Software Construction
Yazılım İnşası

Sedat Görmüş, PhD

Email : sedatgormus@gmail.com

: sedatgormus@ktu.edu.tr

İnşa için Dizayn Yaklaşımları

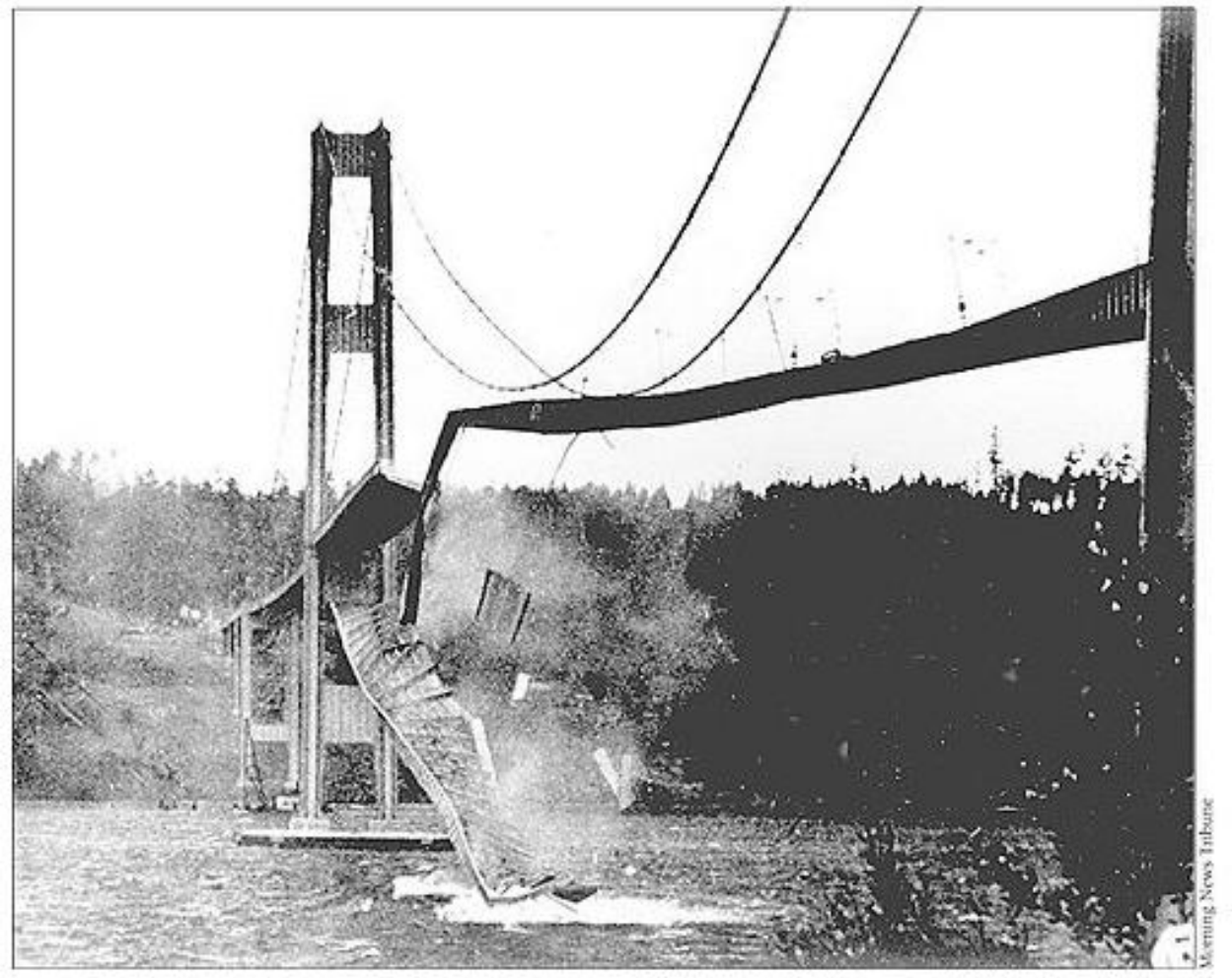
İçerik

- Dizaynda karşılaşılan sorunlar
- Dizayn konseptleri
- Dizayn yapı taşları
- Dizayn pratikleri
- Popüler dizayn metodları

İnşa için dizayn nedir?

- **Dizayn gereksinimleri koda ve hata ayıklamaya bağlayan aktivite olarak adlandırılabilir.**

Bu köprünün yazılım inşasındaki dizaynla ne ilgisi var?



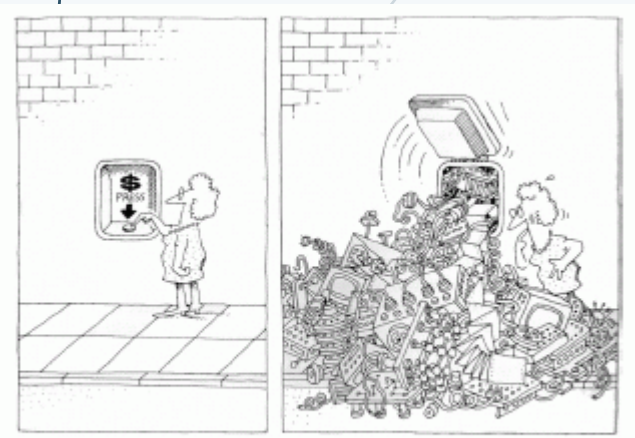
Dizayna Nasıl Yaklaşmalıyız?

- ***Dizayn oldukça külfetli bir prosestir***.
  - *İki yaklaşım çok farklı olmasa bile, iki dizayn başarı anlamında çok farklı olabilir.*
- ***Dizayn öncelikler ve kısıtlamarın harmanlandığı bir prosestir.***
  - *İnşa öncesi yapılan dizayn sistem kısıtlamalarını göze almalı ve öncelikleri gözetmelidir.*
- ***Ne yazıkki dizayn deterministik bir proses değildir.***
  - *Eğer farklı kişileri, aynı problemi çözmek için görevlendirirseniz, birbirinden oldukça farklı dizaynlarla size geri döneceklerdir.*
  - *Dahası her bir dizayn da kabul edilebilir olabilir.*
  - *Dizayn heruristictir, ve belli dizayn kuralları olsa da bunlar genel geçer değillerdir.*
- ***Özetle, dizayn evrimsel bir prosestir***.
  - *Resmi olmayan tartışmalar ve resmi gözden geçirme toplantılarıyla dizayn belli bir olgunluğa getirilir.*

Dizayn Konseptleri

- Karmaşıklığı yönetmek (Kontrol altında tutmak)
  - Complex != Complicated
- Dizaynın istenen özelikleri.
- Dizayn seviyeleri.
- Gerçek hayata dait objeler bul.
- Soyutlamarınız kararlı ve tutarlı olsun

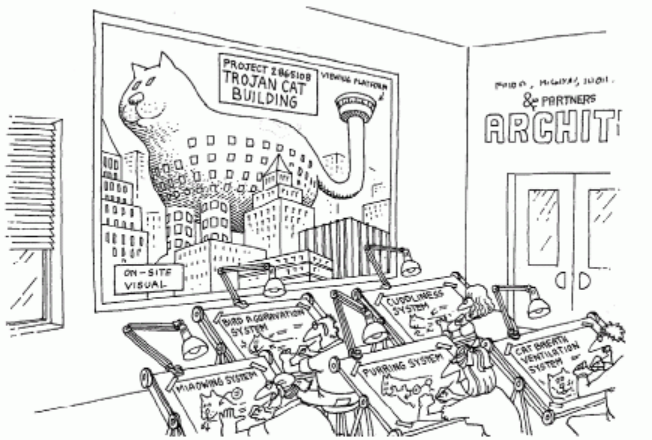
Karmaşıklığı Yönetmek (managing complexity)



The task of the software development team is to engineer the illusion of simplicity.

- Yazılımda karmaşıklığın iki kaynağı oladuğunu söylemek yanlış olmaz.
  - Detay olan karmaşıklıklar (accidental, or incidental)
  - Gerekli karmaşıklıklar
- Bir örnek üzerinde tartışalım
  - Sizden bir top yapmanızı istiyorum
  - Bana gerekli karmaşıklıkları
  - Ve Detay Karmaşıklıkları sıralayın
- Tartışmamızdan da anladığımız gibi gerekli karmaşıklıklar, yazılımın müşteri tarafından kabul edilebilir olması için gerekli dizayn gereksinimleridir.
  - Bir aracın, otomobil olarak tanımlanması için motoru lastikleri ve bir gövdesi olması gerekir.
  - Lastik kalınlığı, motorunun dizel yada benzinli olması gibi özelikler detay (incidental) gereksinimlerdir.
- Dizaynı karmaşıklığını yönetmek için, dizayn sırasında ilk olarak gerekli olan karmaşıklıkların üstesinden gelmenizdir.

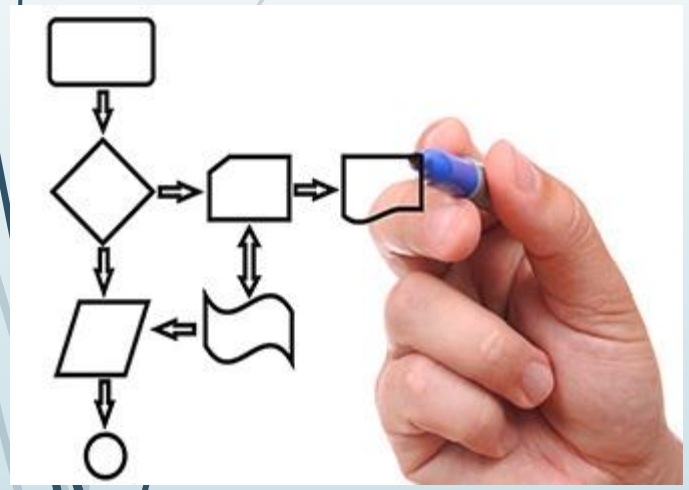
Karmaşıklığı Yönetmek



The architecture of a complex system is a function of its components as well as the hierarchic relationships among these components.

- Rutinler basit ve kısa tutulmalıdır.
- Problem he programcı tarafından kolayca anlaşılabilen parçalara bölünmelidir.
- **Karmaşıklıkla Başa Çıkarken Dikkat Edilmesi Gereken Ana Nokta İyi Bir detaylı Dizayn Prosesidir**
- **Kötü dizayn üç kaynaktan beslenir**
  - **Kolay probleme karmaşık bir çözüm üretmek**
  - **Zor bir probleme basit ve yanlış çözüm üretmek**
  - **Zor ve karmaşık probleme uygun olmayan karmaşık bir dizayn üretmek.**
- **Detaylı dizayn sırasında, gerekli karmaşıklıklar öncelikle halledilmelidir.**
- **Eksta fonksiyonlar için gerekli karmaşıklıklar ise ikincil olarak halledilmeli ve gerekli karmaşıklıklarla olan ilişkileri açıkça ifade edilmelidir.**
- **Mimari anlamda karmaşıklık, yazılımı modüllere ayırarak azaltılabilir.**

İyi Dizayn Karakteristikleri



- Dizayn prosesinin birinci amacı yazılımdaki karmaşıklığı azaltmaktır.
- Lütfen, ‘kurnaz’ yada ‘zeki’ dizaynlardan uzak durun.
- Zira, bu tür dizaynların genel olarak anlaşılması zordur.
- Bunun yerine akıllı, basit ve herkesin anlayabileceği bir dizayn yapmalıyız
- Eğer dizaynınızda tasarladığınız bir modül sizin sürekli diğer modüller hakkında düşünmenize neden oluyorsa yanlış bir dizayn yapmışsınız demektir.
- Kolay bakım(maintanence) dizayn prosesinin karakteristiklerinden biridir. (Bakımdan ne anlıyoruz?)

İyi Dizayn Karakteristikleri

- Gevşek İlişki(Loose Coupling); Moduller arasındaki ilişkilerin minimum düzeyde tutulması anlamına gelir.
  - Bu integrasyon sırasında yapılacak ekstra işlerin minimize edilmesi anlamına gelir.
- Genişletilebilirlik; bu sisteme farklı fonksiyonlar eklendiğinde sistemi oluşturan yazılımlarda ölümcül bir etki yaratmamak olarak anlaşılabilir.
- Tekrar kullanılabilirlik(Reusability); bu bir sistem parçasının farklı projeler için kullanılması anlamına gelir (Bir örnek vermenizi istiyorum).
- Yüksek dizayn içi kullanım (High Fan-in); bu herhangi bir ana sınıfın alt sınıflar tarafından ne kadar kullanıldığını gösterir. İyi bir dizaynda ana sınıflar alt sınıflar tarafından sıkca inherit edilir.

İyi Dizayn Karakteristikleri

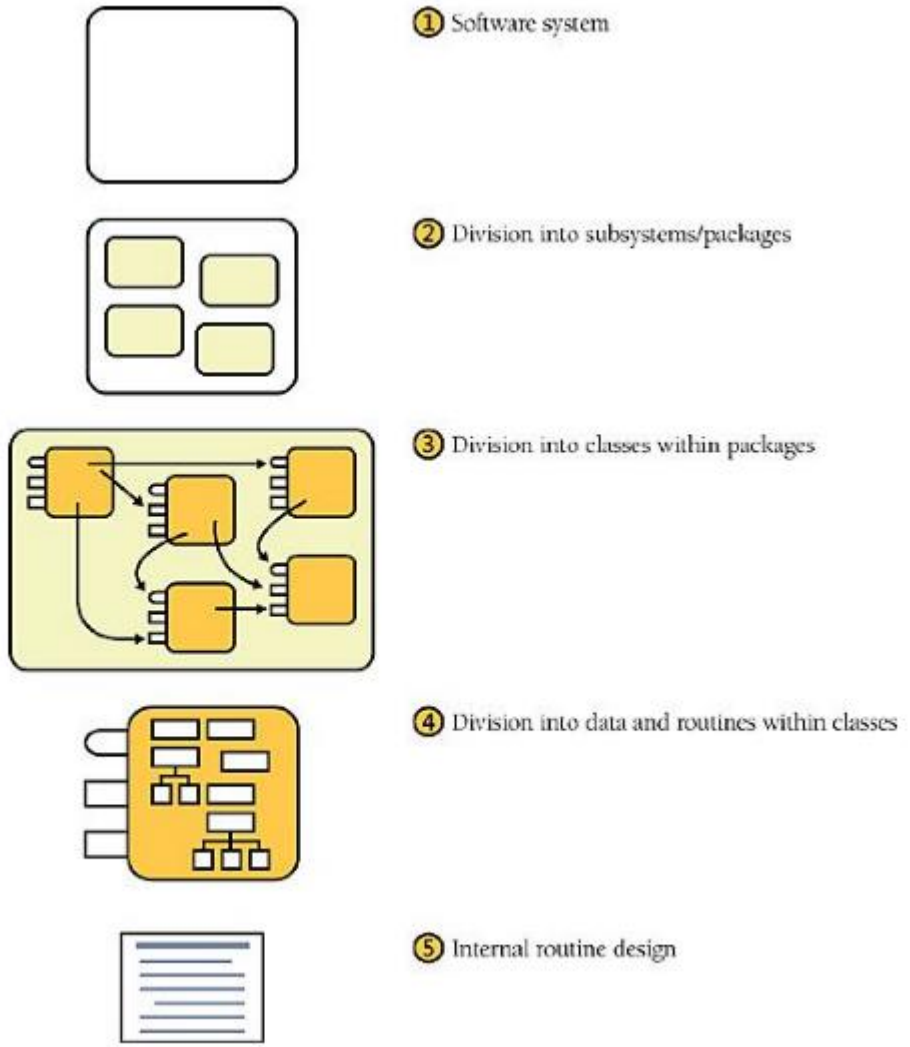
- **Orta ve Düşük dizayn içi sınıf kullanımı (Low to Medium Fan-out); bu herhangi bir sınıfın ne kadar sınıfı kullandığını gösterir. Ortalama olarak 7den fazla sınıfın aynı anda başka bir sınıf tarafından kullanılması, bu sınıfın çok karmaşık olduğunu gösterir.**

- **Taşınabilirlik (Portability); Bu herhangi bir yazılım çözümünün geliştirildiği sistemden başka sisteme kolayca taşınması anlamına gelir. (Buna bir örnek verebilir misiniz)**

- **Yalınlık (Leanness); Bu gereksinim ise, kodun yapması gereken fonksiyonların haricinde fonksiyonları içermemesidir.**

- **Standart teknikler (**Standard techniques); İyi dizayn herkesin anlayacağı standard dizay tekniklerini kullanır (Buna daha sonra tekrar değineceğiz).

Dizayn Seviyeleri

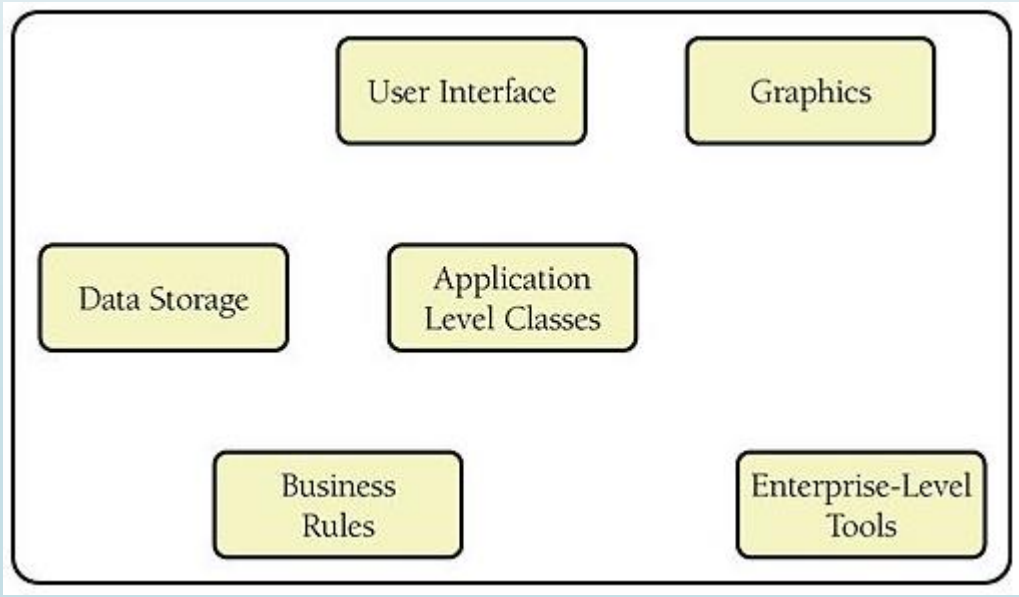


- Bir yazılım projesi şekildende görüleceği gibi farklı seviyelerde ayrıntılandırılmış dizaynlara gereksinim duyar.
- **Seviye 1 : Yazılım sistemi**
- **Seviye 2 : Sistemin parçaları (packetleri)**
- **Seviye 3: paketler içindeki sınıflar**
- **Seviye 4: Sınıflar içindeki veri ve rutinler**
- **Seviye 5: Rutin içi dizayn**

Dizayn Seviyeleri

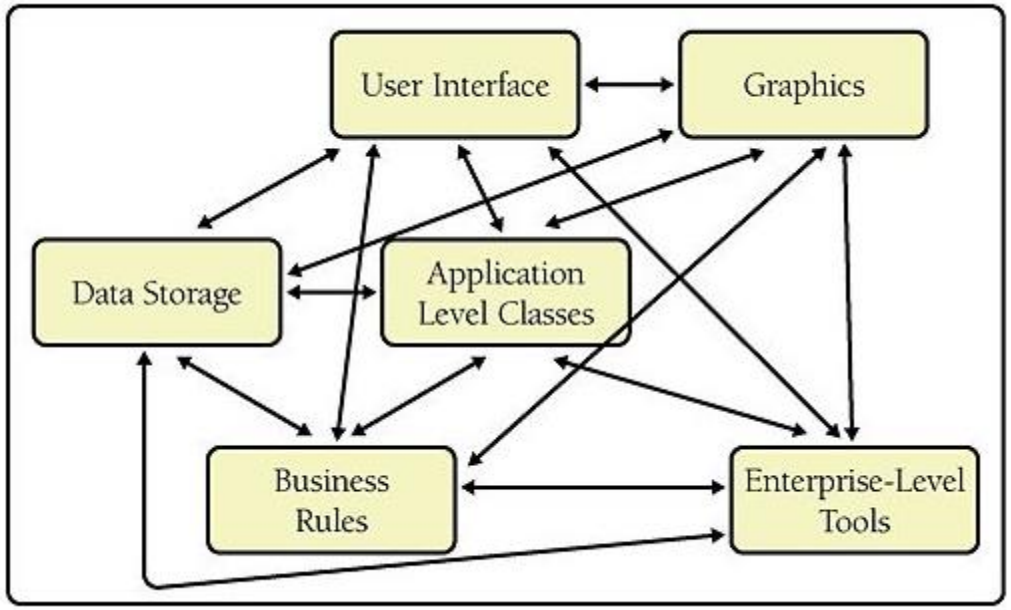
- **Seviye 1:** Bu yazılacak programın tamamıdır, bazı programcılar direkt bu seviyeden sınıf dizaynına geçerler ancak bu yazılımın geliştirilmesinde bazı riskleri içinde barındırabilir.

- **Seviye 2:** Bu seviyede ana alt sistemler ve bu sistemlerin birbirleriyle olan etkileşimleri belirlenir. Bu seviyede bir parcalama 5 haftadan uzun sürecek projeler için gereklidir.



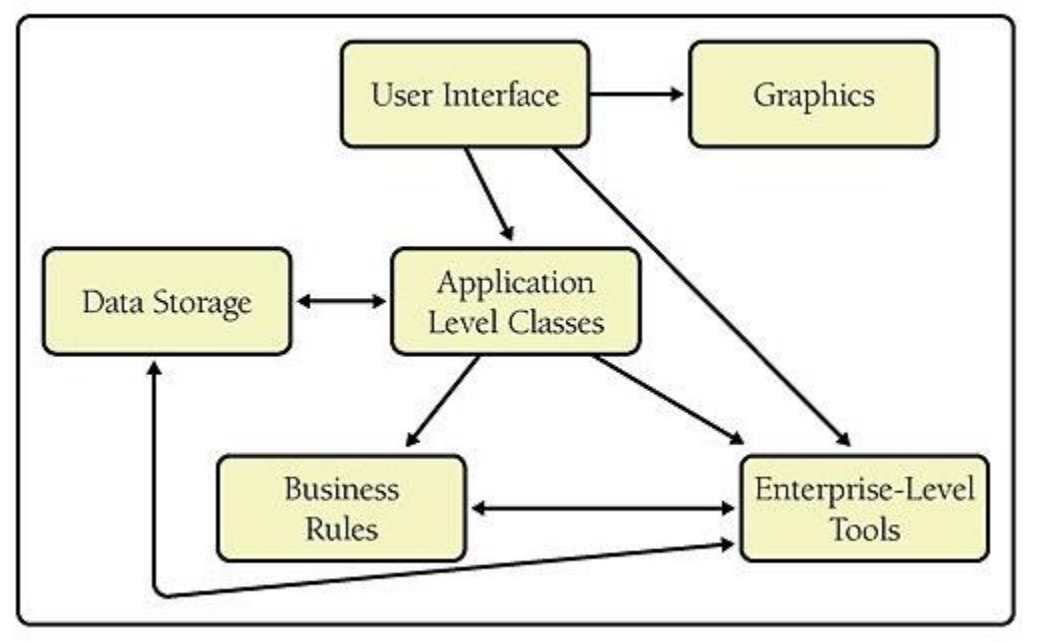
Seviye 2 parçaları arasındaki haberleşme

- Önemli bir dizayn yaklaşımı seviye 2deki parçaların birbiriyle olan iletişimnin kısıtlanmasıdır.
- Eğer bu kısıtlama dizayna dahil edilmezse, sistemin kararlılığı azalacaktır.
- Bu şekilde haberleşme ihtiyacı olan bir sistemin aşağıdaki sorulara cevap vermesi gerekir.
  - **Grafik sisteminde bir değişiklik için geliştiricinin diger alt parçalar hakkında ne kadar bilgi sahibi olmalıdır.**
  - **Buradaki parçaları başka bir sistemde tekrar kullanırsanız ne olur**
  - **Sisteme yeni bir kullanıcı arayüzü koymanız gerekirse ne olur.**
  - **Veri tabanını uzuk bir sisteme kurulursa haberleşme yükünün sisteme maliyeti ne olur (burda anlatılmal istenen nedir?)**



Seviye 2 parçaları arasındaki haberleşme

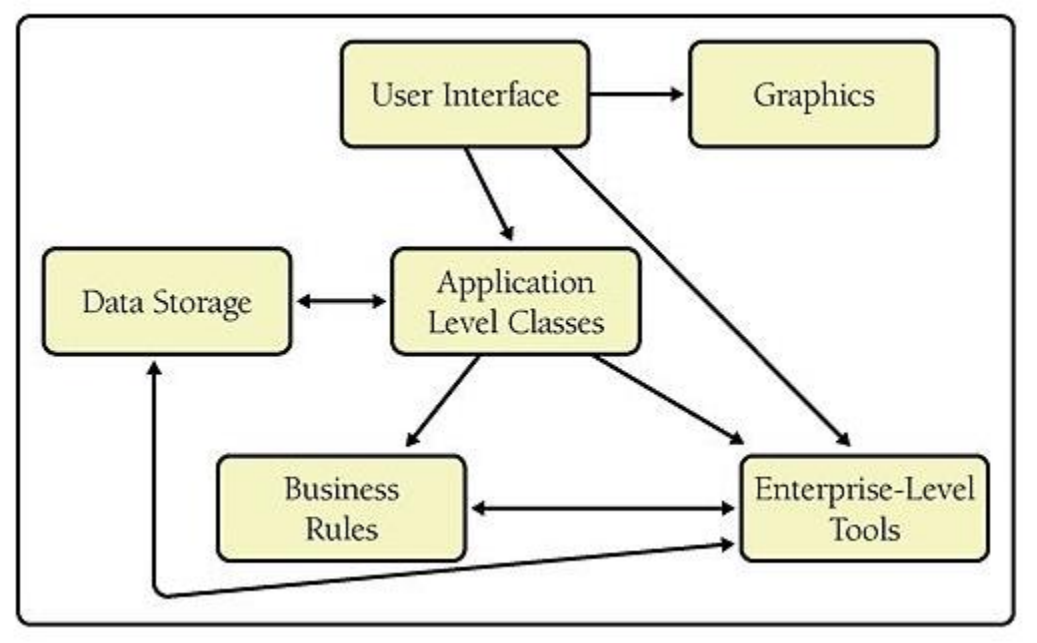
- **İyi bir sistem level dizayn diagram şekildeki gibi acylic bir graph olmalıdır (Bunu tartışalım).**
  - Bir graphın acyclic olması ne anlama gelir?
- **Daha ayrıntılı olarak açıklamak gerekirse, Eger bir sınıf A sınıf B yi kullanıyorsa ve Sınıf B de Sınıf C yi kullanıyorsa ve Sınıf C de Sınıf A yı kullanıyorsa bu cyclic bir sınıflar arası etkileşim örneğidir.**
- **Kullanıcı arayüzü diğer parçalardan izole edilerek kullanıcı arayüzündeki değişikliklerin diğer sistem parçalarına zarar vermeden yapılması sağlanabilir.**
- **Veri tabanı erişimi sağlayan rutinlerin detayları ise veri tabanı parçası tarafından gizlenerek veri tabanının yaptığı detaylı işler programcıdan saklanılabilir.**
- **Bu durumda uygulama detaylarını saklayan alt parçalar sistemin komplekslığinin kontrol edilmesine faydalı olacaktır.**



Seviye 2 parçaları arasındaki haberleşme

- Ayrıca, sistemin çalıştığı ortama ait bağımlılıklarını içeren bir alt sistemin oluşturulması, yazılımın farklı isletim sistemleri arasında taşınmasını kolayca sağlayacaktır.

- Örneğin **Windows** için geliştirilen bir programın işletim sistemi olan arayüzlerinin tama bir alt parçaya koyulması durumunda, sadece bu alt parçanın içeriği değiştirilerek, **Linux** gibi farklı bir işletim sisteminde de programın koşması sağlanabilir.



Seviye 3: Alt parçaların Sınıflara Bölünmesi

- Bu seviyedeki dizayn, sistemin bütün sınıflarının tanımlanmasını içerir.
  - Örneğin veri tabanı alt sistemi, veri erişim ve veri işleme sınıflarını içerebilir.
- Bu seviyede her sınıfın sistemin geri kalanıyla olan ilişkisi tanımlanmalıdır.
- Buradaki ana fikir, sistem alt parçaları gerekli sınıflara ayrılarak, sınıfların içindeki fonksiyonların gerçeklenmesi sağlanabilir.
- Burada nesneye yönelimli dizayna bir geri dönüş yapalım
  - Kim, bir sınıfla, nesne arasıdaki farkı yada ilişkiyi anlatabilir?

Seviye 4: Rutinlerin oluşturulması

- Bu seviyedeki dizayn sınıfların rutinlere ayrılmasını içerir.
- Seviye 3 teki sınıf arayüz rutinleri, sınıfın public rutinlerini tanımlarken, bu seviyede, sınıfa ait private rutinler dizayn edilmeli ve tanımlanmalıdır.
- Bu seviyenin detaylı dizaynı sıkça sınıf arayüzünü oluşturan public fonksiyonların değişmesine neden olabilir.
- Bu seviyedeki rutinlerin tanımı ve implementasyonu genelde programcıya bırakılır.

Seviye 5: Rutinlerin iç dizaynı

- Rutin seviyesinde dizayn rutinin iç tasarımı, yerleşimi ve fonksiyonlarının gerçeklenmesini içerir.
- Rutin içi dizayn,genellikle programcıya bırakılır.
- Bu seviyedeki dizayn genellikle Pseudo kod yazımı,
- Algoritmaların referans kitaplardan bulunması,
- Kodun formatının belirlenmesi,
- Ve programın kodlanmasını içerir.



Dizayn Yapı Taşları: Heuristics

Heuristic Bir Yöntem olarak Nesneler Dizayn Etme

- Nesnelerin metodlarını ve özeliklerini belirle (**Methods and attributes**)
- Her nesneye ne yapılabileceğini belirle. (**Diğer nesnelerle etkileşim**)
- Her nesnenin diğer nesnelere ne yapmasına izin vereceğini belirle.(**Diğer nesnelerle etkileşim**)
- Her nesnenin diğer nesneler tarafından görülebilir kısımlarını belirle. (Public and private parts)
- Her nesnenin public arayüzünü belirle.

Heuristic Bir Yöntem olarak Nesneler Dizayn Etme

- Bilgisayar programları çoğu zaman gerçek hayattaki nesneler üzerinde çalışır.
- Nesnelerin özeliklleri ve metodları nesnelri dizayn etmekten daha zor değildir.
- Şimdi hep birlikte bir bakkal dükkanı yönetim programı için nesneler geliştirelim.

Bakkal Dükkanı İçin Nesneler Ne Olabilir?

- Müşteri
- Toptancı
- Stok
- Alacak defteri
- Borç Defteri
- **Şimdi hep birlikte Müşteri ve Alacak Defteri Nesnelerini dizayn edip aralarındaki etkileşime bakalım.**

Şimdi Veresiye Defteri İle Müşteri Nesnelerini Dizayn Edelim



Teşekkürler