2. Hafta
2. Grup
Başlangıç

Bu kitabın üzerinde durduğu konu yazılım mühendisliği olduğu için, proje yönetimi de yazılım ağırlıklı olarak ele alınmaktadır. Bundan sonraki kısımlarda proje planlama süreçlerine daha ayrıntılı olarak değinilecektir.

3. Proje Tümleştirme Yönetimi

Proje Tümleştirme Yönetimi, çeşitli proje unsurlarının uygun bir eşgüdüm halinde tümleştirilmesini ve denetlenmesini sağlayan süreçleri tanımlar. Bu yönetim altında proje planı oluşturma, proje planının yürütülmesi ve tümleşik değişiklik denetimi bulunur.

3.1 Proje Planı

Proje Planı, diğer adıyla *Proje Yönetim Planı* (Project Management Plan) veya *Proje Geliştirme Planı* (Project Development Plan) stratejik planların uygulanabileceği şekilde projenin yürütülmesi ve denetlenmesine ilişkin planlamaların yer aldığı bir belgedir. Yürütmeye ilişkin çeşitli kılavuzluk bilgileri, tahminler, paydaşlar arası ilişkiler, iş paketleri, temel zamanlamalar, önemli aşama noktaları, kısıtlamalar, ölçme ve sabitleme için dayanaklar ile gerekli görülen diğer ayrıntılar bu belge içinde tanımlanır. Eğer farklı sözleşme yüklenicileri aynı proje altında çalışıyorlarsa bu plan *Tümleşik Proje Planı* adını alır. Yazılım projeleri için plan hazırlanmasında *IEEE 1058-1998 Software Project Management Plans* ve IEEE/EIA 12207 dikkate alınabilir. Örnek olarak bir şablon Ek-A.4 te verilmiştir.

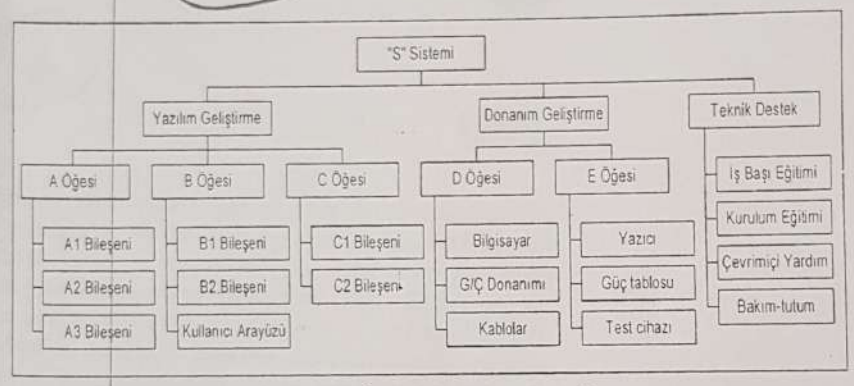
Proje Planı, belirlenen kıstaslara göre yürütülür ve uygulamalar Proje Yöneticisi tarafından denetlenir. Bu amaçla örgüt politikaları kullanılır, gerekiyorsa risk önleyici ve düzeltici tedbirler alınır. Uygulanan her aşama sonunda bir ürün, yani teslimat kalemi (deliverable item) ortaya çıkar.

Bu aşamada ortaya çıkabilecek değişiklik gereksinimleri düzenleşim yönetimi denetiminde yapılır.

3.2 İş Dağılım Ağacı

İş Dağılım Ağacı (Work Breakdown Structure), proje kapsamını tanımlayan ve düzenleyen bileşenlerin ürüne veya hizmete göre gruplanmasıdır. Bazen, *iş kırınım ağacı* veya *iş kırınım yapısı* ya da *iş dağılım yapısı* olarak adlandırılır. Genellikle sıradüzensel bir ağaç yapısı şeklinde veya bir liste halinde gösterilir. Her azalan düzey daha fazla ayrıntıda proje öğesinin tanımını temsil eder. Her öğe için *maliyet kodu* adıyla ayrı bir tanımlayıcı belirlenir. En alt düzeydeki öğeler *iş paketi* olarak tanımlanır. Her iş paketi için bir sorumlu (örgüt veya birey) belirlenir ve resmi olarak atanır. İş paketi tanımları genelde bir sözlük içinde toplanarak iş paketi tanımları oluşturulur.

Şekil-13.2 grafiksel yapıda bir İş Dağılım Ağacı örneği göstermektedir. Bu gösterim şekli çok karmaşık olmayan sistemlerin tamamını veya karmaşık sistemlerin yalnızca belirli bir düzeydeki bileşenlerini göstermek için elverişlidir.



Şekil-13.2. İş dağılım ağacı örneği.

Projelerde yürütülecek etkinliklerin ve işlerin tanımlaması yapılırken İş Dağılım Ağacı kullanılır. Bu durumda, sıralı bir liste yapılarak herbir iş için ayrı bir kod verilir. Buna *İş Kırınım Yapısı* veya *İş Dağılım Yapısı* demeyi tercih ediyoruz. Eğer liste maliyeti de içeriyorsa buna *Maliyet Kırınım Yapısı* (Cost Breakdown Structure) adı verilir.

Tablo-13.1. İş kırınım yapısı.

İKY Kodu	İş	Ederi
1	Malzeme alımı	6500
1.1	Bilgisayarlar (6 adet)	6000
1.2	Ağ elemanları	500
2	İsterler çözümlemesi	280
2.1	Yazılım isterleri	250
2.1.1	Bileşen-A çözümlemesi	100
2.1.2	Bileşen-B çözümlemesi	150
2.2	Güvenlik isterleri	30
3	Yazılım tasarımı	250
3.1	Bileşen-A tasarımı	100
3.2	Bileşen-B tasarımı	150
4	...	...

Bir bilgi sistemi geliştirme projesi için IEEE/EIA 12207 süreçleri göz önüne alınarak kullanılabilecek bir İş Kırınım Yapısı örneği Ek-B de verilmektedir. Bu liste herhangi bir projede temel alınıp ilgili kısımlar daha alt düzeylere indirgenebilir veya uygun olmayanlar çıkartılabilir. Önemli olan temel süreçlerin ve temel etkinliklerin atlanmadan uygulanmasıdır.

4. UML

Model kavramı, belirli bir amaca dönük sadeleştirmeyi ifade eder. Karmaşıklık ve ayrıntılardan kurtulmak için oluşturulur. Model oluşturma, yani modelleme, kendini kanıtlamış ve yaygın bir şekilde kabul görmüş önemli bir mühendislik tekniğidir. İnşaat alanındaki modelleme görsel algılamayı kolaylaştırır. Matematik modelleri birçok alanda fiziksel etkilerin gözlemlenmesine ve anlaşılmasına yardımcı olur. Prototipler makine ve araç üretiminin temelidir. Ekonomi modelleri ve iş yönetimi kavramları kamu ve iş dünyasında önemli yer tutarlar. Benzer şekilde, yazılım sistemlerinin modelleri yapılarak geliştirilecek sistemin daha iyi tanımlanması ve anlaşılması sağlanır. Modelleme, müşteri ile iletişimin kurulmasına, kullanılacak mimarinin değerlendirilmesine, sistem isterlerinin ve davranışlarının daha iyi anlaşılmasına yardımcı olur. Sistemin gelişimi ve güncellenmesi gibi işlemlerin daha az maliyetli bir şekilde yapılmasına olanak tanır.

Yazılım modelinin anlaşılabilir ve kullanılabilir olması, kararlı bir biçimde kullanılan ve anlaşılan ortak bir gösterimi gerektirir. Ortak gösterime sahip, tam ve doğru tanımlanmış bir yazılım modeli, geliştiricilere paralel çalışma olanağı da sağlar.

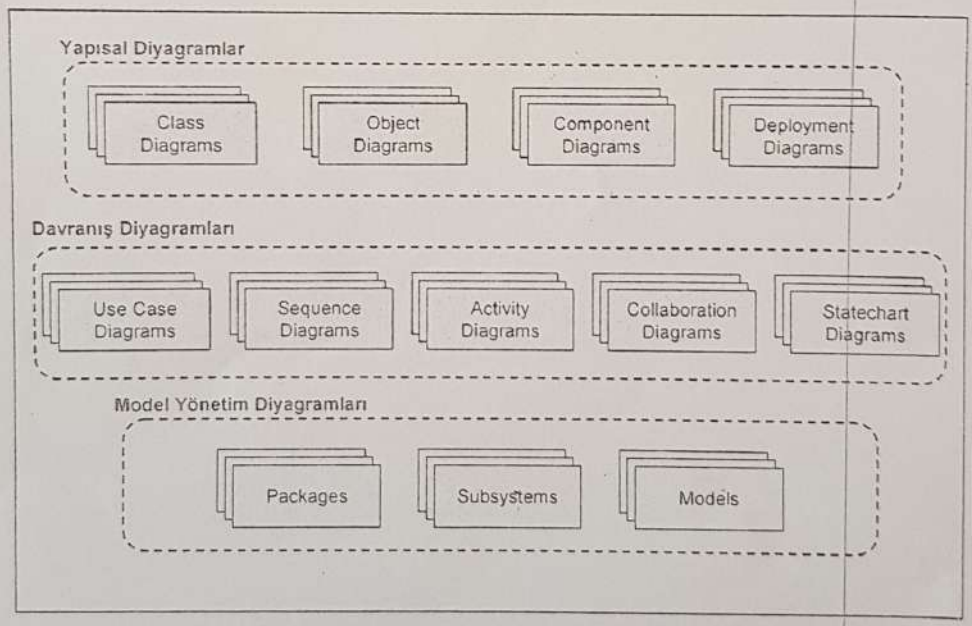
Unified Modelling Language (UML) son yıllarda ortaya çıkmış en güçlü yazılım çözümleme ve tasarım yöntemlerinden biridir. Hem metinsel hem de grafiksel olarak nesneye yönelik çözümlemenin ve tasarımın yapılabilmesine yardımcı olur. UML, görsel öğeler kullanan bir modelleme dilidir, bir süreç değildir. En iyi pratiklerden elde edilmiş deneyimlerle oluşturulmuş bir açık standarttır ve çeşitli yazılım araçları tarafından desteklenmektedir. SysML adıyla genişletilerek sistem mühendisliğinin de kapsanmasına başlanmıştır.

Çeşitli yöntemlerin birleştirilmesi ile 1996 yılında ortaya çıkarılan UML, dünya çapında yazılım mühendislerinden oluşmuş Object Management Group (OMG) tarafından 1997 yılında ilk standart olarak kabul edilmiştir. Halen *UML Altyapı* ve *UML Üstyapı* belirtimlerinin Şubat 2007 tarihli ve 2.1.1 nolu sürümleri yürürlüktedir.

Bu kısımda UML modellerinde kullanılan temel unsurlar kısaca tanıtılmaktadır. Anlatımlar ve tasarım aşamaları özet halinde verilmiş olup temel bir kaynak olarak alınmamalı, bu maksatla daha ayrıntılı olarak hazırlanmış başka kaynaklar kullanılmalıdır. UML belirtimleri ile ilgili bilgiler http://www.omg.org adresindedir.

4. UML Diyagramları

UML in temel yapı taşları varlıklar, ilişkiler ve diyagramlardır. Varlıklar da yapısal ve davranışsal olarak ikiye ayrılmaktadır. Yapısal varlıklar UML modellerindeki kavramsal veya fiziksel elemanları temsil eden isimlerdir. Davranışsal varlıklar ise modellerdeki zaman ve uzaydaki davranışları gösteren dinamik kısımlar, yani eylemlerdir. Varlıklar arasında sıradüzensel yapılar kurmak ve özelliklerine göre gruplandırmak üzere ilişkiler kullanılır. Diyagramlar bir varlık kümesinin grafiksel gösterimidir. Bir sistemi değişik açılardan görsel bir şekilde modellemeye yararlar. UML üç gruba ayrılan toplam 12 adet diyagram tanımlamaktadır. Bunlardan dört tanesi uygulamanın durağan yapısını gösterirken, beş tanesi dinamik davranışları gösterir; üç tanesi de uygulama modüllerini düzenlemeye yarar. Bu diyagramlar İngilizce isimleri ile Şekil-C.1 de gösterilmektedir. Daha sonraki kısımlarda Türkçe karşılıkları kullanılacaktır.



Şekil-C.1. UML diyagramları.

4.1. Modelleme Elemanları

UML ile kullanılan modelleme elemanları şunlardır:

- *Aktör* (Actor), sistemin kullanıcılarını tanımlamak için kullanılır. Tasarlanmakta olan sitemin kullanıcısı bir insan, bir başka sistem ya da bir cihaz olabilir. Aktör, sistemden hizmet isteğinde bulunabilir, sisteme hizmet verebilir. Birincil Aktör (Primary Actor) sistemden asıl yararı sağlar; genellikle işlemleri başlatan kullanıcıdır. Destek Aktörü ise sisteme bilgi veya destek sağlar; genellikle bir bilgisayar sistemidir.

- *Kullanım Senaryosu* (Use Case), sistemdeki aktörler için, belirli bir sonuç değerini oluşturan, bir dizi eylem bütünü olarak düşünülebilir. İşlevsel isteklerin anlaşılmasına yardımcı olan bir yöntemdir.
- *Yapısal Elemanlar* (Structural Elements) sistemde bulunan yapı taşlarını, yani sınıfları, nesneleri ve paketleri içeren durağan kısımlardır.
- *Davranışsal Elemanlar* (Behavioral Elements) kullanılan eylem sözcükleri yardımıyla yazılım modelinin dinamik kısımlarını tanımlar.
- *Gruplama Elemanları* (Grouping Elements), diğer adıyla paketler, yazılım elemanlarını gruplamak amacıyla kullanılırlar. Paketlerin içinde sınıflar, arayüzler, diyagramlar, hata başka paketler bulunabilir.
- *Nesneler* (Object), yazılım modeli içindeki belirli ve sonlu sayıda olan sistem elemanlarıdır. Nesnelerin karakterleri ve bu karakterlerin üzerinde duran davranışları vardır. Nesneler bu yapılarıyla (karakter, davranış) sistem üzerinde kendilerine verilen görevi yerine getirirler.
- *Sınıflar* (Class), nesnelerin sistem içerisinde oynadıkları rolü, yani nesnelerin öznitelikleri ile iş yapan yordamlarını (metot) kapsayan yapılarını belirlerler. Sınıflar nesneler için bir kalıp görevi görürler. Sistem içerisinde nesneler, bu kalıba göre oluşturulurlar, belirli etkinlikleri yerine getirirler ve sonlanırlar. Sınıflar, bir yandan varlıkların belirli özellikleri ve davranışlarına odaklanarak soyutlama sağlarken diğer yandan varlıkların bazı özelliklerini ve davranışlarını kapsayıp dış dünyadan gizleyerek karmaşıklığı azaltırlar.
- *Aktif Sınıflar* (Active Class) kendi başlarına var olabilen, uygulamanın akışını kontrol edebilen nesneleri tanımlarlar.
- *Arayüzler* (Interface), nesnelerin davranışlarını belirleyen kurallar bütünü olarak düşünülebilir. Sınıflar, davranışları belirleyen arayüz yordamlarını gerçekleştirirler.
- *İşbirliği* (Collaboration), belirli bir amaca yönelik kullanım senaryolarının birleştirilmiş halidir.
- *Bileşenler* (Component) yazılım isterlerinin belirli kısımlarını gerçekleştiren modüllerdir.
- *Düğümler* (Node), sisteme, hesaplanabilir ve ölçülebilir durumda olan herhangi bir kaynak sunan fiziksel elemanlardır.

4.2. Nesneler Arası İlişkiler

Nesneler arasındaki ilişkilerin tanımlanması UML modellerinin en önemli özelliklerindendir. Bu amaçla aşağıdaki ilişki türleri vardır:

- *Bağımlılık* (dependency) nesneler arasındaki en zayıf ilişkidir. Bir nesnenin kısa süreliğine başka bir nesneye olan bağlılığını ifade eder. Bağımlı olan nesne, ilişkide bulunduğu nesnenin yordamlarını ve alanlarını kullanabilir.
- *Birliktelik* (association) nesneler arası uzun süreli ilişkidir. İki tür birliktelik vardır: Topluluk (aggregation) ve oluşum (composition). Topluluk ilişkisi, bir

nesnenin bütünün bir parçası olma durumuna işaret eder. Katılımcı nesne, birden fazla topluluk ilişkisinde de bulunabilir. Ayrıca katılımcı nesne, topluluk ilişkisinde bulunduğu yapıdan bağımsız olarak da var olabilir. Oluşum ilişkisi bir tür sahiplik ilişkisidir. Bir nesnenin bütünün bir parçası olmayıp, bütünün sahip olduğu bir parça olma durumuna işaret eder. Katılımcı nesne, birden fazla oluşum ilişkisinde bulunamaz ve bütün yapıdan bağımsız bir şekilde düşünülemez.

- *Çokşekillilik* (polymorphism) kavramı, nesneye yönelik programlama deyimlerinden biri olarak soyutlamanın çoklu uygulamaları olarak tanımlanmaktadır. Çokşekillilik, sistem içerisinde bir soyutlamanın birden çok uygulamasını kullanmak suretiyle, uygulamaya esnek bir yapı kazandırır. Çokşekilliliğin bir avantajı da, uygulamaya genişleyebilirlik katmaktır.
- *Genelleştirme* (generalization), genel bir yapıya sahip sistem elemanı ile, özelleşmiş bir nesne arasındaki ilişkidir.
- *Gerçekleştirme* (realization) bir sınıfın bir arayüze erişerek arayüzün yordamlarını kullanılır hale getirmesidir.

4.3 Diyagramlar

Diyagramlar, modelleme elemanları ile bunların birbirleriyle olan ilişkilerini belirten yazılım modellerinin görsel şekilleridir. Başlıca UML diyagramları şunlardır:

- **Kullanım Senaryosu Diyagramı (use case diagram)**
  Kullanım senaryosu, yazılım sisteminin işlevlerinin belirli bir bölümünün yerine getirilmesi için uygulanır. Herbir senaryo, sistemle aktör arasındaki etkileşimi belirleyerek yazılım sisteminin işlevselliğini ortaya koyar. Temel öğeleri, yazılım sisteminin ilgilenilen kısmının modeli olan sistem, bir anlatım olan kullanım senaryosu, aktör olarak tanımlanan sistemin kullanıcıları ya da sistem üzerinde rolleri olan diğer yapılar ve aktörlerle veya diğer kullanım senaryolarıyla olan ilişkilerdir.
- **Sınıf Diyagramı (class diagram)**
  Diğer nesneye yönelik yöntembilimlerde kullanıldığı gibi, UML de de sınıf diyagramları sistemin durağan yapısını tanımlarlar. Temel öğeleri normal ve aktif sınıflar, erişim durumu ve ilişkilerdir.
- **Nesne Diyagramı (object diagram)**
  Sınıf diyagramlarının doğruluğunun test edilmesi için kullanılan nesne diyagramları sistemin belirli bir andaki durumunu gösterirler.
- **Ardıllık Diyagramı (sequence diagram)**
  Belirli bir zaman dilimine yayılmış nesne etkileşimlerini gösterir. Nesneler arasında zamana bağlı iletişimi ve işlevselliği tanımlar. Nesneler ve iletilerden oluşur.

7

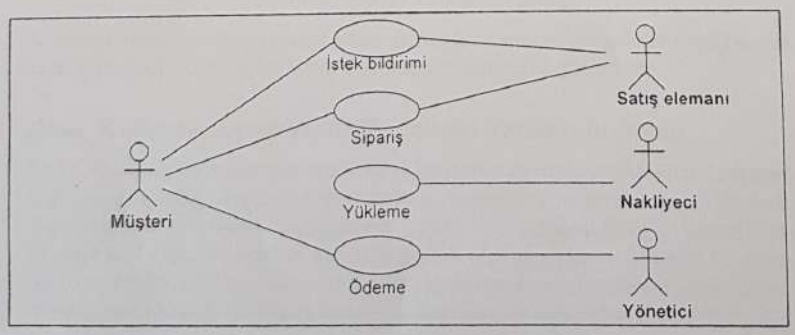
- **İşbirliği Diyagramı (collaboration diagram)**
  İleti yollayan ve alan nesnelerin yapısal örgütlenmelerini gösteren etkileşim diyagramıdır. Sistemin durağan ve dinamik davranışlarını birlikte gösterir.
- **Durum Diyagramı (statechart diagram)**
  Sistem içindeki sınıfların dışarıdan yapılan uyarılara ve tetiklemelere karşı sergilediği davranışları gösterir. Nesnelerin gelişen olaylara karşı içinde bulunduğu durumları modeller. Genel yapısında başlangıç ve sonlanma durumları, geçiş koşulları ve davranışlar bulunur.
- **İşleklik Diyagramı (activity diagram)**
  Bunlar da sistemin akışını, dolayısıyla da işlevselliğini gösterirler.
- **Bileşen Diyagramı (component diagram)**
  Sistemde bulunan bileşenlerin örgütlenmelerini ve birbirleriyle olan bağlılıklarını gösterir.
- **Konuşlandırma Diyagramı (deployment diagram)**
  Sistemdeki düğümleri, bileşenleri ve çeşitli bağlantılar içeren fiziksel varlıkları gösterir.

1. Hafta
2. Grup
SON

5. Kullanım Senaryoları

3. Hafta
1. Grup
Başlangıç

UML'in çözümleme yöntemi kullanım senaryolarıdır (use cases). Bir kullanım senaryosu, sistemin iç yapısını göstermeden kullanıcıya bir sonuç değeri veren, sistemin belirli bir işlevidir. Sistem bir kara kutu olarak ele alınarak sistemin yerine getirmesi gereken sorumluluklar tanımlanır. Genel olarak müşteriyle görüşülerek belirlenirler. Kullanım senaryolarının birer amacı vardır. Var olan, gözlemlenebilir ve ölçülebilir bir değeri kullanıcıya verirler. Senaryo, anlamlı bir sonuca ulaşmak için aktör ile sistem arasında gerçekleşen olaylar zinciridir. Olası senaryoların tamamı ayrı ayrı tanımlanır. Şekil-C.2, kullanım senaryosu diyagramına örnek olarak aktörleri ve kullanım senaryoları ile olan ilişkileri göstermektedir.



Şekil-C.2. Kullanım senaryosu diyagramı.