Öneri. C.3. S.14.
Haziran 2000. ss.89-94.

JACOBSON'UN ÖNERİLERİ İLE NESNEYE YÖNELİK UYGULAMA GELİŞTİRMENİN TÜM YAŞAM DÖNGÜSÜ

Zerrin AYVAZ REİS

İstanbul Üniversitesi Teknik Bilimler Meslek Yüksekokulu Bilgisayar Programcılığı Programı, Öğr.Gör.Dr.

Abstract: Jacobson suggested OOSE to implement for full life cycle software engineering. Aim of this paper; to introduce Jacobson's method which is called OOSE (Object Oriented Software Engineering).

I. GİRİŞ

Yazılım mühendisliğinde tekrar kullanılabilirlik, üretimselliğin önemli bir şekilde arttığı kritik bir konu olduğundan uzun bir süredir düşünülmektedir. Ancak hala, yazılım mühendisliğinde yaygın olarak ilgi görmemektedir ve bunun da bazı sebepleri mevcuttur.

Nesneye yönelik teknoloji, tekrar kullanımı arttırdığından gelecek vaad eden bir yaklaşımdır.. Bileşenlerin yeniden kullanımı bizim programlama çabalarımızı olgusal seviyede arttıracağından, böylece üreticiliğimizi de arttıracaktır.

Var olan sistem geliştirme metotları (a)fonksiyon/veri metotları ve (b)nesneye yönelik metotlar olmak üzere ikiye ayrılır. Fonksiyon/veri metotları da fonksiyon ve verilere ayrılır. Öyle ki fonksiyonlar temelde aktiftirler ve kendine ait davranışları vardır ve veriler fonksiyonlar tarafından etkilenen pasif bilgi saklayıcılardır. Sistem tipik olarak fonksiyonlara bölünür ve veri bu fonksiyonlar arasında birbirlerine gönderilir. Fonksiyonlar da kendi içlerinde daha ufak parçalara bölünür ve kaynak koda dönüştürülürler.

Fonksiyon/veri metodu kullanarak sistem geliştirmek genelde zordur. Fonksiyon/veri metodundaki ana problem temelde bütün fonksiyonların verinin nasıl saklandığını ve hangi tipte olduğunun bilinmesi gerekliliğidir. Genellikle değişik veri tipleri arasındaki veri formatlarında çok az fark vardır. Bu da demektir ki, genellikle veri tiplerini doğrulamak için durum şartlarına ihtiyacımız olur. Bu yüzden programlar aslında hiçte fonksiyonel olmayan ve sadece değişik veri formatları ile ilgilenen "if-then" veya "case" tiplerine ihtiyaç duyarlar. Böylece eğer veri formatları ile değil de sadece programın fonksiyonelliği ile ilgileniyorsak programı okumak çok zorlaşır. Daha kötüsü, veri tipini değiştireceğimiz zaman o veri ile ilgili bütün fonksiyonları da güncellemek gerekir. Bu metotları kullanarak geliştirilmiş bir sistem kolayca kararsız hale gelebilir ve üzerinde yapılan ufak değişiklikler diğer alanlarda da istenmeyen sonuçlar yaratabilir.

Şunu bilmeliyiz ki nesneye yönelik bir yöntemle sistem, içten de gereksinim spesifikasyonları (özellikleri) doğrultusunda düzenlenir.

Şimdi fonksiyon/veri sistemlerini değiştirmenin daha zor olmasının sebebi üzerinde duracağız. Fonksiyon/veri tasarımları davranışlara göre yapıldığı için her hangi bir değişiklik çok daha büyük değişikliklerin sebebi olur. Nesneye yönelik metotlarda problemin merkezindeki nesne (item) temel alınarak sistem kurulur. Bu çoğunlukla sistem tanımlamak için daha doğal bir yöntemdir. Bu nesneler normalde sabittirler ve çok az değişirler. Yapılan değişiklikler normalde sadece bir tane veya az sayıda nesneyi etkiler, bu da demektir ki sistemde değişiklikler genellikle yerel olarak yapılır. Eğer sağlam bir sistem istiyorsak; neyin değişime meyilli olduğunu belirlemeliyiz ve buna göre tasarım yapmalıyız. Peter Coad ve Ed Yourdon (1991) Tablo.1.in yapılabilmesine ilham kaynağı oluşturmuşlardır.

Çoğu nesneye yönelik yöntem, temellerini değişme ihtimali az olan nesneler üzerine kurar. Özellikle süreç programlarken veya tasarlarken de nesneye yönelik bir bakışla ele alınabileceği söylenir. Nesneye yönelik programlama fonksiyon/veri geliştirme metoduna göre farklı bir yaklaşım gerektirir

Bu yaklaşımdaki kaymanın sebebi, hangi nesnenin sistem yapısında asli olduğunun geliştirme işleminde çok geç belirlenmesidir. Nesneye yönelik programlama ile ilişkili genel görüş, uygulamanın temelindeki önemli maddelerin nesne olarak gösterilmesidir. Eğer biri nesneye yönelik

programlama dili ile ilişkili katı fonksiyonel analiz metodu kullanıyorsa nesne tasarım aşamasına kadar tanımlanmayacaktır. Müşteri ve son kullanıcı ile ilişkisi olmayan insanlar tarafından tanımlanacaktır. Bu tanımlanmış nesnelerin muhtemelen uygulamanın temelindeki bütün önemli maddeleri temsil etmemesi önerilir.

Tablo.1. Nesne-Değişiklik Tahmini

NESNE	DEĞİŞİKLİK TAHMİNİ
Uygulamadan nesneye	Düşük
Uzun süreli bilgi yapıları	Düşük
Pasif nesne nitelikleri	Orta
Davranışın sırası	Orta
Dış dünya ile arayüz	Yüksek
Fonksiyonel	Yüksek

Şimdi nesneye yönelik bir bakış açısından değişik sistem geliştirme süreçlerini tartışmalıyız. Bir şekilde var olan bir gereksinim özelleştirilmelerinin gerçekleştirilmiş olduğunu kabul edelim. Bu, hangi nesnelerin sistem içinde olması gerektiğini bulmak amacıyla analiz edilsin. Ele geçen nesne modelleri tasarım işlemini gerçekleştirdiğimiz sırada tasarlanmıştır.

II. NESNEYE YÖNELİK ANALİZ

Nesneye yönelik analizin amacı, diğer tüm analizlerle birlikte uygulamanın (sistemin foksiyonel gereksinimleriyle) anlaşılmasıdır.

Nesneye yönelik analiz ile fonksiyon/veri analizi arasındaki fark, daha önceden belirtildiği gibi ifade farkıdır. Fonksiyon/veri analiz metodu sistem davranışı ve/veya veriyi ayrıca incelerken, nesneye yönelik analiz bunları birleştirir. Nesneye yönelik analiz, sistem davranışının ve açıklamasının analizi arasında bir tekrarlama olarak karakterize edilebilir.

Nesneye yönelik analiz her hangi bir sırada aşağıdaki aktiviteleri içerir.

- Nesneleri bulma,
- Nesneleri düzenleme,
- Nesnelerin ilişkilerini açıklama,
- Nesnelerin operasyonlarını (yapacağı işleri) tanımlama,
- Nesneleri içsel (internaly) tanımlama.

II.1. Nesneleri Bulmak

Nesneler, doğal olarak uygulama temelinde sahip oldukları değerler olarak bulunabilirler. Bir nesne tipik olarak temelde isim olur. Bu yüzden problem kökü için en iyi başlangıç terminolojiyi öğrenmektir. Nesneleri bulmak aslında çok kolaydır, asıl zor ve önemli olan sistemin asıl nesnelerini bulmaktır. Bunlar, sistemin yaşam döngüsü içinde sürekli ayakta kalan (önemli olan) ve en az değişim geçiren nesnelerdir. Bir su tankını kontrol eden uygulama için bu nesneler; su tankı, regülatör, vana olacaktır. Bir bankacılık uygulaması için; müşteri, hesap, vezne, vb. olacaktır.

Var olan nesneye yönelik metotların büyük bir çoğunluğu tek tip nesneye sahiptir. Bu sayede daha kolay ve genel bir modele sahip olurlar. Değişik nesne tipleri kullanmak için çeşitli sebepler vardır. Tek tip bir nesneyle nesneler arasındaki farkları görmek daha zordur. Ayrıca çeşitli nesne tipleri kullanmanın diğer avantajı, sistemi geliştirmek ve desteklemek için değişik kuralları olan nesneler kullanabilmektir. Örneğin; sabit bilgiler içeren pasif nesneler, arayüzle ilişkili olan nesnelerle ilişkili olmamalıdır. Zira, arayüzde değişiklikler daha fazladır.

Değişik nesneler değişik karakterlere göre organize edilebilirler.

II.2. Nesneleri Organize Etmek

Nesnelerin ve nesne sınıflarının sınıflandırılması için bir çok kriter vardır. Birincisi nesne sınıflarının birbirlerine ne kadar yakın olduğunu incelemektir. Bu kalıtım hiyerarşisinin temellerini oluşturur. Bir sınıf başka bir sınıfa miras bırakabilir. Bir diğeri hangi nesnelerin, hangi nesnelerle çalıştığını veya hangi nesnelerin nasıl parçası olduğunu incelemektir. Yakın bir sınıflandırma ise, bir nesnenin bir şekilde diğer nesnelerle nasıl bağımlı olduğunu incelemektir. Böylece alt sistemlere gruplamanın temellerini oluşturan değişiklikler içerir.

II.3. Nesne Etkileşimi

Nesnelerin sistem içindeki yerlerini anlayabilmek için nesnenin diğer nesnelerle ilişkilerini ve diğer nesnelerin parçası olduğu senaryolar veya kullanım durumları (use case) tanımlayabiliriz. Bunlara bakarak, nesnenin arayüzünü tamamen hazırlayabiliriz. Nesnelerin diğer nesnelerle nasıl iletişim kuracağını ve nesneden neler beklendiğini görebiliriz. Aynı zamanda belli nesnelerin nasıl parçası olduğu üzerinde düşünebiliriz.

II.4. Nesneler Üzerinde Operasyonlar

Nesne operasyonları arayüzler düşünüldüğünde gündeme gelir. Nesneler doğrudan uygulama tarafından da tanımlanabilir. Otomatik tanımlanan nesneler tek iş yapabilir veya daha az karmaşık olabilir. Örneğin; bir kaç nesneden bilgi alıp rapor hazırlanabilir, ama mümkün olduğunca çok karışık nesnelerden kaçınmalıdır.

II.5. Nesnelerin Gerçekleştirimi (Implementation)

Son olarak nesneler içsel olarak tanımlanmalıdır. Bunun içine, nesnelerin içereceği bilgileri tanımlamak, her nesnenin içerebileceği örnek sayısını belirtmek girer. Bazı nitelikler miras bırakılabilir.

Yukarıdaki maddeleri açıklamak gerekirse, bütün geliştirme basamakları birbirine bağlıdır ve etkileşimli şekilde çalışırlar. Analitik safhada temel problemi anlamak ve onun üzerinde yoğunlaşmak önemlidir ve bunun sonuçları bütün işi etkiler. Tecrübeler göstermiştir ki, uygulama temelinde tanımlanan nesneler çok sağlamdır ve biraz çalışma sonrasında bütün iş ve dökümanlar bunlar çevresinde dönmeye başlarlar. Daha önceden belirtildiği gibi nesneye yönelik analizin bir avantajı da temel ile metot arasındaki mantıksal boşluğu, insancıl bakış açısıyla en aza indirmesidir.

Bir avantajı da, az değişim ihtimali olan nesneler doğal olarak tanımlanmıştır ve çok değişme ihtimali olan maddeler erken evrelerde izole edilebilir.

OOSE bu maddeleri biraz değişik bir sırada içerir. Burada tartışılan bazı nesneler OOSE'de değişik şekilde bulunur.

III. NESNEYE YÖNELİK YAPIM

Nesneye yönelik yapım, kaynak koduyla tasarım ve tamamlamanın yapılması demektir. Bu kaynak kod hedef ortamda çalıştırılır.

Bütün tasarım aktivitelerinde olduğu gibi, yapı ve verimlilik arasında iyi bir denge kurmak zordur. Analiz modeli bize olabildiğince ideal, izlememiz gereken bir yapı sunmuştur. Bu değiştirilmeye karşı korunmuştur. Tasarım sırasında biri bütün sınırlayıcı istekleri denetlemelidir (Örneğin; hedef ortamına bağlı olarak, en az hafıza kullanımı , güvenilirlik, cevaplama zamanı) Bunların hepsi yapıyı etkileyebilir.

Amaç olarak analiz aşamasında tanımlanan bütün nesneler, tasarım aşamasında da olmalıdır. Buna izlenebilirlik (tracebility) diyoruz. Bu yüzden tasarım aşamasından; analiz ve programlama aşamasına geçerken katı kurallarımız olmalıdır.

Nesneler daha önce yazılmış kodlar kullanılarak da tasarlanabilir. Bu parçalara bileşenler (component) diyoruz. Bu bileşenleri oluşturmak nesneye yönelik ortamda fonksiyon/veri bütünleşmesi sayesinde daha kolaydır.

Nesneye yönelik programlama nesneye yönelik sistem geliştirme gerektirir. Aksi halde zor bir paradigma kayması olur. Nesneye yönelik analizden klasik programlamaya geçmek de bir paradigma kaymasına sebep olur. Bununla beraber bu daha az probleme yol açar. Nesneye yönelik olmayan dillerle, nesneye yönelik bir sistem yapılabilir. Ek olarak, diğer diller için bile nesneye yönelik stilde programlar yapabiliriz. Nesneye yönelme sadece kalıtım, çok şekillilik, kapsüle etmeyi içeren bir dil olmakla kalmaz, aynı zamanda bir programlama tekniğidir. Bir örnek kayması olur ama, göreceli olarak yumuşatılabilir. Daha sonra, örneğin tipik bir fonksiyon/veriye yönelik ilişkisel veri tabanını bir nesne içine yerleştirip, paradigma kaymasını gerçekleştirmeyi yerel olarak göreceğiz.

Tabi ki fonksiyon/veri metodunda da çok iyi teknikler geliştirilmiştir ve bunlar nesneye yönelik metotlarda kullanılabilir. Sadece örnek kayması yapabildiğimiz için bütün öğrendiklerimizi unutmamalıyız. Örneğin; nesneye yönelik programlamada kullanılan bir teknik de durum diyagramlarıdır.

IV. NESNEYE YÖNELİK TEST ETME

Nesneye yönelik geliştirilmiş bir sistemi test etmek, diğer metotlarla geliştirilmiş bir sistemi test etmekten daha farklı değildir. Her şekilde sistemi doğrularız., gereksinime göre yapılandırdığımız sistemi isimlere göre kontrol ederiz. Bu doğrulama en erken sürede başlamalıdır. Programın testi en alt seviyeden birim testi ile başlar, böylece birimlerin birbirleriyle doğru etkileşim kurduğunu doğrularız. Son olarak geçerli sistem için test biter.

Geleneksel olarak entegrasyon test, sistem geliştirilmesi sırasında çok önemlidir ve bir "big bang"(büyük patlama) olayıdır. Bu seviyede, geliştirilmiş parçalar birleştirilir ve gerçekten birlikte çalışabildikleri kontrol edilir. Nesneye yönelik sistemlerde böyle bir "big bang" çarpıcı değildir.

Nesneye yönelik bir sistem, bir çok ilişkili nesneden oluşur. Bu nesneler davranışa ve veriye sahiptirler. Bu da onları, klasik bir sistem geliştirme metodundakilere göre, daha büyük birimler yapar. Bir nesnenin operasyonları belli bir veri çevresinde olur ve bir geliştirici, normalde bütün operasyonları bir nesne üzerinde geliştirir. Bu birim testini, klasik sistem geliştirmedekine göre daha geniş hale getirir. Entegrasyon testi erken bir evrede başlar ve daha üst seviyelere kadar devam eder.

Bununla beraber sınıflar arası kalıtım, test etmede yeni zorluklar yaratabilir. Bildiğimiz gibi kalıtım, bir sınıf için tanımlanan operasyonun diğer bir operasyona miras kalması ve onun tarafından kullanılabilmesi demektir. Bunun için sınıfların ortak bölümlerini tutan sanal sınıflar olabilir. Bu sanal sınıfları, eğer operasyonu bütün olası durumlarda test etmek yerine bir kere test ediyorsak, test etmeye değer. Sanal sınıftaki operasyon duruma göre değişen özelliklere sahipse, normal olarak operasyon her yeni şekilde test edilmelidir.

Bu sebeple, kalıtım hiyerarşilerinin daha sıhhatli bir metotla test edilmesi gerekir. Sistemin operasyonda nasıl görüneceğini bilmeliyiz. Örneğin; kalıtım hiyerarşisi seviyesinde test edilen operasyonun alt seviyede test edilmesinin gerekli olmadığı görüşü yanlıştır. Bir operasyon kendini yeni bir çevrede bulabilir ve bu çevrede daha önce test edilmemiş olabilir. Bu demektir ki, kök seviyedeki bir operasyonu değiştirirsek, bu operasyonu muhtemel tüm durumlarda test etmeliyiz. Aynı şekilde, yeni bir olasılık eklersek, bununla ilgili tüm operasyonları yeni çevrelerinde test etmeliyiz.

Bu yüzden geri çekilme ve otomatik test, nesneye yönelik geliştirilmiş sistemleri test etmede büyük rol oynar. Burada hangi test verisinin kullanıldığına dikkat edilmelidir. Eğer olası operasyonlar atlanırsa, hazırlanmış test verisi yeni operasyonlar için yeterli olmaz. Test verileri her ne kadar bir operasyonu kullanmanın muhtemel yönlerini araştırarak işe başlasa da, eski test verisinin yeni operasyonları test edebileceği kesin değildir. Uç durumlarda, her kalıtım hiyerarşisi seviyesi için özel test verisi hazırlamamız gerekebilir. Nesneye yönelik yazılımı test etmenin diğer problemleri çok şekilliliği içerir.

Bir başka problem de, bir nesne kendisine bir uyarı gönderirse ortaya çıkar. Böyle bir uyarı belki sadece bir ihtimaldir. Bu yüzden bu durumda sanal sınıfı test etmek gereksizdir. Çok iş yapılmasını gerektirir. Bu tür problemlere Yoyo problemi denir.

Nesneye yönelik bir yaklaşımla, testi de bir nesne gibi kabul edebiliriz. Böylece testin bir sınıfı ve bir veya daha çok örneği vardır. Sadece sonucuyla ilgilendiğimiz,, çalışmasıyla ilgilenmediğimiz için kapsüle edilmiştir (encapsulated). Hatta bazı parçalarını kalıtımla karşı karşıya bırakabiliriz. Örneğin; başlangıç gibi basit parçalarını kalıtlayabiliriz. Sonraki durumda, bir test gereksinimi, değişik sistem versiyonları için, değişik versiyonlar geliştirebilir.

V. Objectory ve OOSE

Jacobson metodolojisi aşağıdaki adımları sağlamaktadır;

1. Gereksinim Modeli,
2. Analiz Modeli,
3. Tasarım Modeli,
4. Geliştirme Modeli,
5. Test Modeli,

Analiz işlemi kurulacak sistemi açıklamak ve belirginleştirmek amacını taşır. Iki model geliştirilmiştir. Gereksinim modeli ve analiz modeli. Her iki model de mevcut uygulama çevresinin her hangi bir gerek şartını içermeyen anlamda mantıksaldırlar.

V.1. Gereksinim Modeli

Gereksinim modeli; aktörleri ve kullanım durumlarını kullanarak sistemin her ayrıntısını kullanıcının açısından tek tek açıklamakta kullanılır. Aktörler, sisteme dahil olan makineler ve insanlar gibi dış faktörleri oluşturan bir modeli kurarlar. Kullanım durumları (use case) iş akışlarıdır (flows) ve bu aktörler sistemde kullanılacaklardır. Use case'ler, teknik olmayan personel tarafından algılanacak ve böyle potansiyel bir kullanıcının da katılımı ile sistemin fonksiyonel ihtiyaçlarının tanımlanması ve iletişimi ile ilgili ana yapı şekillenecektir.

Gereksinim modelinin açıklanmasına bir destek olarak, problem merkezli nesne modeli, sistemin neyi ele alacağına karar verecek yerel bir platform oluşturacağından kuvvetli bir aracı oluşturmaktadır. Yüksek derecede bir desteklenebilmeye ulaşmak durumunda, bu problem merkezli nesneler, sistemin gerçekleşmesi için bir temel oluşturmayacaktırlar.

Kullanım durumları ve problem merkezli nesnelerin bütünlük oluşturması için, gereksinim modeli, sistem arayüzlerinin tanımlamalarda birlikte kullanılmaları ayrıca gereklidir.

Kullanım durumu (use case) modeli, nesneler üzerinde tanımlayıcı işlemlerden sistemlerin yapılanmasını ve, aynı değerde, entegrasyon testini yapacak bir araç olarak da, OOSE'deki tüm gelişme çalışmalarının kullanımında bağ görevi oluşturacak bir anlam kazanacaktır.

V.2. Analiz Modeli

Analiz modeli, analiz işleminde geliştirilmiş ikinci bir modeldir. Bu model sistemin yapısına onarılabilir ve mantıklı bir şekil kazandırmayı amaçlar. Mevcut uygulama çevresini dikkate almamak durumu mantıklıdır. Buna sebep, ana amacın gerekli sistem fonksiyonelliği üzerine odaklanmasıdır. Sistemi kurmada üç tip nesne kullanılmıştır; arayüz nesneleri, varlık nesneleri ve kontrol nesneleri. Arayüz nesneleri sistem arayüzlerini ilgilendiren tüm fonksiyonel modeli oluştururlar.

Varlık nesneleri, sistemde uzun süre tutulacak olan mevcut bilgiyi ele alırken; kontrol nesneleri böyle herhangi bir nesneye doğal olarak bağlı olmayan (sıklıkla davranış ağırlıklı) fonksiyonelliğin modelini oluştururlar. Bu nesne tipleri kullanım durumu (use

case) analiz edilip kopukluk olduğunda tanımlanabilir. Nesneler, kullanım durumunun (use case) tüm fonksiyonelliğini sağlamalıdırlar.

Alt sistemler, sistemin daha büyük ünitelerini oluşturmada kullanılırlar. Alt sistemler nesneleri analiz modelinde gruplar. Bu modellerin gelişimi, bir örnekleme işlemidir ve istikrarlı şekle gelene dek pek çok değişikliğe uğrayacaklardır. Bu nedenle, olgunlaşmış bir seviyeye ulaşmadan önce bu modellere fazla detay konulmamalıdır. Yine de erken modeller geliştirilirken bile, bir sonraki modellerin taslağı çıkarılmalıdır. Erken modellerde istikrar sağlandığında, bir sonraki modeller üzerinde çalışma başlatılabilir.

Yapılandırmada sistemin tasarımı ve uygulanmasını gerçekleştiririz. Yapılacak iş analiz sonucu ortaya çıkacak bilgi üzerine kuruludur. Analiz modeli, mevcut çevreyi hesaba katmaksızın sistemi ideal şartlar dahilinde açıklar. Bu modelin amacı tüm sistemin yaşam süreci üzerinde sağlam ve desteklenebilir bir yapıyı elde etmektir. Yapılandırmada bu ideal modeli bilinen şartlara uyarlamalıyız.

Analiz modeli, olgunlaşmış bir düzeye varmaya başladığında, uygulama stratejisi bu modele eklenmelidir. Bu bize mevcut sistem mimarisie ilk gerçek yaklaşımı verecek. Bu mimarinin tutarlı ve güçlü olmasını istediğimizden analiz modelinden olabildiğince az sapma yapmak büyük önem taşıyor.

V.3. Tasarım Modeli

Tasarım modeli, analiz sırasında belirginleşmiş kullanım durumlarının (use case) uygulanmasıyla rafine hale gelmiştir. Kullanım durum (use case) tasarımını yaparak etkileşim diyagramlarını çizme tekniği kullanarak her bir nesnenin komple arayüz özelliklerini geliştireceğiz.

Burada nesnelerin sistem dahilinde birbirlerine gönderdikleri her bir uyarımı (stimulus) . Bu ayrıntılar, nesnenin davranışını çok detaylı gösterir şekilde -her nesnenin durum geçiş şeması (event trace diagram) yoluyla- ileri derecede saflaşmasına yol açacaktır. Böylece nesneler bir ya da birkaç nesne modülü kullanılarak yapılandırılıp uygulanabilecekler. Bu nesne modülleri, bir nesneye yönelik dil sınıfları ile irtibata geçecekler, fakat dil nesneye yönelik değilse bunlar modül kavramı ile mevcut dil yoluyla ilişkilendirilirler.

Yazılım elemanlarının kullanımı bu yapılanmada önemlidir.

V.4. Geliştirme Modeli

Uygulama seçilmiş olan dilde doğrudan gerçekleştirilir. Nesneye yönelik dil; OOSE'de kullanılan tüm önemli kavramları doğrudan uygulamaya uyarladığından tercih sebebidir. Eğer dil nesneye yönelik değilse, bazı değişiklikler yapılmalıdır. Yine de nesneye yönelik bir yapı nesneye yönelik olmayan şekilde uygulanmış sistemler için bile tamamen mümkündür.

Gerçek sistem gelişiminin yapılanmasını ele almak için, çeşitli seviyelerde kesin olmayan soyutlama mekanizmalara ihtiyaç duymaktayız.

Blok, kaynak kodunda sınıflar için kesin olmayan bir mekanizmadır. Karmaşıklık ele alındığında içerik olarak belki de en önemli olgu alt sistemlerdir.

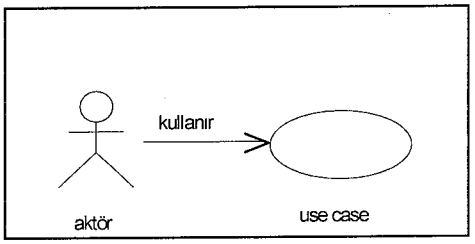
Bu alt sistemler, tasarım modelini parçalara ayırarak ve blok arayüzleri ya da sınıf arayüzleri anlamında belirgin arayüzlerin tanımlanmasında kullanılabilirler.

V.5. Test Modeli

Test modeli , sistemi test ettiğinde geliştirilir. Test bazı değişik düzeylerin çözünürlüğü şeklinde yapılır, aşağı düzey nesneleri test edilir. Kullanım durumu (use case) modeli entegrasyon testi yapılırken ana araç olarak kullanılır.

Gerçek zaman sistemlerini test etmek, ayrı bir zaman boyutundan dolayı çok zordur. Tüm uygulamalar zamana bağlı olduğundan hataların tekrar edilmesi; neredeyse tamamen imkansız bir haldedir. Işte bu yüzden kalite güvencesi ve doğrulama işlemlerine sahip olmak belki de neredeyse tüm gelişimden daha önemlidir. O zaman OOSE'de takip edilebilirlik belli başlı bir değeri oluşturmaktadır.

Objectory ve OOSE'deki nesneye yönelik kavramlar ve nesne modelleme diğer metodolojilere çok benzer. Jacobson'daki en büyük ayrım ise kullanım durumları-use case-dır. Bir use case'in tanımında bir aktör ve bir sistem arasındaki etkileşimin tanımı ve bir diyagram yer alır. Bu aktör son kullanıcı veya sistemdeki başka nesneler olabilir. Örneğin; sipariş giriş uygulamasındaki use case'de aktörün (kullanıcı) sisteme bir siparişi girişindeki her bir adımdaki etkileşimi tarifler ve istisnai tüm durumları da tanımlayabilir.



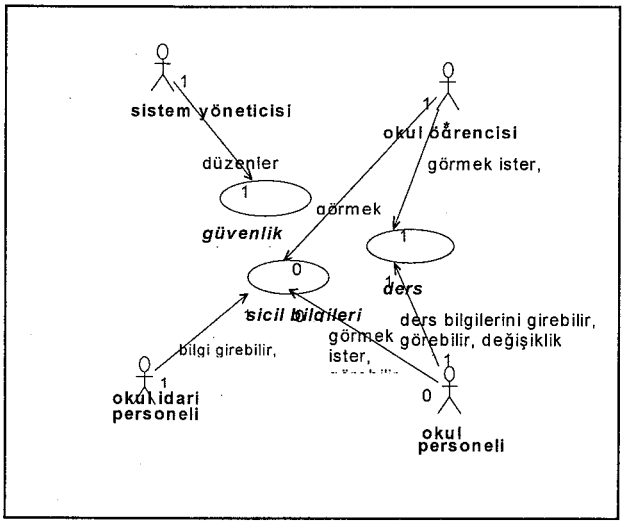
Şekil.1. Jacobsen Modeli

Jacobson'a göre bir use case, bir sistemin veya uygulamanın kullanım yolunu tarifler veya aktörlerin kara kutu halindeki uygulamaları nasıl kullandığını kapsayan üst düzey sınıf kullanım senaryosudur. Bir aktör sisteme bir arayüzdür. Bir kişi veya bir program olabilir. Jacobson, işlemler sırasının, bir kullanıcının sistem ile diyaloğunu tanımlayan davranışsal ilişkiler olarak da tanımlar.

VI. SONUÇ

Jacobson'ın OOSE metodunu kullanarak büyük ölçekli iş uygulamalarında, nesneye yönelik yazılım mühendisliği projeleri geliştirmek kolaylıkla mümkün olacaktır. Özellikle yöntemin tüm adımlarının gerçeklenmesi esnasında gereken dikkat ve ayrıntıların incelikli bir şekilde ortaya konması, yazılım mühendisliği uygulamalarında bir projenin yaşam döngüsündeki geri dönüş zorunluluğunun olabildiğince az kullanılması için sebep teşkil edecektir.

Aşağıda OOSE notasyonları kullanılarak, öğrenci işleri otomasyonu için yapılan çalışma sonucu elde edilen bir diyagram sunulmuştur.



Şekil.2. OOSE Notasyonu ile Öğrenci İşleri Otomasyonu

KAYNAKLAR

[1] Jacobson, I., **Object-Oriented Software Engineering**, Addison Wesley,1995.

[2] Jacobson, Magnus, Christerson, Patrik, Jonsson, Gunnar, Övergaard, **Object-Oriented Software Engineering A Use Case Driven Approach**, Ivar, Addison-Wesley, 1995.

[3] Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorenson, W., **Object-Oriented Modeling and Design**, Prentice Hall, 1991.

[4] http://www.rational.com.

[5] http://www.insight-tech.com/vourses/track/OOP.html