PAMUKKALE ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ YIL : 1995
PAMUKKALE UNIVERSITY ENGINEERING COLLEGE CİLT : 1
MÜHENDİSLİK BİLİMLERİ DERGİSİ SAYI : 2-3
JOURNAL OF ENGINEERING SCIENCES SAYFA : 73-80

YAZILIM TEKRAR KULLANIMI VE NESNEYE YÖNELİK YAKLAŞIM

Halil ŞENGONCA, Yasemin TOPALOĞLU, Oğuz DİKENELLİ
Ege Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Bornova-İzmir

ÖZET

Yazılımın tekrar kullanımı, bir yazılım sisteminin geliştirilmesinde kullanılan bilginin ve ürünlerin, başka bir yazılım sistemi için kullanılmasıdır. Yazılımın tekrar kullanımı için kaynak kod bileşenlerinden oluşan bir kütüphane ve karmaşık sistemlerin bu yapı bloklarından oluşturulması fikri, ilk olarak 1968'te Dough McIlroy tarafından önerilmiştir. Günümüzde programlama dilleri, kullanılan teknikler ve araçlar çok değişmesine rağmen yazılımın tekrar kullanımı yazılım geliştirme maliyetlerini azaltmak ve kaliteyi yükseltmek için hala bir çözüm olarak görülmektedir. Nesneye yönelik programlama, kalıtım, koruma gibi özellikleriyle, kaynak kod bileşenlerinin oluşturulmasını ve kullanımını desteklemektedir. Yine de yazılım tekrar kullanımını bir kuruluş düzeyinde gerçekleştirmek için planlı ve koordineli bir programa gereksinim vardır. Bu makalede, yazılım tekrar kullanımı tanımlanmış ve nesneye yönelik yaklaşım ile yazılım tekrar kullanımını amaçlayan bir yazılım geliştirme süreci tanıtılmıştır. Ayrıca bir kuruluşun yazılım tekrar kullanımını kurumsallaştırma süreci için bir strateji önerilmiştir.

Anahtar Kelimeler: Yazılım tekrar kullanımı, Nesneye yönelik programlama, Kurumsallaştırma süreci.

SOFTWARE REUSE AND OBJECT ORIENTED APPROACH

ABSTRACT

Software reuse is defined as the reuse of information and products of a previously developed software system in the development process of a new software system. The purpose of software reuse is to reduce the development cost and the time, while increasing the software quality. Constructing a library of software components and using them in developing complex software systems was first proposed by Dough McIlroy in 1968. Since then programming languages, tools and techniques have changed a lot, but component oriented software reuse is still accepted as an important solution to solve the software crisis. Object oriented approach encourages component reuse, by inheritance and encapsulation. But in order to accomplish a succesful reuse program in an organization, a well-planned and coordinated managenment is essential. In this paper, an object oriented software development process is introduced and in order to instituionalize software reuse a strategy is proposed.

Key Words: Software reuse, Object oriented programming, Institutionalizing process.

1. GİRİŞ

Tekrar kullanım genel bir kavramdır ve çeşitli bilimsel alanlarda olduğu gibi, günlük hayatımızda da sık sık yer alır. Örneğin, aynı tip tuğlalarla, inşaatcılar farklı tipte yapılar oluştururlar, mimarlar aynı tasarım ilkelerini çeşitli mimari tasarımlara uygularlar, matematikçiler, aynı formülleri çeşitli problemleri çözmek için kullanırlar. Tekrar kullanım kavramı için günlük yaşantımızdaki ve çeşitli bilimsel alanlardaki deneyimlerimize dayanan bir

tanım yapacak olursak, tekrar kullanım, aynı nesnenin veya aynı bilginin farklı uygulamalara birden çok uygulanmasıdır, tanımını yapabiliriz.

Benzer şekilde, yazılımın tekrar kullanımı, bir sistemin geliştirilmesinde kullanılan bilginin ve ürünlerin, başka bir sistem için kullanılmasıdır (Krueger,1992). Böylece yeni bir sistem geliştirilirken veya bakımı yapılırken, gerekli olan çaba azaltılmış olmaktadır. Tekrar kullanılabilecek ürünler, yazılım geliştirilmesindeki tüm ürünleri kapsar. Örneğin; yazılım bileşenleri, belgeler gibi. Tekrar kullanılabilecek bilgi ise sistemin geliştirilmesi ile ilgili bilgileri, örneğin analiz ve tasarımdaki bilgileri içerir. Yazılım açısından tekrar kullanım, sadece kaynak kod parçalarına değil, yazılım geliştirme sürecinde oluşturulan tüm ara ürünlere uygulanabilir. Örneğin; gereksinimlerle ilgili bilgiler, sistem özellikleri, tasarım yapıları, sınamalar gibi (Barnes,1991).

Güvenilir ve büyük ölçekli yazılım sistemlerinin uygun bir maliyetle elde edilmesindeki problemler olarak tanımlanan yazılım krizi, ilk olarak 1968'de dile getirilmiştir. Dough McIlroy, 1968'te, NATO'nun yazılım mühendisliği konferansında sunduğu bildiride yazılımın tekrar kullanımının, yazılım krizine bir çözüm olabileceğini öne sürmüştür ve tekrar kullanılabilir bileşenlerden oluşan bir kütüphane önermiştir. McIlroy, bileşen kütüphanelerinin sayısal hesaplama, G/Ç dönüşüm işlemleri, metin işleme gibi konularda etkin olarak kullanılabileceğini belirtmiştir (Krueger, 1992).

1968'den bu yana, 25 yılı aşkın bir süre geçmesine rağmen, yazılımın tekrar kullanımı, yazılım mühendisliğinin gelişimi için hala güçlü bir yol olarak görülmektedir (Krueger, 1992). 1968'den beri yazılım mühendisliği anlayışı, programlama dilleri, araçlar, yöntemler çok değişmiştir, ama yazılım geliştirme maliyetini tekrar kullanım ile azaltmak fikri hala yaygın olarak kabul edilmektedir. Endüstri, tekrar kullanımı, zaman ve bütçe kısıtlamalarına karşı bir kurtarıcı olarak görmektedir. Ancak, yazılımın tekrar kullanımı, yazılım oluşturulmasında standart hale gelememiştir. Bunun teknik nedenlerinin yanısıra, yönetim, kültür ve hukuktan kaynaklanan nedenleri vardır ve teknik olmayan problemlerin çözümü, teknik problemlerin çözümünden zor olabilmektedir (Diaz, 1993).

McIlroy'un standart kaynak kod bileşenlerden oluşan bir kütüphane oluşturulması ve karmaşık sistemlerin bu küçük yapı bloklarından oluşturulması fikri 1970'lerden sonra çok geliştirilmiş ve çeşitli şirketlerdeki uygulamalara temel oluşturmuştur. Sonuçlar başarılıdır. Çeşitli araştırmalar ve raporlar, tekrar kullanımın uygulanması ile yazılım geliştirmenin ve bakımın maliyetinin önemli ölçüde azaldığını göstermiştir. Isveç'teki NobelTech firması, bir dizi sistemi, ortak yazılım altsistemlerinden oluşturmuş ve %70 tekrar kullanım ile üretkenliğini ikiye katlamıştır. Benzer başarılı sonuçlar Japonya'da Toshiba firmasında elde edilmiştir (Davis, 1994). Üretkenlik artışı, maliyetin azalması ve kalitenin artmasının sonucudur. Bunun nedenleri şöyle açıklanabilir: Geleneksel yazılım geliştirme yaşam döngüsü, analiz, tasarım, kodlama, sınama ve bakım aşamalarından oluşur. Bu yaşam döngüsü içinde en fazla maliyete sahip aşamalar, sınama ve bakım aşamalarıdır. Tekrar kullanım ile kullanılan yazılım bileşenleri, sınama aşamasını ve bakım sürecini kısaltacakları için, yazılım geliştirme maliyetini düşürürler. Yazılımın tekrar kullanımı, yazılım kalitesini de arttırır. Bir çok matematik formülü ve fizik kanunu kusursuzdur. Binlerce kez kullanıldıkları için, çok sayıda değerlendirmenin ve düzeltmenin ürünüdürler. Benzer şekilde, bir yazılım bileşeninin sık olarak tekrar kullanımı, çok sayıda değerlendirme ve gerekiyorsa yenilenmeye neden olacağı için, yüksek kaliteli yazılım bileşenleri oluşturacaktır. Bunların ışığında, yazılımın tekrar kullanımının, yazılım krizini aşmada önemli bir etken olduğu söylenebilir (Jacobson, 1993).

2. NESNEYE YÖNELIK YAKLAŞIM VE TEKRAR KULLANIM

2.1 Nesneye Yönelik Yaklaşımın Özellikleri

Nesneye yönelik yaklaşım, yazılım geliştirme işlemini basitleştirmeye çalışır ve program yapısında veriyi merkezîleştirir (Booch,1994). Nesneye yönelik yaklaşımda veriler ve üzerinde çalışan fonksiyonlar birbirine bağlanır. Prosedürel yaklaşımda, bir programın temel birimleri fonksiyonlardır. Nesneye yönelik yaklaşımda ise temel birim nesne (object)dir. Nesneler, hem veri hem de o verilerin üzerinde çalışan fonksiyonları içerirler. Bir nesnede bulunan verilere, sadece o nesnede yer alan fonksiyonlar erişebilir. Diğer nesnelerdeki fonksiyonlar ulaşamazlar. Başka nesnelerdeki fonksiyonların, verilere ulaşabilmesi için, verinin bulunduğu nesnenin parçası olan fonksiyonları kullanmaları gerekir. Bir nesnedeki fonksiyonlara yöntem (metod) adı verilir. Yöntemler,

nesnelerin verilerini işlerler. Verilerin ve o veriler üzerinde çalışan fonksiyonların, birarada bulunması koruma (encapsulation) özelliğidir. Nesneler, sınıfların (class) elemanlarıdır. Bir sınıf, ortak bir yapı ve ortak bir davranışı paylaşan bir çok nesneden oluşur. Nesneler somut, sınıflar ise soyut birimlerdir.

Nesneye yönelik yaklaşımın bir diğer özelliği kalıtım (inheritance)dır. Kalıtım özelliği, bir nesneden yeni bir nesne oluşturulmasını sağlar. Bu özellik kullanılarak, bir nesnenin tüm özelliklerini kullanan ve bazı başka yeni özellikler içeren nesneler tanımlanabilir. Bunların ışığında, nesneye yönelik yaklaşım açısından tekrar kullanımı değerlendirirsek:

- Gerçek dünyayı modelleme, nesneye yönelik yaklaşım tarafından desteklenir. Sınıflar, bir problem uzayındaki en sürekli elemanlardır. Sınıflar soyut birimler olduğundan bir problem uzayına ilişkin sınıflar, benzer problem uzaylarında da var olabileceklerdir. Bu nedenle, sınıflar etrafında yapılanmış programlar, değişen gereksinimleri karşılama açısından daha güçlüdürler. Sınıfların var olması, aynı alandaki programlarda bileşen tekrar kullanımını sağlar.

- Koruma özelliği, kullanıcıların bir nesneyi, nesnenin sistemin diğer parçalarıyla olan ilişkileri, ve gerçekleştirimindeki ayrıntılarla ilgilenmeden kullanmasını sağlar. Kullanıcılar, nesnenin arayüzü ile iletişim kurar, uygulamanın diğer bölümlerinden korunurlar. Bu özellik, gerçek dünyada bulunan soyut sınıfların, uygulama ortamında da özelliklerini aynen korumasını sağlar.

- Kalıtım özelliği, veri tiplerinin sürekli yenilenmesini destekler. Böylece, ilk geliştirici tarafından düşünülmeyen özellikler ortaya çıkabilir. Kalıtım özelliği ile bir sınıftan yeni sınıflar türetilebilir. Böylece tekrar kullanımın gerçekleşmesine olanak sağlanır. Bileşenler, gereksinimlerde küçük değişiklikler olduğu zaman, bu yeni gereksinimleri karşılayabilecek duruma getirilirse tekrar kullanım olasılığı artar. Geleneksel altyordam kütüphaneleri ile ilgili en önemli problem, parametreler ile sağlanan sınırlı destektir. Parametreler, sadece ilk geliştiricinin öngördüklerine izin verir. Daha çok esneklik, daha çok parametre ile başarılabilir. Genelliği çok parametre ile başarmaya çalışan altyordam yaklaşımının tersine, kalıtım özelliği, genelliği basitleştirme ile destekler (Coleman, 1994).

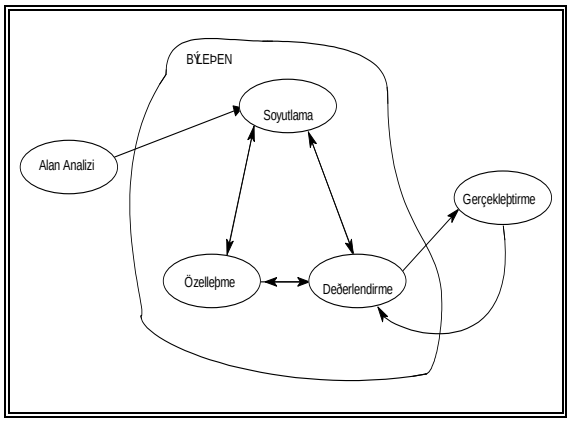
- Çok yapılılık (polymorphism) özelliği, kalıtım ile ilişkilenen bir çok nesnenin, aynı yöntemleri kullanmasına izin verir. Böylece yeni özelleşmiş bileşenlerinin, çevreyi değiştirmeden eski bileşenler ile aynı çevrede kullanılmasını sağlayarak, tekrar kullanımına katkıda bulunur.

2.2 Tekrar Kullanım Amaçlanarak Nesneye Yönelik Yaklaşımla Bileşen Geliştirilmesi Için Bir Süreç

Booch tarafından özellikle vurgulandığı gibi, nesneye yönelik geliştirmeler yinelemeli (iteratif) veya artışlı (incremental) olarak geliştirmeye uygundurlar (Booch, 1994). Yinelemeli türde, bir problem için her biri gereksinimleri karşılamaya daha yakın bir dizi çözüm geliştirilir. Geliştirilen her çözüm tamdır ama duyarlılığı ve kabul edilebilirliği her geçişte artmaktadır. Artışlı türde ise, yine bir dizi geliştirme yapılır ancak herbirinin sonucu tam bir çözüm olmamakta, çözümün bir parçası olmaktadır. Nesneye yönelik yazılım geliştirilmesinde hangi yöntem kullanılırsa kullanılsın düşünce aynıdır. Bir sistemi bir defada oluşturmak yerine sistemi bir dizi sonunda, her defada bir öncekine bir şeyler ekleyerek oluşturmaktır.

Tekrar kullanılabilir nesneye yönelik yazılım geliştirilirken yinelemeli bir yaklaşım kullanılması ve bir alandaki tek bir problem veya uygulama yerine, alandaki bir dizi problemin düşünülmesi gereklidir. Böylece, bir sıradüzenin bir parçası veya bir uygulama alanı için bir sınıflar çerçevesi oluşturacak kod tasarlanabilir. Bir sınıflar çerçevesi, tekrar kullanılabilir bir kavramın soyutlamasını gösteren bir grup sınıf olarak tanımlanmıştır (Gossain,1990).

Gossain ve Anderson tarafından tanıtılan nesneye yönelik tekrar kullanılabilir yazılım geliştirme süreci 5 ana aşamaya ayrılabilir. Sürecin şekilsel gösterimi Şekil 1'de görülmektedir. Süreç, her aşamada geri dönüş ve tekrara olanak sağlar. Bu nedenle, nesneye yönelik yazılım sistemlerinin yinelemeli gelişimine uygundur. Süreçdeki beş aşama:



Şekil 1 Nesneye yönelik yazılım geliştirme süreci

- Alan Analizi
- Soyutlama
- Özelleştirme
- Değerlendirme
- Gerçekleştirme

olmaktadır.

Alan Analizi: Alan analizi, sistem analizinden önce yapılan bir etkinlikdir. Alandaki tüm sistemlerin, benzer işlemlerinin ve nesnelerinin özelliklerini ortaya çıkaran bir alan modeli üretir. Böylece, bir alandaki nesneler, işlemler ve alana ilişkin sınıflar belirlenir. Alan analizi aşaması, çok sayıda yinelemeye sahip olabilir. Alan analizi sonucunda sınıflar çerçevesi oluşturulabilir. Böylece alandaki başka problemler için özelleştirilebilecek ortak soyutlamalar elde edilir.

Bir bankanın işlemler sisteminin hesaplarla ilgili alanını düşünürsek, alan analizi sonuçları aşağıdakiler olabilir.

- Ortak özellikler: Hesapların açılması/kapanması, para aktarımı, para yatırılması/çekilmesi gibi.
- Özel kavramlar: Farklı tipteki hesaplar/hesap sahipleri.
- Soyut kavramlar: Müşteri, vadesiz hesap, vadeli hesap.
- Genel ilişkiler: Müşteri A, 3657126 numaralı hesabı açar.
- Soyut ilişkiler: Müşterinin ismi vardır, hesabın numarası vardır.
- Aday sınıflar: Vadesiz hesap, vadeli hesap vb.

Alan analizinin önemi, yazılım tekrar kullanımının dikey ve yatay tekrar kullanım olarak sınıflandırılması ile açığa çıkar (Banker, 1993). Dikey tekrar kullanım, bir kuruluştaki yazılım geliştirme uygulamalarının çoğu, tek bir veri işleme etkinliğinin gerçekleştirimi olduğu ve bir çok yazılım nesnesi, diğerleriyle paylaşılabildiği durumda oluşabilir. Yatay tekrar kullanım ise, çeşitli uygulama alanlarında yer alabilir. Dikey tekrar kullanım, daha fazla potansiyel yarar sağlamasına karşın daha az sıklıkla gerçekleşir. Dikey tekrar kullanım için, geliştiricilerin, olası benzerliklerin en fazlasını içerecek sistemler tasarlaması gereklidir. Bunun birinci koşulu ise iyi ve tam bir alan analizidir.

Soyutlama: Soyut sınıflar, alandaki sınıfların ortak özellikleri, tarifleri ve ilişkilerinden belirlenir. Soyut sınıflar, alandaki nesnelerin temsilcileridir ve diğer sınıfların türetileceği temeli oluştururlar. Ancak, soyut sınıfları bulmak kolay değildir ve çok tekrar gerektirebilir. Soyutlama işlemi adım adım açıklanamaz. Soyutlama, uzmanın bilgisine ve tasarımcının yeteneğine bağlıdır. Bir bileşen ortamının bulunması, tasarım örnekleri vererek tasarım işlemine yardımcı olabilir. Ayrıca, aşağıdaki özelliklerden bir veya daha fazlasını paylaşan bir sınıflar kümesi, soyut sınıfların belirlenmesinde yardımcı olur. Bunlar:

- Ortak temel kimlik
- Ortak amaç
- Ortak davranış
- Ortak yaklaşımlar

olmaktadır.

Banka sistemi örneğinde, çeşitli hesap tipleri için bir soyut sınıf oluşturulabilir. Genel bir sınıf, hesap tüm sınıfların ortak davranışını içerir. Örneğin: para yatırma ve çekme. Alt sınıflar ise özel hesap tiplerinin davranışını gösterir.

Özelleştirme: Genel soyut sınıflar belirlendikten sonra, eldeki problem için kullanılabilecek sınıflar türetilir. Özelleştirme aşaması, her soyut sınıfın, sistem analistine göre yeterli seviyeye gelene kadar somutlaştırılmasını gerektirir. Bu aşamada, uygun bir gösterimle, örneğin Booch tarafından önerilen gösterim ile, sınıflar ve ilişkileri tariflenebilir (Booch, 1994). Bu gösterim, sınıfların özelliklerini, sınıflar arası ilişkileri ve ilişkilerin sırasını belirtir. Bu aşamada, olası değişikliklerin göz önüne alınması, tekrar kullanımın ön koşuludur. Sık karşılaşılan hatalar, yanlış alt sınıflara ayırma ve fazla özelleşmedir. Bunun sonucu olarak, birbirlerine çok benzeyen sınıflar veya çok ayrıntılı sınıflar oluşur. Bu aşamanın sonunda, problem uzayındaki

kavramsal nesnelerle gerçekten uyuşan bir dizi sınıf elde edilmelidir. Bu sınıfların özellikleri ve sınıflar arası ilişkiler tanımlanmış olmalıdır.

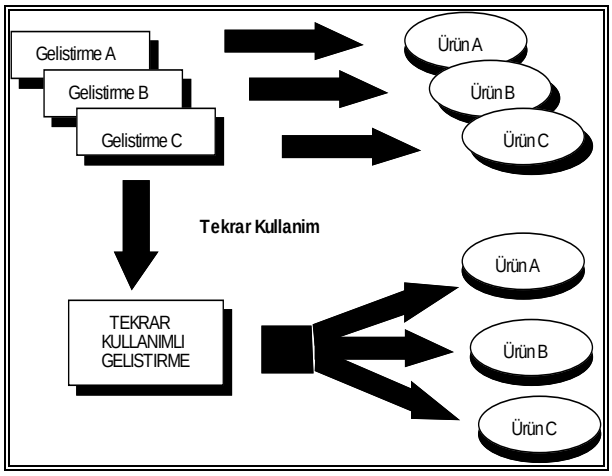
Banka örneğine dönersek, bir soyut sınıf olan hesabın altsınıfları, vadesiz hesap ve vadeli hesap olabilir. Bir vadeli hesabın açılması için, en az bir miktar paranın yatırılması gerekebilir. Vadeli hesabın altsınıfları vade zamanına göre olabilir. Her yeni sınıfın eklenmesi, tekrar kullanım olasılığını arttırır.

Değerlendirme: Bu aşamada, sınıflarla ilgili ayrıntılar ele alınarak, arayüzlere, gerçekleştirimlere, sınıf yapılarına düzeltmeler yapılmalıdır. Gerekli değişiklikler, sınıflar bir senaryo ile sınandığı zaman ortaya çıkar. Her seviyedeki sınıflar, sınıf-içi ve sınıflar-arası değişikliklerin belirlenmesi için incelenirken, bir alt sınıfta işlevsellik eksiği, çok fazla genel bir sınıf veya fazla özelleşme bulunabilir. Bu tip hataların bulunması, önceki aşamaların tekrarını gerektirebilir.

Gerçekleştirme: Bu aşamada, çözümler oluşturmak için sınıflar kullanılır ve kodlanır. Gerçekleştirmede karşılaşılan bir zorluk, önceki tasarım aşamalarındaki bir hatalı değerlendirmeyi işaret eder.

3. TEKRAR KULLANIM AMAÇLI GELİŞTIRMELER IÇIN BIR YÖNETIM STRATEJISI

Literatürde sıklıkla belirtildiği gibi, bir yazılım geliştirme sürecinde tekrar kullanım kendiliğinden olmaz. Tekrar kullanımın gerçekleşmesi için yatırım ve planlama zorunludur (Davis, 1994), (Pitmann, 1993). Aslında programlama sürecinde programcılar, kod, altyordam, ve algoritmaların tekrar kullanımını yapmaktadırlar. Ancak bunlar rasgele tekrar kullanıma örnektir ve proje düzeyinde ya da kuruluş düzeyinde değildir. Bireysel tekrar kullanımdır. Istenen ve gerekli olan ise, bireysel olarak tekrar kullanımdan öte, kuruluş düzeyinde hatta daha geniş olarak yazılım endüstrisi genelinde tekrar kullanımın planlı olarak gerçekleştirilmesidir. Bir kuruluşta tekrar kullanımın gerçekleşmesi için ortak problemler belirlenmelidir. Daha sonra ortak çözümler geliştirmek için kaynak ve planlama gereklidir. Geleneksel olarak yazılım geliştirme yönetimi, tek bir yazılım projesi üzerinde yoğunlaşır. Tekrar kullanım amaçlı geliştirmelerde ise birden çok proje düşünülmelidir. Şekil 2'de tekrar kullanıma yönelik süreç dönüşümü görülmektedir.

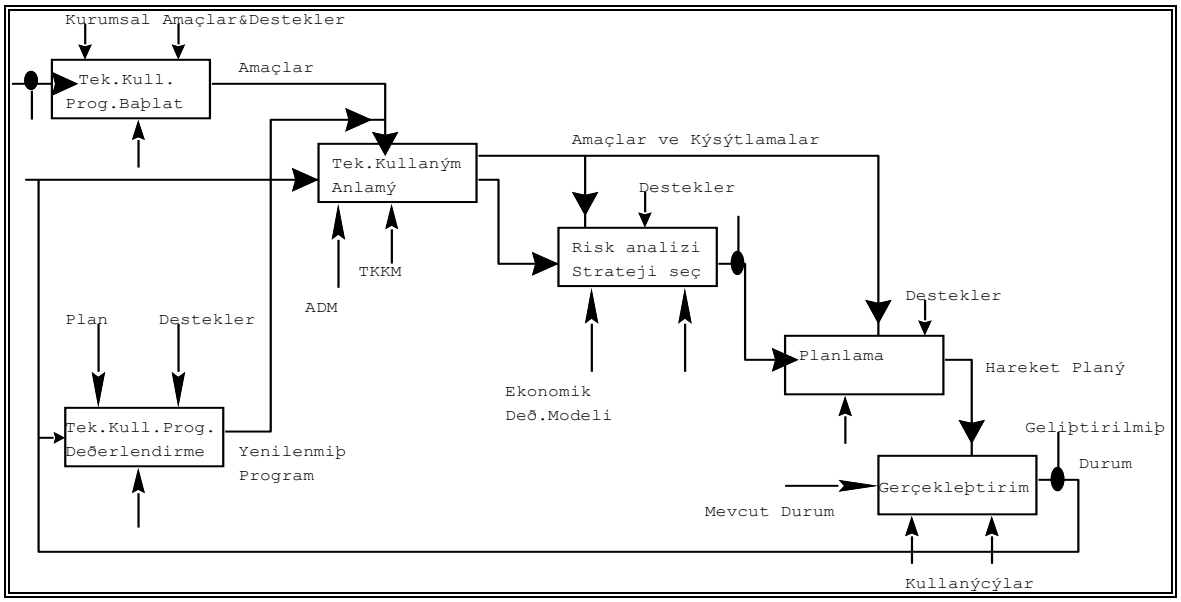


Şekil 2 Tekrar kullanıma yönelik süreç dönüşümü

3.1 Tekrar Kullanımı Kurumsallaştırma Süreci

Tekrar kullanımı kurumsallaştırmak, bir kuruluşun işlerini yürütme şeklini değiştirmeyi gerektirir. Bunun için bir programa ve koordineli çalışmaya gerek vardır. Bir çok kuruluşta yaşanan deneyimler, sistem tasarım ve yazılım tekrar kullanımın tüm kuruluşta uygulanan bir program ile gerçekleştirilebileceğini göstermiştir. Davis, tekrar kullanımı kurumsallaştırmayı, bir kuruluşun çalışma şeklini, özgün sistem geliştirmelerinden, tekrar kullanım yaklaşımına yönelik sistem geliştirmelerine dönüştürme süreci olarak tanımlamıştır (Davis, 1994). Bu dönüşüm, yönetimsel, kültürel, sosyal ve ekonomik bazı problemleri ortaya çıkarır. Bunların çözümü, planlı bir yönetim yaklaşımıdır. Bu amaçla önerilen bir kurumsallaştırma süreci Şekil 3'de görülmekte ve bu bölümde açıklanmaktadır.

Bir Program Başlatmak: Başlamak en fazla çabayı gerektirir. Çünkü, tekrar kullanımın yararlarını, maliyetini araştırmak için kaynak gereklidir. Kaynak ise insan, zaman ve paradır. Bu çabayı kolaylaştırmanın yolu, tekrar kullanım yaklaşımının, kurumsal amaçlara ulaşmaya nasıl faydalı olacağını ortaya çıkarmaktır.



Şekil 3 Tekrar kullanımı kurumsallaştırma süreci

Tekrar Kullanımın Anlamı: Bir kuruluş, tekrar kullanım açısından nerede olduğunu bildikten sonra, nereye gitmek istediğini ve nasıl gideceğini belirleyebilir. Eğer nerede olduğunuzu bilmiyorsanız, bir haritanın faydası olmaz (Davis, 1994). Başarı için iki anahtar; bir iş alanındaki tekrar kullanım olanaklarını bulmak ve kuruluşun bu olanakları kullanabilme yeteneğini bilmektir. Birincisi için, Alan Değerlendirme Modeli (ADM) yardımcı olur. ADM'nin ilk adımı önerilen süreçte de bulunan alan analizidir. ADM, bir kuruluşun tekrar kullanım potansiyelini etkileyen çevre ve ürün faktörlerini belirler (Davis, 1994). Bu faktörler:

- Piyasa potansiyeli
- Alanda var olan tekrar kullanılabilir ürünler
- Benzerlikler ve farklılıklar
- Alan sürekliliği
- Alandaki standartizasyon

olmaktadır.

ADM ile, bir tekrar kullanım programının ne derecede yararlı olabileceği ve çabaları nerede yoğunlaştırmak gerektiği belirlenebilir. Bir kuruluşun, tekrar kullanım olanaklarını kullanabilme yeteneği ise izlediği süreçlere, yöntemlere ve araçlara bağlıdır. Tekrar Kullanım Kapasite Modeli (TKKM), bir kuruluşun tekrar kullanımı gerçekleştirmesini etkileyen kurumsal ve süreçsel faktörleri ortaya çıkarır (Davis, 1994). TKKM, bir Değerlendirme Modeli ve bir Gerçekleştirme Modeli olarak iki bölümden oluşur. Değerlendirme modeli, kuruluşun yazılım geliştirme sürecini, tekrar kullanım açısından inceler, yeteneklerini ve geliştirme olanaklarını değerlendirerek tanımlı bir tekrar kullanım sürecine dönüştürmek için yapılması gerekenleri ortaya çıkarır.

Strateji Belirlenmesi: Bir kuruluş, alan ve değerlendirme modelinin sonuçlarıyla, tekrar kullanım ile ulaşılabilecek gerçekçi hedefler belirledikten sonra, bu hedeflere ulaşmak için strateji geliştirmelidir. Strateji aşağıdaki 6 önemli noktayı ele almalıdır:

- Geliştirilecek bileşenler: Birden çok kez kullanılmaya gereksinim duyulacak bileşenler düşünülmelidir. Bu nedenle analiz yapılarak, tekrar kullanım potansiyeli fazla olan ürünler belirlenmelidir.
- Yasal ve kültürel konular: Bir kuruluş içinde hatta aynı projede çalışan kişiler bile sadece kendi yazdıkları koda güvenme eğilimindedirler. Ayrıca var olan bir kod için talep edilecek ücretin düşük olacağı düşüncesi yaygındır. Bu ve buna benzer inanışlarla mücadele edilmelidir.
- Geliştirme süreci: Kullanılacak sürecin tekrar kullanımı amaçlaması ve bu doğrultuda devam etmesi sağlanmalıdır.
- Kuruluş yapısı: Gerektiği takdirde, kuruluştaki var olan yapılanma tekrar kullanım düşünülerek değiştirilmelidir.
- Geliştirme ortamı: Geliştirme ortamında tekrar kullanım için gerekli donanım, yazılım, ve araçların bulunması sağlanmalıdır.

- Dönüşüm: Yukarıdaki noktalar ışığında, kuruluşun var olan geliştirme sürecinin, tekrar kullanımı gerçekleştiren bir sürece dönüşmesi için bir yöntem belirlenmelidir.

Planlama: Bir kuruluş, amaçlarıyla kısıtlamaları uyumlu ve riskten çok yarar getiren bir strateji bulduktan sonra, stratejisini bir hareket planına dönüştürmelidir. Plan, hangi işlerin kimler tarafından ve ne zaman gerçekleştirileceğini belirlemelidir.

Gerçekleştirme: Gerçekleştirmede, yapılması gerekenler önceliklerine göre sıralanır ve amaçlar belirlenir. Ana amaç, kuruluşun izlediği süreci, tekrar kullanımdan beklenen yararı arttıran ve riski azaltan bir tekrar kullanım sürecine dönüştürmektir. Gerçekleştirme modelinde, dönüşüm için gerekli çabayı tahminleyen bir taslak hazırlanması yararlı olur. Gerçekleştirim sürecinde dikkat edilmesi gereken noktalar şunlardır:

- Iyi tanımlanmış ve ölçülebilir tekrar kullanım amaçlarının belirlenmesi.
- Tekrar kullanım çabasının, tekrar kullanım amaçlarına etkisinin izlenebilmesi ve ölçülebilmesi için gerekli ölçümlerin belirlenmesi. Bu amaçla toplanacak verilerin tanımlanması.
- Kalıtım özelliğini kullanan çerçeveler.
- Deneyimli personel kullanılması veya personelin gerekli eğitimden geçirilmesi.
- Tekrar kullanılabilir bir bileşeni geliştirmenin, sınamanın ve belgelemenin, tekrar kullanılabilir olması amaçlanmayan bir bileşene göre çok daha fazla çaba gerektireceği ve maliyetinin yüksek olacağının düşünülmesi ve tüm ekibe elde edilecek yararların anlatılması.

Başka projelerde geliştirilmiş var olan bileşenlerin tekrar kullanımında dikkat edilecek noktalar ise:

- Bileşen tekrar kullanımını değerlendirmek için maliyet/fayda analizleri yapılması
- Bileşenlerin kalitesi
- Bileşenlerin amaca uygunluğunun belirlenmesi

olmaktadır.

Tekrar kullanım programının gelişimi, plana ve amaçlara göre izlenmelidir. Gelişime, elde edilen sonuçlara ve ortaya çıkan problemlere göre, planda uyarlamalar yapılması gerekebilir.

4. SONUÇ

Nesneye yönelik yaklaşımın ticarî alanda benimsenmesinin en önemli nedeni, potansiyel tekrar kullanım olanaklarıdır (Pitmann, 1993). Çünkü, bileşen tekrar kullanımını sağlamanın en iyi yolu nesneye yönelik yaklaşımdır. Fakat sadece nesneye yönelik yaklaşım ve nesne tabanlı programlama dillerinin kullanımı, tekrar kullanımın oluşması için yeterli değildir. Kullanılan yazılım geliştirme sürecinin nesneye yönelik geliştirmelere uygun olarak değiştirilmesi ve tekrar kullanılabilir sınıf kütüphanesi, etkin arama ve alma mekanizmaları oluşturulması gereklidir. Nesneye yönelik yaklaşım, çeşitli alanlarda, artan sayıda tekrar kullanılabilir kod kütüphanelerinin ve çerçevelerinin gelişimini desteklemiştir. Ancak, düzenli olarak tekrar kullanılabilir analiz, tasarım ve bileşen geliştirilmemektedir. Bunun önemli bir nedeni, tekrar kullanımın, nesneye yönelik geliştirmelerin önemli bir amacı olmasına rağmen, gereksinimlerde veya planlarda ender olarak belirtilmesidir. Pitmann'ın da özellikle vurguladığı gibi, tekrar kullanımı başarmanın yolu, teknikler ve araçlar olduğu kadar izlenen yönetim stratejisidir (Pitmann, 1993). Bunun için, tekrar kullanım ile ilgili konuların, proje yöneticisi ve lideri tarafından proje planına dahil edilerek, amaçların ve izlenecek sürecin tüm ilgililerce anlaşılması şarttır.

Bir kuruluşta uygulanan bir tekrar kullanım programının başarısını belirleyen önemli bir unsur, aşağıda belirtilen iki maliyetin göreceli büyüklüğüdür:

- Var olan bir yazılım nesnesini bulmak, amaca göre uyarlamak ve yeni bir uygulamaya katmak.
- Aynı fonksiyonu sıfırdan oluşturmak ve yeni yazılıma dahil etmek (arama ve adaptasyon maliyetlerinin elimine edilmesi).

Bu nedenle, tekrar kullanımı amaçlayan bir kuruluşta, yazılım bileşenlerinden oluşan ve iyi bir sınıflandırma uygulanmış, bir yazılım kütüphanesinin bulunması gereklidir. Tekrar kullanım düzeyi, daha fazla tekrar kullanım adayının hazır olması ve programcıların arama maliyetlerinin azaltılması ile arttırılabilir.

Bir kuruluş, bir tekrar kullanım programını uygularken, bir ekonomik değerlendirme modeli ile, yazılım tekrar kullanımının mali portresini çıkarmalıdır. Böyle bir model, tekrar kullanım ile sistem geliştirme maliyetini, ve tekrar kullanımın

üretkenlik ve kalite üzerindeki etkisini ortaya çıkarır. Yazılım geliştirme üretkenliğini ölçmek, tekrar kullanım için yapılan yatırımları ve geliştirilen yazılım maliyetlerini kontrol etmek için yararlıdır. Böylece bir kuruluşta, tekrar kullanım ile geliştirilen sistem sayısı arttıkça, üretkenliğin arttığı görülür. Banker ve Zweig'e göre, herhangi bir endüstriyel ihtilalin gücünü ölçmek için gerekli olan şey, üretkenlik ölçümüdür (Banker, 1994).

Özellikle son yıllarda kullanımı yaygınlaşan nesneye yönelik yaklaşım ile kaliteli bileşen üretimin artması ve tekrar kullanımın yazılım geliştirme projelerinde, gerçekleştirilecek hedeflerden birisi olarak yer alması beklenmektedir. Planlı bir şekilde izlenecek süreçler, tekrar kullanımı, yazılım üreten kuruluşlar ve yazılım endüstrisi genelinde yaygınlaştıracaktır. Tekrar kullanım kısa vadede kaynak tüketir, ancak uzun vadeli yatırımlarda kazançlı ve faydalıdır.

5. KAYNAKLAR

Krueger, C. 1992. Software Reuse, ACM Computing Surveys, 24 (2), pp 131-182.

Barnes, B., Bollinger, T. 1991. Making Reuse Cost Effective, IEEE Software, pp 13-19.

Diaz, R. 1993. Status Report: Software Reusability, IEEE Software, pp 61-66.

Davis, T. 1994. Adopting a Policy of Reuse, IEEE Spectrum, pp 44-48.

Jacobson, I. 1993. Object Oriented Software Engineering, Addison-Wesley.

Booch, G. 1994. Object Oriented Analysis and Design, The Benjamin/Cummings Publishing Company.

Coleman, C., Arnold, P. 1994. Object Oriented Development-Fusion Method, Prentice Hall.

Gossain, S., Anderson, B. 1990. "An Iterative-Design Model for Reusable Object-Oriented Software", **Proceedings of OOPSLA'90**, pp 12-25.

Banker, R., Kauffman, R., Zweig, D. 1993. Repository Evaluation of Software Reuse, IEEE Transactions on Software Engineering, pp 379-389.

Pittman, M. 1993. Lessons Learned in Managing Object-Oriented Development, IEEE Software, pp 43-53.

Banker, R., Kauffman, R., Zweig, D., Wright, C. 1994. Automating Output Size and Reuse Metrics in a Repository-Based CASE Environment, IEEE Transactions on Software Engineering, pp 169-185.