Öneri. C.3. S.14.
Haziran 2000. ss.1-7.

BOOCH METODOLOJİSİ İLE BAŞARILI NESNEYE YÖNELİK PROJELERİN ÖZELLİKLERİ

Zerrin AYVAZ REİS[1], Mithat UYSAL[2]

[1]*İstanbul Üniversitesi, Teknik Bilimler Meslek Yüksekokulu Bilgisayar Programcılığı, Öğr. Gör. Dr.*
[2]*Mimar Sinan Üniversitesi, Enformatik Bölümü, Prof. Dr*

***Abstract:** Booch defined iterative and incremental technic for analysis, design and implementation of object oriented systems. He has suggested 2 different ways to construct Micro and Macro applications. Aim of this paper; to determine steps of successful project's and to present it with micro application.*

I. GİRİŞ

Başarılı bir proje, teslim edilen kısımları müşterilerin beklentilerini tatmin eden ve muhtemelen daha fazlası olan; zaman ve ekonomik sınırlar içinde geliştirilmiş; değişim ve adaptasyona açık olan bir projedir. Bu anlamda irdelenen tüm başarılı nesne-tabanlı sistemlerde, genel olarak yokluğu hata olarak değerlendirilebilecek ve var olduğu gözlemlenen iki tane özellikten bahsedilebilir:

- Güçlü bir yapısal vizyonun varlığı
- İyi yönetilmiş bir döngüsel ve gelişen yaşam döngüsü uygulaması (iterative and incremental development life cycle)

II. YAPISAL BAKIŞ AÇISI

Yapısal bir mana ifade eden bir sistem kavramsal bütünlüğe sahiptir. Brooks' un da açıkça söylediği gibi. "Kavramsal bütünlük, sistem tasarımındaki en önemli noktadır" [1]. Nesne tabanlı yazılım sisteminin yapısı, farklı düzey ve bölümlerden organize edilmiş kendi sınıf ve nesne yapısını içerir. Bu arada bu sistem yapısı, son kullanıcıyı pek ilgilendirmez. Bununla beraber, Stroustrup' un işaret ettiği gibi "temiz bir iç yapıya" sahip olmak, anlaşılabilir; geliştirilebilir ve yeniden organize edilebilir; bakımı yapılabilir ve test edilebilir bir sistemi oluştururken çok önemlidir [2]. Genel soyutlama ve mekanizmaları keşfetmek, ancak sistem yapısını net olarak anlamakla mümkün olur. Bu genel noktaları anlamak, daha basit ve dolayısıyla daha küçük ve güvenilir sistemleri geliştirmeye olanak tanır.

Soyutlamaları sınıflandırmanın "doğru" bir yolu olmadığı gibi, belli bir sistemin yapısını oluşturmanın da "doğru" bir yolu yoktur. Herhangi bir uygulama alanında bir çözümün yapısı tasarlanırken, kesinlikle bazı basit yollar ve ara sıra da dahice yollar bulunabilir. Buna bağlı olarak, iyi bir yapıyı kötüsünden nasıl ayıracağız ?

Temel olarak, iyi yapılar nesne-tabanlı olma eğilimindedir. Bu, bütün nesne-tabanlı yapılar iyidir demek olmadığı gibi, yalnızca nesne-tabanlı yapılar iyidir demek de değildir. Nesne-tabanlı ayrıştırma prensiplerine dayalı bir uygulamanın, organize edilmiş karmaşık bir sistemin istenen özelliklerini içeren yapıların ortaya çıkmasını sağlayacağı gösterilebilir.

İyi yazılım yapıları genel olarak birçok ortak özelliğe sahiptir:

- İyi tanımlanmış soyutlama düzeylerinden oluşmuştur. Bütünlük taşıyan ve uyumlu (coherent) bir soyutlamayı ifade eden her düzey, iyi tanımlanmış ve kontrollü ara yüzlerle sağlanır ve aynı şekilde iyi tanımlanmış ve kontrollü özellikler içeren daha alt soyutlama seviyeleri üzerine inşa edilir.
- Her düzeyin arayüzü ve uygulaması arasında, kullanıcılar tarafından yapılan varsayımları sarsmadan düzeyin uygulamasını değiştirebilme olanağı sağlayan, net bir kavramların ayrıştırılması söz konusudur.
- Yapı basittir: Genel davranışlar, genel soyutlamalar ve genel mekanizmalarla sağlanır.

Stratejik ve taktik yapısal kararlar arasında bir ayrımdan bahsedilebilir. *Stratejik* bir karar, daha genel ve formal yapısal anlamlar taşır, ve bu yüzden de daha üst düzey yapıların organizasyonunu içerir. Hata tespit ve düzeltme mekanizmaları, kullanıcı arayüz paradigmaları, hafıza yönetim ve nesnelerin duruş politikaları, gerçek-zamanlı uygulamalardaki işlem senkronizasyon yaklaşımları, stratejik yapısal kararları ifade eder. Bu duruma ters olarak, *taktik* bir karar, yalnızca yerel yapısal anlamlar taşır ve genellikle soyutlama arayüz ve uygulamalarının detaylarını içerir.

Güçlü bir yapısal vizyonun temelinde, bu stratejik ve taktik kararlar arasındaki dengenin devamlılığını sağlamak yatar. İyi stratejik kararların yokluğu durumunda, sınıf çok iyi tasarlanmış olsa bile, asla tam

lamıyla doğru oturmayacaktır. Hassasiyetle alınmış ratejik kararlar, eğer bağımsız sınıfların tasarımına gereken hem verilmezse ciddi şekilde zarar görür. Her bir durumda, apısal vizyonu ciddiye almamak işe yaramayan bir azılımla baş başa bırakacaktır.

III. DÖNGÜSEL VE GELİŞEN YAŞAM DÖNGÜSÜ

Başarılı nesne - tabanlı yapıların oluşturulmasına anak tanıyan çalışmalar, hem döngüsel hem de gelişen zelliklere eğilimlidir. Böyle bir çalışma döngüseldir, öyle ki esne-tabanlı yapının sürekli ilerleyen, gelişen anlamda bir vizyonunu ifade eder. Bu revizyonun temelini, bir önceki ürümün sonuçlarını ve biriken deneyimi bir sonraki analiz tasarıma aktarmak oluşturur. Böyle bir çalışma sürekli elişen özellik gösterir, öyle ki analiz/tasarım/evrim öngüsünden her geçiş, stratejik ve taktik kararların yeniden özden geçirilip geliştirilmesini ve sonuçta son kullanıcının erçek (genellikle de baştan belirlenmemiş) ihtiyaçlarını arşılayan bir çözüm ortaya çıkmasını sağlar. Üstelik bu özüm basit, güvenilir ve adapte edilebilirdir.

Döngüsel ve gelişen bir geliştirme süreci, geleneksel aterfall modelinin anti-tezidir ve dolayısıyla ne kesin bir ukarıdan aşağı (top-down) ne de aşağıdan yukarı (bottom-p) tarzı ifade etmez.

Nesne-tabanlı geliştirmenin ne tam olarak üstten-alta de alttan-üste olmadığını tecrübeler göstermektedir. unun yerine, Druke' ın da önerdiği gibi, iyi yapılanmış mpleks sistemler en iyi şekilde "round-trip gestalt design"* ullanarak yaratılabilir. Bu tasarım tarzı, sistemin eliştirilmesinde döngüsel ve gelişen yapısının altını çizer. unu yaparken esas, sistemin farklı ve tutarlı mantıksal ve ziksel görünümlerini bir bütün olarak algılayıp yenilemek geliştirmektir. "Round-trip gestalt design" nesne-tabanlı sarım işleminin temelidir.

Sınırlı birkaç uygulama alanı için, bu alanda pek çok rklı uygulamayla beraber çözülecek problem çok iyi nımlanmış olabilir. Burada, geliştirme işlemini hemen emen bütünüyle kodlamak mümkündür: Böyle bir gulama alanı içindeki yeni bir sistemin tasarımcıları zaten emli soyutlamaların ne olduğunu anlıyordur; zaten hangi ekanizmaların kullanılması gerektiğini biliyordur ve nellikle böyle bir sistemden beklenen hareket aralığını liyordur. Yaratıcılık böyle bir süreçte hala önemlidir, fakat rada daha ziyade sistemin stratejik kararlarına reslenmiştir. Bu tür durumlarda, geliştirme riskinin büyük r kısmı elimine edilmiş olduğu için, çok yüksek üretim rformansı yakalamak mümkündür [3]. Çözülecek problem kkında ne kadar çok şey bilirsek, onu çözmek o kadar laylaşır.

Endüstri ağırlıklı yazılım problemlerinin büyük bir smı böyle değildir: çoğu, fonksiyonel ve performans rekliliklerin oluşturduğu özgün bir kümenin dengesini rektirir ve bu iş de geliştirme takımının maksimum rformans yaratıcı enerjisine ihtiyaç duyar.

Daha da derine inersek, yaratıcılık gerektiren her insani aktivite, deneyim, zeka ve bütün takım elemanlarının katılımına dayanan döngüsel ve gelişen çalışmayı gerektirir.

Doğal olarak hazır reçeteler sunmak imkansızdır.

IV. AKILCI BİR TASARIM İŞLEMİNE DOĞRU

Bununla beraber net olarak tanımlayıcı olmaya ihtiyacımız vardır. Aksi durumda herhangi bir organizasyon için asla olgunlaşmış, tekrar edilebilen bir geliştirme işleminden bahsedemeyiz. Bu yüzdendir ki iyi yönetilmiş bir gelişen ve döngüsel süreçten bahsediliyor: iyi yönetilmişten kasıt, işlemin kontrol edilebilir ve ölçülebilir olması, yaratıcılığa olanak sağlayacak yeterli bir özgürlük alanı bırakan bir sınırlama sistemine sahip olmasıdır.

Tanımlayıcı bir işleme sahip olmak, bir yazılım organizasyonunun olgunluğu anlamında bir temel oluşturur. Humprey' nin tanımına göre, işlem olgunluğunun beş farklı seviyesi vardır [4] :

- İlk (Initial)

Geliştirme işlemi "ad hoc" özellik taşır -yalnızca kendisi için vardır - ve genelde kaotiktir. Organizasyonlar basit proje kontrolleri oluşturarak gelişebilir.

- Tekrar Edilebilir (Repeatable)

Organizasyon, planları ve taahhütleri üzerinde makul bir kontrole sahiptir. Organizasyonlar iyi tanımlanmış bir işlemi kurumsallaştırarak gelişebilir.

- Tanımlanan (Defined)

Geliştirme işlemi, makul bir şekilde iyi tanımlanmış, anlaşılabilir, uygulanabilirdir; gelişimi öngörebilen ve takımı kalibre eden durağan bir kurum olarak hizmet eder. Organizasyonlar geliştirme pratiklerini araçsallaştırarak gelişebilir.

- Yönetilen (Managed)

Organizasyon işleminin nicel ölçülerine sahiptir. Organizasyonlar bu veriyi toplama maliyetlerini düşürerek ve işlem üzerine etkimesini sağlayan pratikler yaparak gelişebilir.

- Optimizasyon (Optimizing)

Organizasyon, tahmin edilebilir, zaman ve maliyet verimliliğini sağlayan bir yapıda, tutarlı olarak yüksek kaliteli ürünler üretebilen, ince ayarı yapılmış bir işlerlik noktasındadır.

Tasarım elemanları bir rehbere ihtiyaç duymaktalar, eğer ad hoc bir zeminde hareket etmekten çok, kendimizi işleme kaptırmayı denersek, daha gerçekçi bir işleme yakınlaşacağız. Bir organizasyon pek çok yazılım projesi üstlendiği zaman, standart bir prosedüre sahip olunmasında

---

Round-trip gestalt design :Nesneye yönelik tasarıma hberlik eden, gelişen ve döngüsel bir tasarım sistemidir.

yantajlar vardır.... Eğer ideal bir işlem üzerinde fikir birliği ursa, projenin gelişimini ölçmek çok daha kolaylaşacaktır"].

Geliştirme organizasyonlarımızı daha yüksek gunluk seviyelerine getirdiğimizde, daha kontrollü yönetim ygulama ihtiyacıyla beraber yaratıcılığı nasıl sağlayacağız ? u sorunun yanıtı geliştirme işleminin mikro ve makro emanlarını ayırt etmekte yatıyor gibi gözüküyor. Mikro lem, daha ziyade Boehm' in spiral geliştirme modeline dukça yakındır ve geliştirmeye döngüsel ve gelişen bir rçeve çizer ve bir zemin oluşturur [6].

Makro işlem, daha çok geleneksel waterfall odeline yakındır ve mikro işlem için kontrol altyapısı luşturur. Bu iki ayrı işlemi kaynaştırarak, tümüyle gerçekçi ir geliştirme işlemine varabiliriz böylelikle tanımlanan azılım işleminin olgunluk seviyesine bir altyapımız olur.

V. MİKRO GELİŞTİRME İŞLEMİ

GÖZDEN GEÇİRME

Nesne-tabanlı geliştirmenin mikro işlemleri, makro mler tarafından sürekli üretilen senaryolar ve yapısal nler tarafından belirlenir. Biraz daha açarsak, mikro lemler, küçük bir geliştirme takımı veya bireysel bir lemanın günlük aktivitelerini ifade eder.

Mikro işlemler yazılım mühendisi ve yazılım himarına eşit mesafededir. Mühendisin bakış açısından hikro işlemler, günlük üretim ve yapının adaptasyonunda ok sayıda karara rehberlik eder. Mimarın bakış açısından e, yapının evrimine ve alternatif tasarımların keşfine bir atı oluşturur.

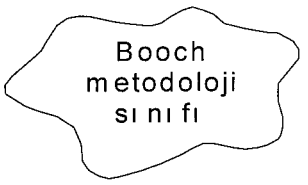
Mikro işlemlerde, geleneksel analiz ve tasarım fazları açınılmaz olarak bulanıklaşır, ve işlemler oportünist gerekenler yerine yapılabilenlerin ön plana çıktığı) bir ontrol altında yürür. Stroustrup' un gözlemlediği gibi, Tasarım ve programlamada zeka, deneyim ve iyi koku alma eteneğinin yerini alabilecek "hazır bir reçete, tarif ookbook)" metotlar yoktur.... Bir yazılım projesinin asarım, programlama ve test gibi farklı aşamalarını net larak birbirinden ayırmanın imkanı yoktur" [2].

Mikro işlemlerin geliştirilmesi aşağıdaki eğilimdedir:

- Verilen soyutlama düzeyinde sınıf ve nesnelerin anımlanması.
- Bu sınıf ve nesnelerin semantiğinin tanımlanması.
- Bu sınıf ve nesneler arasındaki ilişkilerin anımlanması.
- Arayüzün belirlenmesi ve sonra bu sınıfların ve esnelerin uygulanması .

VI. SINIF VE NESNELERİN TANIMLANMASI

Amaç Sınıf ve nesnelerin tanımlanmasının amacı, eldeki problemin sınırlarını belirlemektir. Ek olarak bu aktivite, geliştirilen sistemin nesne-tabanlı ayrıştırılmasındaki ilk aşamadır.



Şekil.1. Booch Metodolojisi Notasyonu ile Tanımlanmış Bir Sınıf Örneği

Analizin bir parçası olarak, problem kümesinin sözlüğünü oluşturan soyutlamaları keşfetmek için bu adımı uygularız. Böylelikle ilgi odağımızda nelerin olduğuna ve nelerin olmadığına karar vererek problemimizin sınırlarını belirlemeye başlarız. Tasarımın bir parçası olarak, çözümün parçalarını oluşturan yeni soyutlamaları keşfetmek için bu adım uygulanır. Uygulama sürerken, üst düzey soyutlamaları şekillendiren daha alt düzey soyutlamaları keşfetmek ve sistemin yapısını basitleştirmeye olanak sağlayan, varolan soyutlamaların ortak özelliklerini anlamak ve ortaya koymak için bu adımı uyguluyoruz.

Ürünler Bu adımın temel ürünü, geliştirme süresince güncellenen bir (data dictionary) veri sözlüğüdür. Başlangıçta, bütün önemli sınıf ve nesneleri içeren ve anlamlarını çağrıştıracak şekilde isimlendirilmiş bir "şeyler listesi" (list of things) oluşturmak yeterli olabilir [7]. Geliştirme sürerken özellikle de sözlük genişlerken, bir depolama, dosyalama sistemi oluşturmak gerekli hale gelecektir. Bu iş, muhtemelen basit bir ad hoc veritabanı oluşturarak veya metodu direk destekleyecek daha formal bir araç oluşturarak yapılabilir. Daha da ileri gidersek veri sözlüğü, geliştirme işleminin bütün diğer ürünleriyle ilişkili, çeşitli diyagramlar ve nesne-tabanlı geliştirme notasyonunun özelliklerini içeren bir indeks olarak ele alınabilir.

Böylelikle veri sözlüğü, sistemle ilişkili soyutlamaların yer aldığı merkezi bir depo gibi hizmet eder. Başlangıçta sözlüğün önünü açık tutmakta sakınca olmamakla birlikte, geliştirme sürecinde zamanla bazı şeyler sınıf olmaya, nesne olmaya ve diğer soyutlamaların bir parçası veya eşleniği olmaya başlar. Zaman ilerledikçe, bu sözlük yeni soyutlamalar eklenerek, alakasız olanlar çıkarılarak ve benzer olanlar birleştirilerek yeniden yapılandırılacaktır.

Bu aktivitenin bir parçası olarak bir veri sözlüğü oluşturulmasının üç önemli faydası vardır. Bunlardan ilki, bütün proje boyunca kullanılabilecek genel ve tutarlı bir sözlük oluşturmaya yardımı olur. İkincisi, projenin bütün birimlerini gelişigüzel şekillerde sorgulamak ve görebilmek için etkili bir araçtır. Bu özellik, geliştirme takımına yeni üyeler katıldığında özellikle faydalıdır. Zira yeni üyelerin geliştirilmeye devam edilen çözüme çok çabuk adapte olmaları gerekir. Üçüncüsü, bir veri sözlüğü,

mimarların projeye genel olarak bakabilmelerini sağlar. Böylece genel ve ortak özellikleri keşfetmek mümkün hale gelir.

Aktiviteler Sınıf ve objelerin tanımlanması iki aktivite içerir: buluş ve keşif.

Her geliştirici bu aktivitelerde yetenekli olmak zorunda değildir. Genellikle alan uzmanlarıyla beraber çalışan analistler, soyutlamaları bulmakta; problem alanına bakıp, anlamlı sınıf ve nesneleri tespit etmekte çok iyi olmalıdır. Benzer şekilde mimarlar ve daha üst düzey geliştiriciler, yeni sınıf ve nesneleri çözüm alanından çıkartmada yetenekli olmalıdır.

Olayların tipik sırası aşağıdaki gibidir:

- Sınıf ve nesneleri oluşturmak için, nesne-tabanlı analize klasik yaklaşım uygulanır. Hayat akışının ilk aşamalarında, elle dokunulur, net şeyler ve onların fonksiyonları iyi başlangıç noktalarıdır. Daha sonraki aşamalarda gelişen olaylar ilk ve ikincil soyutlamalara olanak tanıyacaktır. Her olay için, olayı tespit etmekten ve/veya ona tepki göstermekten sorumlu nesnelerimiz olmalıdır.

- Sistem fonksiyon noktalarına direk bağlı soyutlamaları tanımlamak üzere davranış analizi tekniklerin uygulanır. Sistemin fonksiyon noktaları makro işlemlerden çıkar ve farklı, net olarak görülebilir ve test edilebilir davranışlar sergiler. Olaylarla çalışırken her bir davranış için, onu başlatan ve parçası olan varlıklara (entity) sahip olmalıyız.

- Makro işlemler sırasında üretilmiş ilgili senaryolara durum analizi teknikleri (use-case analysis) uygulanır. Yaşam akışının ilk aşamalarında, sistemin genel davranışını tanımlayan ilk senaryolar takip edilir. Daha sonraki aşamalarda sistemin biraz daha ayrıntılandırılması arzu edilen davranışın kapalı kalmış köşelerini açıklayabilmek için, daha detaylı senaryolar ve çevre birimleri senaryolarını inceleriz.

Bütün bu yaklaşımlar içinde, CRC(Collaboratio Responsibility Cards) kartlarının kullanımı, problemin çözümünü daha geniş bir vizyonla yapabilmek için beyin fırtınası işlemine etkili bir katalizör olacak ve ilaveten takımın iletişimini sağlama konusunda faydalı olacaktır.

İlk aşamalarda tanımlanan sınıf ve nesneler kaçınılmaz olarak yanlış olacaktır fakat bu ille de kötü bir olay değildir. Başlangıçta belirlenen elle tutulur şeylerin ve ilişkili rollerin çoğu bütün uygulama sürecinde önemli olacaktır çünkü, bizim probleme ilişkin kavramsal modelimiz için temel niteliğindedir. Problem hakkında daha çok bilgimiz oldukça, sorumlulukları yeniden belirleyerek; benzer soyutlamaları birleştirerek; ve nadiren de daha geniş soyutlamaları gruplara bölerek muhtemelen belirli soyutlamaların sınırlarını değiştireceğiz. Böylece çözüme katkısı olan bazı mekanizmalar oluşturmuş olacağız.

Mantıksal olarak durağan bir veri sözlüğüne sahip olduğumuzda, bu aşamayı başarıyla tamamlarız. Mikro işlemlerin döngüsel ve gelişen doğası gereği, geliştirme sürecinin son aşamalarına kadar bu sözlüğü tamamlamayı veya dondurmayı bekleyemeyiz. Daha ziyade soyutlama kümesinin bir örneğini içeren, tutarlı bir şekilde adlandırılmış ve sorumlulukların hassasiyetle ayrıldığı bir sözlüğe sahip olmak yeterli olacaktır.

Dolayısıyla iyiliğin bir ölçüsü, mikro işlemlerdeki her döngüde sözlüğün geniş çaplı değişmemesidir. Hızla değişen bir sözlük ya geliştirme takımının yeterince odaklanamadığının ya da yapıdaki bir çöküntünün işareti olacaktır. Gelişim sürdükçe, soyutlamalardaki yerel değişimleri takip ederek yapının daha alt seviyedeki durağanlığı izlenebilir.

VII. SINIF VE NESNELERİN SEMANTİĞİNİN TANIMLANMASI

Amaç; Sınıf ve nesnelerin semantiğinin tanımlanmasının amacı, bir önceki aşamada ortaya konan soyutlamaların özelliklerini ve davranış şeklini belirlemektir. Zekice ve ölçülebilir bir sorumluluk dağılımı yapmak suretiyle burada soyutlamalarımızı gözden geçiririz.

Analiz anlamında bu aşamada, farklı sistem davranışları için sorumluluklar belirleriz. Tasarım anlamında bu aşamada, çözümümüzün parçaları arasında net bir kavram ayrıştırması yaparız. Uygulama sürerken de sorumluluk ve rollerin serbest şekilde yapılan tanımlarından yola çıkıp, her soyutlama için komple bir protokol tanımlamak maksadıyla her operasyon için net ve kesin bir tabloya ulaşırız.

Ürünler; Bu aşamadan gelen pek çok ürün vardır. İlki, başlangıçta her soyutlamaya sorumluluk iliştirdiğimiz veri sözlüğünün yenilenmesidir. Gelişim devam ederken her soyutlama için spesifikasyonlar yaratabiliriz Bu spesifikasyonlar her sınıfın protokolünü oluşturan adlandırılmış operasyonlardır. Olabildiğince erken, bütün sınıflara belirli bir uygulama dili kullanarak arayüz yazmak suretiyle bu kararları formal hale getirmek gerekir. C++ için bu *.h* dosyalarını ifade eder; Ada için teslim edilecek paket spesifikasyonlarını; CLOS için her sınıf için yazılmış jenerik fonksiyonları; Smalltalk için her sınıfın metotlarını tanımlamayı fakat uygulamamayı ifade eder. Eğer problemimizin veri tabanı elemanlarıyla ilgileniyorsak, özellikle de nesne-tabanlı bir veri tabanı kullanıyorsak , şemamızın ilk halini üretebiliriz.

Doğası gereği daha çok taktiksel olan bu ürünlere ek olarak, makro işlemlerden türetilen senaryoların anlamlarını açığa çıkaracak nesne diyagramlarını ve ilişki diyagramlarını üretebiliriz. Bu diyagramlar bize her senaryonun işlem adımlarını daha formal olarak anlamamızı ve gerekiyorsa yeniden düzenleme yapabilmemize hizmet ederler. Böylece birbirleri ile ilişkili nesneler arasındaki sorumlulukların ayrıştırılmasını yansıtırlar. Bu noktada, belli bazı soyutlamalar için sonlu durum makinası (finite state machines) üretimine başlanabilir.

Daha önceki aşamada olduğu gibi, her soyutlamanın sorumluluklarını takip edebilmek için *ad hoc* bir veri tabanı veya metoda özel bir araç kullanabiliriz. Böylelikle takım tutarlı bir ifade diline doğru evrilebilir. Bir kez formal sınıf arayüzleri ürettiğimizde, tasarım kararlarımızı

test etmek ve güçlendirmek için programlama araçlarını kullanmaya başlayabiliriz.

Bu aşamanın daha formal ürünlerinin temel faydası, geliştiricileri soyutlama protokollerinin pragmatik boyutlarını dikkate almaya zorlamasıdır.

Net semantikleri belirleme konusundaki sıkıntılar, soyutlamaların çöktüğünün bir işaretidir.

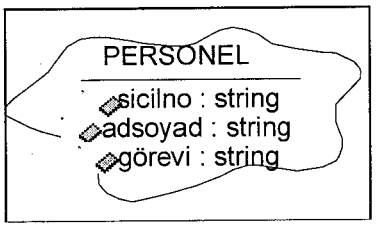
Aktiviteler Bu aşamayla ilişkili üç aktivite vardır: hikayeleştirme, izole sınıf tasarımı ve örnek toplama.

Makro işlemlerden oluşturulmuş temel ve çevresel senaryolar hikayeleştirmenin temel sürücüleridir. Bu aktivite semantiklerin top-down tanımlanmasını ifade eder; sistem fonksiyon noktalarını içerir ve stratejik olguları yerleştirir. Olayların tipik gelişimi aşağıdaki gibi olabilir:

- Tek bir fonksiyonla ilişkili tek bir senaryo veya bir senaryo kümesi seçilir; bir önceki aşamadan senaryoya ilişkin soyutlamalar tanımlanır.

- İstenen davranış alınıncaya kadar her soyutlamaya sorumluluklar verilip, senaryo takip edilir. Gerektiğinde belirli sorumlulukları yerine getirebilmek için yapısal elemanları ifade eden özellikler (attributes) belirlenir.

- Hikayeleştirme sürerken, davranış dağılımının makul bir dengeye geldiği noktaya kadar sorumluluklar yeniden yapılandırılır. Geniş sorumlulukları daha küçük parçalara bölmek genel bir eğilimdir; nadir olmamakla beraber daha az sıklıkla deneysel sorumluluklar da daha geniş davranışlara monte edilir.



Şekil.2. Attribute'leriyle Tanımlanmış Bir Sınıf Örneği

Yeterli olmamakla beraber, senaryoların hikayelerini oluşturma sürecinde CRC kartlarını kullanabiliriz. Daha da iyisi geliştirme takımı, nesne diyagramları veya etkileşim diyagramları çıkarabilir. Analiz sırasında, bu hikayeleştirme işi sınırlı olmamakla beraber tipik olarak bir analist, alan uzmanı, mimar ve kalite elemanından oluşan bir takım tarafından yapılır.

Tasarım ve sonra da uygulama sırasında ise, stratejik kararları düzenlemek maksadıyla mimarlar ve deneyimli geliştiriciler tarafından; ve taktik kararları düzenlemek maksadıyla bireysel geliştiriciler tarfından yapılır. Hikayeleştirme işinde daha fazla takım üyesine sahip olmak, deneyimsiz geliştiricilerin eğitimi ve yapısal vizyonun aktarılabilmesi için hayli etkili bir yoldur.

Geliştirme sürecinin ilk aşamalarında, soyutlamaların sorumluluklarını serbest şekilde yazmak suretiyle, sınıf ve nesnelerin semantiklerini belirleyebiliriz. Genellikle bir kaç kelime veya bir cümle yeterlidir; daha fazlası gerekiyorsa verilen sorumluluk bir hayli karmaşık demektir ve daha küçük parçalara bölünmesi uygun olacaktır.

Geliştirme sürecinin ileri safhalarında, bireysel soyutlamaların protokolü belirlenmeye başladığı için, bütün boyutlarını umursamadan belirli operasyonları isimlendirebiliriz. Pratik olabildiği ölçüde de bütün boyutlarını da ele alabiliriz.

Bu yöntemle, her belirli sorumluluk birbirini tamamlayan operasyonlardan oluşan bir kümeyle doldurulur ve her operasyon bir şekilde soyutlamanın sorumluluklarına katkıda bulunur. Bu noktada, belirli sınıflar için "finite state machines" devreye girebilir, protokollerinin dinamik manalarını keşfetmek üzere, özellikle event-driven veya state-ordered davranış içeren sorumluluklara sahip sınıflar için bu geçerlidir.

Bu aşamada yapıya değil davranışa odaklanmak önemlidir. Özellikler, yapısal elemanları temsil eder ve dolayısıyla özellikle analizin erken safhalarında bir tehlike vardır: belirli özelliklerin olmasını gerektiren uygulama kararlarını çok erken almak tehlikesi. Bu noktada senaryonun kavramsal modelini inşa etmek önemli olduğu için, özellikler tanımlanmalıdır.

İzole sınıf tasarımı semantiklerin bottom-up bir tanımlanmasını ifade eder. Burada dikkatimizi tek bir soyutlamaya yoğunlaştırırız ve sınıf tasarımına geliştirilen yaklaşımlarımızı uygulayarak soyutlamamızı tamamlayacak operasyonları inceleriz. Bu aktivite doğasında daha çok taktikseldir çünkü yapısal tasarımla değil iyi sınıf tasarımıyla ilgilidir. Olayların tipik gelişimi aşağıdaki gibi olabilir:

- Bir soyutlama seçilir ve rol ve sorumlulukları belirlenir.

- Bu sorumlulukları sağlayan yeterli bir operasyon kümesi türetilir. Mümkünse, birbirine rol ve sorumluluk anlamında kavramsal boyutta yakın operasyonları tekrar kullanmak denenir.

- Her bir operasyon incelenir ve ilkelliği, basitliği (primitive) garanti altına alınır. Eğer öyle değilse, izole edip daha basit operasyonlara bölünmelidir. Bileşik operasyonlar sınıfın içinde veya bir sınıf özelliğine (utility) dönüştürülür. Mümkün olduğunca, minimal bir basit operasyonlar kümesi incelenmelidir.

- Geliştirmenin sonraki aşamalarında, üretim, kopyalama ve yok etme ihtiyaçları irdelenir [2]. Aksini gerektiren bir durum olmadıkça, bireysel sınıfların kendine

has hareketinden çok, genel bir stratejik politikaya sahip olunması daha iyidir.

- Bütünlük ihtiyacı irdelenir: orta vadedeki kullanıcılar için gerekmeyen fakat varlığı soyutlamayı tamamlayacak ve böylece gelecekteki kullanıcılara hitap edecek başka ilkel operasyonlar eklenir. Mükemmel bir bütünlüğün olamayacağı gerçeğinden hareketle, karmaşıklaktırmak yerine basitleştirme yoluna gidilmelidir.

VIII. SONUÇ

Booch metodolojisi, bir sistemin analizi, tasarımı ve geliştirilmesi için nesneye yönelik yinelemeli bir teknik tanımlamaktadır. Mikro ve Makro uygulamalar için ayrı önerileri vardır.

Tekrar tanımlamak gerekirse; Makro uygulama geliştirmenin adımları;

- Kavramları anlamak-Gereksinimleri belirlemek, yerine koymak,
- Analiz-Geliştirme Metodu,
- Tasarım- Mimariyi yaratmak,
- Değer kazandırmak- Gerçekleştirmek,
- Bakım,

aşamalarını içerirken, mikro uygulama geliştirme işlemi ise aşağıdaki adımları içerir.

- Nesneleri ve sınıfları tanımlamak,
- Nesnelerin ve sınıfların mantığını tanımlamak,
- Nesneler ve sınıflar arası ilişkileri tanımlamak,
- Sınıfları ve nesneleri geliştirmek.

Bu adımların titizlikle uygulanması sonucu tam bir yazılım mühendisliği ürünü elde edilir.

Bu çalışmada BOOCH metodolojisi kullanılarak gerçekleştirilen bir sistem geliştirme uygulamasının, çözümün uzun olması sebebiyle tüm aşamalarının modellenmesini değil de sadece kod üretme aşamasına hazırlık olarak ortaya koyulan veri tabanının modellenmesine kaynak oluşturacak olan "nesne modeli" diyagramı aşağıda gösterilmiştir. (Diğer sayfada)

KAYNAKLAR

[1] Brocks,F., **The Mythical Man-Month**, Addison-Wesl 1975, s.42.

[2] Stroustrup, B., **The C+ Programming Langua** Addison-Wesley, 1991, s.362, 363.

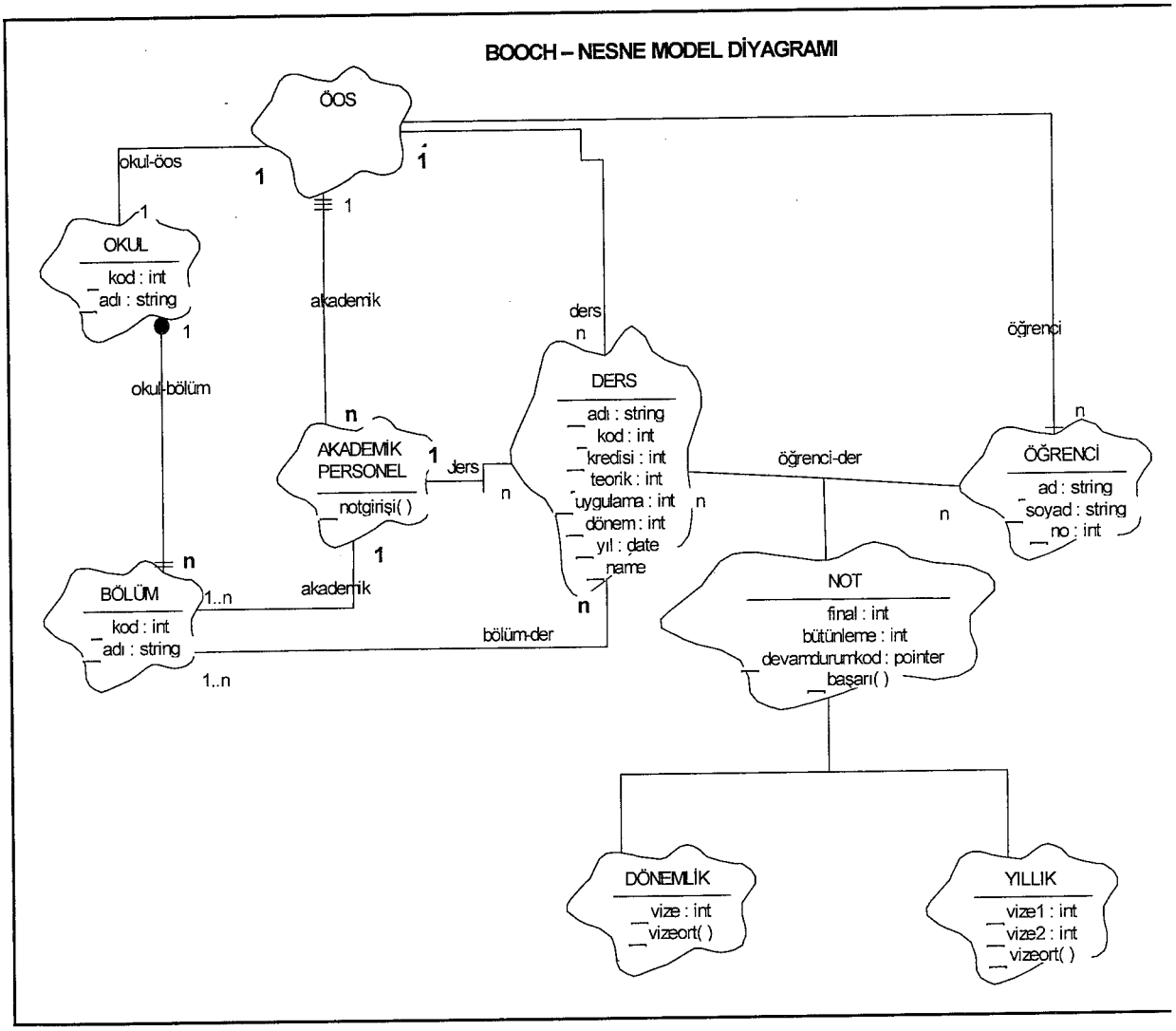
[3] Jones, C., "Reuseability in Programming: A Survey of State of The Ar", **IEEE Transaction on Softwa Engineering**, Vol.SE-10(5), September-1984.

[4] Humprey, W., **Managing The Software Proce** Addison-Wesley, 1989, s.5.

[5] Parnas, D., Clements, P., "A Rational Design Process; H and Why to Fake It", **IEEE Transactions on Softwa Engineering**, Vol. SE-12(2), 1986.

[6] Boehm, B., "A Spiral Model of Software Development a Enhancement", **Software Engineering Notes**, Vol.11(August 1986, s.22.

[7] Booch, G., **Object-Oriented Analysis and Design**, 19 s.236



Sınıflar arası ilişkilerinde gösterildiği nesne diyagram