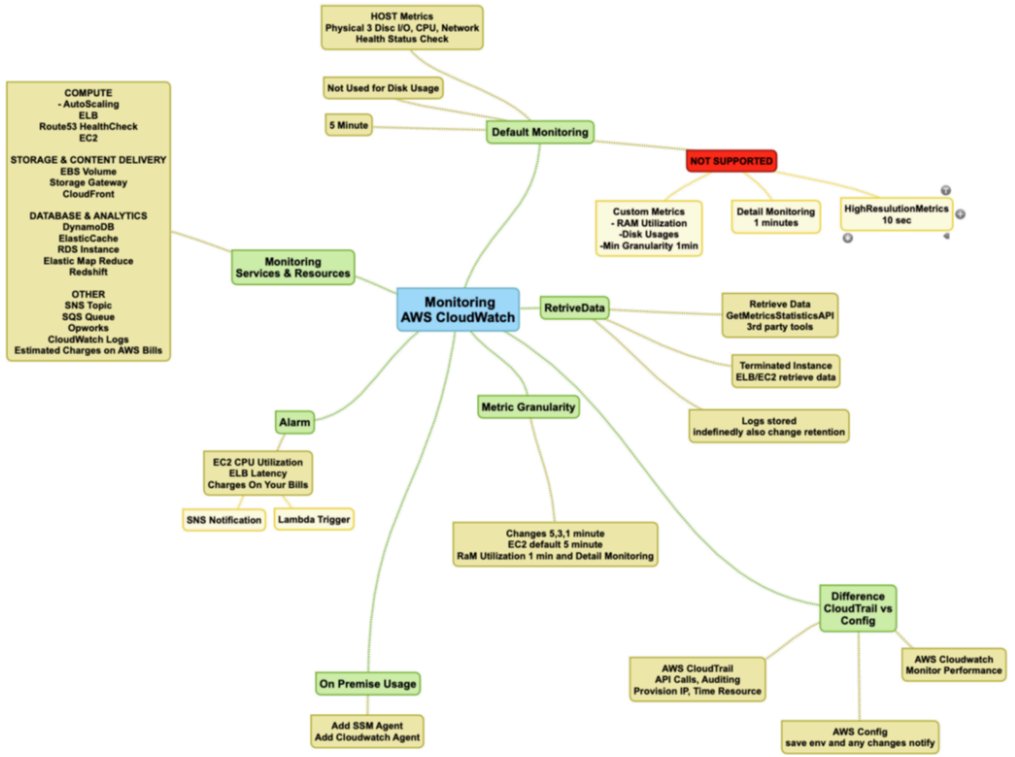



Inci Gün (Architecture Level)
Müşteri Etkisi
Teknoloji Etkisi
İletişim Yöntemleri
Driven Development
Architecture Styles
Architectural Patterns
~~Conway Yasası~~
~~MinViable Product~~
~~Jobs To Be Done~~
~~Teknolojik Gelişmenin Yazılım Mimarisine Etkisi~~
~~Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı~~
UML
Sketching
MindMap
CDD
DDD
FDD
BDD
Layered Arch
SOA
Microservice
Microfrontends
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture
MVC
Flux Pattern
Saga Pattern
Integration Pattern

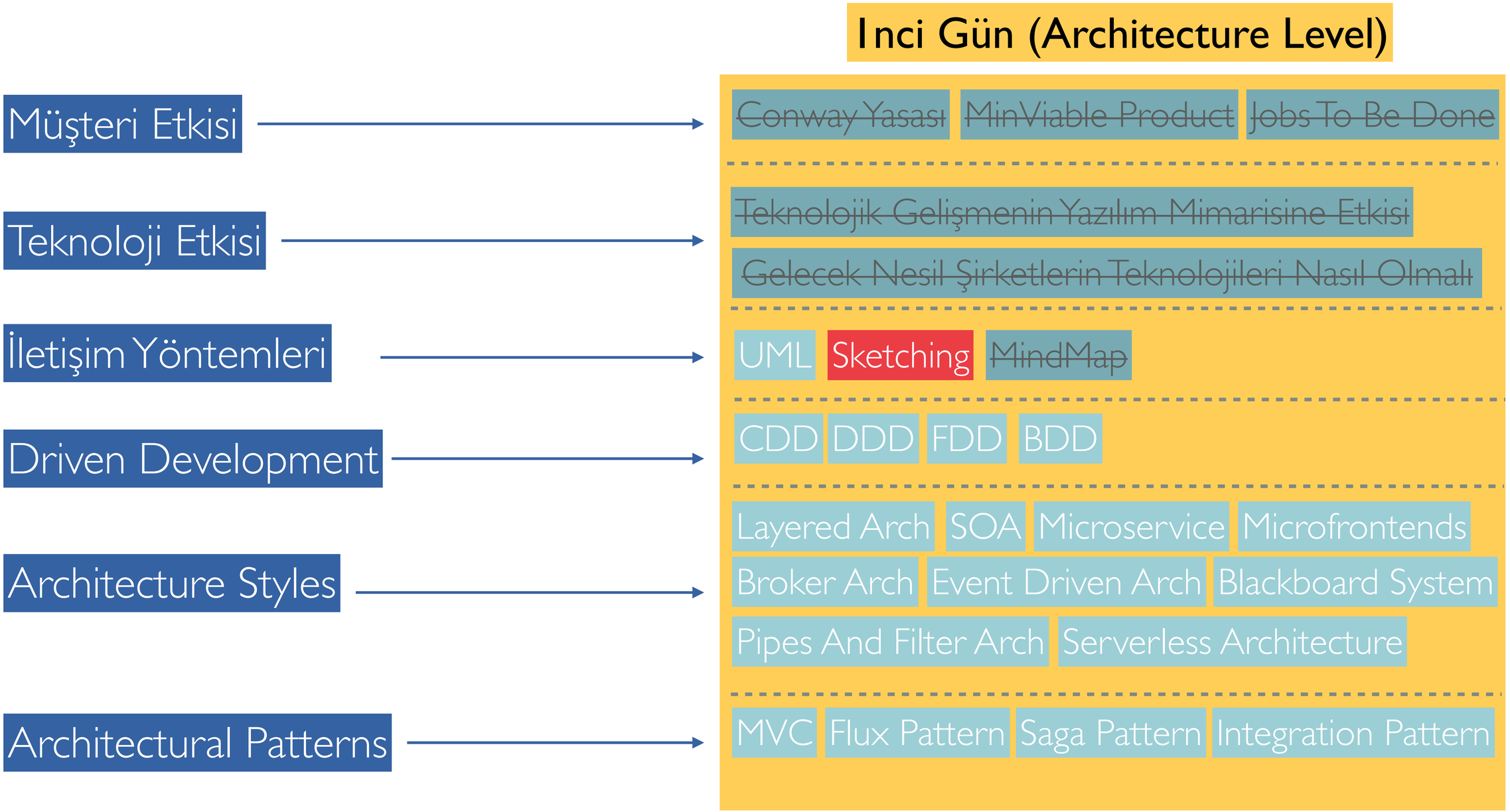
Beyin fırtınası yöntemi ile bir toplantı sırasında gurubun bir konu üzerinde yaratıcı fikirler ortaya çıkarmasını sağlayan toplantılardır. Süreç iyileştirmesi, yeni bir ürün çıkarma, ürüne yeni bir özellik ekleme,pazarlama ve reklam süreçlerinde beyin fırtınasından faydalınarak güzel sonuçlar elde edilebilir.

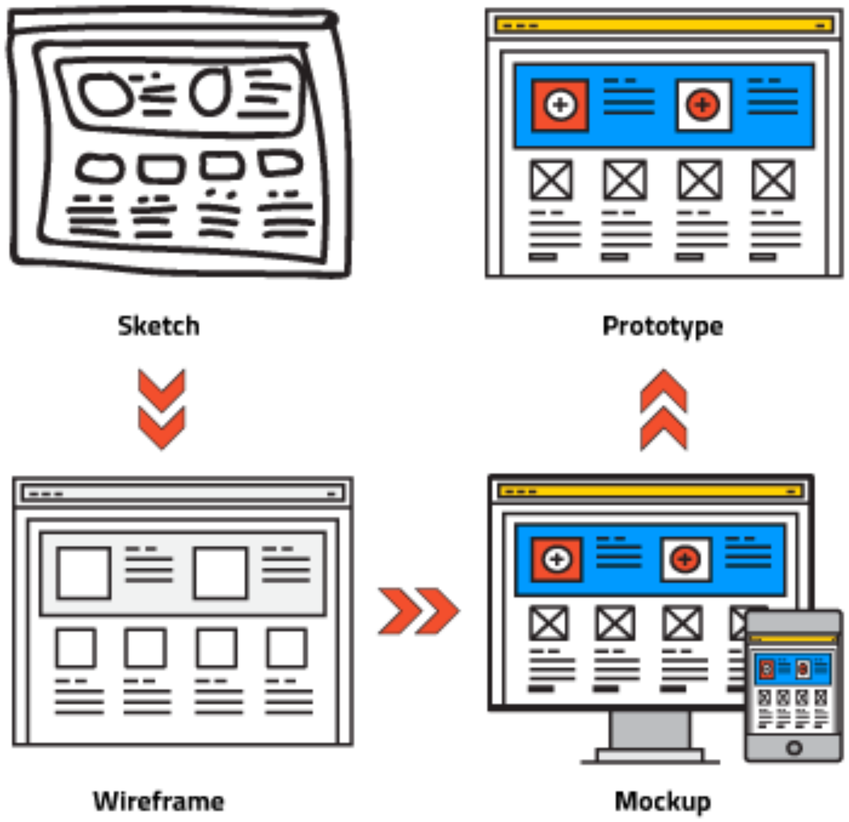
Yaratıcı fikirler oluşturmanın bir diğer yöntemide sistemi , konuşulanları , toplantıları vb.. bir harita üzerinde tümüyle görebilmektir. Bunun için tuttuğunuz notları zihin haritası yöntemi ile tutarsanız bu size dönüp baktığınızda tüm sistemin zihninizdeki haritasını tek seferde gösterebilme imkanı sunar. [Örnek bir MindMap]

Monitoring (AWS CloudWatch)



AWS CloudWatch


Inci Gün (Architecture Level)
Müşteri Etkisi
Teknoloji Etkisi
İletişim Yöntemleri
Driven Development
Architecture Styles
Architectural Patterns
~~Conway Yasası~~
~~MinViable Product~~
~~Jobs To Be Done~~
~~Teknolojik Gelişmenin Yazılım Mimarisine Etkisi~~
~~Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı~~
UML
Sketching
~~MindMap~~
CDD
DDD
FDD
BDD
Layered Arch
SOA
Microservice
Microfrontends
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture
MVC
Flux Pattern
Saga Pattern
Integration Pattern



Ürün arayüzü geliştirme sırasında kodlamaya geçilmeden önce uygulanmasında faydalı olan bazı adımlar/aşamalar vardır. Bu adımları ne kadar hakkıyla uygulayabilirseniz bu sizin projenizde sonradan çok maliyetli olacak değişiklikleri önceden öngörebilmenizi sağlar. Bu adımları ve neleri içerdiklerini aşağıda anlatmaya çalışacağım;

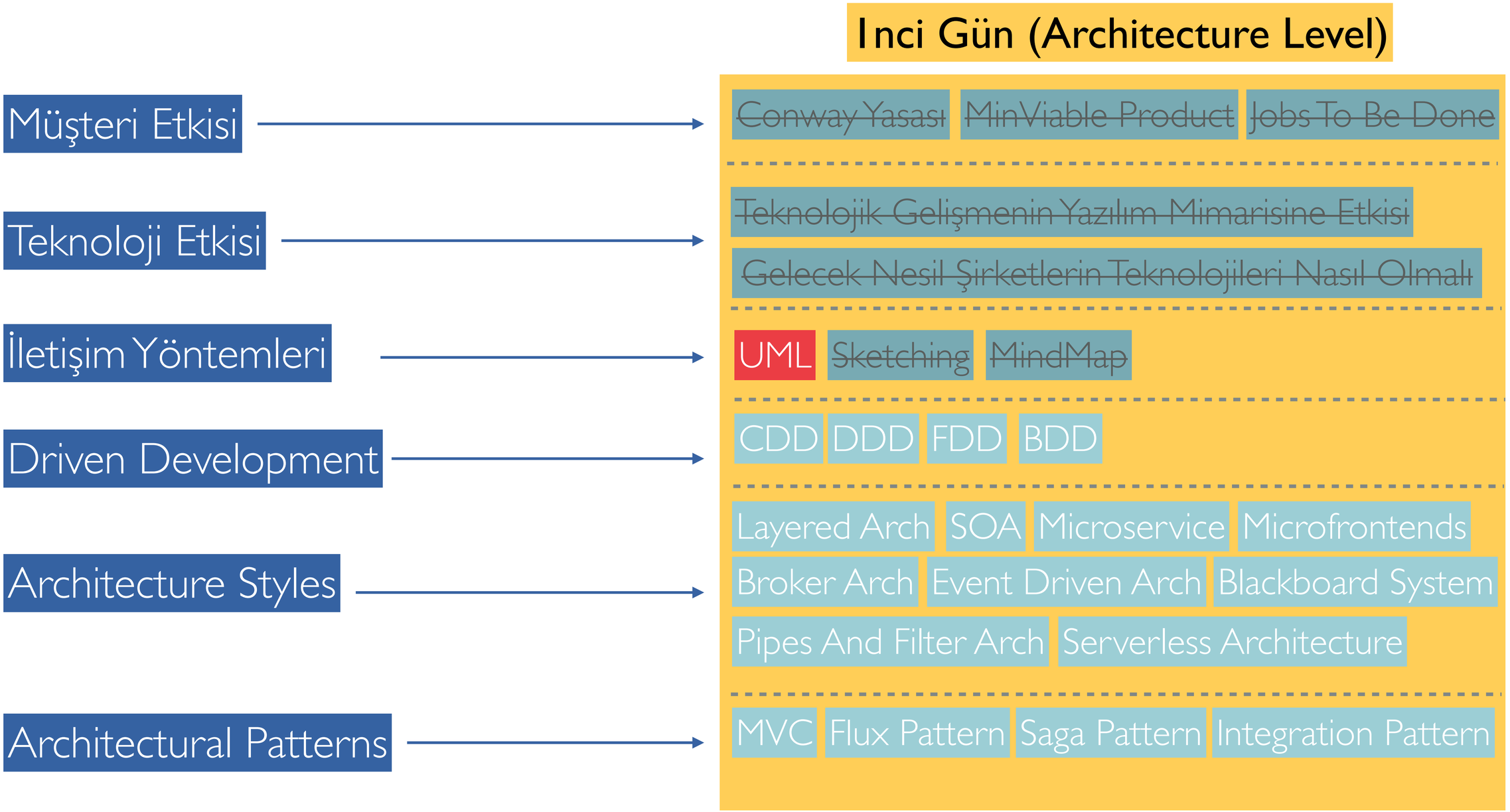
```
Sketch → Wireframe → Mockup → Prototype
```

Sketch: Projenin en başlarında fikrinizi hızlı bir şekilde anlatmanızı sağlayan karalamalar, çizimler. (MindMap 'te kullanabilirsiniz)

Wireframe: Projeniz ilerlemiş fonksiyonel akışlar belirlenmiş bu yönde projenizi anlatmanızı ve tasarım seviyesine geçmenizi sağlar. Ekranlar yinede değişiklik gösterdiği için bu aşamada tasarım ile uğraşmamak daha anlamlıdır.

Mockup: Projenizde yer alan sayfaların görsel tasarımına yönelik , renk paletleri , bileşen görsellikleri üzerine yapılan çalışmadır.

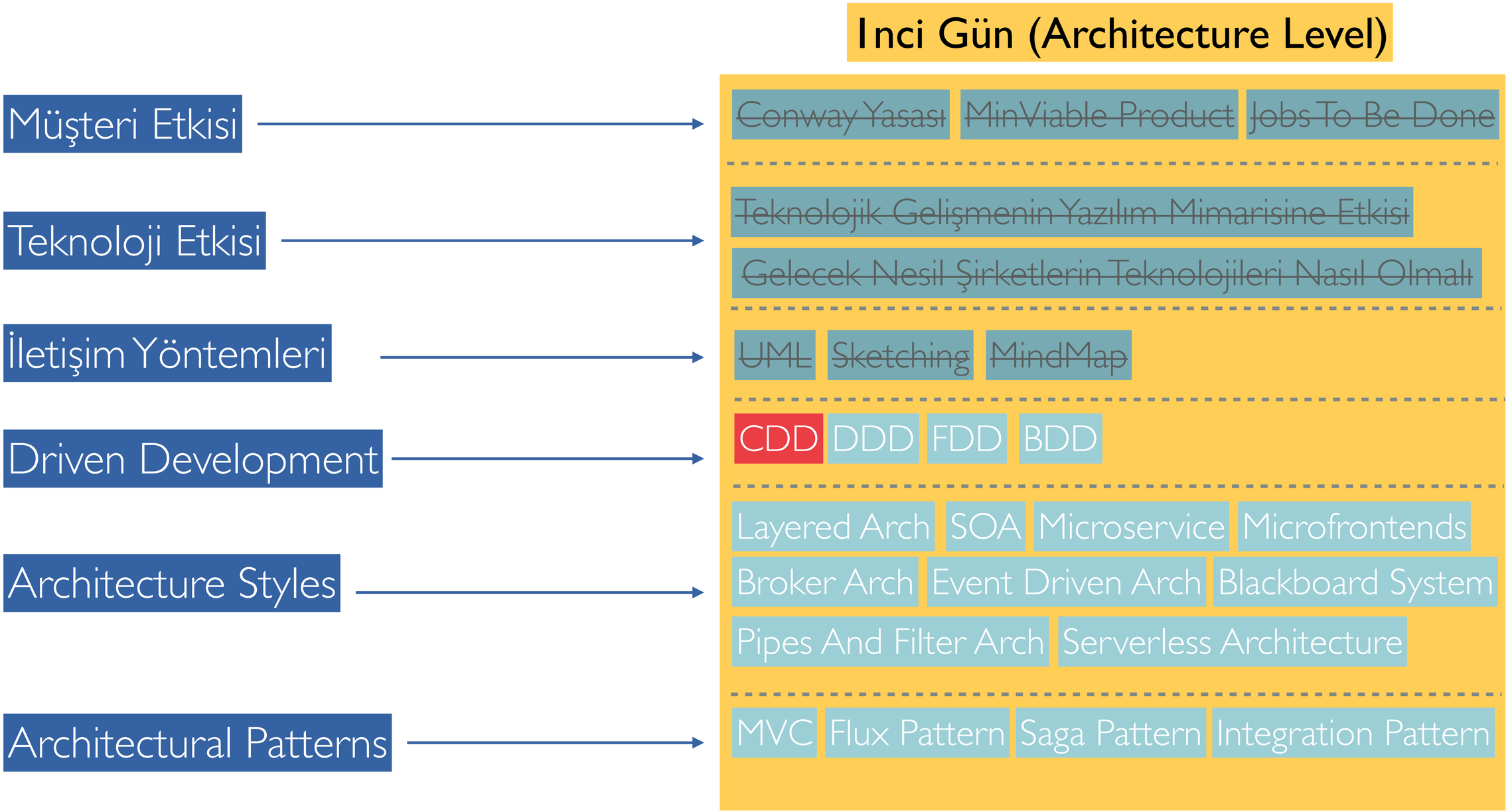
Prototype: Bileşenler ve sayfalar ile kullanıcının etkileşimi test edebilmek için yapılan çalışmadır.


Inci Gün (Architecture Level)
Müşteri Etkisi
Teknoloji Etkisi
İletişim Yöntemleri
Driven Development
Architecture Styles
Architectural Patterns
~~Conway Yasası~~
~~MinViable Product~~
~~Jobs To Be Done~~
~~Teknolojik Gelişmenin Yazılım Mimarisine Etkisi~~
~~Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı~~
UML
~~Sketching~~
~~MindMap~~
CDD
DDD
FDD
BDD
Layered Arch
SOA
Microservice
Microfrontends
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture
MVC
Flux Pattern
Saga Pattern
Integration Pattern

Yazılım geliştirmede tasarım sistemini göstermek için kullanılan temel amaçlı modelleme dilidir. Elinizde gereksinimler ve ilgili paydaşlardan alınmış veriler var ve bunlara göre anlatılanları modellemek veya bunlardan tasarım yapmak istiyorsunuz. Bu hem ekibinizin arasında ortak bir dil oluşturuyor hemde sistemin nasıl işlediğini daha net görebiliyorsunuz. 1994–1995 yılında ***Grady Booch, Ivar Jacobson and James Rumbaugh*** tarafından Rational Software firmasında iken oluşturulmuştur.

Bu modelleme dili Nesne Tabanlı Programlama yapısına uygun bir çok şemaya sahiptir. Bazıları sistemin yapısını, bazılarıda sistemin işleyişini modelleyerek yazılım geliştiricilerin baştan tasarımlar üzerinde konuşabilmesine olanak sağlar.

- **Behavioral Diagram** → Activity, State Machine, Use Case, Timing, Sequence, Communication, Interaction Overview Diagrams
- **Structural Diagrams** → Class, Composite Structure, Component, Deployment, Object, Package, Profile Diagrams


Inci Gün (Architecture Level)
Müşteri Etkisi
Teknoloji Etkisi
İletişim Yöntemleri
Driven Development
Architecture Styles
Architectural Patterns
~~Conway Yasası~~
~~MinViable Product~~
~~Jobs To Be Done~~
~~Teknolojik Gelişmenin Yazılım Mimarisine Etkisi~~
~~Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı~~
~~UML~~
~~Sketching~~
~~MindMap~~
CDD
DDD
FDD
BDD
Layered Arch
SOA
Microservice
Microfrontends
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture
MVC
Flux Pattern
Saga Pattern
Integration Pattern

CDD - Component Driven Development

Web uygulamalarının Native (Win32, MFC, Swing) ortamlardaki kompleks UI versiyonlarının tarayıcı üzerinde çalışabilecek hale gelmesi ve SPA(Single Page Application) daha sonrasında JAMstack yapısının gelişmesi bizim UI bileşenlerinin yapısının kütüphane/framework (Angular,Ember,React, Vue, Svelte …) birlikte evrimleşmesine sebep olmuştur.

Bu UI kütüphanelerinin tek amacı, birbirine kapsayabilen, birlikte iş yapabilen bileşen grupları oluşturmak.

Tekrar kullanabilir ve ölçeklenebilir sistemler tasarlamak için Bileşen Odaklı Geliştirmeye Sahip olmanız gerekiyor. Bileşenleri birleştirerek büyük sistemleri lego benzeri tasalarlama imkanı buluyoruz.

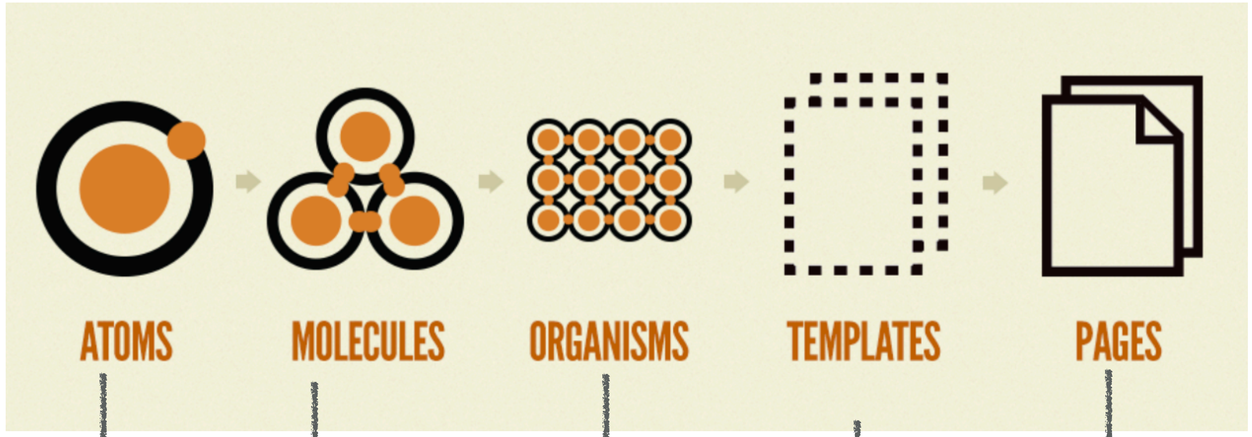
CDD - Bileşen Odaklı UI Geliştirme

Component Explorer (Bileşen Gezgini): Bağımsız bileşenler üreterek bu bileşenlerin farklı state göre çizimlerini direk bu araçlar üzerinden test edip görüntülemenizi sağlayan araçlar. React için (Storybook, React Styleguidist , React Cosmos …).

Design System: Bir çok büyük firma Invision, Salesforce, Atlassian vb.. firma bir çok proje ve uygulama geliştiriyor. UI için ortak bir dil olması gerekiyor . Temelde Color, Typography, Sizing and spacing, Imagery ile bir tasarım dili oluşturulması ve bunlar ile bileşen, sayfa ve geçişlerin(animasyonların) tasarım bütünü diyebiliriz. A comprehensive guide to design systems — Invision

Component Based Mimari: Bu konuda Brad Frost yazmış olduğu Atomic Design kitap ve Tom Coleman yazmış olduğu Component Driven Development bu tarz yapılar oluşturmak için referans oluşturmaktadır.

Bileşen Odaklı UI - Atomic Design



Atoms: Web sayfasını oluşturan en küçük elemanlardır. Daha fazla parçalanamazlar. HTML tags ; ***label, input, button, div, canvas …***

Organisms: Birden fazla molekülün, bazen aynı türdeki molekül gruplarının tekrar ederek oluşturdukları yapılardır. Örneğin; **NavigationBar, Table, ListItems vb..**

Templates: Bir veya birden organizma/molekülün layout üzerindeki yerleşim yapısıdır.

Molecules: Atomların bir araya gelerek oluşturdukları küçük bileşenlerdir. Örneğin bir arama molekül oluşturmak istiyorum;

- Label, Input, Button birleştirerek bu molekülü oluşturabilirim
- ***Label*** kullanıcıya bu bileşenin nerede aradığını, ***Input*** arama yapmak istediği kelimeyi gireceği ortamı, ***Button*** ise arama işlemini tetiklemesini sağlar

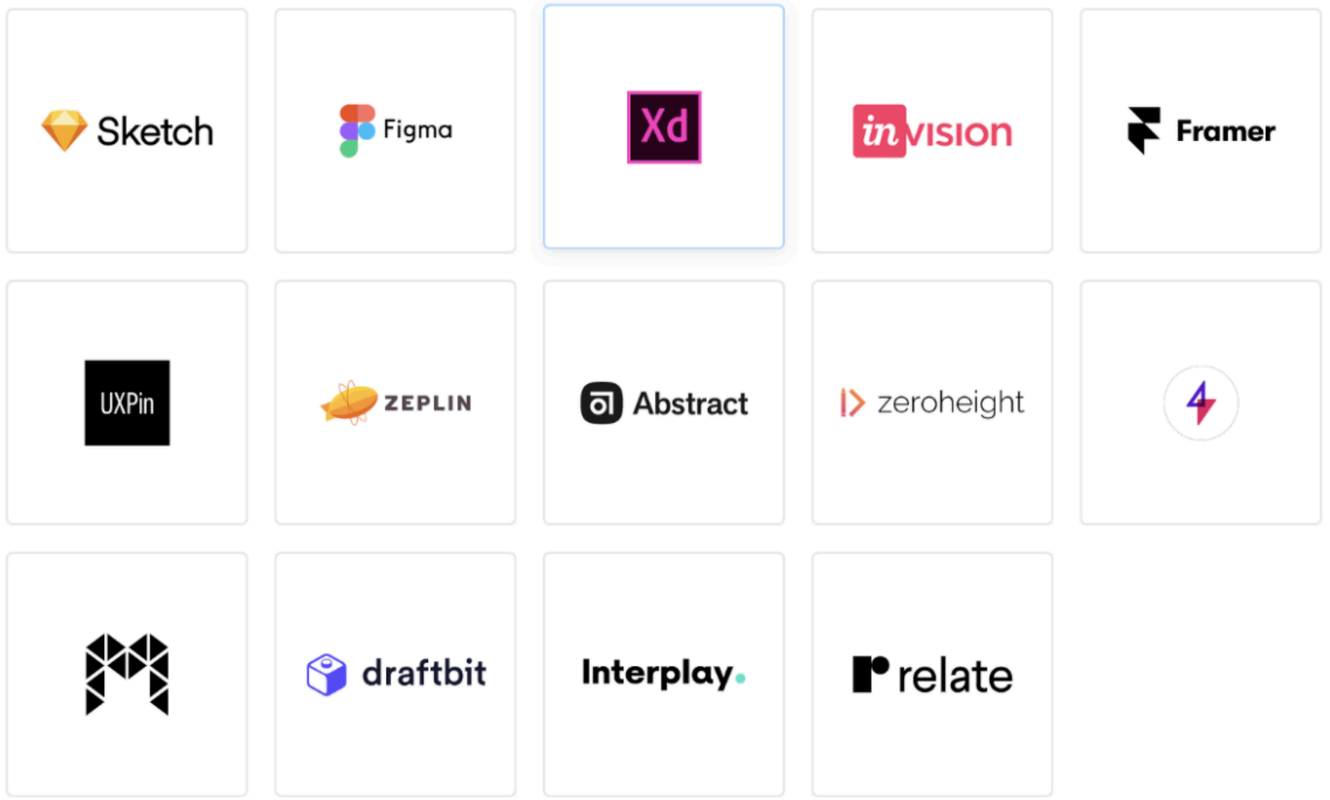
Burada moleküllerin ***sadece bir iş yapması ve onuda iyi yapması*** üzerine tasarlanmalıdır.

Pages: Template belli veri ile doldurularak gösterildiği gerçek sayfadır.

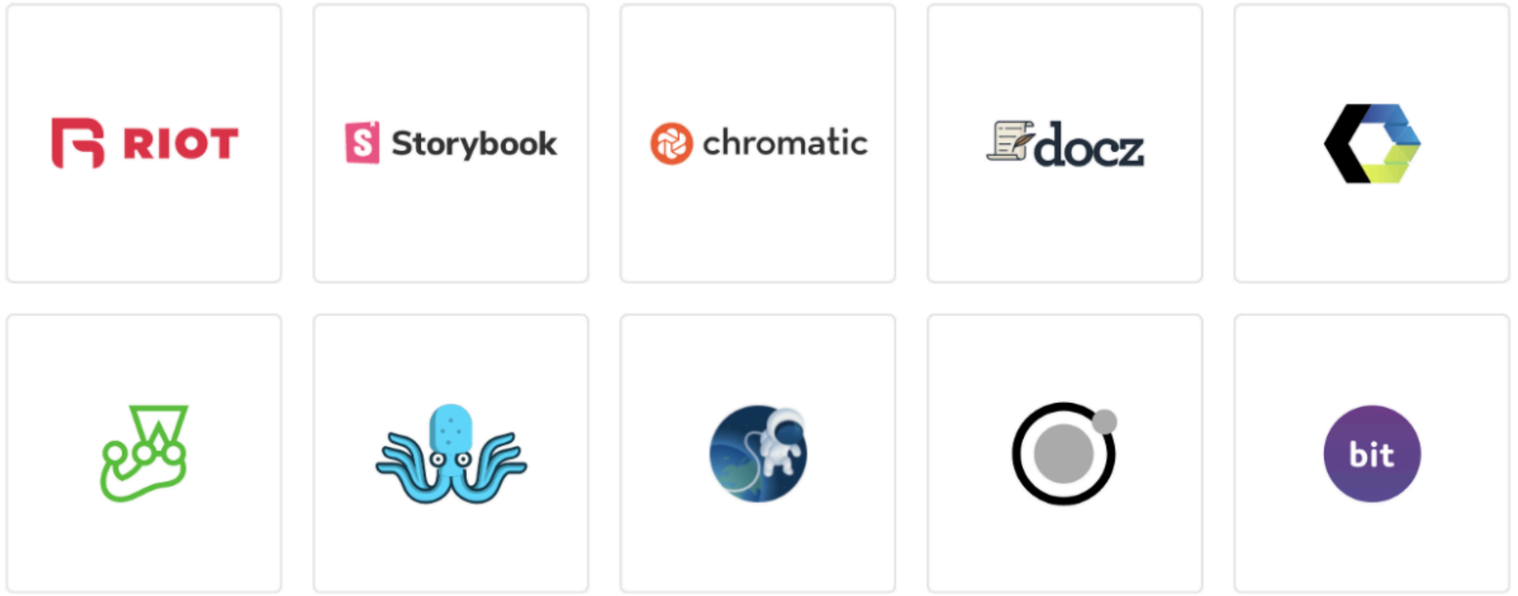
Bileşen Odaklı UI Geliştirme

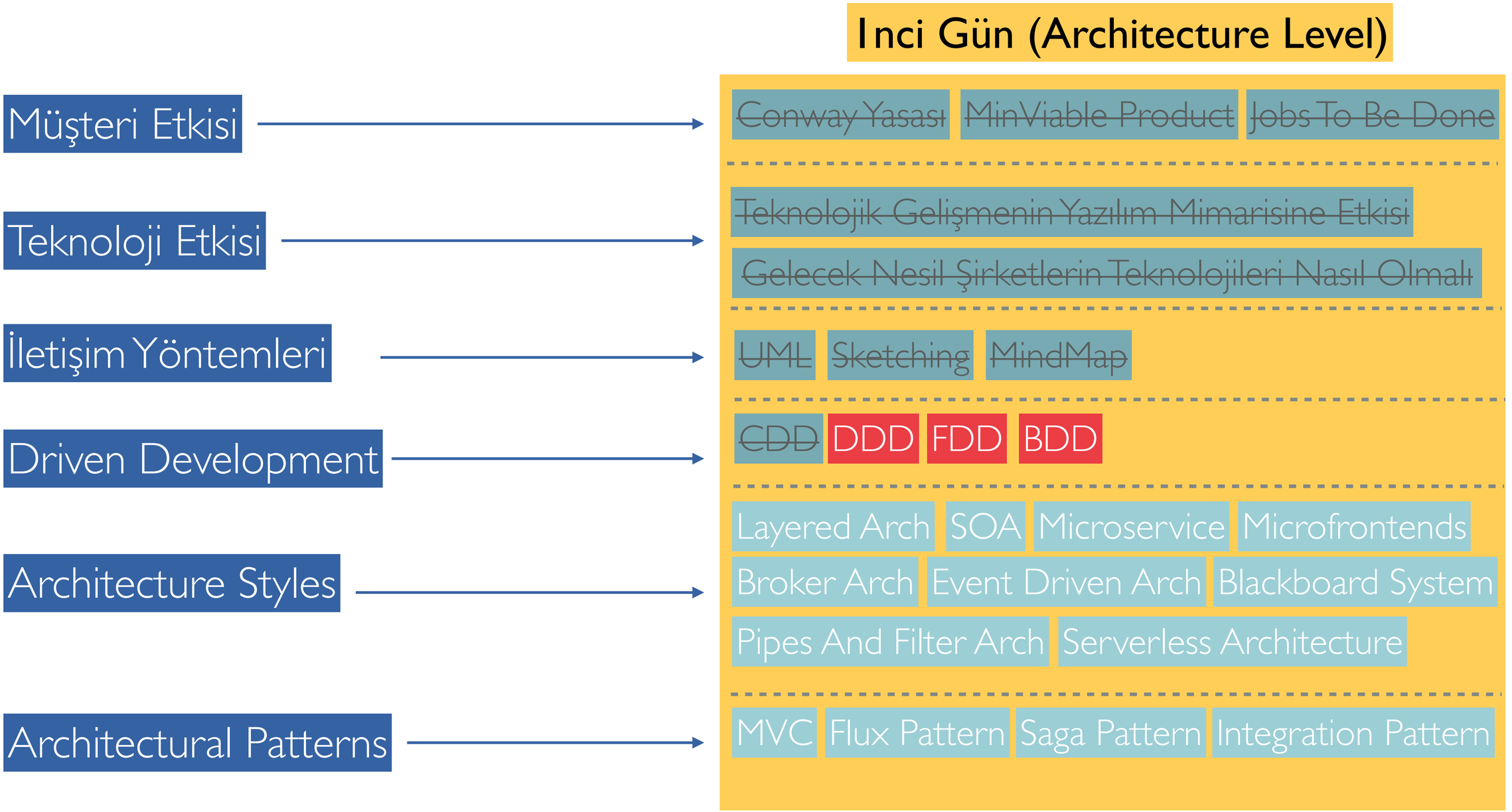
Component → Combine Components → Assemble Pages → Integrate Pages..

Bileşen Odaklı UI Geliştirme - Tasarım Araçları



Bileşen Odaklı UI Geliştirme - Test Ortamı

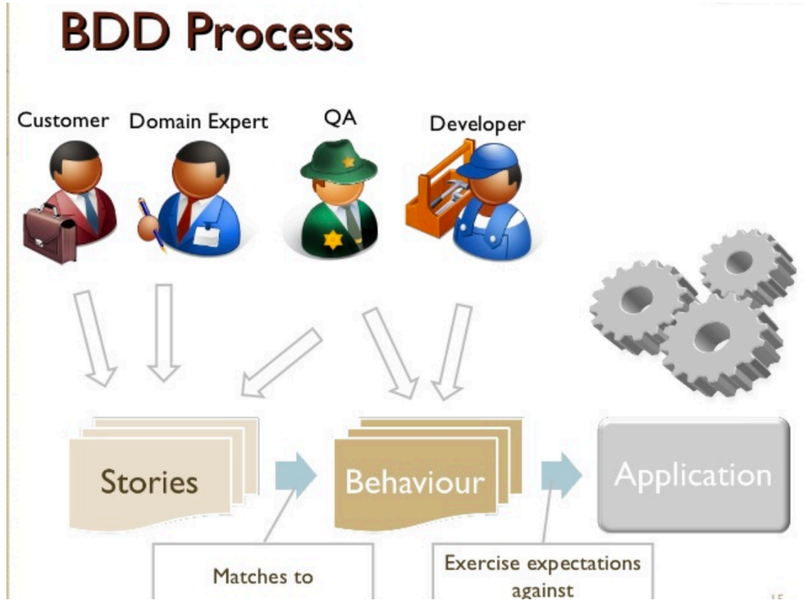


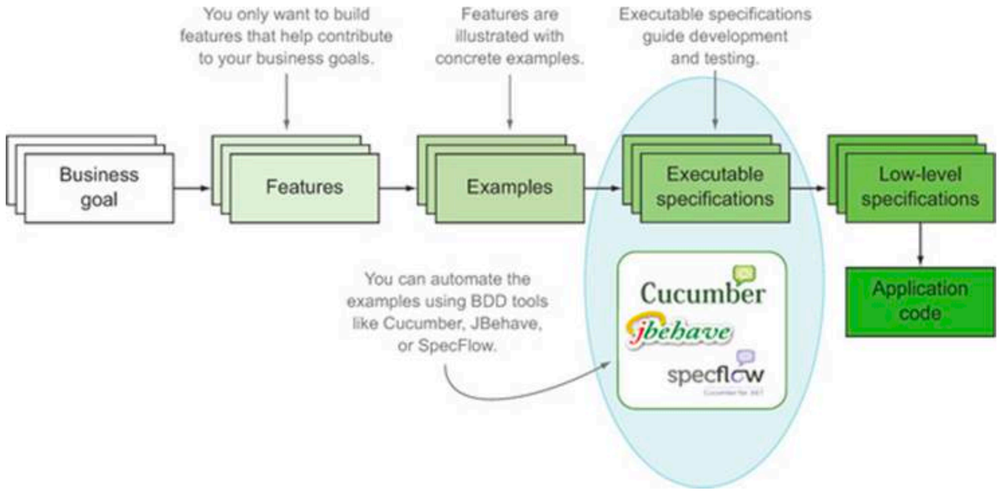

Inci Gün (Architecture Level)
Müşteri Etkisi
Teknoloji Etkisi
İletişim Yöntemleri
Driven Development
Architecture Styles
Architectural Patterns
~~Conway Yasası~~
~~MinViable Product~~
~~Jobs To Be Done~~
~~Teknolojik Gelişmenin Yazılım Mimarisine Etkisi~~
~~Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı~~
~~UML~~
~~Sketching~~
~~MindMap~~
~~CDD~~
DDD
FDD
BDD
Layered Arch
SOA
Microservice
Microfrontends
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture
MVC
Flux Pattern
Saga Pattern
Integration Pattern

BDD (Behavior Driven Development)

Yazılımlara daha yukarıdan bakarsak sadece Developer'ın içerisinde olmadığını **Müşteri, Alan Uzmanı, Kalite vb..** kişilerinde yazılımda rol oynadığını görebilirsiniz.

BDD yönteminde Kullanıcı hikayelerinden çıkan Behavior'lar üzerinde Acceptance Test oluşturularak yazılım davranışı, yazılım tasarlanmaya çalışılır.





FDD (Feature Driven Development)

İstemci(müşteri) tarafından üretilen özellikleri belirlenip geliştirilmesi üzerine kurgulanmış yapıdır. Aşağıdaki yapıda feature tanımlamaları yapılır.

- <action><result><object>
- Calculate the total of a sale
- Validate the password of a user

  Authorize the sales transaction of a customer

Bu feature FDD şeklinde işletilerek RUP — use case/kullanım senaryosuna , Scrum — user stories/kullanıcı hikayelerine dönüştürülerek bunların geliştirilmesi planlanır, tasarlanır ve geliştirilir.

DDD (Domain Driven Development)

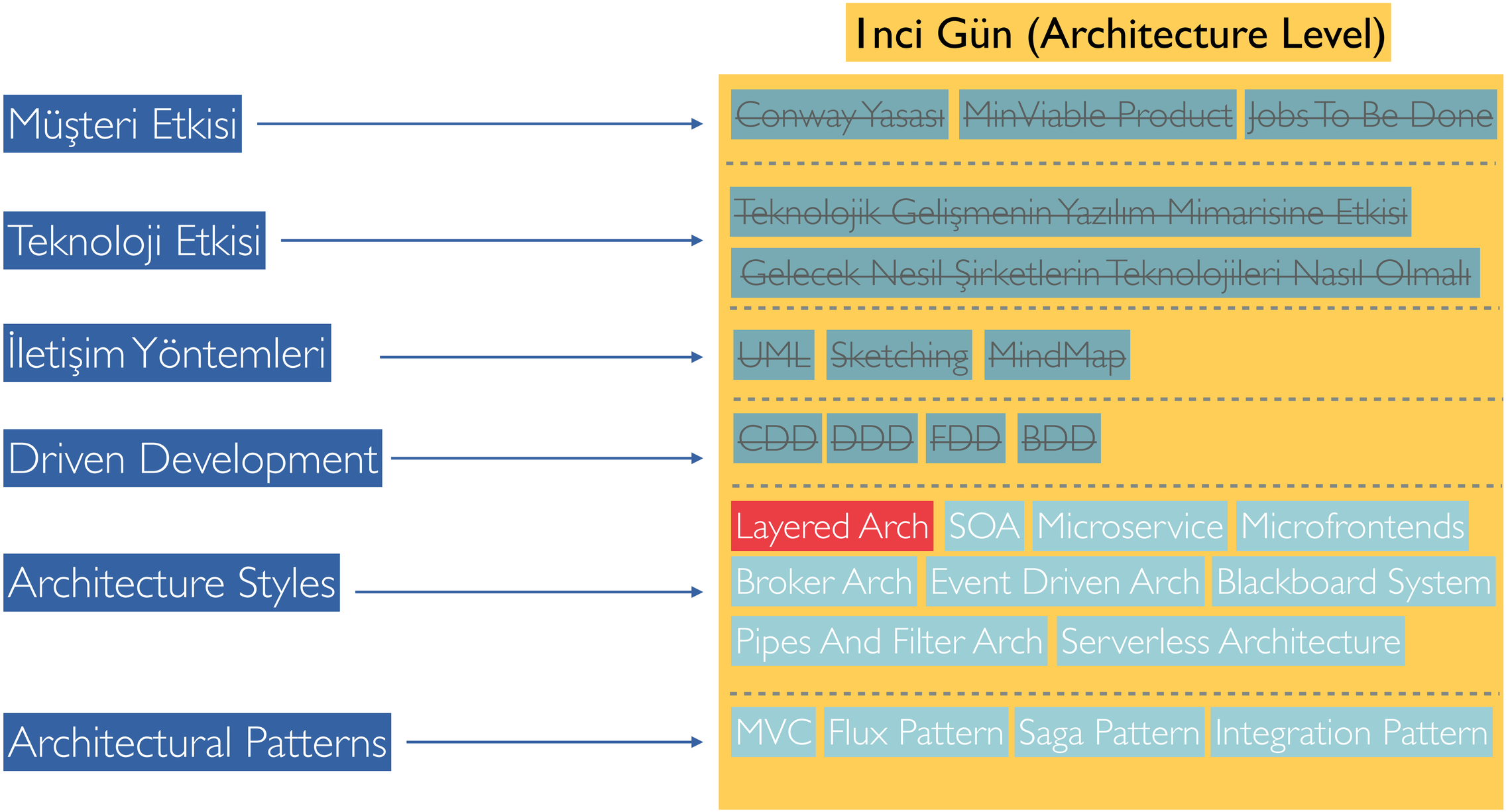
Kompleks ve karmaşık sistemler, Bankacılık, Sigorta, Vergi, Uçuş vb sistemleri geliştirmek için **Domain/Alan** odaklı tasarım ve geliştirme yapılması oldukça önemlidir. Çünkü sistemler birçok alanı ve bu alanlar arası iletişimi içermektedir.

Birde bunun üstüne iş kuralları sürekli değişen ve güncellenen ve tamamı ile sistemdeki bağlantıları etkileyen bu mekanikleri normal yazılım geliştirme yöntemleri ile ilerlemek pek başarılı sonuçlar vermeyecektir.

Bunun yerine **Etki Alanına Dayalı Geliştirme** yöntemi seçilmelidir.

- Alan uzmanları ile ortak dil belirlenmeli (Domain Spesific Language)
- Bounded Context içerisinde Context Mapping yapılmalıdır.

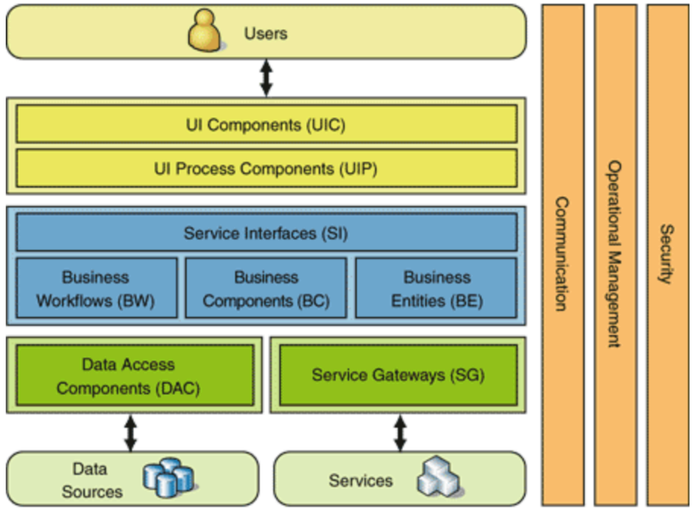
Inci Gün (Architecture Level)



Layered Architecture (Katmanlı)

Mimari patternlerden biriside Layers denilen Katmanlı Uygulama Örüntüsüdür. Burada amaç işleri fonksiyonel gruplara bölerek, bu bölümlerin birbirlerine bu katmanlar aracılığı ile ulaşmasını sağlamaktır.

Çok uzun zamandır tercih edilen bu yöntem bir çok Web ve Enterprise uygulamanın temelini oluşturmaktadır. En çok bilinen ve ismi duyulanda 3 katmanlı mimari yapısıdır.



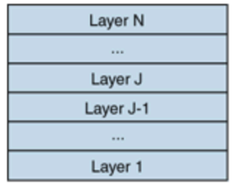
Presentation Katmanı: Bu katman sunum ve arayüz katmanıdır. Kullanıcının uygulama ile etkileşime geçtiği arayüz bileşenlerinin bulunduğu katmandır.

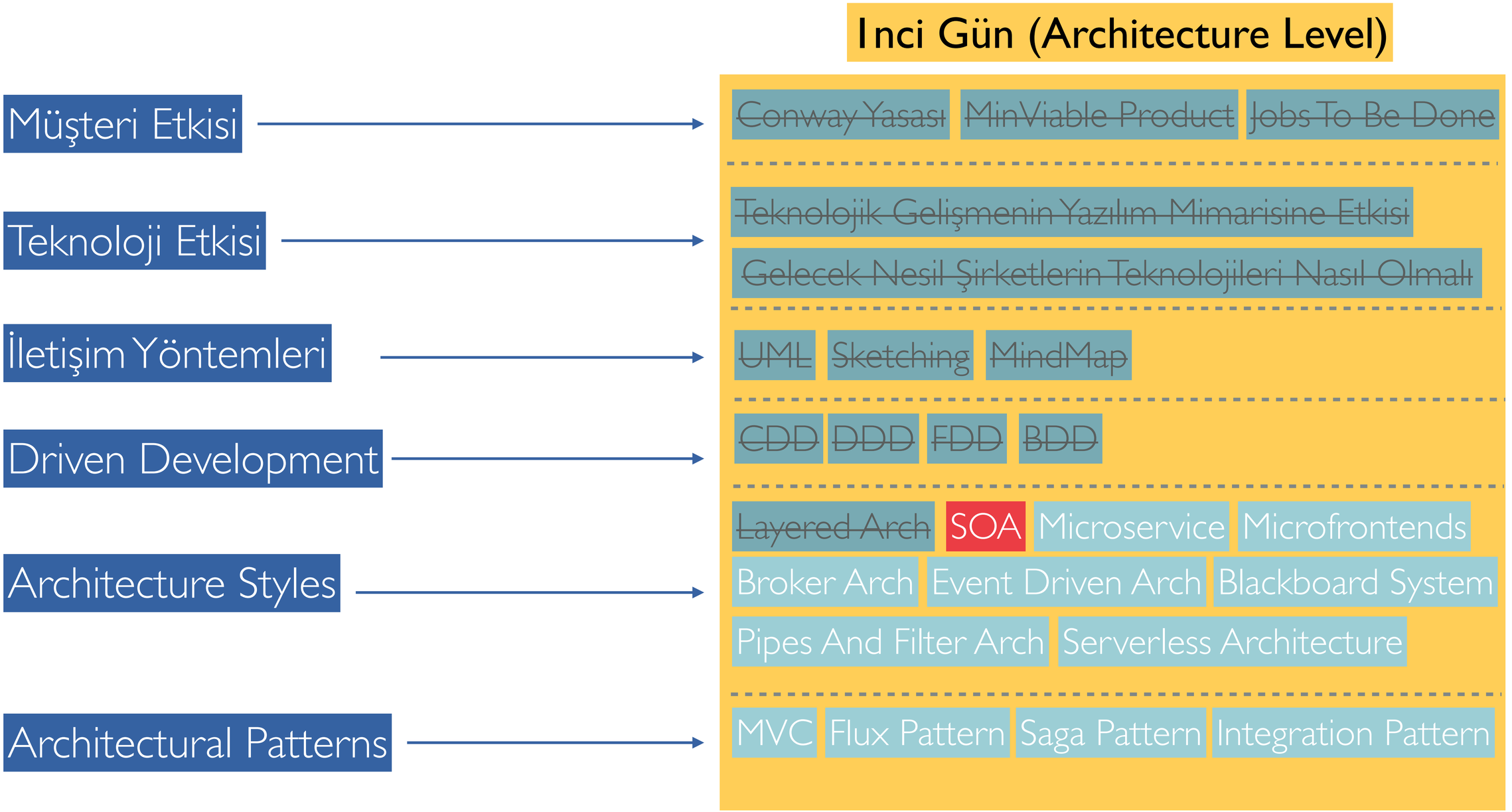
Business Katmanı: İş akışlarını, iş bileşenlerini ve nesnelerini tutan yöneten katmandır.

Data Katmanı: Veri katmanıdır. Veri ya sizin dosya sisteminizde veya veritabanınızda tutulmaktadır.

Uzak Servis Katmanı (RPC) Uygulanız uzak bir sistemden fonksiyon çağırıyor ise buda WebServis/ServiceGateway üzerinden gerçekleştirilir.

Katman sayınızı kendiniz belirleyebilirsiniz. Katmanlar arası iletişimi bir alt katmana erişim sağlayacak şekilde yaparsanız sistem güvenliğini arttırırsınız. Her katman kendi seviyesinde güvenliğini kontrolleri gerçekleştirebilirsiniz.




Inci Gün (Architecture Level)
Müşteri Etkisi
Teknoloji Etkisi
İletişim Yöntemleri
Driven Development
Architecture Styles
Architectural Patterns
~~Conway Yasası~~
~~MinViable Product~~
~~Jobs To Be Done~~
~~Teknolojik Gelişmenin Yazılım Mimarisine Etkisi~~
~~Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı~~
~~UML~~
~~Sketching~~
~~MindMap~~
~~CDD~~
~~DDD~~
~~FDD~~
~~BDD~~
~~Layered Arch~~
SOA
Microservice
Microfrontends
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture
MVC
Flux Pattern
Saga Pattern
Integration Pattern

SOA (Service Oriented Architecture)

SOA (Servis Odaklı Mimari) kavramının ne olduğuna anlatmadan önce bu kavramı ilk defa nerde duyduğum ve ne için kullanıldığını anlatmaya çalışarak SOA kavramını daha iyi anlatabileceğimi düşünüyorum.

Servis Odaklı Mimari'yi ilk defa T2 firmasında Türk Telekom için çalışmaya başladığımızda duydum. Türk Telekom'da Entegrasyon ve Mimari ekibinin sürekli üzerinde uğraştığı konulardan biri de buydu.

Telekom'un içerisinde 100 lerce proje geliştirilmiş. Zaman içinde bu uygulamalar giderek artmış ve aynı veritabanı üzerine kullanılan bir sürü proje olmuş. Her projenin ekstra ihtiyaçları doğrultusunda veritaban tabloları giderek artmış, üzerine yapılan sorgular giderek artmış ve karmaşıklık bu işi başka bir şekilde çözmeye itmiş.

Daha sonrasında veritabanı soyutlanmış ve belli uygulamaların dışarıya servis vermesi şeklinde tasarımlar yapılmış. Fakat ilerleyen sürede uygulamaların birbirlerinden kullandıkları servislerin karmaşası ile karşılaşılmış ve bunu çözmek için tüm uygulamaları SOA yapısı üzerine oturtmayı ve bu servisleri ESB(Enterprise Service Bus) üzerinden vermeye çalışıyorlardı.

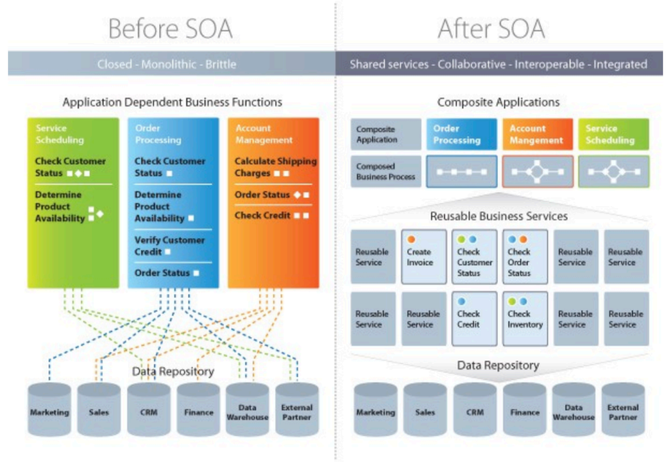
Verilen hizmet çeşitliliği ve kampanyalar bu kadar yaygın değil iken uygulamalar Telekom'un ürün ve hizmetler ihtiyaçlarına göre karar verilip geliştiriliyordu. IT ile ilgili olan bu projeler genelde Telekom'un kendi firması İnnova tarafından geliştiriliyordu. Örneğin

- Müşteri İlişkileri Yönetim Sistemi,
- Kurumsal ve Bireysel Hizmet Merkezleri,
- Çağrı Merkezi
- Arıza Sistemi
- Tahakkuk Sistemi
- Kampanya Yönetim Sistemi
- Ürün Satış ve Pazarlama
- vb…

Bu ve benzeri projelerin kapsamlarının artması, bunları geliştiren firmaların sayıların artması ve çeşitlenmesi Accenture(CRM), Etiya(Product Catalog), Innova, T2(OHM, Mobil, Tahakkuk) vb..

Bu ve benzeri projelerin kapsamlarının artması, bunları geliştiren firmaların sayıların artması ve çeşitlenmesi Accenture(CRM), Etiya(Product Catalog), Innova, T2(OHM, Mobil, Tahakkuk) vb..

Bu şirketlerin geliştirdiği uygulamaların birbirlerine servis vermeleri ve bunların yönetilmelerini zorunlu hale getirmiştir. Daha önceden olduğu gibi herkesin aynı veritabanı üzerinde geliştireceği uygulamalar ile çözülebilecek bir yöntemden farklı bir yönteme geçilmesi gerekmektedir.



Temelde problem ***veri'den*** kaynaklanıyor. Örneğin Müşteriye erişebileceğiniz telefon numarasını düşünün

- Bu verinin sahibi veya sahipleri kimlerdir ?
- Bu veri kim üretir ?
- Bu veriyi kim günceller ?
- Bu veriyi kim görüntüler ?
- Bu veriyi kim siler ?
- Bu veri güncellenince nereler nasıl etkilenir ?
- Bu verinin tüm alanlarını görme yetkisi var mıdır ?

Veriyi uygulamalardan soyutlarak bir servis aracılığı ile sağlarsak. Hem verinin güvenilirliğini, hemde sorunsuz güncellenmesini, hem de sorunsuz dağıtılmasını sağlamış oluruz.

ESB (Enterprise Service Bus)

Buraya kadar herşey güzel peki bu ESB nerden, hangi ihtiyaçtan doğmuş. SOA tek başına yeterli bir şey değilmi ?

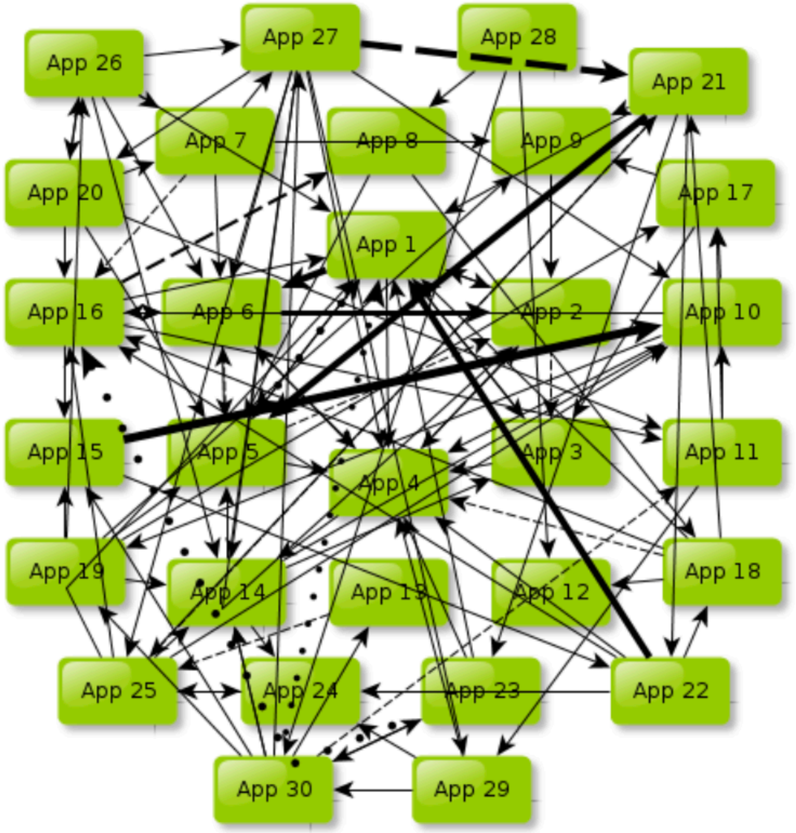
Buraya kadar herşey güzel peki bu ESB nerden, hangi ihtiyaçtan doğmuş. SOA tek başına yeterli bir şey değilmi ?

Uygulamaların giderek artması sonucu karşımıza aşağıdaki gibi sorunlar çıkar ?

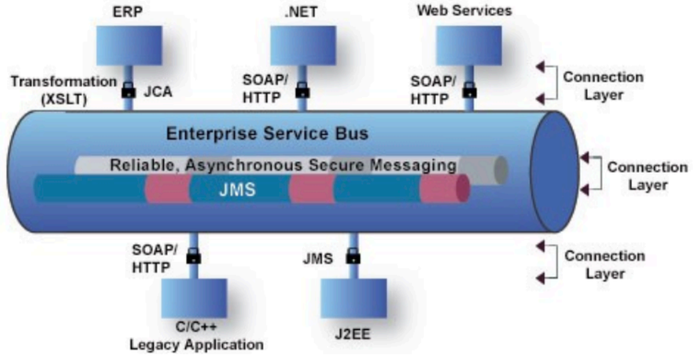
- Uygulamalar birbirlerine verecekleri servisleri hangi protokol üzerinde sağlıyacakları HTTP,AMQP,SMTP, FTP, SOAP vb üzerinden sağlıyacakları, Bu servislerin güvenlik seviyeleri, Servislerde sağlıyacakları veri formatları JSON, HTML, XML, Binary vb.. olması
- Uygulamaların görmeleri gerek veri içeriğinin birbirinden farklı olması
- Verilmesi gereken servisteki verinin yeterli olmaması 2,3 servis verisinin birleştirilerek bir veri oluşturma isteği.
- Servislerin versiyonları. Bir servis eski iş mantığına göre çalışırken, diğer servis yeni bir versiyonla çalışması gerekebilir.
- vb..

SOA

Tüm bunlar servislerin yönetilemez bir hal almasına sebep olabilir.

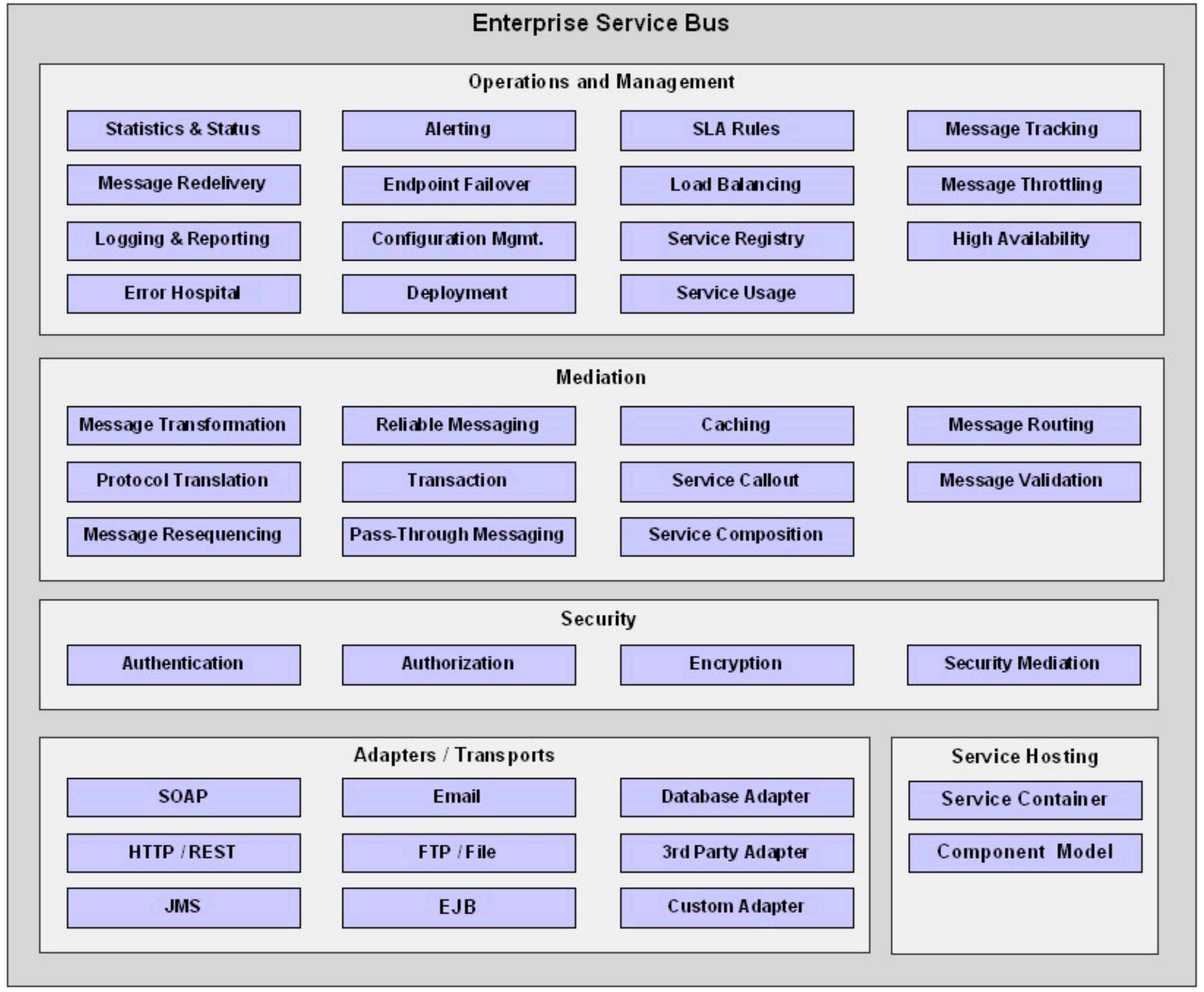


Bunun için ESB ürünlerine ihtiyaç duyulmaktadır. ESB, servislerin bir kanal üzerinden akacak şekilde yönetilmelerini sağlar. Farklı farklı projeler ve ortamlar tüm bu servisleri bu Bus üzerinden alıp kullanabilirler.



ESB'nin iç yapısına baktığımızda Adapters/Tranports,Service Hosting, Security, Mediation ve Operations and Management bölümlerinden oluşur.

SOA



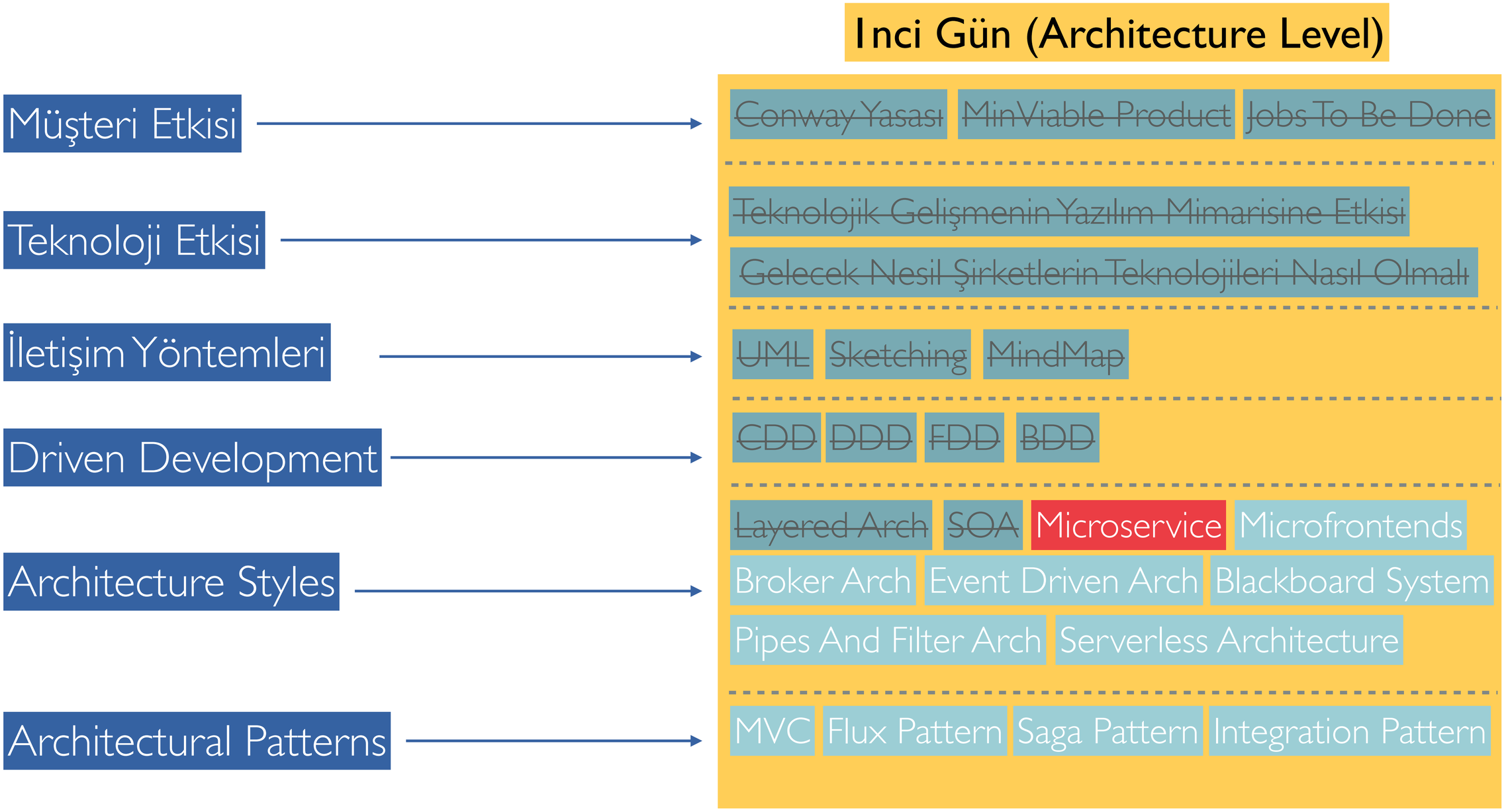
Adapters / Transports: Uygulamaların değişik ihtiyaçlarına göre servislerle erişim protokolleri, veritabanı adaptörleri ve 3rd Party Adaptörleri sağlaması

Service Hosting : Servislerin yüklendiği, servislere erişimin sağlandığı, yaşam döngülerinin yönetildiği kısım.

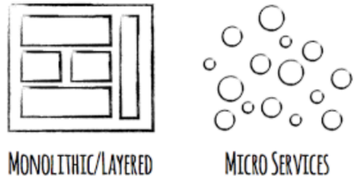
Security: Transport ve Mesaj seviyesinde güvenlik sağlandığı kısımdır. Servisler için kimlik doğrulaması ve yetkilendirmenin yapılabilmesine olanak sağlar. Mesajların Encryption/Decryption yapılmasını sağlar.

Mediation: Mesaj dönüşümlerinin, protokol dönüşümlerinin, mesaj sıralamalarının gerçekleştirildiği modüldür.

Operation And Management: ESB hakkında istatistik toplama, performansı hakkında bilgi alma ve buna göre yükü dağıtma işlemlerinin gerçekleştirildiği kısımdır. Mesajın gereğinden fazla olması durumunda kuyruğa atılıp işlenmesi , konfigurasyon yönetimi vb bir sürü operasyon ve yönetim bu modül aracılığı ile gerçekleştirilir.


Inci Gün (Architecture Level)
Müşteri Etkisi
Teknoloji Etkisi
İletişim Yöntemleri
Driven Development
Architecture Styles
Architectural Patterns
Conway Yasası
MinViable Product
Jobs To Be Done
Teknolojik Gelişmenin Yazılım Mimarisine Etkisi
Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı
UML
Sketching
MindMap
CDD
DDD
FDD
BDD
Layered Arch
SOA
Microservice
Microfrontends
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture
MVC
Flux Pattern
Saga Pattern
Integration Pattern

Microservices



Kurumsal uygulamalar geliştirilirken eskiden Monolithic/Layered dediğimiz esasa göre gerçekleştiriliyordu. N Tier dediğimiz bu mimariyi basitçe Veri Katmanı, İş Katmanı ve Arayüz Katmanı olarak ayırıyorduk. Tüm bileşenler veya uygulamalar tek bir Container içerisinde katmanlı yapı içerisinde kodlanıyordu.

Daha sonra sistemler ve projeler biraz daha geliştikten sonra WSDL dediğimiz kavramlar ile birlikte SOA(Servis Odaklı Mimari) ler gelişmeye başladı. Bunun ile birlikte Servis yönetimini ve SOA sorunlarını gidermeye çalışan ESB'ler çıktı.

SOA dediğimiz kavramı bir önceki yazımızda anlatmıştık. http://bit.ly/1MLfYEb SOA ve ESB'ler dışarıdan görüldüğü gibi kolay şeyler değil tabi.

- Ekiplerin birbirleri ile planlı çalışması gerekiyor.
- Entegrasyon Takımının sürekli olarak ekipleri toplayıp servislerin versiyon kontrolünü ve değişen servis kontrolünü takip etmesi gerekiyor.

Bu seferde servislerin yönetimi her ne kadar ESB gibi altyapılar olsa bile işin doğası gereği çok sancılı bir sürece dönüşüyor.

Normalde **EKİP**'leri incelediğimizde ekiplerin

Arayüz ekipleri : Mobil (IOS, Android, Windows Phone Developer) / Web (HTML, JS, CSS, Angular, React vb..) konularda arayüz, usability vb konularda uzmanlaşmış ekipler oluyor.

Middleware ekipleri: Spring, Integration, Messaging Queue, J2EE, Web sunucuları, ESB, SOA mimarileri, Web Servisleri gibi business katman konusunda uzmanlaşmış ekipler

Veritabanı ekipleri : MSSQL , MySQL, Oracle, NoSQL veritabanlarından, tablo yapılarından, nasıl sorgu yazılması gerekliliğinden, backup, replikasyon alma sorumluluğunda olan başka ekiplerdir.

Sistem Ekipleri: Uygulamaların kurulacağı, depolanacağı ortamların kurulması, Firewall ve güvenlik ayarlamaları, yetki ayarlamaları, Bulut ve Container ortamlarının hazırlanması sorumluluğunda olan ekiplerdir.

Microservices

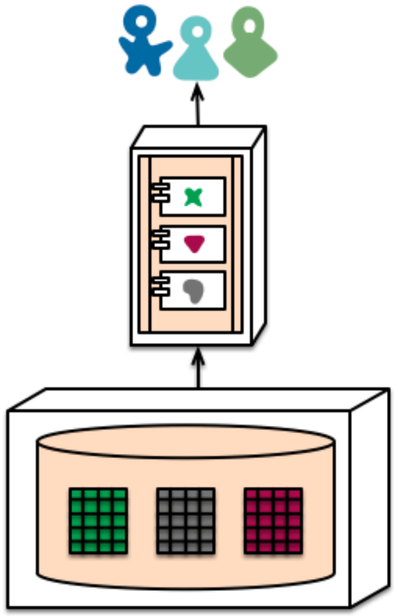
Mikroservis sadece bir işi yapan, bir fonksiyonaliteyi gerçekleştiren çok küçük kod parçacıkları. Geliştirme süreçleri, bağımlılıkları, boyutları olabildiğince küçük olan atomik servislerdir diyebiliriz.

Bu servisler herhangi bir dil ile geliştirilebilir. java, c#, ruby, phyton vb. diller kullanarak geliştirme yapabilirsiniz.

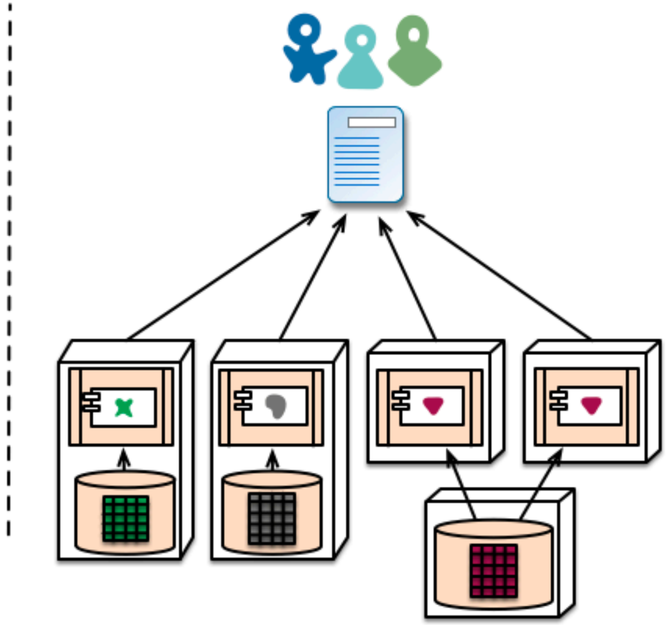
Microservisler stateless'dir. Herhangi bir nesnenin state'ini tutmaz.

Microservisler WSDL'daki XSD kontraktları yerine Consumer Driven Contract'lar oluşturarak servis sağlayanın bir kod değişiminin hangi servis tüketicilerini etkilediği , ilgili servis tüketicilerin testleri çalıştırılarak gerçekleştiriliyor.

Ekip ve Şirket yapısının değişmesi gerekiyor. Demin yukarıda bahsettiğimiz farklı konularda uzmanlaşmış birbirlerinden ayrı çalışan ekip elemanlarının (Arayuz, Spring, Veritabanı ve Sistemci) olacak şekilde ekipler haline getirilmesi gerekir ki, bu da çok maliyetli bir iştir. Bunun yerine son dönemde **Full Stack Developer** kavramı popülerleşmeye başlamıştır. Uygulama yükleme ve geliştirme araçlarının giderek gelişmesi Bulut/Container Servislerinin giderek artması, Yazılım kısmında frameworklerin artması (Spring, Hibernate, AngularJS). Artık uygulama geliştiricilerinin Full Stack Developer olmasına imkan tanımaktadır.



monolith - single database



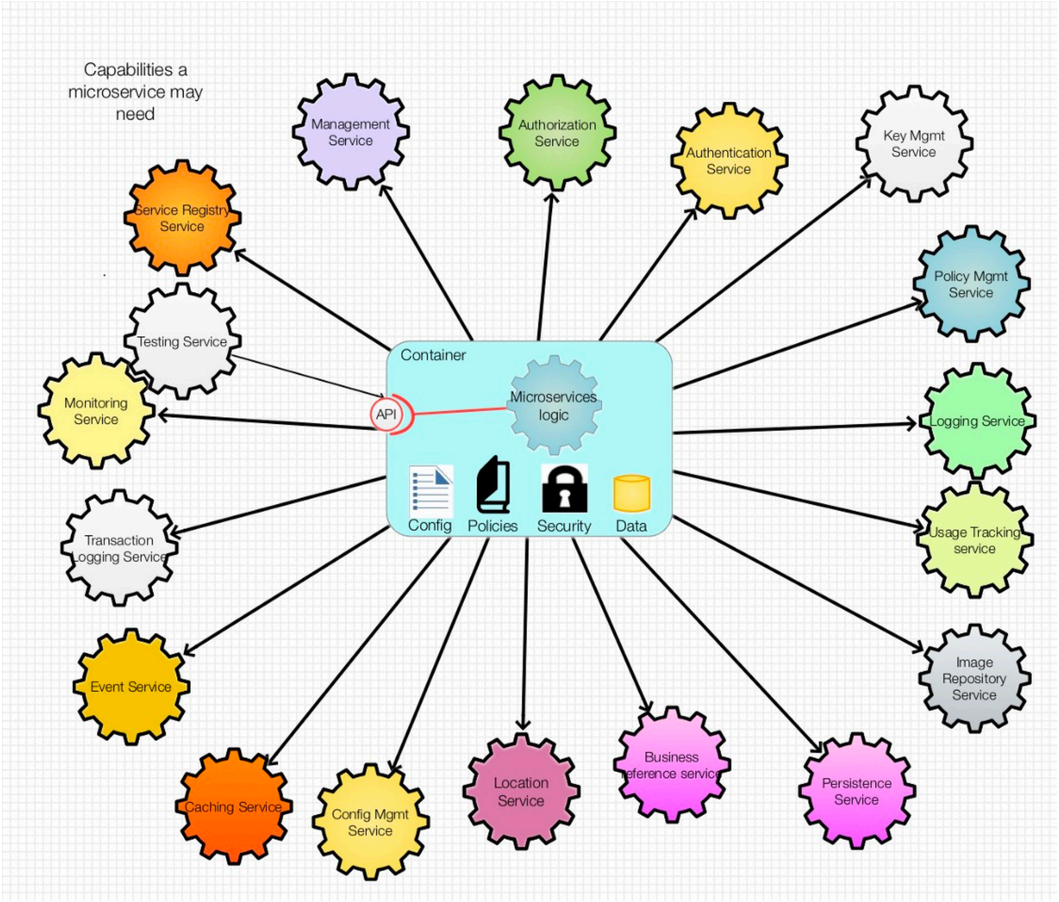
microservices - application databases

Microservices

MicroServislerin faydaları mimarinin en baştan kurulmaya çalışmaması, Ürün geliştikçe mimarinin gelişmesi. Ekibe sonradan katılan bir kişinin bu kadar büyük bir yapıya, mimariyi öğrenmesi yerine, bu kişinin hangi dilde ve hangi DB ortamında işi yapmasını biliyor ise bu ortamda bu ufak iş mantığını geliştirme imkanı vermesi. Aynı zamanda bu atomik yapıların sırasını ve hiyerarşini değiştirme imkanı vermesi, uygulamanın daha esnek, daha reusabe ve daha scalable olmasına imkan vermesi diyebiliriz.

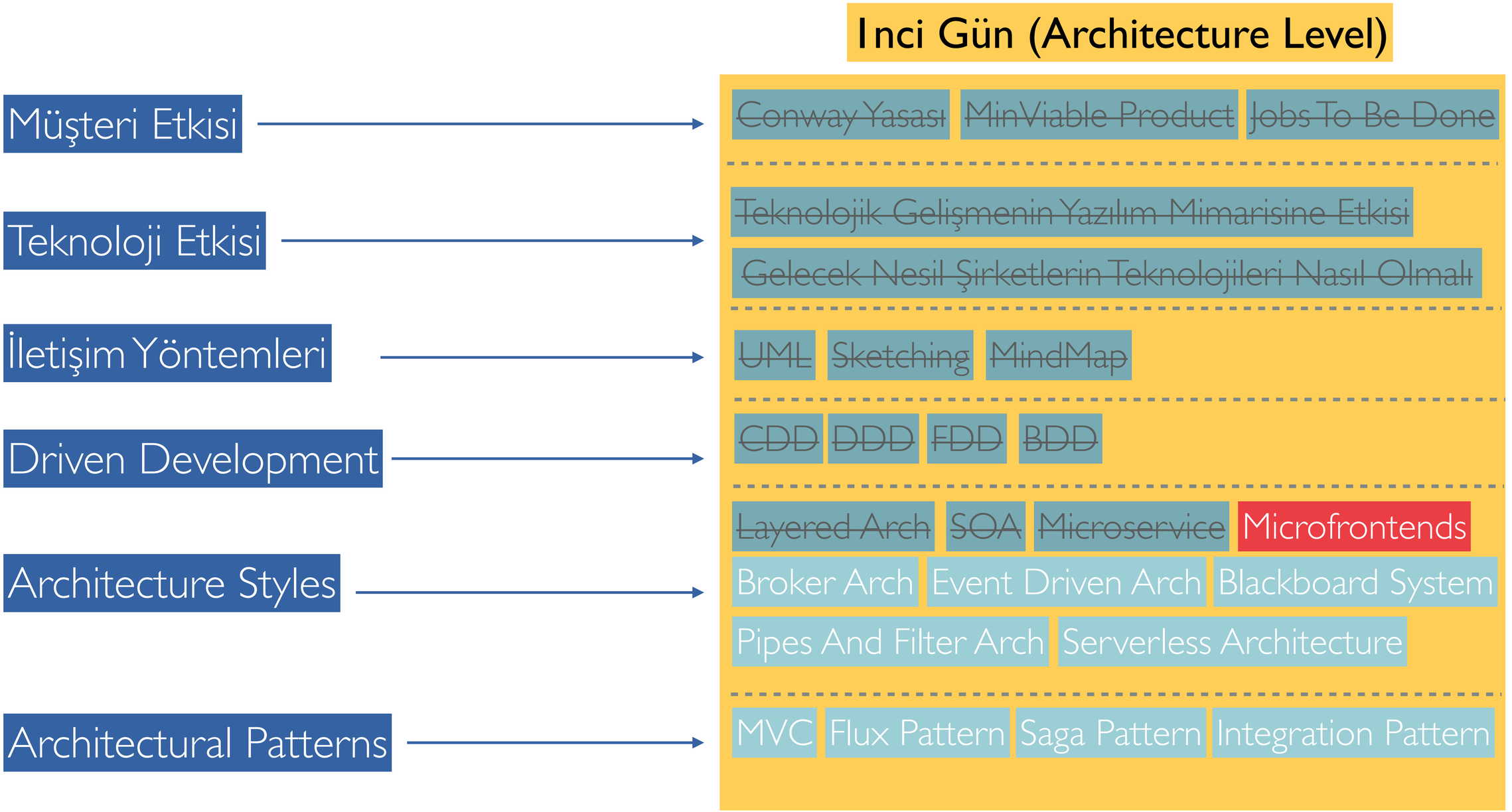
SOA mimarisinde merkezde orchestration mekanizması(ESB) ile tüm business flow'lar merkezdeki yapı tarafından organize ediliyor. Önceki yazılarımda Mediator pattern'i ile benzerlik kuran bu yapı. Giderek merkezin yükünü arttırmaya başlıyor. Hem network , hemde çalışan sorumluluğu açısından bu servisleri orkestra eden kişilere bağımlılık inanılmaz boyutlara ulaşıyor.

Microservislerin çok ufak şekilde hızlı bir şekilde gerçekleştirebilmeniz için sistem içerisinde aşağıdakine benzer mircroservislerinizin hazır olması gerekiyor.

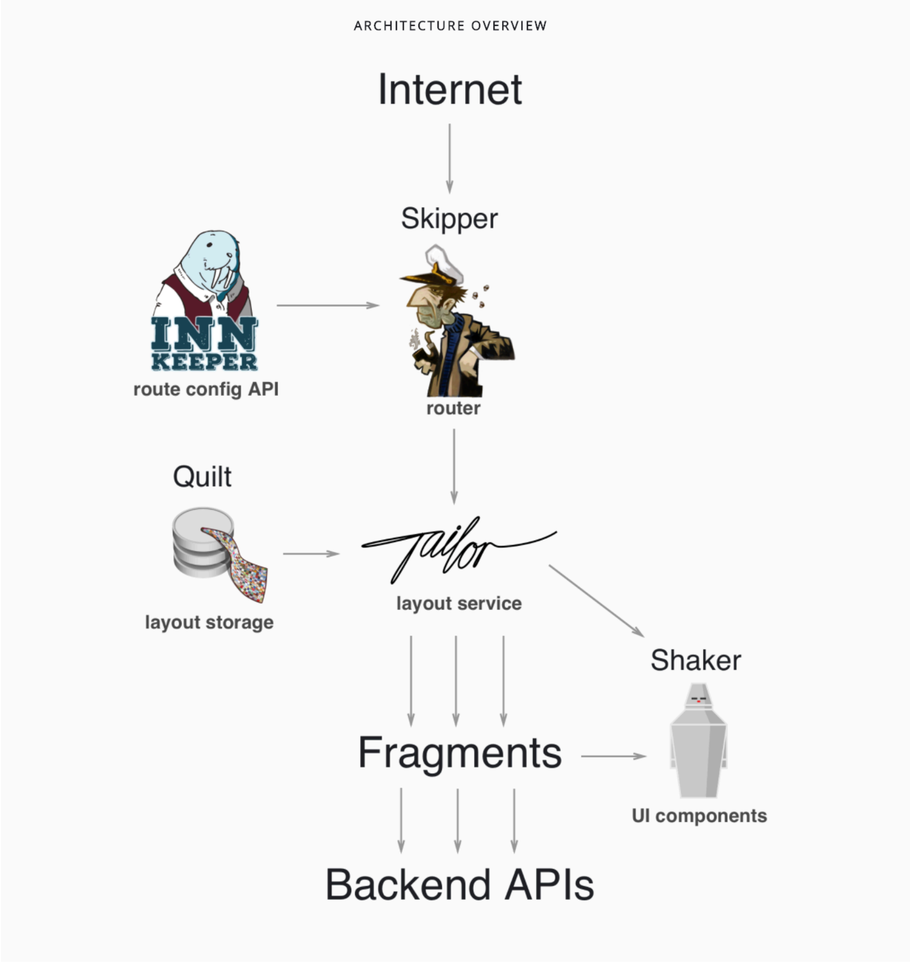


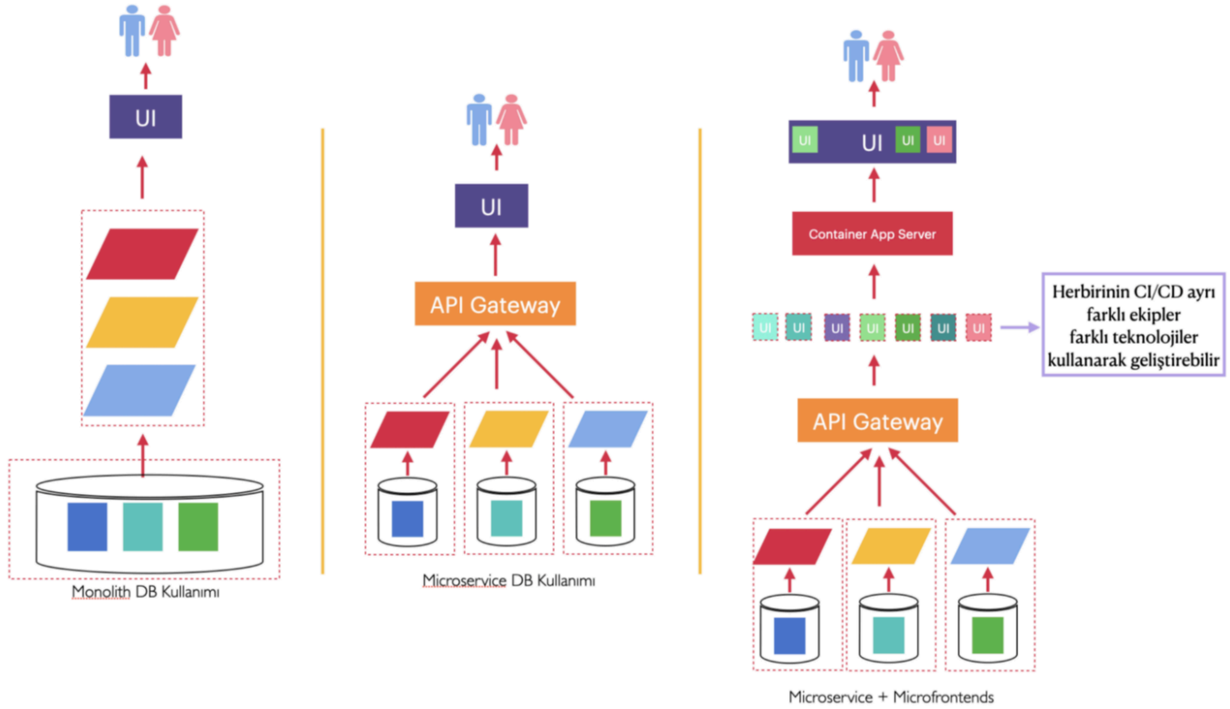
Bu kadar çok servisin bir sürü sanal makineye kurulumunun manual olarak yapılması imkansız. Bunun için "Infrastructure As a Code" IAC kavramını geliştirmişler. Infrastuctrure'ları kod ile ayağa kaldırıp uygulamaların bu infrastructure'lar üzerine hızlıca çalışmasını sağlıyorlar.

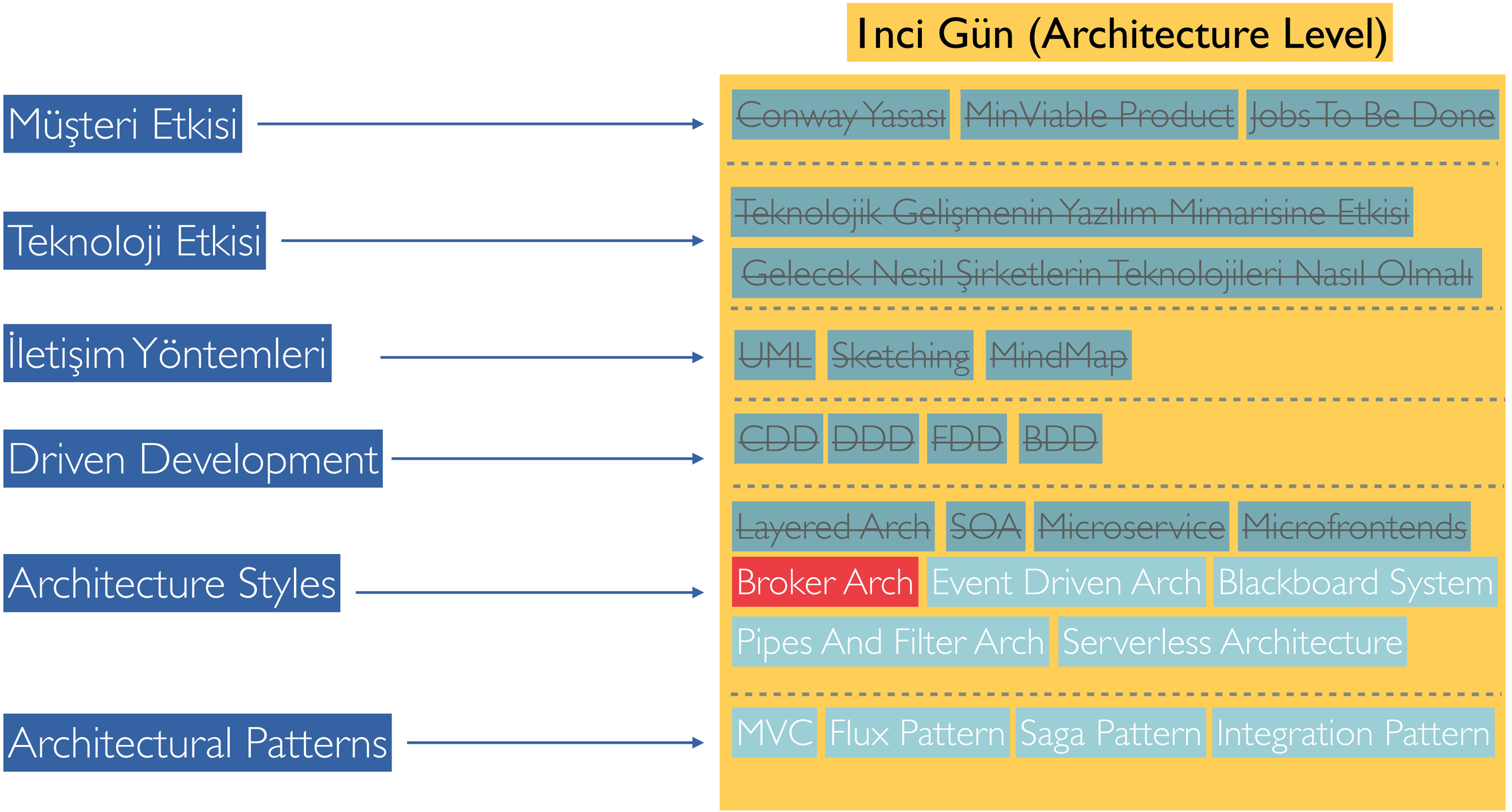
Continues Integration ve Continues Delivery kavramlarının çok iyi oturması gerekiyor. Herhangi T anında belirlenen X sunucularına, Y,Z Feature set değişikliklerini deploy edebilme yeteneğidir.


Inci Gün (Architecture Level)
Müşteri Etkisi
Conway Yasası
MinViable Product
Jobs To Be Done
Teknoloji Etkisi
Teknolojik Gelişmenin Yazılım Mimarisine Etkisi
Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı
İletişim Yöntemleri
UML
Sketching
MindMap
Driven Development
CDD
DDD
FDD
BDD
Architecture Styles
Layered Arch
SOA
Microservice
Microfrontends
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture
Architectural Patterns
MVC
Flux Pattern
Saga Pattern
Integration Pattern

Microfrontends






Inci Gün (Architecture Level)
Müşteri Etkisi
Teknoloji Etkisi
İletişim Yöntemleri
Driven Development
Architecture Styles
Architectural Patterns
~~Conway Yasası~~
~~MinViable Product~~
~~Jobs To Be Done~~
~~Teknolojik Gelişmenin Yazılım Mimarisine Etkisi~~
~~Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı~~
~~UML~~
~~Sketching~~
~~MindMap~~
~~CDD~~
~~DDD~~
~~FDD~~
~~BDD~~
~~Layered Arch~~
~~SOA~~
~~Microservice~~
~~Microfrontends~~
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture
MVC
Flux Pattern
Saga Pattern
Integration Pattern

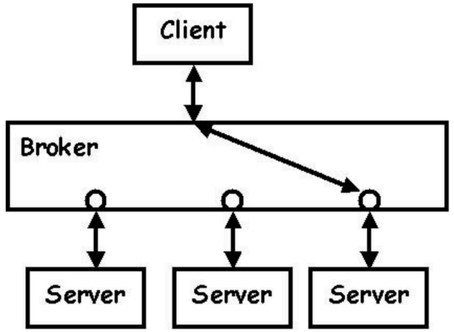
Broker

Brokerlik kavramını çoğu kişi finans sektöründe duymuştur aslında.

“Broker; finans piyasalarında alım ve satım işlemlerine belirli bir komisyon ücreti karşılığında aracılık eden, işlemleri taraf olduğ kişi yada kurumlar için görünürde kendi adıyla yapan piyasa uzmanları verilen isimdir.”

Yukarıda ki metinden de anlaşılıyor ki finans, borsa, piyasa alanında uzmanlığı olan bu kişiler sizin ile borsa uygulamaları arasında aracı olu sizin adınıza finansal işlemler gerçekleştirir.

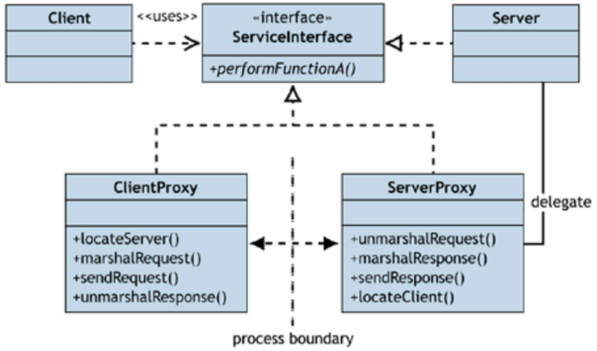
Yazılım mimarisinde de aynısı geçerlidir. Peki neden bu aşağıdaki şekild bir broker ihtiyaç vardır. Neden istemci direk sunucuya erişmezde araya broker koyarız. Broker’ın uzmanlık alanı nedir ?



Yukarıdaki resimdeki Client (İstemci Uygulama) .dır. Şimdi basitten karmaşığa doğru anlatırsak .

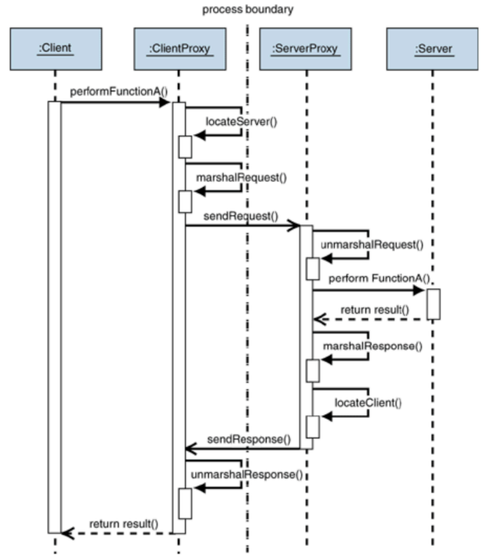
Adım 1. Dağıtık olmayan bir uygulamada istemci sunucudan bir işi gerçekleştirmesini ister. Hiç bir problem yok.

Adım 2. Dağıtık uygulamada araya katmanlar giriyor çünkü isteğin sunucuya taşınması gerekir. Aynı zamanda sunucudan dönen cevabın istemciye taşınması gerekir. Bu taşımaların gerçekleştirilebilmesi, bağlantıların kurulabilmesi için aralarda ClientProxy ve ServerProxy inin yer alması gerekir.



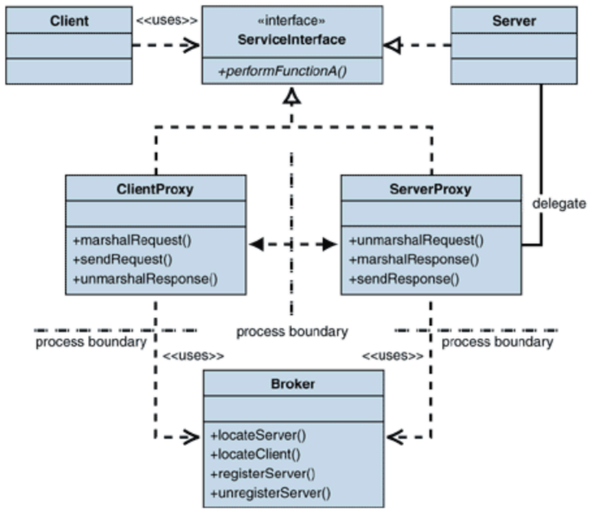
Proxy’lerin Görevleri : WebService’lerde(SOAP) bu şekilde çalışır. Sunucunun yerini tespit etmek. Mesajın dönüşümlerini marshall/unmarshall edilmesinden sorumludur. Aşağıda akışı görebilirsiniz. Aslında oldukça zahmetli bir olay dağıtık çalışmak.

- sunucunun bulunması
- entity’den xml dönüştürme
- network’den gönderme
- bunun sunucu tarafından alınması ve cevaplanması



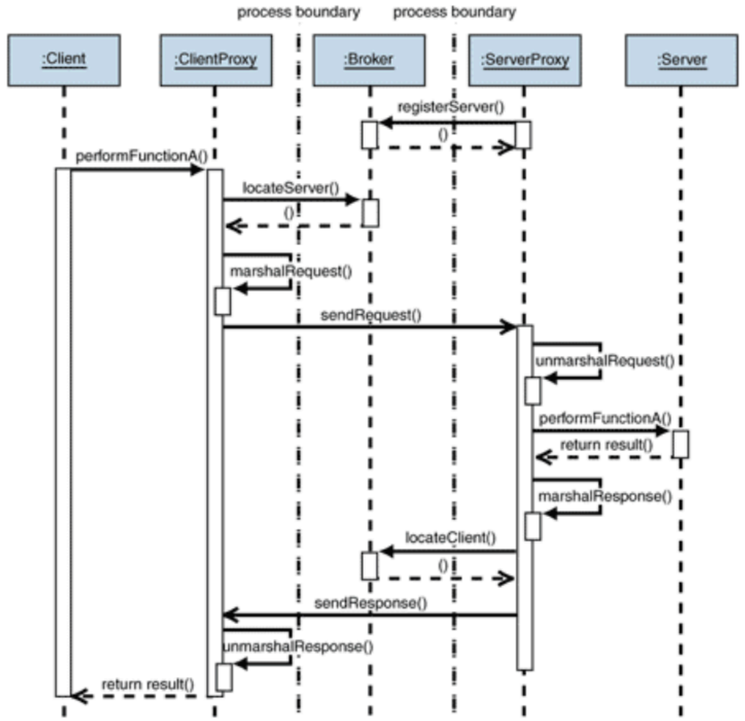
Broker

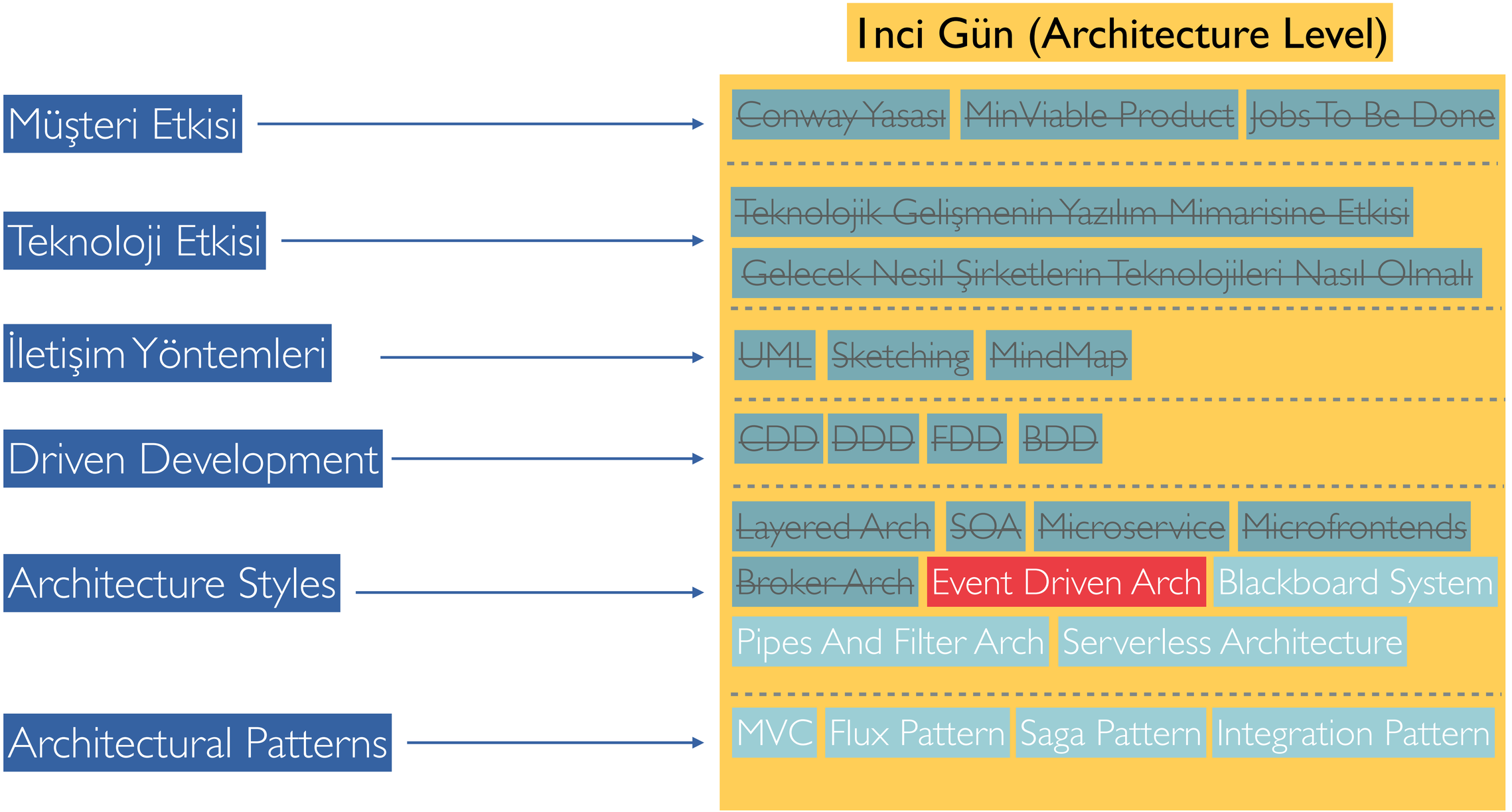
Adım 3. Bir önceki adımda her şey güzel gibi çalışıyordu. Fakat Proxy'ler birbirlerini network'de bulma gibi sorumlulukları var ve bunu compile time'da gerçekleştiriyorlar. Run-time sunucunuzun yerini değiştirmek istiyorsunuz. Veya yeni bileşenler eklemek istiyorsunuz. vb.. Bu esnekliği 2nci adımda kaybediyorsunuz. Bunun için bize arada bu işi çözümlüyecek bir broker gerekiyor.



Yukarıdaki resimde Broker'ın sunucunun yerini bulma ve birbirlerine register olma işlemlerini üzerine almış olduğunu görebilirsiniz. Bu sayede runtime'da birleşme/ayrılma ve yer değiştirme işlemleri yerine getirebiliriz.

Yukarıdaki resimde Broker'ın sunucunun yerini bulma ve birbirlerine register olma işlemlerini üzerine almış olduğunu görebilirsiniz. Bu sayede runtime'da birleşme/ayrılma ve yer değiştirme işlemleri yerine getirebiliriz.




Inci Gün (Architecture Level)
Müşteri Etkisi
Teknoloji Etkisi
İletişim Yöntemleri
Driven Development
Architecture Styles
Architectural Patterns
Conway Yasası
MinViable Product
Jobs To Be Done
Teknolojik Gelişmenin Yazılım Mimarisine Etkisi
Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı
UML
Sketching
MindMap
CDD
DDD
FDD
BDD
Layered Arch
SOA
Microservice
Microfrontends
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture
MVC
Flux Pattern
Saga Pattern
Integration Pattern

Event Driven Architecture

Event-Driven Mimari Stili 2 ana topolojiden oluşur

- Mediator
- Broker

Mediator topolojide olaylar(events) önce merkezi bir mediator bileşeni tarafından ele alınıp çoklu adımları olacak şekilde orkestra edip yeni event serisi oluşturur, **Broker topolojide** ise zincir olaylar şekilde merkezi bir mediator kullanmadan arka arkaya işletilir.

1. Mediator Topoloji

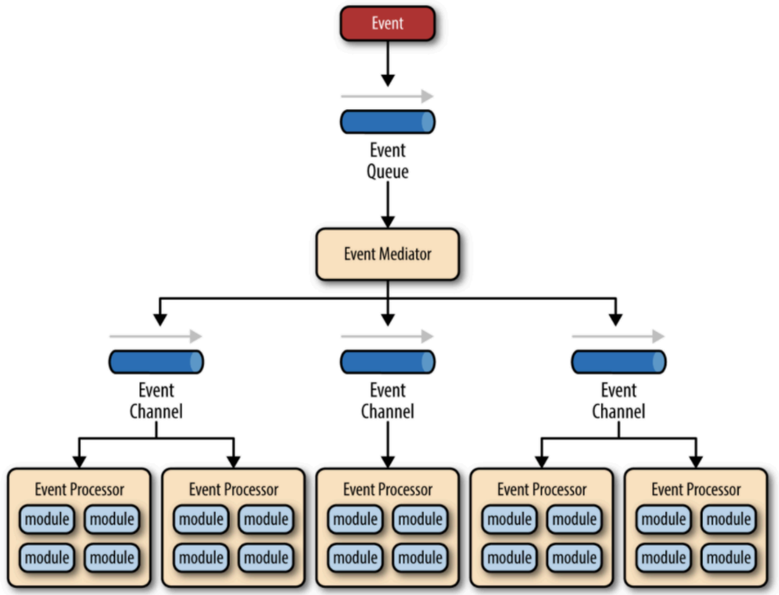
Merkezi bir yerde gelen eventler ele alarak bunlar ile ilgili yapılacak adımları orkestra edip bunların seri ve paralel olarak işleten yapıya mediator topoloji mimarisi denir .

Örneğin hisse senedi almak istiyorsunuz. Gittiniz ekrandan senedi seçtiniz ve al dediniz. Al Eventi arkaplanda Mediator tarafından bir dizi olay adımına dönüştürülür ve çalıştırılır.

- Alışverişi doğrulamanız.
- Uyum kurallarını kontrol etmeniz
- Komisyoncuya atamanız
- Komisyonu hesaplamanız
- Alışverişi tamamlamanız

Tüm bu adımların hangi sırada olacağı bu adımların seri mi yoksa paralel mi işletileceğini mediator bilir.

Mediator topoloji 4 mimari bileşenden oluşur;



https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html (Mediator)

- **event queue (olay kuyruğu):** Olayları event mediator taşıyan yapıdır.
- **event mediator:** başlangıçtaki işlenmemiş olayı alıp buna orkestra ederek yeni eventlerde ekleyerek asenkron olarak event channel bu dizi eventi yollayan bileşendir
- **event channel (olay kanalları) :** Burda artık olaylar belli işleyiciler tarafından işlenmesi için ilgili kanallara aktarılır.
- **event processor:** Kanallardaki eventleri dinleyerek bunları işletirler.

Event Driven Architecture

Bu mimari event queue(olay kuyruklarını) nasıl oluşturacağınız konusunda, kaç tane olması gerektiği konusunda bir sınırlandırma ve tanımlamada bulunmaz. Bu bir mesaj kuyruğu, web servis endpoint, Restful servis endpoint vb kombinasyonlarda olabilir.

2 tip event mevcut, birincisi ***initial event*** yani dış kaynaklardan ilk event mediator giriş yapan işlenmemiş eventler birde event mediator tarafından ***processing event*** olarak oluşturulan processor/worker bileşenlerin alıp işleyeceği eventler.

Event mediator bileşeni initial eventi alarak ***business logic*** (iş mantığının) detayında neler yapılacağını bilmeden üst seviye adımları bilecek şekilde processing eventleri oluşturan mekanizmadır. Bu bişeni kendiniz geliştirebilirsiniz veya açık kaynaklı hazır kütüphanelerden faydalanabilirsiniz. Bunlar Enterprise Integration Pattern gerçekleştirimlerini içeren kütüphanelerdir. (Detay için bu konudaki yazımı okuyabilirsiniz)

- Spring Integration Framework
- Apache Camel
- MuleSoft

Burada daha önceden anlatmış olduğum SOA kısmındaki ESB(Enterprise Service Bus) Mediation/Orkestrasyon işlemini gerçekleştirildiği kısımlardır. Olay akışlarının genelde hard coded veya DSL(Domain Spesific Language) gerçekleştirilir. Çok karmaşık iş akışları içinse BFEL((business process execution language)) gibi XML tabanlı diller ile yazılarak gerçekleştirilir. Bazı iş akışlarında ve orkestrasyon adımlarında insanlar ile etkileşim (imza/onay/yönlendirme) ihtiyaçları olabilir bunun içinde BPM (Business Process Manager) kullanılır.

Event processor bileşeni uygulama iş mantıklarının içeren, bağımsız, sadece kendi işini yapmaya odaklanmış bileşenlerdir . Bu bileşenlerin büyüklüğü yapacağı işin büyüklüğü ve karmaşıklığına göre değişiklik gösterir. Alışveriş üzerindeki vergiyi hesaplayan bileşen büyüklüğünden , sigorta için verelicek poliçe fiyatını hesaplayacak bileşen büyüklüğü gibi;

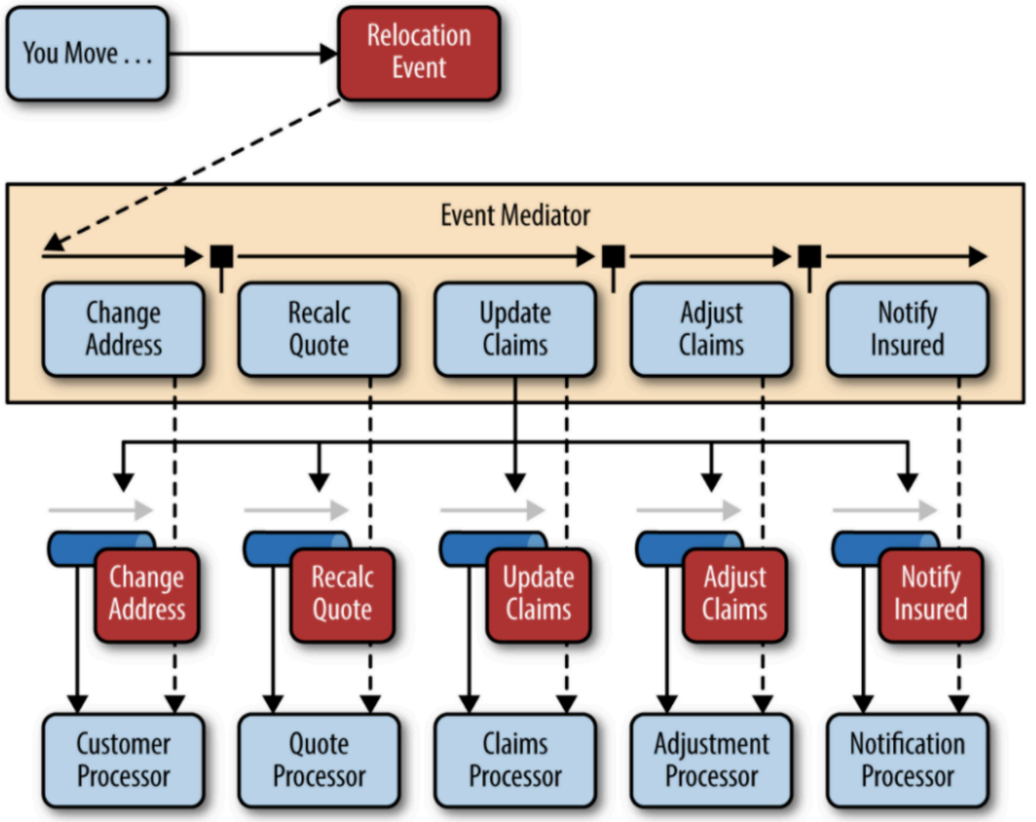
- ***f(order)***= order * 0.18
- ***fInsurance(yaş, cinsiyet, hastalık, şehir …)***= algorithm, decision tables …

Burada asıl önemli olan bileşenlerin diğer bileşenlere ihtiyaç duymaması ve tek bir işe odaklanarak bunu gerçekleştirmesidir.

Event Driven Architecture

Aşağıdaki örnekte Sigortacılık anlamında bir kişinin adresinin değişmesinin sigorta sistemine nasıl yansıtıldığı anlatılmış..

- **Adım 1:** Relocation (Yer değiştirme ***initial eventi*** oluşturuyor)
- **Adım 2:** ***Event mediator*** bu eventi alarak kendi içerisinde 5 tane processing event adımı oluşturuyor (Adresi değiştir, Hesaplamayı yenile, Sigorta taleplerini güncelle, talepleri ayarla, ve bu güncellemeler ile ilgili uyarı yap)
- **Adım 3,4,5…..** ilgili işlemciler kendileri ile ilgili processing eventleri işletir.



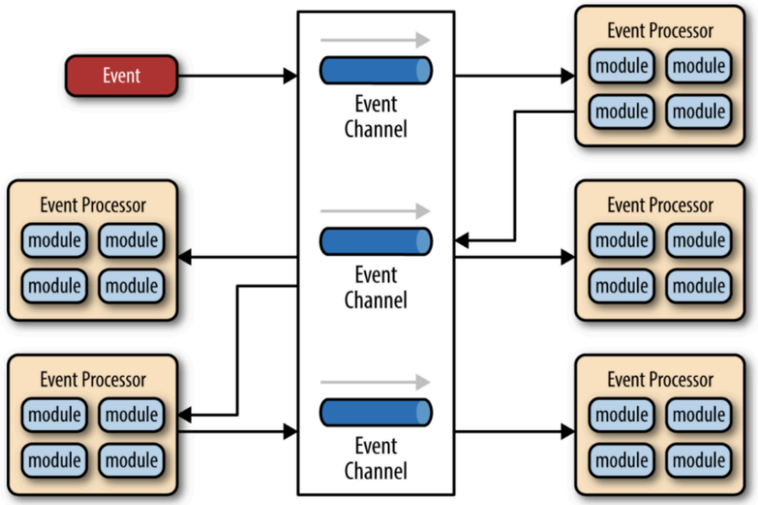
https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html (Mediator)

Event Driven Architecture

2. Broker Topoloji

Broker Topoloji'de merkezi bir ***event mediator (yönetim)*** mekanizması yerine mesaj akışının dağıtık olarak event processor üzerinden zincir şeklinde yapılan daha lightweight mesaj broker olan (ActiveMQ, RabitMQ, HornetQ) gibi sistemler kullanılır. Bu topoloji basit olay/iş akışlarının olduğu merkezi bir olay orkestrasyonuna gerek olmayan mimarilerde tercih edilmelidir.

Broker topoloji 2 mimari bileşenden oluşur;



https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html (Broker)

Broker: Broker merkezi veya gruplar halinde iş akışında kullanılan tüm olay kanallarını (event channels) içerirler. Bu event channel mesaj kuyruğu, mesaj topic veya bunların birleşimi olabilir.

Event Processor: Biraz önce yukarıda da bahsettiğim iş mantıklarını gerçekleştiren işlemci/işçilerdir.

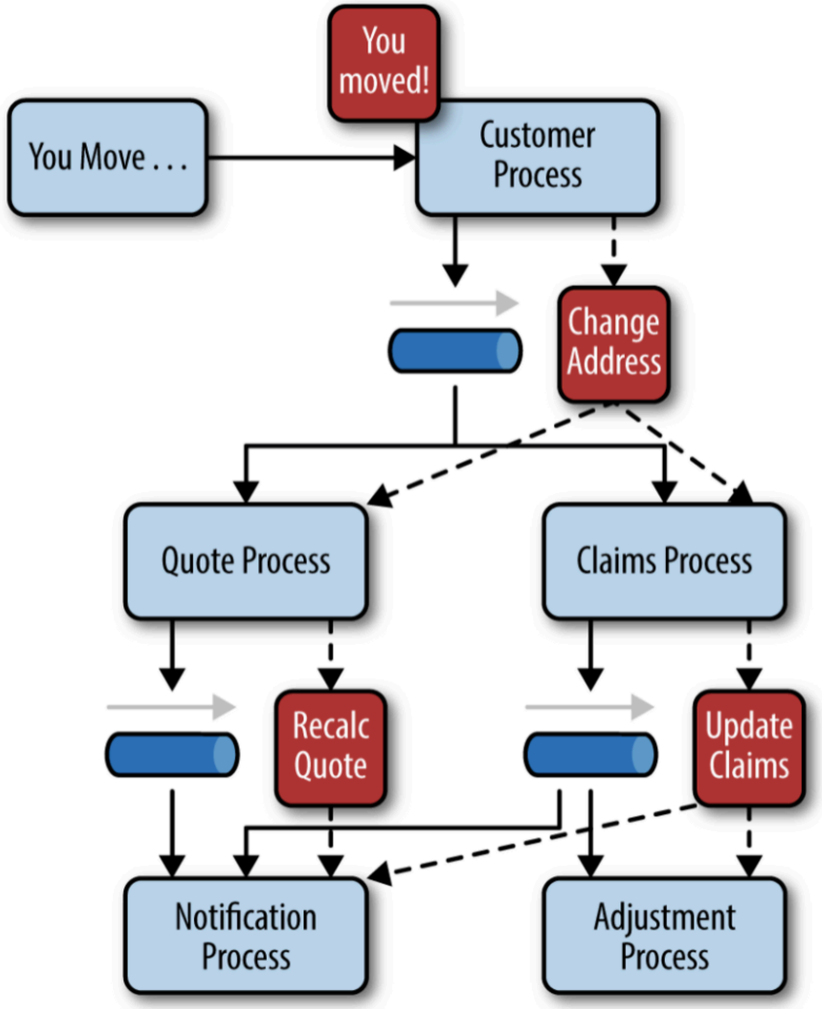
Bu mekanizmada merkezi akıllı bir orkestrasyoncu yerine bileşenler initial event işledikten sonra yeni bir event oluşturarak bunu broker iletir. Bunu başka bir bileşen alarak ilerletir. Burda ilerletilen eventleri bir processor alıp işletmesi ve ilerletmesi temel önceliktir.

Event Driven Architecture

Aşağıda aynı sigortacılık yer değiştirme örneğinin Broker topoloji ile nasıl yapıldığını görebilirsiniz. Bu sefer Processor kendilerin yaptığı işlemden sonra ortaya attığı eventleri diğer processor bileşenlerinden ilgilileri alıp işletir. Bunların seri veya paralel olacağı dağıtık bir şekilde bileşenlerde durur. Bundan dolayı çok karmaşık iş/olay akışları olan mimari için broker mimarisi çok uygun bir yapıda değildir.

Broker mimarisinin işleyişini anlatan en iyi görsellerden bir taneside bayrak yarışıdır. Processor bir tanesi bayrağı diğer processor vermek için taşır sonra bir diğeri …

Usain Bolt of Jamaica accepting the baton to go on to win gold at the Olympics 2016



https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html (Broker)

Event Driven Architecture

3. Avantaj ve Dezavantajları

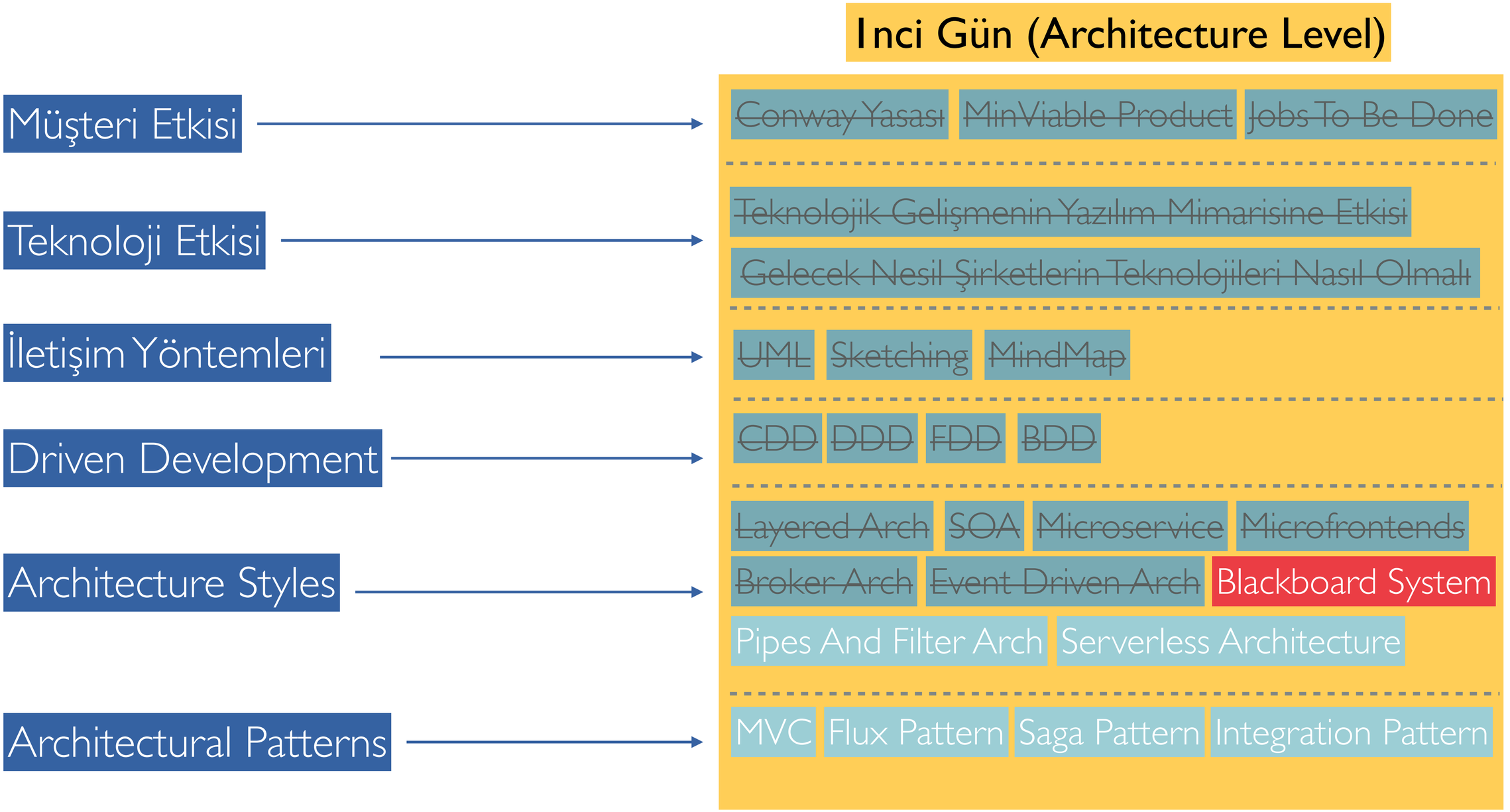
Event Driven Mimari stili **avantajları**;

- Ölçeklenebilir, Dağıtık Çalışabilir, Yüksek performanslı, Çevik , Her bir parçanın ayrı ayrı yüklenebilmesi, bileşenlerin sadece kendi mantıklarını bilmesi ve buna göre test yapılması açısından oldukça güzel bir mimari stildir. Bundan dolayı zaten dağıtık yapılarda sıkça tercih edilir.

Ama bazı ***dezevantajlarıda bulunur***;

- Consistency(tutarlılık), atomicity (atomik), sistemin takip edilebilirliği, bir hata oluştuğunda yapılan işlemlerin geri alınması, tüm sistemin testi vb..

Inci Gün (Architecture Level)



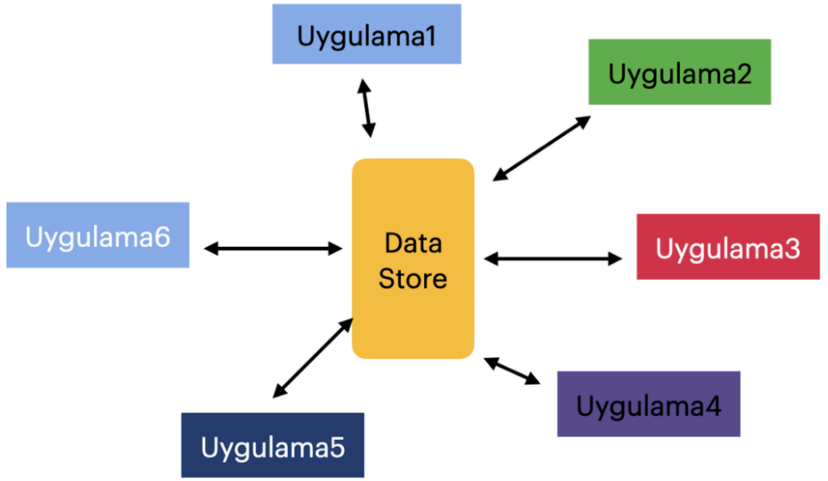
Blackboard

Blackboard ve Repository Mimari Stillerinde ortak veritabanın bileşenler/uygulamalar sürekli güncellenen veriye erişerek elde ederler. Temel mantık çok güçlü bir veritabanı üzerinden tüm sistemin çalışması üzerine kuruludur.

Bileşenler paylaşılmış veri deposuna bağımsız bir şekilde ulaşarak veriyi değiştirirler ve okurlar.

En çok bilinen veri-merkezli mimaride ***veritabanı mimarisidir.*** Veri tanımı protokolleri üzerinden ortak veritabanları şeması oluşturulur ve bu tablolar birbirleri ile ilişkilendirir. RDBMS üzerinden her istemci kendi query/sorgusunu veya insert/update gerçekleştirerek sistemi kullanır.

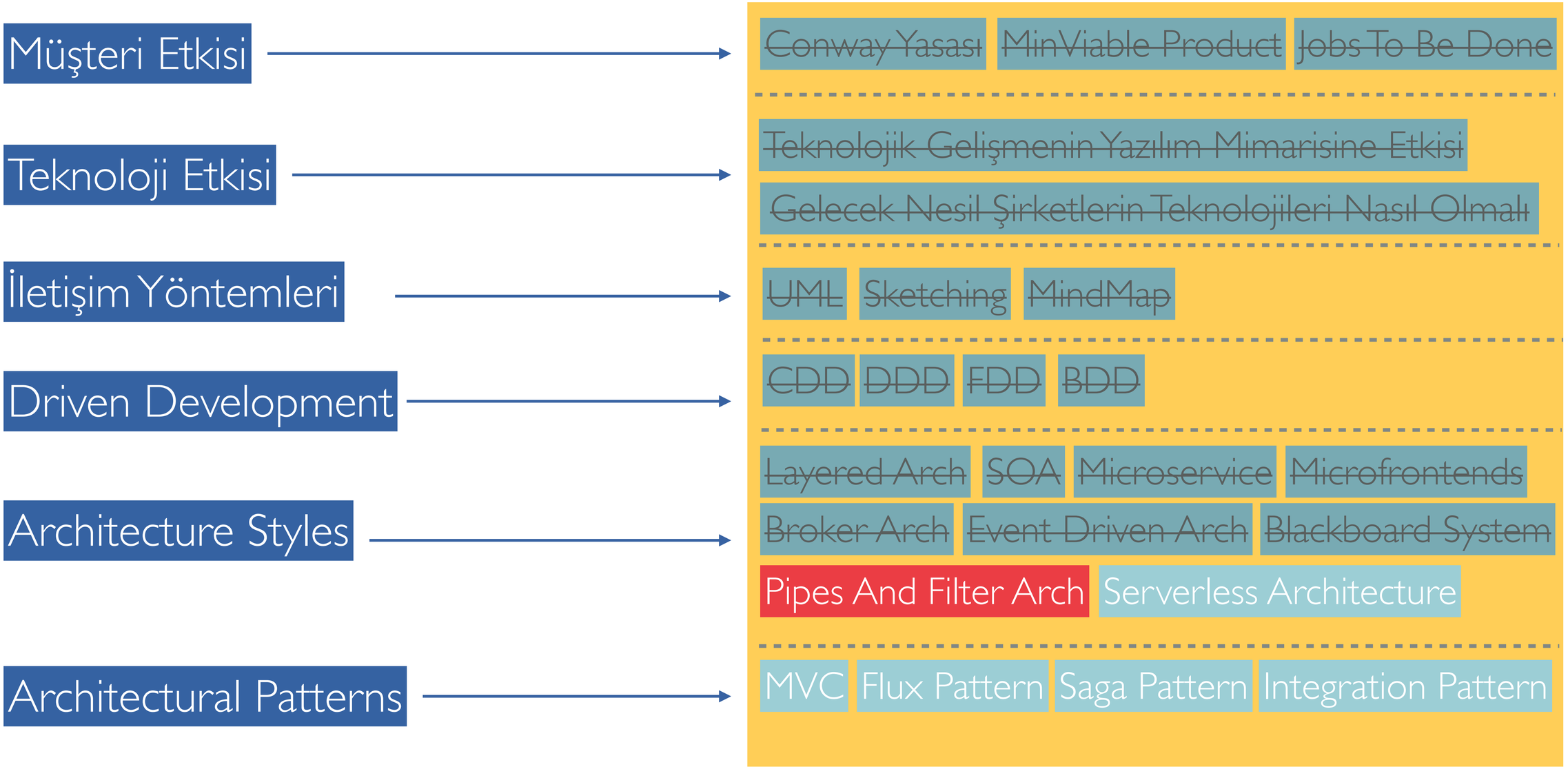
Bir diğer örnekte ***web mimarisidir.*** Ortak bir veri şeması üzerinden (Web'in meta-yapısı) hypermedia veri modeli ve iletişim yöntemleri ile web-tabanlı veri servislerini kullandırtarak ortak veri merkezi sağlamış olur.



2 tip bileşen var. Bir ***merkezi veri*** deposunu mevcut durumu ve kalıcı veriyi tutan billeşen, İkinciside ***veri erişimcisi ,*** merkezi veri deposundaki veriye erişip, bunu işleyen ve geri koyan bileşen

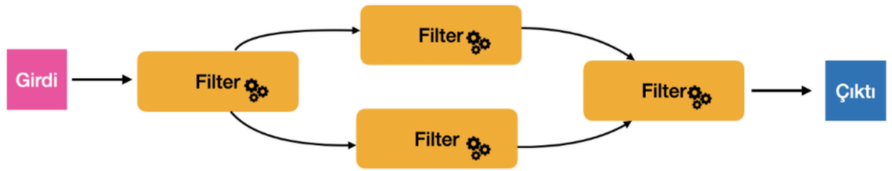
Uygulama bileşenlerinin merkezi veri deposu ile iletişim yöntemine göre ayrılan 2 tip mimari stil var. ***Repository Mimari ve Blackboard Mimari***

Inci Gün (Architecture Level)



Pipe And Filter

Kanallar üzerinde verinin aktığını düşünün (stream data) ilerleme sırasında bu veri üzerinde sırası ile farklı farklı işlemleri gerçekleştiren filtreler/workers bulunur. Bunlar veriyi değiştirip bir sonraki aşamaya yönlendirir. Bu sayede sadece belli işleri yapan birbirine bağımlı olmayan bileşenlere sahip olursunuz.



Pipe And Filters

Burda kanallar bir veritabanı, kuyruk vb ilerleyen veriyi kaydedip bir sonraki aşamaya iletebilecek yapılar olmalıdır. SQL veritabanları, ActiveMQ, RabbitMQ veya Kafka benzeri araçlar bu kanalları oluşturabilirsiniz. Burda Filtreler birbirine bağımlılıklarını ortadan kaldırmış olursunuz. Hepsinde kendisi ile ilgili veri ile ilgilenir, ***sistem paralel*** çalışabilir. High cohesion, low coupling sağlayabildiği için çokça tercih edilen ve kullanılan bir mimaridir.

Görüldüğü gibi Compiler, Unix Shell, Analitik Veri İşleme gibi verinin bir yerden akmaya başladığı ve yol boyunca farklı aşamalardan geçmesi gereken sistemlerde bu mimariyi tercih edebilirsiniz. İnteraktif yani kullanıcı etkileşimi olan yapılar için uygun değildir. Burada amacımız performans ve filtrelerde bulunan farklı iş mantıklarını birbirinden soyutlayarak büyük sistemleri geliştirebilme imkanı sağlarız.

Filtreleri yazacak ekipleri ve kişileri ayırabiliriz. Kodlama dilleri birbirinden farklı olabilir örneğin Görüntü İşleme, OCR gibi kısımlar C++ gibi kütüphaneler ve diller ile yapılırken, burdan çıkan verilen Makine Öğrenmesi kısmında Python ile gerçekleştirilebilir.

Burdaki Worker/Filteler ayrı makinelerde, Kubernetes üzerindeki Docker olabilir, Serverless FaaS Fonksiyonlar (Lambda ..) olabilir, PaaS üzerinde ElasticBeanstalk üzerinde Worker olabilir.

Veya aynı makinedeki farklı process, thread veya fonksiyonda olabilir. Bu tamamen sizin tasarımınıza ve ihtiyaçlarınıza bağlıdır.

Mantıksal parçaları bu şekilde birbirinden ayırmak size ilerde yük geldiğinde sistemi ölçeklendirebilmek ve monolith kocaman bir yapı olmaktan kurtarır.

Pipe And Filter

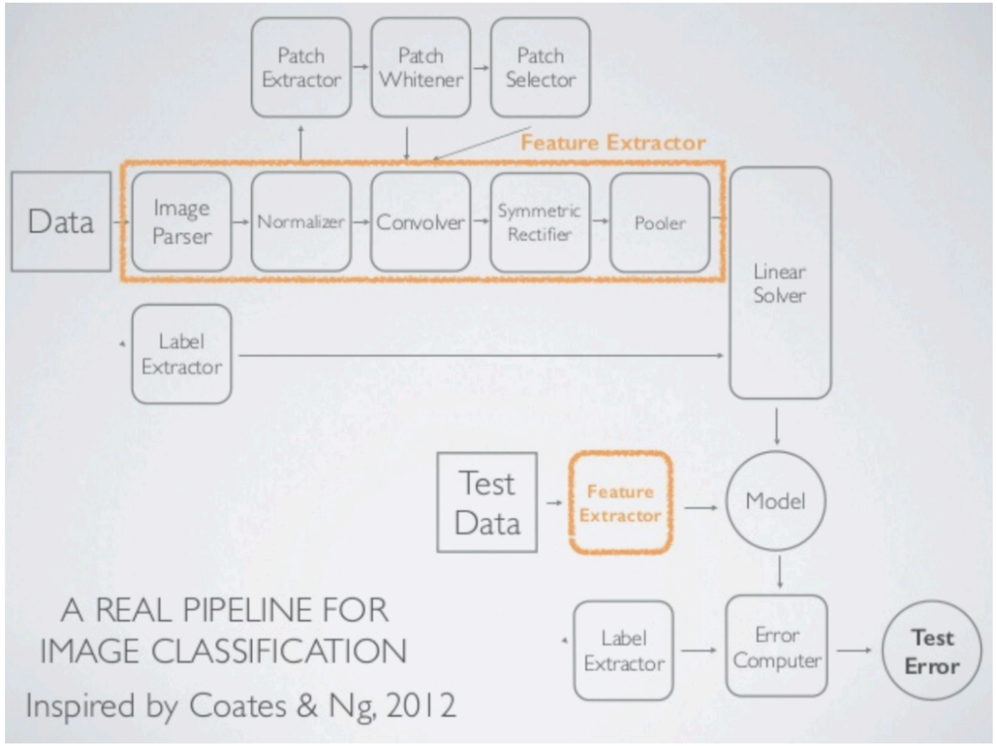
Compiler

Compiler geliştiricilerin yazmış oldukları kodu makine diline çevirirler. Bu sayede bizde yazılımı daha soyut bir şekilde düşünüp geliştirebiliriz. Compiler bu işlemi yaparken bizim yazdığımı ASCII Text kodlarını sırası ile Lexical Analysis → Syntax Analyzer → Semantic Analyzer → ve Code Generator geçirir. Burde her bir filtre bir öncekinin çıktısını bekler ve kendinin sorumlu olduğu kapsam girdiyi işler.

Compiler Pipe And Filter

Makine Öğrenmesi Pipeline

Feature Extractor kısmındaki ImageParser → Normalizer → Convolver → Symmetric Rectifier → Pooler Worker işlenerek veri ilerler



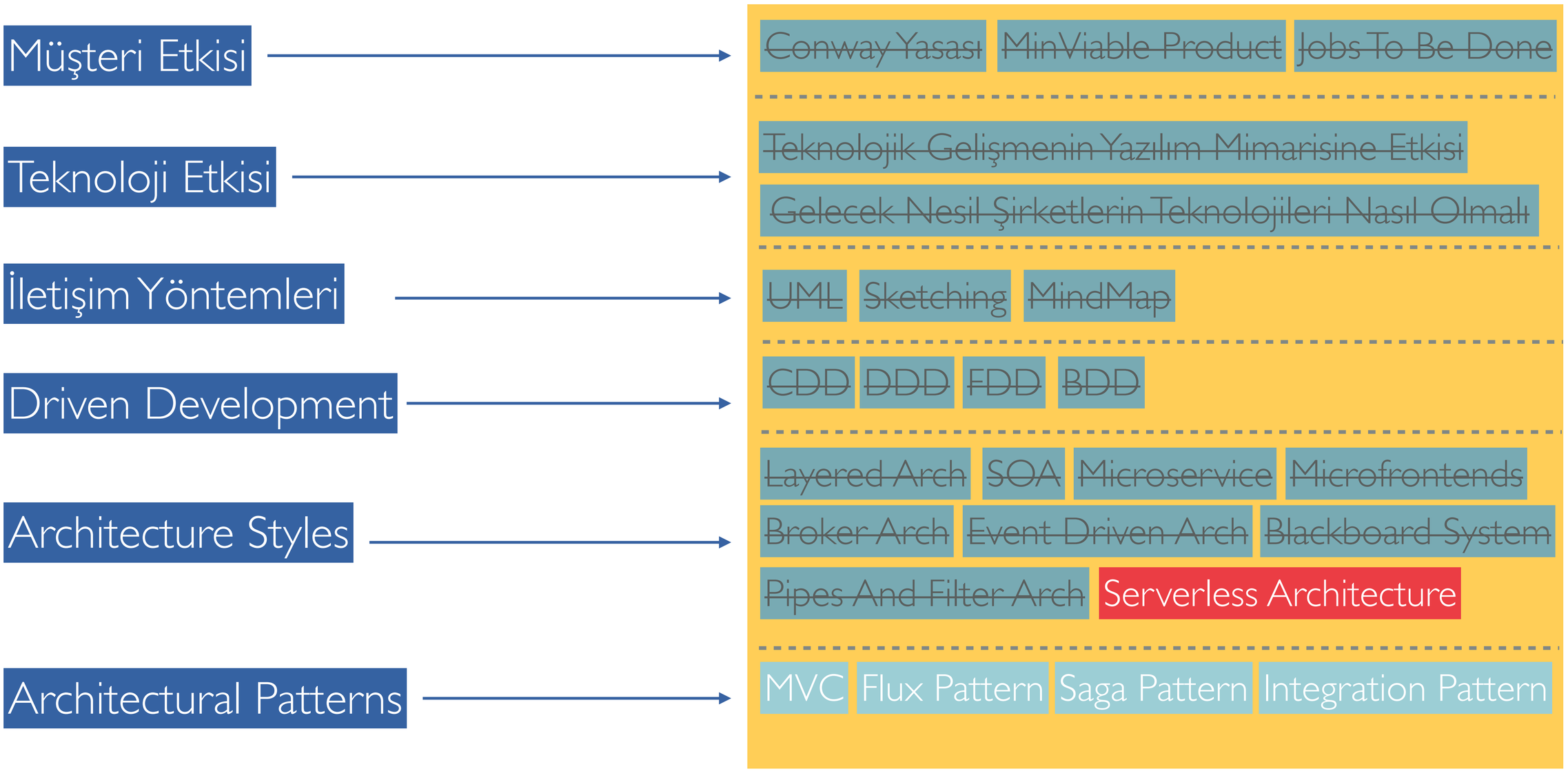
https://www.slideshare.net/jeykottalam/pipelines-ampcamp

Pipe And Filter

Peki bu mimarinin hiç problemli yan etkisi yok mu. Hem performansı arttırıyor hemde High Cohesion, Low Coupling sağlıyor. Ama dikkat edilmesi gereken kısımlar var;

- Öncelikle her yerde kullanabilir değil. Akan bir veriniz ve bu verinin yol üzerinde parçalar halinde işlenme mantığında olması lazım.
- Filtrelerin farklı farklı sistemlere dağıtılabilme kolaylığı ve çok küçük yazılmaları durumunda çok fazla filtrenin olması ve bunların bir karmalıklığı yol açması
- Kanallardaki verinin kaybolmaması lazım. Bunun için bellekte veya kapanınca verinin kaybolacağı sistemler yerine veritabanı ve kuyrukları tercih edin
- ***Idempotency Patterns ,*** İleti üzerinde işlem yaparken bir worker/filtre hata alması durumunda , diğer başka bir worker aynı iletiyi alıp aynı işlemi 2 kez yapmaya çalışırsa örneğin veritabanında X sayısını 10 kuruş arttırmak yerine 20 kuruş arttırır. Bu nedenle kanal üzerindeki filtrenin ileti için sadece 1 kez çalıştırılacak şekilde geliştirilmelidir.
- Kanallar üzerinde aynı iletinin 2 veya daha fazla kopyasının oluşturulması engellenmelidir.
- İş hattındaki her filtre çevresinden yalıtılmış olarak şekilde çalışmaktadır. Bundan dolayı ihtiyaç duyduğu Context(Bağlam) ya bu iletilerle taşınacaktır. Context çok büyük boyutlarda olduğu durumlarda Filtre network üzerinden veya kendi dosya sistemi üzerinden I/O işlemi ile bağlamın eksik kısımlarını almaya çalışacaktır. Bu da performans konusunda ekstra maliyet anlamına gelir.

Inci Gün (Architecture Level)



Serverless

FaaS (Function As A Service) nedir ?

Biraz daha derine inerek Eventlere takılabilen Fonksiyonları ***Function As A Service*** olarak sunacak Provider'lar çıkmaya başladı.

- Amazon Lambda
- Google Cloud Functions
- Microsoft Azure Functions
- IBM OpenWhisk
- Iron.io
- Webtask.io

Bunlar provider kendi ortamlarının eventlerine bağımlı fonksiyonlar oluşturmanızı sağlayarak kendi içerisinde sundukları servisler üzerinde programlama yababilecek, servisleri birbirine bağlayabilecek altyapıyı kurmuş oldular.

- Tomcat, Apache sunucusu yerine APIGateway kullanabilirsiniz ama APIGateway , DynamoDB/RDS servisi ile bağlayacak iş mantığını nerede kodlayacaksınız. İşte bunu FaaS/Lambda sağlıyor.
- SES(Email), Kinesis(Stream), iOT(Telemetry),S3(File),CloudWatch(Log), SNS(Topic) gelen ne olursa olsun bu verinin işlenebilmesi için bu iş mantığının kodlanabileceği bir ortam gerekiyor. İşte bunu FaaS/Lambda sağlıyor.

PaaS bu işin neresinde kalıyor ?

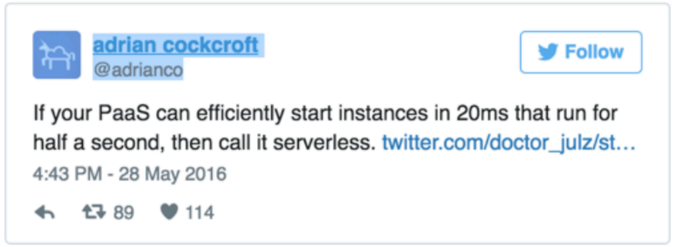
PaaS servisleride benzer bir amaçla çalışır. Server kurma, yönetme, ölçeklendirme, güvenlik maliyetlerinden sizi kurtarıp sadece uygulamanızı,iş mantıklarınızı kurmanıza odaklamanızı sağlar.

- Heroku,
- AWS BeansTalk
- Openshift
- CloudFoundary/Pivotal

PaaS ile FaaS arasındaki farkı aşağıdaki Twitter'daki aşağıdaki post özetlemiş aslında

Comparison with PaaS

Given that Serverless FaaS functions are very similar to 12-Factor applications, are they in fact just another form of 'Platform as a Service' (PaaS) like Heroku? For a brief answer I refer to Adrian Cockcroft



In other words most PaaS applications are not geared towards bringing entire applications up and down for every request, whereas FaaS platforms do exactly this.

Sunucusuz Uygulamalar (Serverless)

Analiz (Analitik)

Makine Öğrenimi

IOT

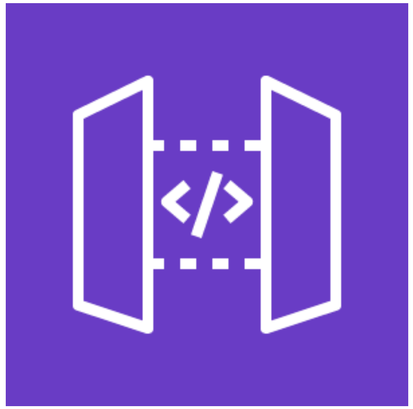
Finansal Hizmetler Sektör

High Performance Computing

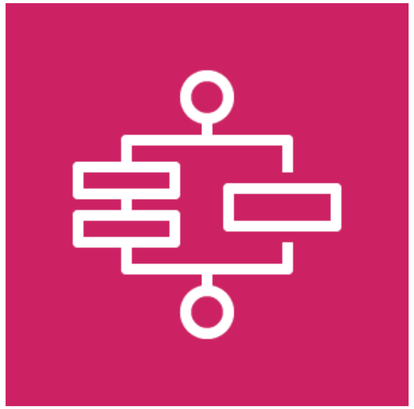
Serverless Nedir ?

COMPUTE LAYER (HESAPLAMA KATMANI)

Lambda



APIGateway



Step Functions

DATA LAYER (VERİ KATMANI)

DynamoDB
DAX
Streams

S3

ElasticSearch

AppSync

MESSAGING AND STREAMING LAYER

SQS

SNS

Kinesis
Data Analytics

Firehose

USER MNG AND IDENTITY LAYER

Cognito

EDGE LAYER

Cloudfront

SYSTEM MONITORING AND DEPLOYMENT

CloudWatch

X-Ray

SAM

Restfull Microservices

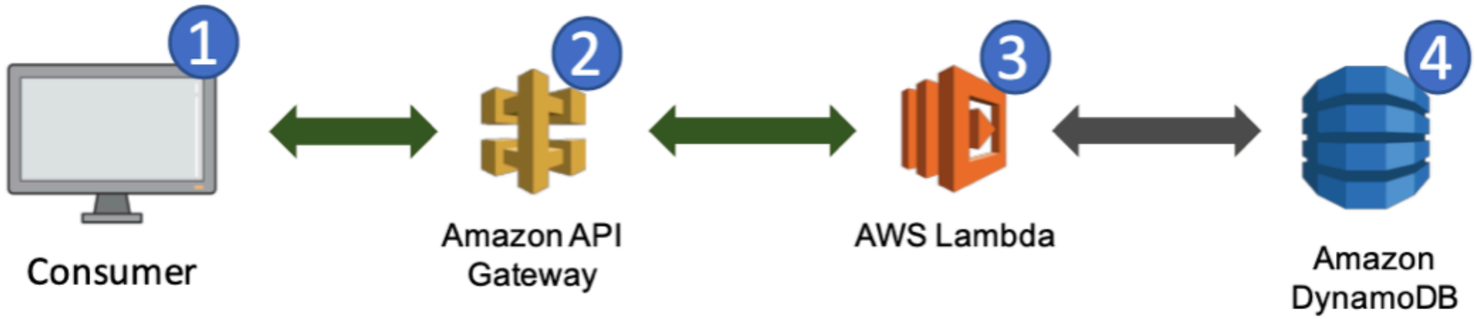


Figure 1: Reference architecture for RESTful microservices

Alexa Skills

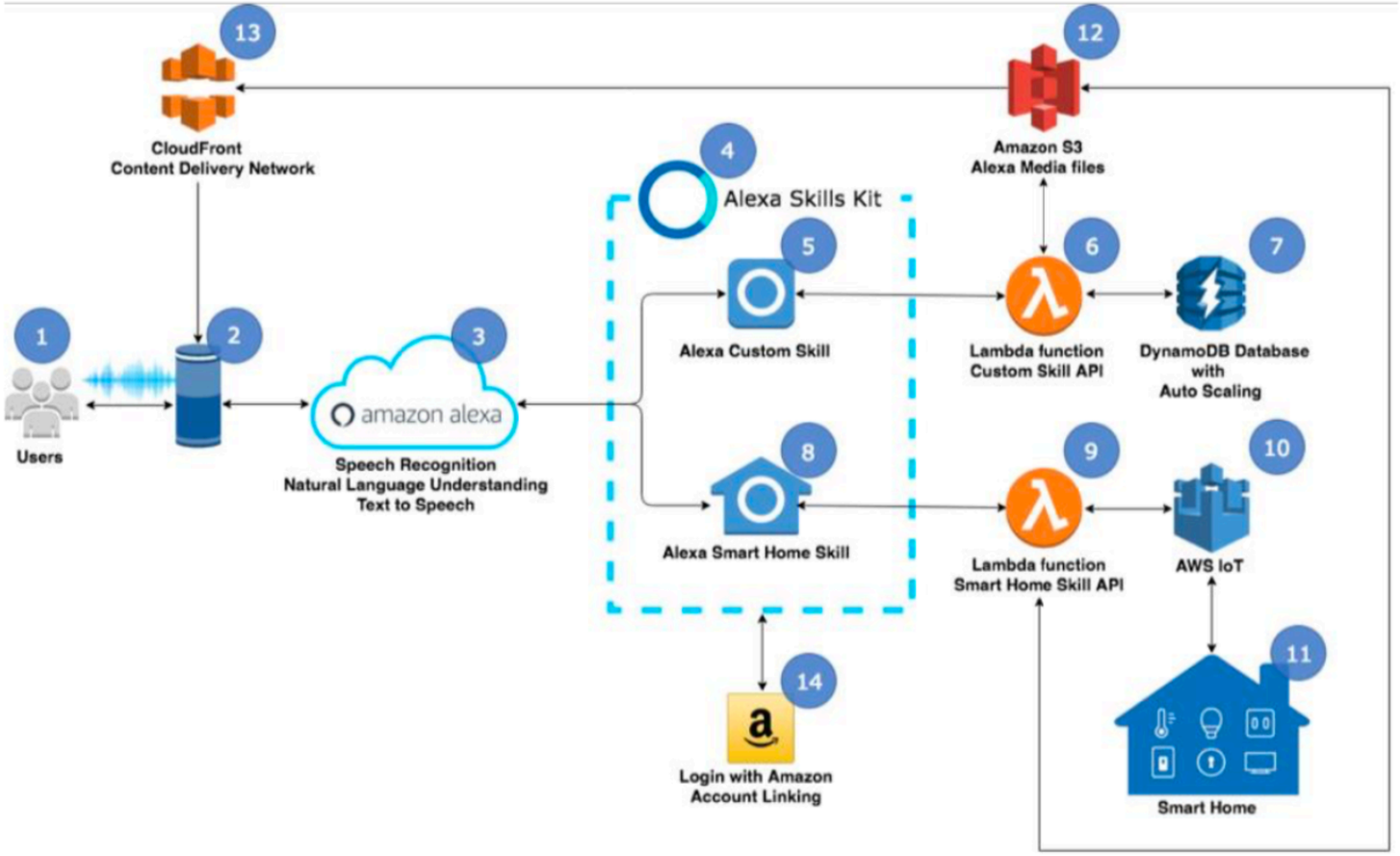


Figure 3: Reference architecture for an Alexa Skill

Mobile Backend

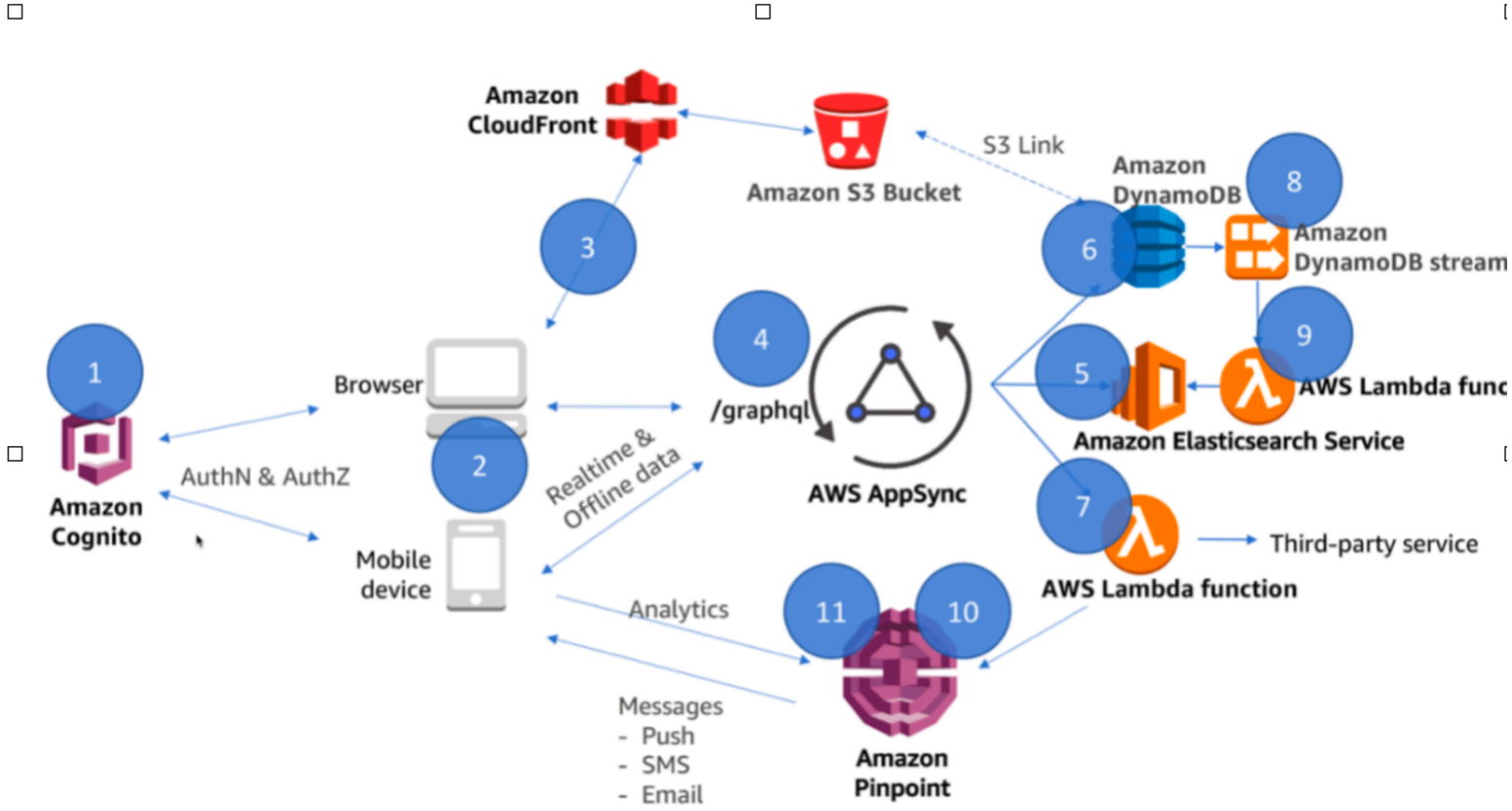


Figure 2: Reference architecture for a mobile backend

Stream Processing

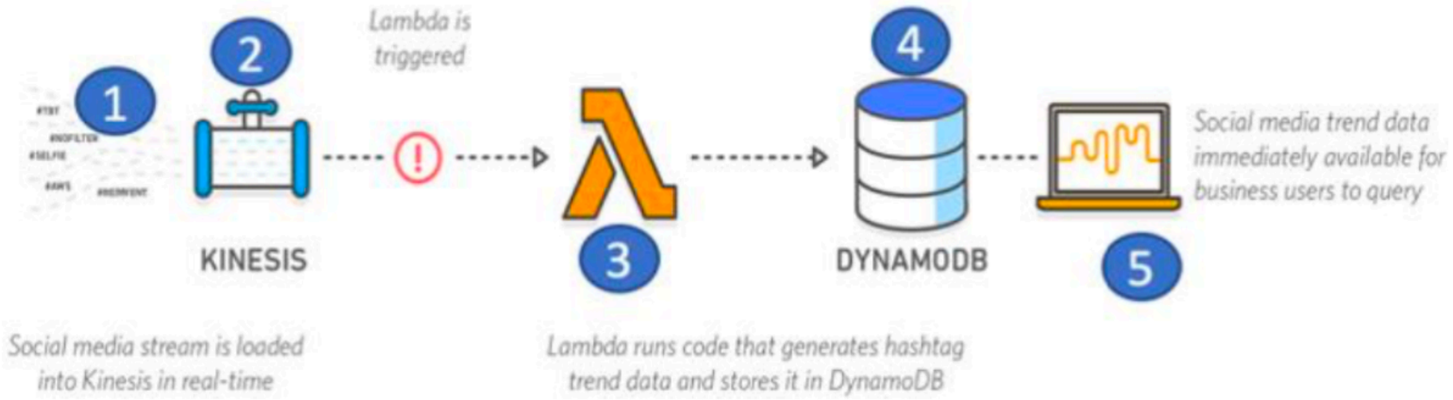
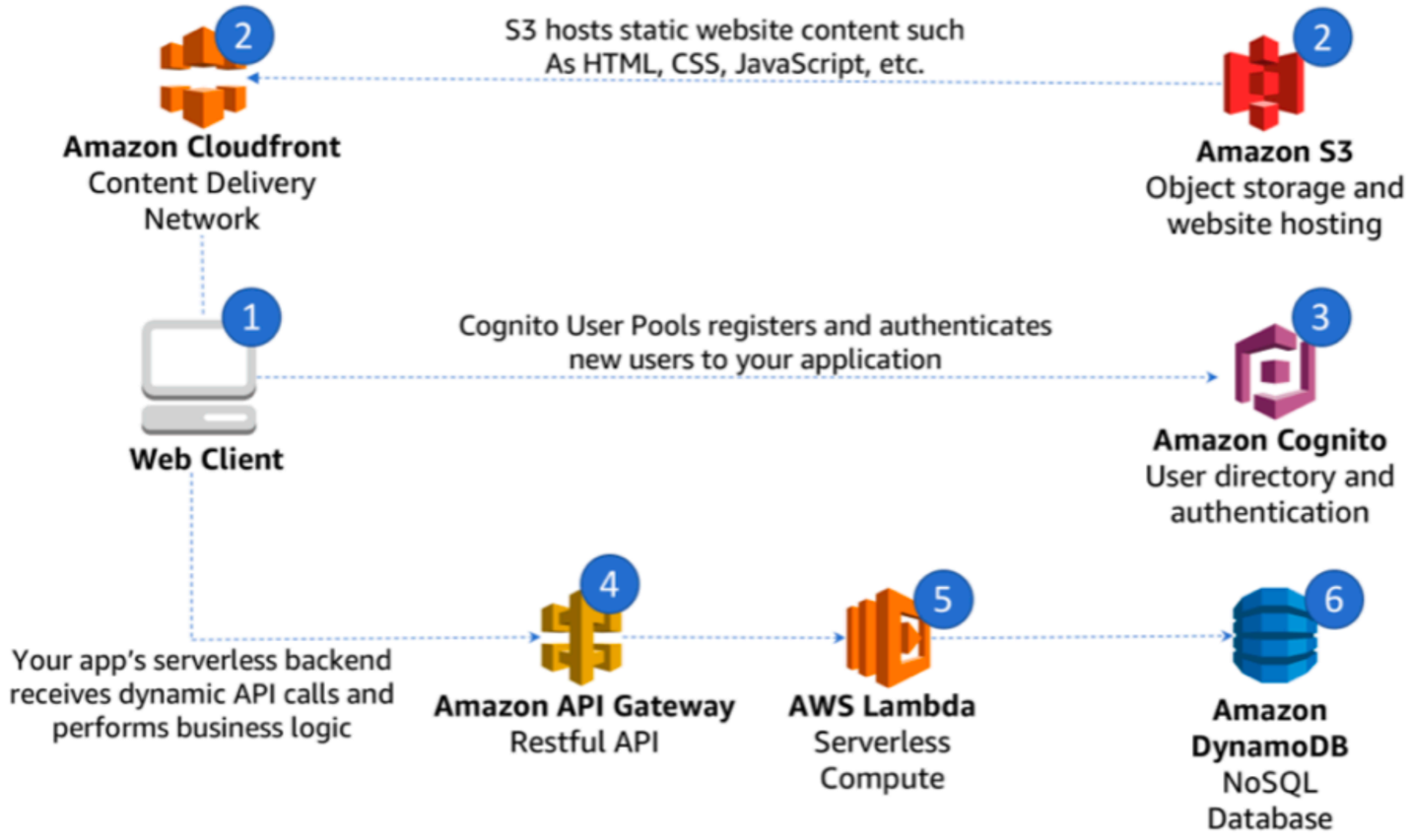


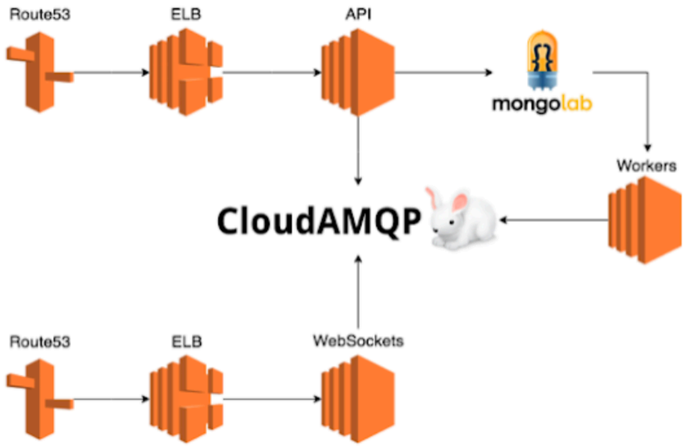
Figure 3: Reference architecture for stream processing

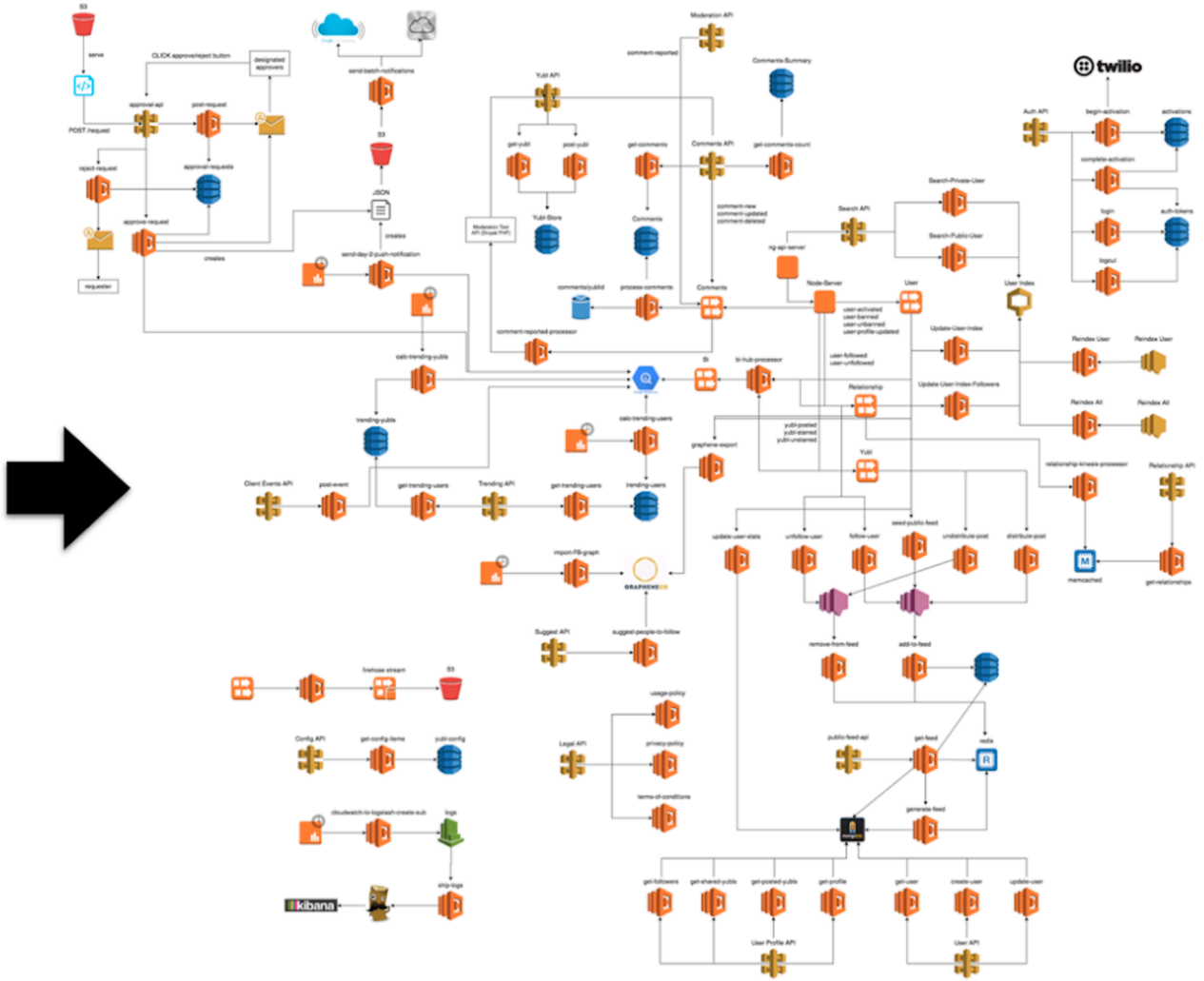
Web Application



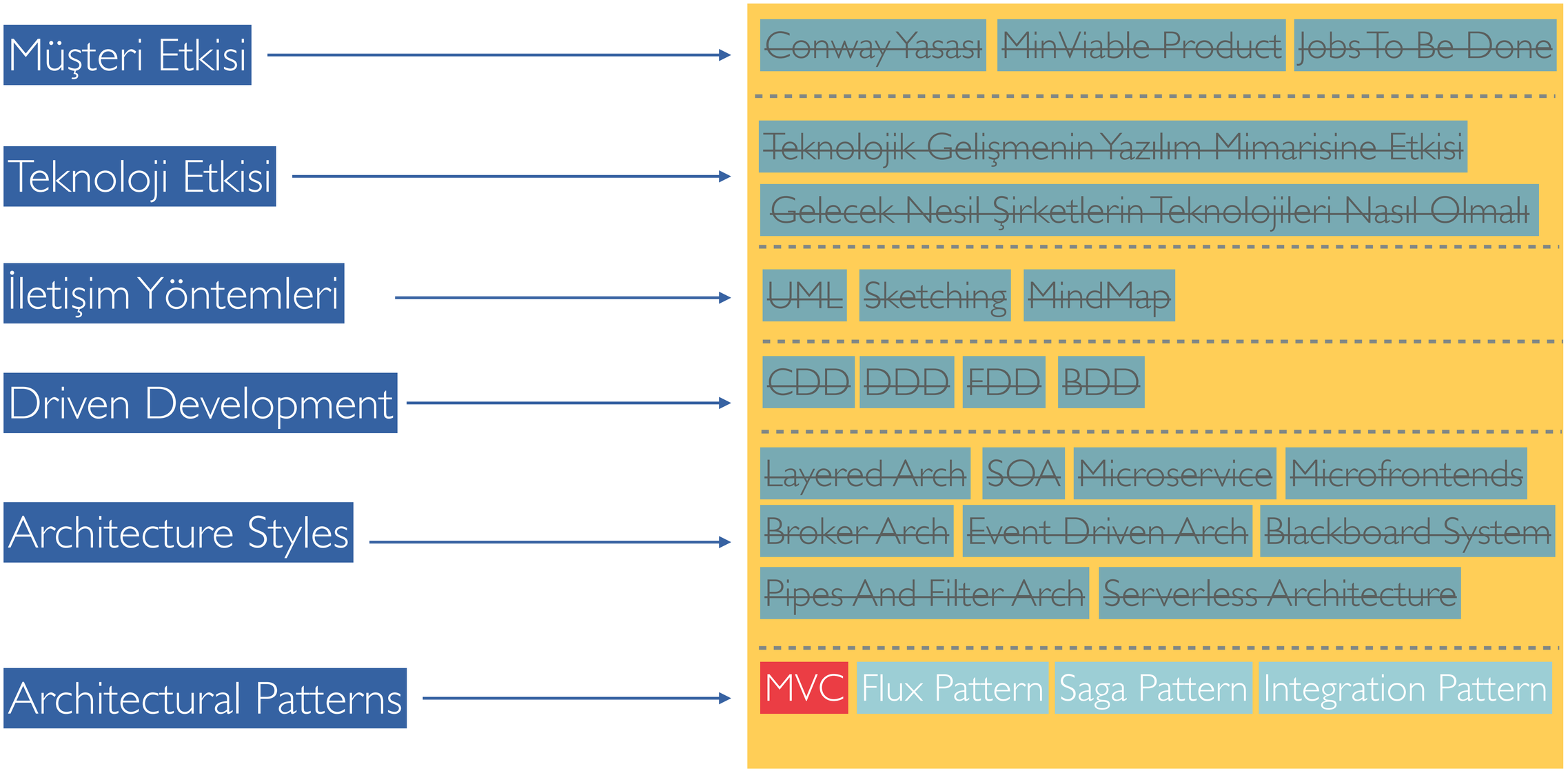
Real World Architecture
Social Network (Yubl)

As we migrated our system to serverless we **reduced operational cost by about 90%**. The **speed of delivery increased ten-fold** without hiring more developers. In under 6 months, we went from 4-6 production deployments a month to around 80. Our system also became more stable, and more scalable as well.

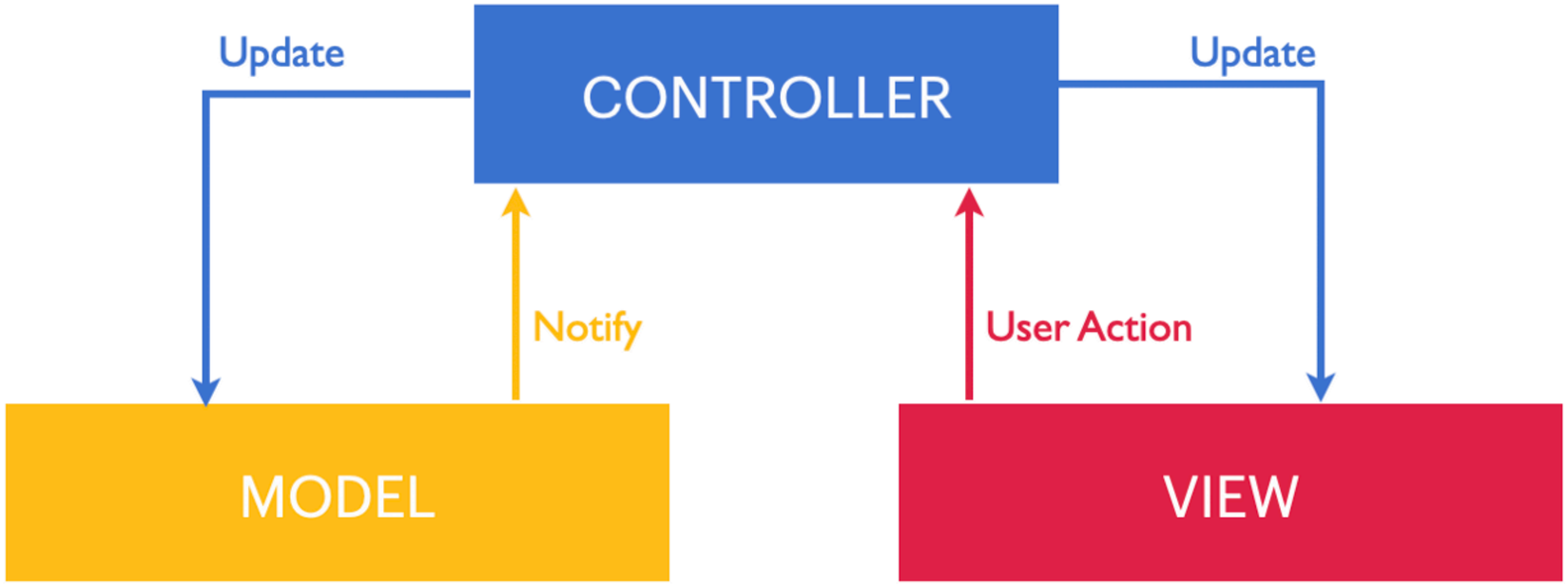




Inci Gün (Architecture Level)



MVC



MVC Bileşenleri

- **View:** Kullanıcı ile etkileşimi olan arayüzleri içeren kısımdır. Kullanıcı girdilerini dinler ve bunları yönetmesi için Controller iletir. Bu girdilere örnek vermek gerekirse Mouse, Keyboard, Ses, Dokunmatik Ekran, Hareket Sensörü olabilir bu girdiler. Aynı zamanda view içeriğinde uygulama state(durumu) içeren verileri kullanıcıya göstermekten sorumlu birimdir.

- **Model:** Uygulamada ekranda gösterilecek olan veriyi tutan kısımdır. Bu kısımlar uzak sistemlerden gelen güncellemeler veya kullanıcı girdilerini sonuçlarının yansıtıldığı verinin en güncel halini yapılarında tutarlar.

- **Controller:** Hem model , hemde view referansını tutarak ikisi arasındaki koordinasyonu sağlar. Yapılması gereken business işlemleri yerine getirir.

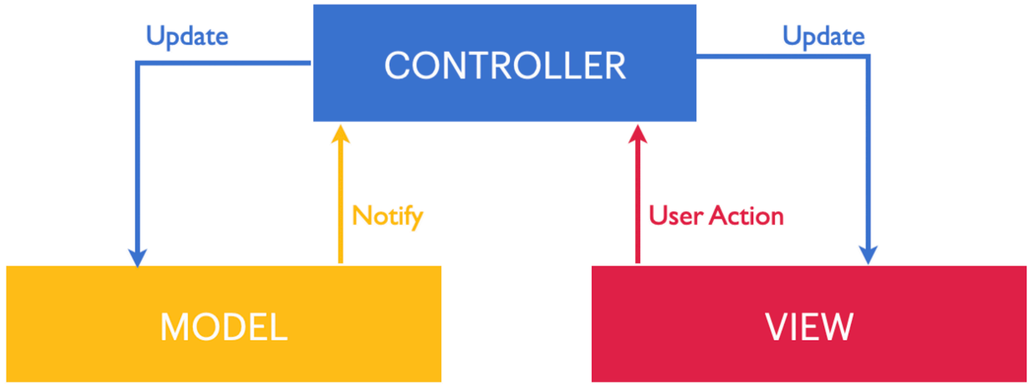
MVC Senaryoları

- **Senaryo 1 (Model Verisini Ekranda Gösterilmesi)**

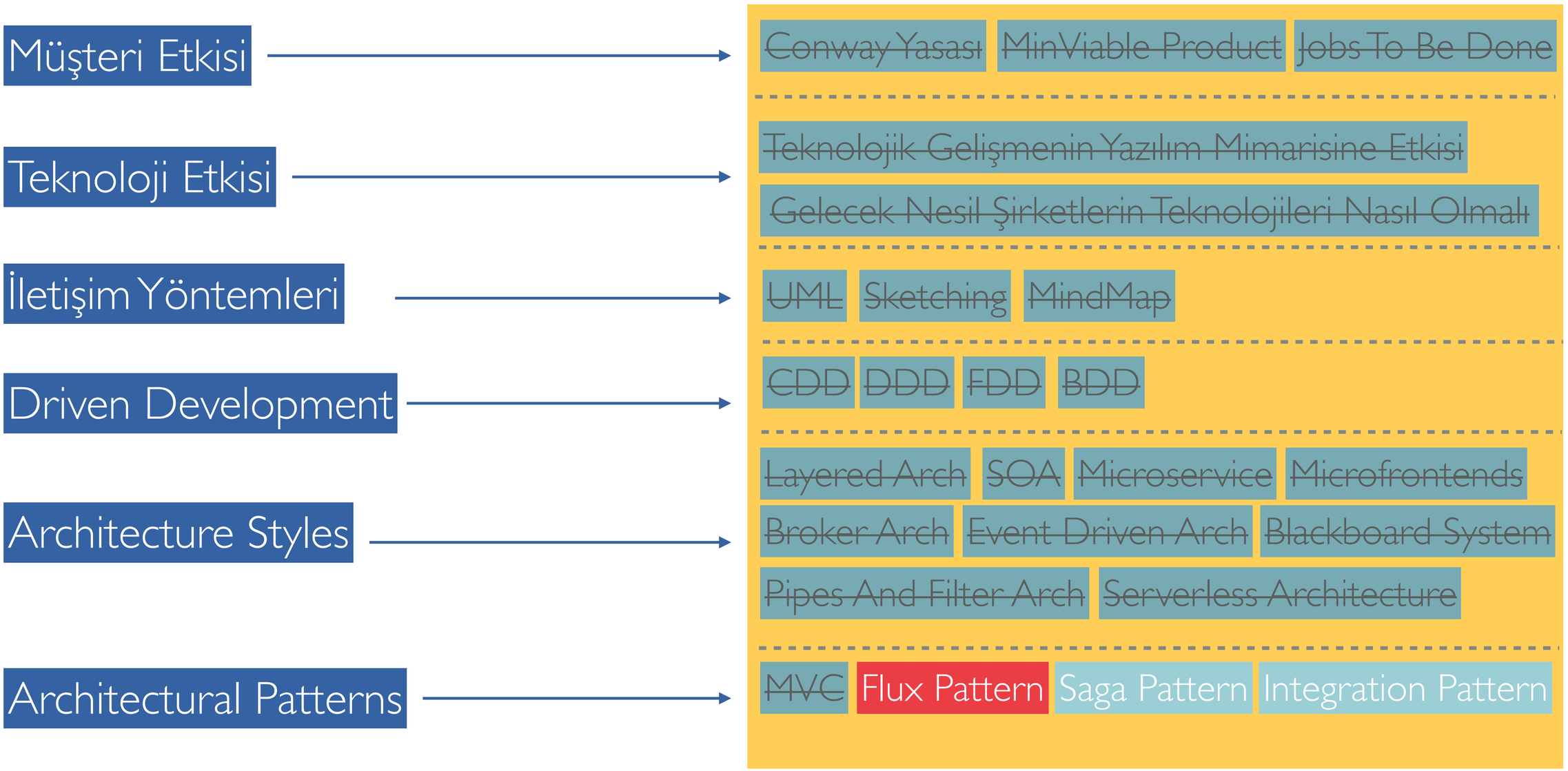
Model → Controller → View akışı gerçekleşir.

- **Senaryo 2 (Kullanıcının , Tablodaki Elemanlardan Bir Tanesini Silmesi)**

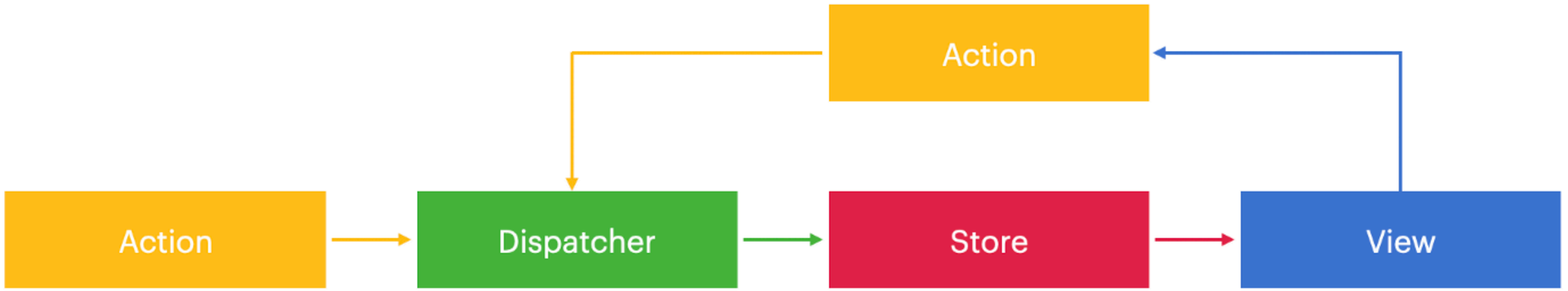
View → Controller → Model → Controller → View akışı gerçekleşir.



Inci Gün (Architecture Level)



Flux



Yukarıdaki resimde görüleceği gibi UI etkileşimi ve data akışı tek yönlü olacak şekilde geliştirilmiştir.

- **Dispatcher** sadece Action tarafından tetiklenebilir.
- **Store** sadece Dispatcher taradından tetiklenebilir.
- **View** sadece Store tarafından (**Not:** kendi iç state değişimleri ve Routing değişimleri hariç)

View

View: Veriyi gösteren UI bileşenleri, kullanıcının bu bileşenler ile etkileşimini sağlayan parçadır. Gerçekleşen herhangi bir kullanıcı etkileşimini Aksiyonlar ile Dispatcher iletir. **(React, Vue, Svelte)**

Action

Aksiyonlar kullanıcının veya client yer alan programın yaptığı sistemin **state değiştiren** komutlardır. Bunlar Dispatcher tetikler bunu söyle düşünebiliriz. Ana amacı Store değiştirmek olan her türlü aksiyon

- Ekranda ilk açılırken sunucudan veri çekme işlemide
- Sunucudan Veri Çekilmesini Sağlatmak Bir Aksiyondur.
- Ekrandaki bir düğmeye basıp bazı bileşenlerin gizlenmesini ve gösterilmesi için
- Mevcut UI State değiştirmek için yapılan işlemler

Dispatcher

Merkezi bir hub gibi çalışır. Store kendisine register edilmiş callback çağırmaktan sorumludur. Olabildiğince basit ve az kod içerir. İş mantığı içermez sadece ilgili ***callback*** çağırır.

Store

Tüm uygulamanın **state(durumunu)** ve **olası geçebileceği durumların neler** olduğunu içerir. Tüm modellerin durumları arasında geçişlerin nasıl gerçekleşeceğinin mantığı Store içerisinde yer alır.

Store İçerisine Bakalım

```
import { createStore } from 'redux'

function todos(state = [], action) {
  switch (action.type) {
    case 'ADD_TODO':
      return state.concat([action.text])
    default:
      return state
  }
}

const store = createStore(todos, ['Use Redux'])

store.dispatch({
  type: 'ADD_TODO',
  text: 'Read the docs'
})

console.log(store.getState())
// [ 'Use Redux', 'Read the docs' ]
```

Reducer

Store Creation

Action

Flux Gerçekleştirimleri

- React → Redux, MobX
- Vue → VueX
- Angular → NgRx

Neden Flux Mimarisine İhtiyaç Var

Birbirinden bağımsız bileşenlerin uygulama state ve birbirleri ile etkileşimini soyutlayıp, Pure bileşenler geliştirebilmesi, uygulama bileşen bağımlılık karmaşıklığını azaltmak ve veri değişimlerindeki hataları ortadan kaldırmak için Redux benzeri State yönetim araçlarına ihtiyaç var.

- **Global:** Her yerden Erişilebilen State (Okuma/Read)
- **State Değişimi Kontrollü:** Store State değişimi tek yönden gerçekleşiyor Action → Dispatcher → Store şeklinde

Nasıl Sağlıyor

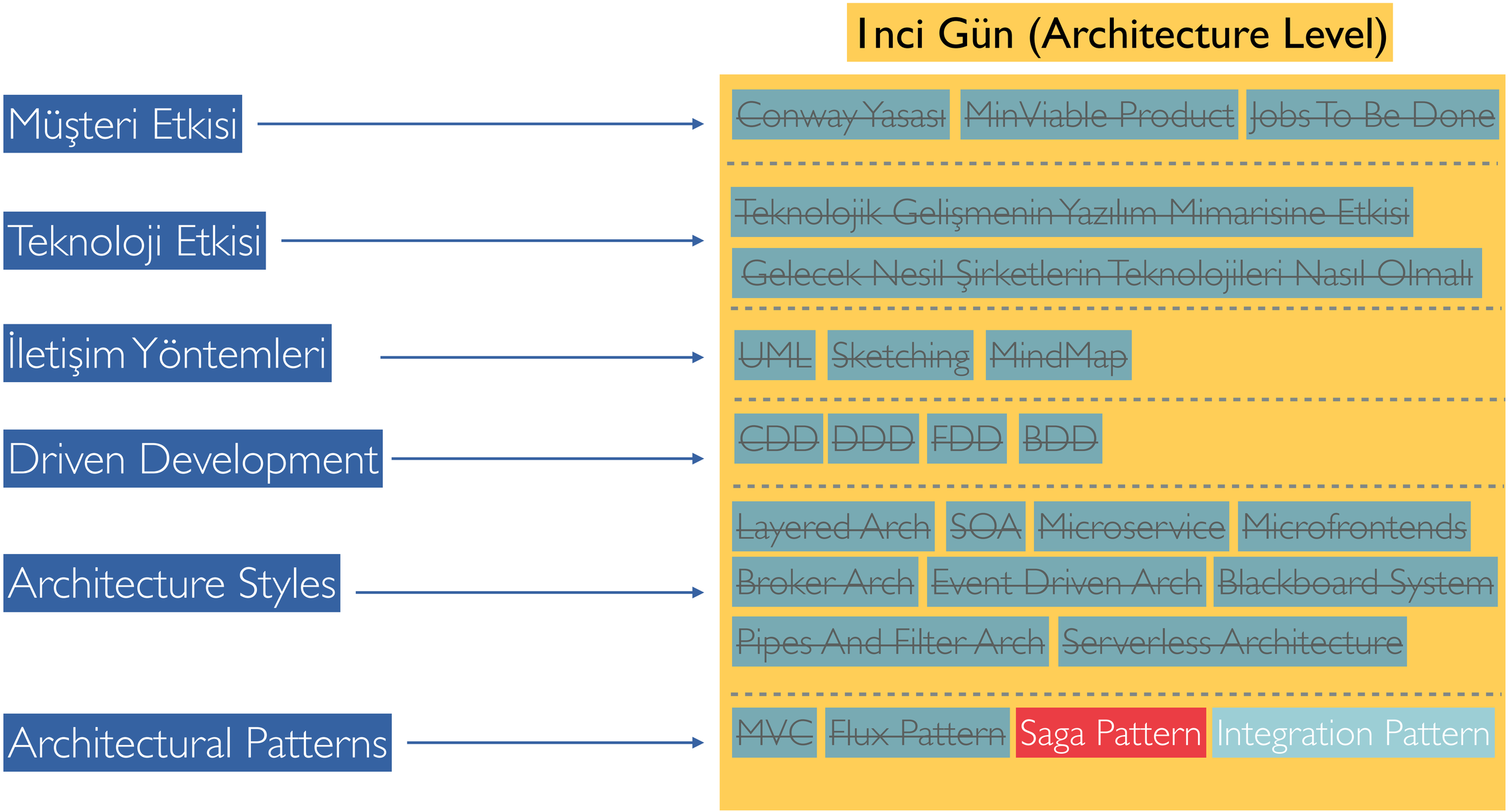
immutable →

Redux is a small library that represents state as (**immutable**) objects. And new states by passing the current state through pure functions to create an entirely new object/application states.

Özetle Store obje kullanacak View verdiğinde objenin Deep clone olan referans bağlantısı olmayan bir versiyonunu veriyor . Bu sayede Store orjinal obje değişmemiş oluyor

Tek yönlü Akış →

İster View'den gelsin, İster sunucu Request/Response veya Sensörden gelsin akış her zaman için tek yönlü gerçekleşiyor


Inci Gün (Architecture Level)
Müşteri Etkisi
Conway Yasası
MinViable Product
Jobs To Be Done
Teknoloji Etkisi
Teknolojik Gelişmenin Yazılım Mimarisine Etkisi
Gelecek Nesil Şirketlerin Teknolojileri Nasıl Olmalı
İletişim Yöntemleri
UML
Sketching
MindMap
Driven Development
CDD
DDD
FDD
BDD
Architecture Styles
Layered Arch
SOA
Microservice
Microfrontends
Broker Arch
Event Driven Arch
Blackboard System
Pipes And Filter Arch
Serverless Architecture
Architectural Patterns
MVC
Flux Pattern
Saga Pattern
Integration Pattern

Saga Pattern

Saga Örüntüsü aslında veri tabanındaki uzun transaction işlemleri sırasında oluşacak hataları engellemek için geliştirilmiş ***Hata Yönetim Örüntüsü***'dür Yani uzun bir transaction işlemini parçalara bölerek T(all)

```
T(all) = T1,T2,T3 ......Tn
```

Aynı zamanda her bir transaction parçasında olabilecek hataları sonrasında Kompanse edecek mekanizmaların yazılması anlamına gelir.

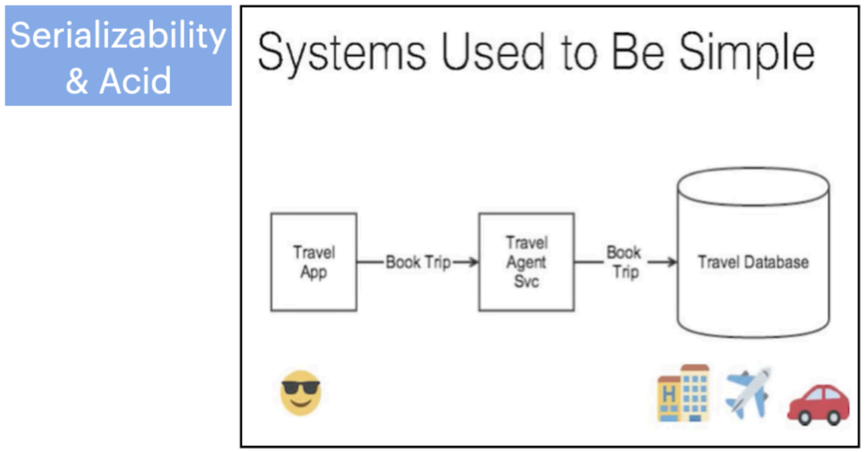
```
C(all) = C1,C2,C3 .....Cn
```

Burada Kompanse edecek mekanizma bazen rollback olurken bazende üzerine yazma anlamına gelebilir. Örneğin eposta attınız bunu geri alabilme şansınız yok bunun yerine onun üzerine bir öncekinin geçerli olmadığına dair yeni bir eposta atabilirsiniz mesala.

Saga Örüntüsü konusunu aşağıdaki video üzerinden çok implementasyon detaylarına girmeden anlamaya çalışırsak

Halo Oyunu Microsoft tarafından XBox üzerine getirilirken çok büyük tek SQL veritabanı üzerinde milyonlarda transaction yönetmenin imkansız olması ve ölçekleneMEmesi yüzünden NoSQL ve dağıtık yapılara web servislerini taşıma ve bu taşıma sırasında Atomicity ve Serialization kaybetmenin önüne nasıl geçtiklerinin üzerine bir konuşma.

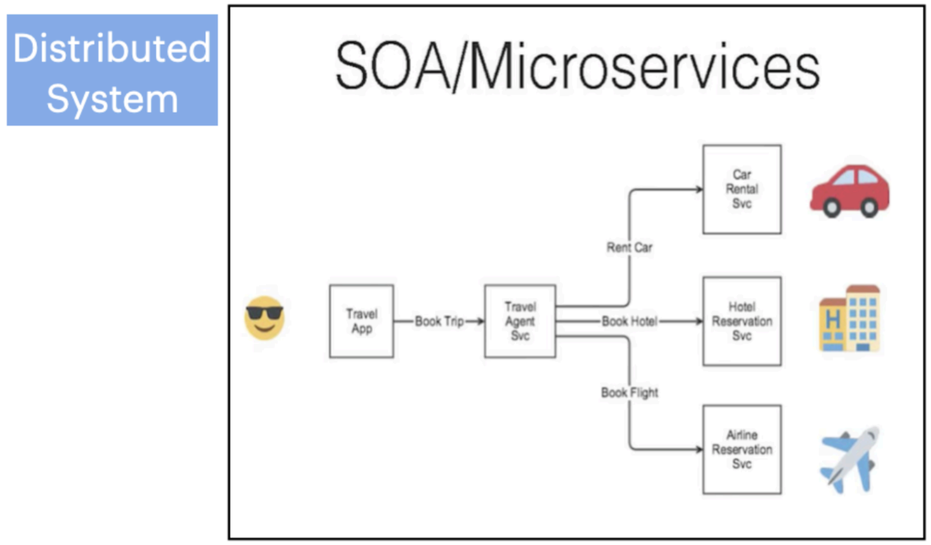
İlişkisel veritabanlarında transaction yapısı için tanımlanmış özellik seti olan ACID . sayesinde işlemleri birbirine karıştırmadan veya herhangi bir problem olduğunda geri (rollback) yapabilme imkanı bulabiliriz. Yönetimi dağıtık sisteme göre daha basit olan bir yöntem.



Single DB (Mono)

Saga Pattern

İşler kompleks hale geldikçe *CAP Teoremi* ve buna uygun hazırlanmış dağıtık veritabanlarında bu işleri nasıl yöneteceğiz. Çünkü dağıtık bir sistemde Transaction'dan söz etmek pek de mümkün değil. Burda Servisler bölümmüş Car, Hotel ve Airline Reservason sistemleri ayrı ayrı hizmet veren servisler hale gelmiş durumda burada oluşacak **Hata Durumunu** yönetmek bir önceki Single DB göre oldukça zor.

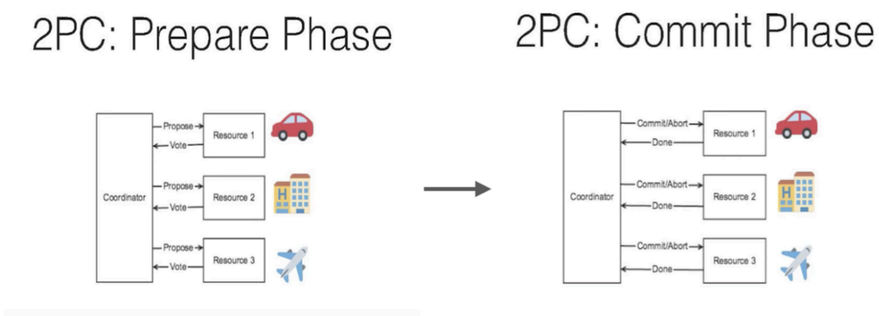


Dağıtık DB

2PC (Two Phase Commit)

Bu durumu yönetmek için değişik teknikler kullanılıyor. Bunlardan bir tanesi 2PC (Two Phase Commit) Protokolü. Burada öncelikle Coordinator gibi bir yapıya ihtiyaç duyulur. Coordinator

1. **Safha(Prepare Phase):** Tüm kaynaklardan bir işlem yapmasını ister.
2. **Safha(Commit Phase):** Tüm kaynaklardan olumlu sonuç dönüldüyse *Commit* mesajı gönderir. Eğer resource en az bir tanesinden olumsuz bir sonuç gelmiş ise bu durumda *Abort* mesajı göndererek işlemi sonlandırır.



Two Phase Commit

Saga Pattern

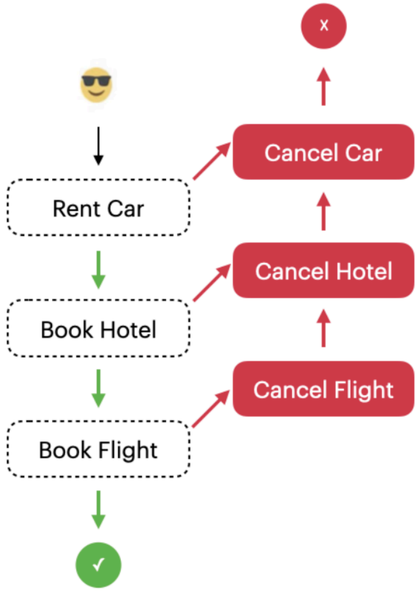
Bu yöntem probleme bir çözüm getiriyor ama ölçeklenmiyor. Nedeni mesaj sayısı kompleksitesini arttırıyor, Throughput azaltıyor ve Coordinator Single Point of Failure durumu oluşturuyor

Spanner (Google Globally-Distributed DB)

Google dağıtık veritabanları arasında özel fiber kabloları ve TrueTimerAPI(Atomic Clock) ve GPS ile veri merkezleri arasında o kadar küçük zaman aralıklarında yapabiliyor ki bu CAP Teoremindeki problemleri bir şekilde egale edebiliyor. Tabi çok pahalı ve tam anlamıyla synchronization sağlamış değil. Olay zaman kapsüllerini ne kadar küçük yapabildiğiniz ile ilgili . Örneğin TV ve Bilgisayar ekranı 40 Frame Per Second ile render edilirken insan gözüne bu problem oluşturmadığı için hiç bir takılma vs hissetmiyoruz. Ama saniye/40 da bir ekrandaki resim değişiyor. Aynısı Google Veritaban senkranizasyonları içinde geçerli.

Saga

Dağıtık sistemler ve CAP teoreminden vazgeçemiyorsak diğer bir yöntemde belki Serializability ve ACID gibi kavramlardan vazgeçerek bu durumu kod/uygulama business mantığı çözmeye çalışmak olabilir. Yukarıda bahsettiğim Transaction parçalara bölme ve her işlem için bir Kompanse sistemi oluşturmak gerekiyor. Bu durumda aşağıdaki gibi Tüm Transaction bölümlememiz gerekiyor



Saga Pattern

Saga Pattern

Öncelikle Transaction bütünümüzü parçalara ayırmamız gerekiyor.

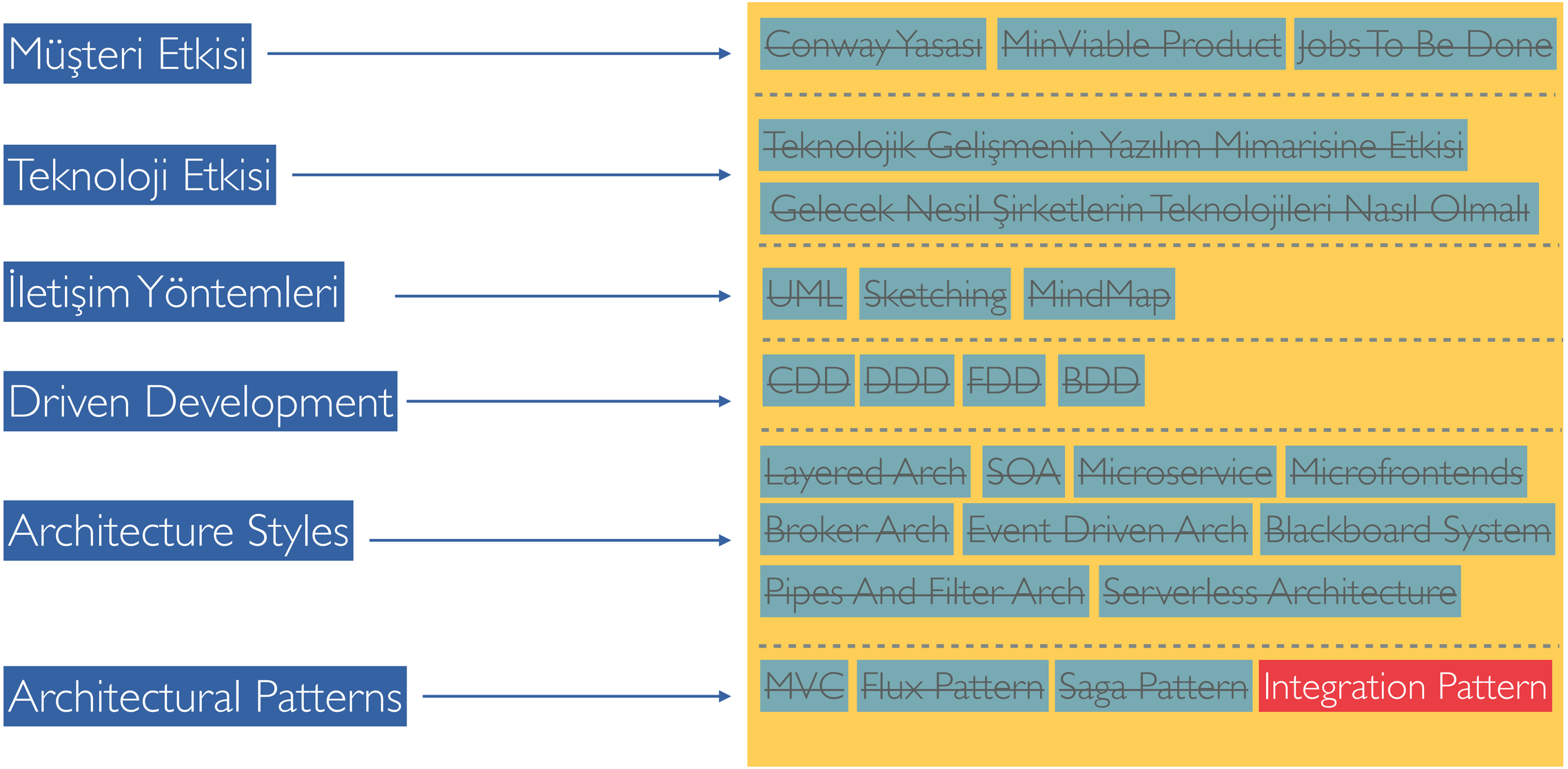
```
T: T(rentcar), T(bookhotel),  T(bookflight)
```

Şimdide herhangi bir aşamada ortaya çıkacak hata durumunda Companse adımlarını tanımlamamız gerekiyor

```
C: C(cancelcar), C(cancelhotel), C(cancelflifht)
```

Görüldüğü gibi istediğimiz gibi birbirinden bağımsız ölçeklenebilir sistemler tasarlayabiliriz artık. Bu sistemleri tasarlarken ya kendimizin yazacağı DB üzerinde persist yöntemi ile bu süreci yönetebiliriz veya AWS Step Functions gibi hazır mekanizmaları kullanabiliriz.

Inci Gün (Architecture Level)



ENTERPRISE INTEGRATION PATTERNS

- Dosya Transferi
- Paylaşılmış Ortak Veri Tabanı
- Uzak Procedure Çağırma (RPC)
- Mesajlaşma

ENTERPRISE INTEGRATION PATTERNS

- İletişim Yöntemleri
- Terminoloji
- Mesajlaşma Sistemlerini Neden Kullanılıyoruz?
- Örüntüler

İletişim Yöntemleri

Uygulamalar birbirleri ile 4 şekilde haberleşirler.

Dosya Transferi: Bir uygulama dosyaya yazar. Diğer uygulama'da bu dosyayı okur. Dosya adresi ve formatı konusunda iki uygulamanın anlaşmış olması gerekir. Dosyanın ne zaman yazılacağı, ne zaman okunacağı ve ne zaman silinebileceği konusunda uygulamaların anlaşmış olması gerekmektedir.

Paylaşılmış Ortak Veritabanı: Uygulamalar aynı veritabanı üzerindeki aynı tablolardaki veriler üzerinden çalışır. Bir uygulama yazar , bir diğeri okur.

Uzak Producure Çağırma (RPC): Başka bir uygulamadaki metodu çağırma SOAP buna bir örnektir. Diğer uygulamadan function call nesnesi xml dönüştürülüp karşıdaki uygulamaya gönderilir. Karşıdaki uygulama yine bu nesneyi method ve nesneye dönüştürüp karşıdaki fonksiyonu trigger eder. Bu işlem senkron bir işlemdir.

Mesajlaşma: Bir uygulama ortak bir mesaj kanalı üzerinden mesajını yayınlar. Diğer uygulamalar ilerleyen süre içerisinde bu kanaldan mesajı alır ve kullanırlar. Mesajı gönderen ve alanın mesaj kanalı ve formatı konusunda anlaşmış olmaları gerekir. Bu işlem asenkron olarak gerçekleştirilebilir.

TERMİNOLOJİ

Message(Mesaj): Uygulamalar arasında gönderilen/alınan veri paketleridir. Mesaj header/body kısmından oluşur. Header kısmında genellikle kimin gönderdiği, kim/kimlere gönderdiği, veri içeriği ve formatı hakkında bilgiler taşır.

Channel (Kanal): Mesajın uygulamalar arasında taşındığı kuyruk yapıları ve sunuculardır.

Sender/Producer: Mesajı kuyruğa koyan, oluşturan.

Reciever/Consumer: Mesajı kuyruktan alan, kullanan ve silen.

Mesajlaşma Sistemlerini Neden Kullanıyoruz ?

Uzak İletişim: Bir verinin/fonksiyonun/olayın mesaj ile kaplanarak serialize edilip diğer uygulamaya geçirilmesi oldukça zahmetli ve problemli bir işlemdir. Bu tip bir zahmete katlanmanız için uzak bilgisayarlarla iletişim ihtiyacınız olması gerekir. Aynı makine üzerinde bu tip işlemler yapıyorsanız ortak bellek , ortak collection yapıları ile bu iletişimini halledebilirsiniz.

Platform/Dil: Uygulamayı geliştiren ekipler farklı platformları Linux, Windows, Unix ve aynı zamanda farklı dilleri C++, Java, Phyton vb.. kullanmış olabilirler. Bu durumda mesajlaşma hiç bir ortamın kendi yapısını bozmadan iletişimini sağlar.

Asenkron: Uygulamalar aynı anda birbirleri ile iletişmek için müsait olmayabilirler. X zamanında gönderici mesajı göndermek için müsaitken alıcı bunun için uygun olmayabilir. Bu durumda mesajların kanallarda beklemesi gerekmektedir. Gönderici ve alıcı uygulamalarının birbirleri ile iletiştikleri kısımları asenkron olarak geliştirmeleri gerekmektedir.

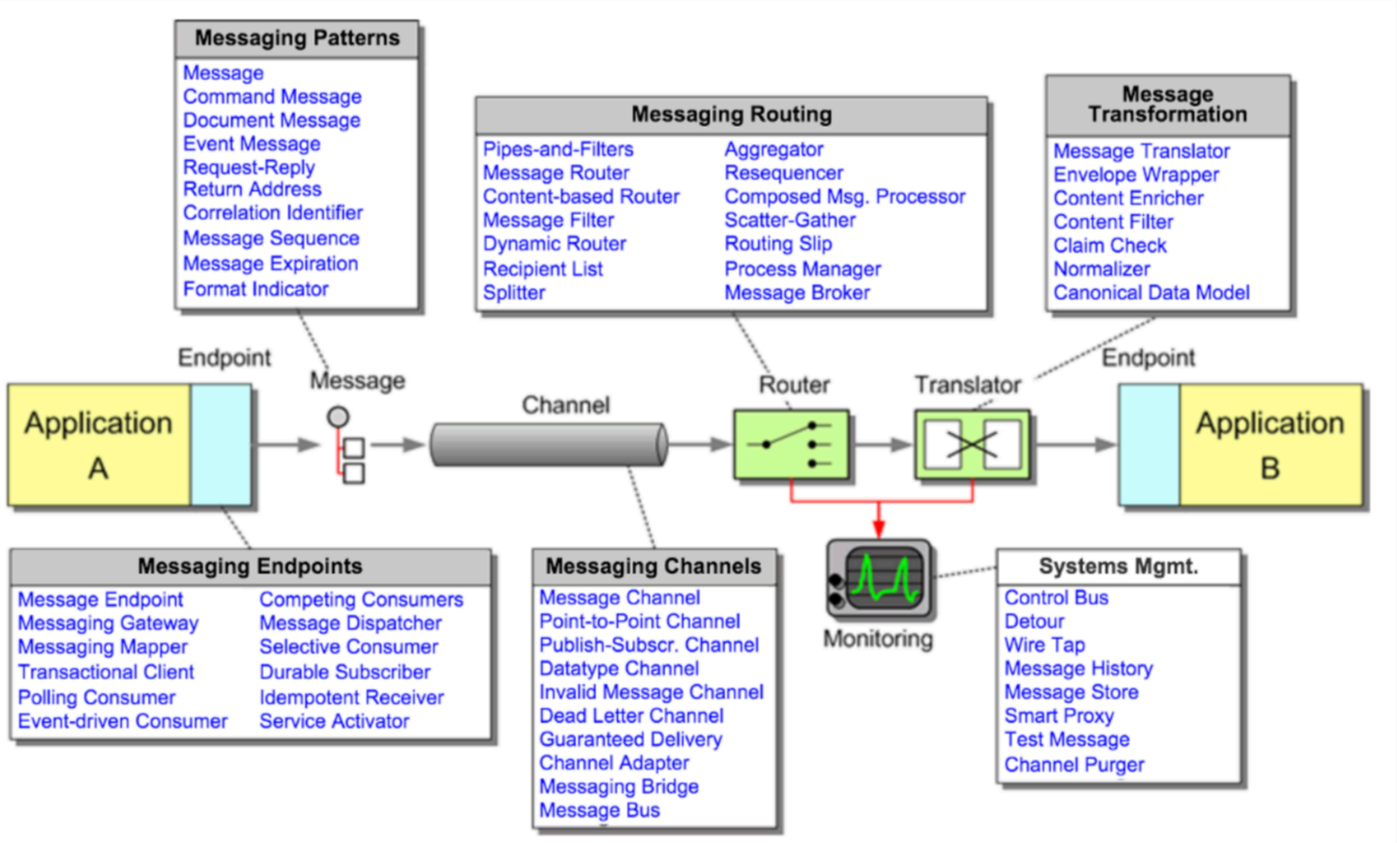
Değişken Zamanlama: Uygulamaların bazılarında isteği bulunanın cevap beklemesi ve bunun sonucunda diğer işlemleri geçebilmektedir. Ama arka planda yapılacak işlem çok uzun sürecek işlemler ise mesajı alan kısım bunu işliyorum şeklinde bir cevap dönerek mesajları batch olarak uzun bir süreçte işleyebilir, bu sayede uygulamalar işlemlerin bitmesini beklemeden diğer işlemlerine geçebilir.

Throttling: RPC senkron olmasından dolayı istemci sistemlerin alıcı uygulamayı dar boğaz ettiği durumlar için oldukça iyi mesajlaşma sistemi oldukça uygun bir yöntemdir.

Güvenilir İletişim: Bazen alıcı sistemin çökmesi veya ağ problemlerinden dolayı RPC fonksiyonu çalışmaz ve işlem yapılamadan yarım kalabilir. Mesajlaşlaşma sistemlerinde mesajın atomic bir veri olarak tutulduğundan bir kopyasının diskte tutulması, replikasının tutulması sayesinde iletişim kaybı veya mesaj kaybı yaşanmaz.

Offline Çalışma: Bazen iletişim olmadan tüm çalışmaların local'e kaydedilmesi sonrasında bağlantı kurulduğunda senkronizasyon yapma ihtiyacı olabilir. Bu durumda tüm işlemler client kuyrukta tutularak iletişim kurulduğunda diger sistemlere aktarılabilir.

Mediation: Mediator örüntüsünde olduğu gibi. Tüm entegrasyonun merkezi bir yerden yönetiliyor olması sayesinde sonradan sisteme katılacak bir uygulamanın diğer uygulamarı bilmesi gerekmez. Sadece mesajlaşma sistemine entegre olması yeterli olacaktır.

Thread Management: Mesajlaşma sistemi bloklanan threadlerin sayısının atmasınıda engelleyerek, uygulamaların kilitlenmesini engellemiş olur.

Enterprise Integration Patterns