HAVACILIK VE UZAY TEKNOLOJİLERİ DERGİSİ
TEMMUZ 2003 CİLT 1 SAYI 2 (1-12)

YAZILIM MÜHENDİSLİĞİNDE KALİTE VE UML

Dr. Sefer KURNAZ
Hava Harp Okulu Komutanlığı
Yeşilyurt/İSTANBUL
s.kurnaz@hho.edu.tr

Ömer ÇETİN
Hava Harp Okulu Komutanlığı
Yeşilyurt/İSTANBUL
domri2003@yahoo.com

Prof.Dr. Fuat İnce
Hava Harp Okulu Komutanlığı
Yeşilyurt/İSTANBUL
fuat.ince@superonline.com

ÖZET

Gelişen yazılım sektöründe, yazılım geliştirme süreçleri içersinde kaliteye ne denli önem verildiği ve mevcut kalite kriterlerinin nasıl değerlendirildiği deneysel olarak anlatılmaya çalışılmış ve Yazılım Mühendisliği Kalite Kriterlerine UML (Unified Modelling Language) Katkısı deneysel olarak görülmeye çalışılmıştır.

Sonuç olarak üretilen yazılımında bir tüketim maddesi olduğu fikrine dayanarak üretici ve tüketici arasındaki gereksinim ilişkileri ve bu ilişkilerin oluşmasında iletişimin sağlanmasının kaliteye etkisi, çalışma ortamlarının ve proje bağlı olarak çalışan kişi sayısının kaliteye etkisi deneysel olarak görülmeye çalışılmıştır.

Anahtar Kelimeler : *Bilişim teknolojileri, Nesneye Yönelik Programlama, Görsel Modelleme*

1.GİRİŞ

Bilişim teknolojilerinde (BT) son on yıllık devrimsel atılım bilgi çağını başlatmıştır. Özellikle kişisel bigisayarların (Personel Computer-PC) hayatımıza girmesi yaşamımızın her anında bilgisayarlarla iç içe olmamıza olanak vermiştir. Artık iş yerimizde, evimizde yaşantımımızın her anında bilgisayarlarla karşılaşıyoruz. Bu gelişmeler doğal olarak beraberinde yazılımlara da yansımış, yazılımlara olan ihtiyaç arttıkça daha etkin yazılım geliştirme yollarına ihtiyaç duyulmuştur. İnsan oğlunun işlerini kolaylaştırmak ve hızlandırmak yazılımların her geçen gün biraz daha karmaşık yapılar haline gelmesine sebep olmuştur. Çünkü yaşantımızda teknolojik gelişmeler hiç durmuyor ve bunun paralelinde gittikçe daha karmaşık sorun ve ihtiyaçlara çözüm getiriyor.

Donanımsal ilerleme bilgisayar dünyasında çok hızlı olmuştur. Mikroçip üreticileri için çok önemli olan ve yaygınca bilinen Moore's Law (Moore Yasası) bu ilerlemeyi çok güzel göstermektedir. Moore Yasası 'Her 18 ayda silikon çiplerin üstüne yerleştirilebilen tranzistor sayısının iki katına çıkacağını öngörür' ve bu son yirmi yılda doğruluğunu ortaya koymuştur. Ancak yazılım uzun yıllar donanımlara uyumlu kalmak ve donanımlarla sınırlı kalmak zorunluluğunda ilerlemiştir. Fakat Bilişim Teknolojilerinde ilerleyen yıllarda, beklenenin aksine donanımların yazılımlara yön vermesi yerini sürpriz bir biçimde donanımların yazılımlara yetişme ve uyum sağlama ihtiyacını doğurmuştur. Donanımda ki ilerleme yazılıma uzun yıllar yansımamıştır. Donanım 1970'ler, 1980'ler 1990'lar boyunca 18 ayda bir performansını ikiye katlarken yazılım bocalamış, başarısızlıklar artmıştır. Ancak son yıllarda ki teknik ilerlemeler ile yazılım ciddi gelişmeler içinde olduğu söylenebilir. Öyle ki bir çok donanım üreticisi firma artık ürettikleri donanımsal parçalara "belirli bir yazılıma uyumludur" diye belirtme ihtiyacı duyar olmuşlardır. Bu gelişmeler ve az önce bahsettiğimiz ihtiyaçlardan dolayı yazılım sektörü donanım sektöründen daha önemli bir hal almıştır. Yine de yazılım sektöründe ki ilerleme yeni yeni gelişim içersine girmektedir.

2.YAZILIM KARMAŞIKLIĞI VE MODELLEMENİN ÖNEMİ

Karmaşık sistemleri anlayabilmek insanoğlu için fizyolojisi gereği her zaman güç olmuştur. Karmaşık bir yapıyı bir bütün olarak ele almak, zihinde canlandırmak elbette çok güçtür. Her insanın karmaşayı zihninde çözebilmesinin belli bir sınırı vardır. Muhakkak bu sınırı belirleyen bir çok iç ve dış etki vardır. Bu sebepten sınır sürekli kişiden kişiye ve zamana göre değişkenlik gösterir. Fakat bir sınır her zaman vardır. Bunu aşmak için bir çok yöntem geliştirilmiştir. Bunlardan en önemlisi karmaşık yapıyı daha kolay

anlaşılabilecek parçalara bölmek ve canlandırmak için modellemek ve daha sonra bu parçaları aralarındaki ilişkileri inceleyerek bir bütün haline getirmek için sırasıyla birleştirerek sonuca ulaşmaktır. Bir örnek vermek gerekirse; size evinizin arka bahçesine bir köpek kulubesi yapın dersek tereddüt etmeden bir kaç parça tahta ve bir kaç çivi alıp hemen inşa etme çabası içine girip amacınıza çok kısa bir süre içersinde ulaşır ve sonucu, daha işe başlamadan önce tahmin ettiğiniz gibi bir çırpıda hallederek, elde edersiniz. Peki size bir ev inşa edin dersek. Bu durumda da hemen işe koyulmazdınız. Önce bir planlama yapma gereksinimi duyardınız. Belki daha bu aşamada bu işi beceremiyeceğinizi düşünürdünüz. Bunun sebebinin sizin kişisel yeteneklerinizle de bir ilişkisi olduğu düşünülebilr. Yani sizin bir ev yapmak için yeterli bilgiye sahip olmadığınız yada bu konuda yeteneğinizin bulunmadığı düşünülebilir. Yani bu işi bu sebeplerden dolayı yapamıyacağınızı düşünebilirsiniz. Bu olasılıkları ortadan kaldırmak için sizin bir inşaat mühendisi olduğunuzu ve daha önceden benzer bir kaç iş yaparak yeterli tecrübeye sahip olduğunuzu da varsayalım. Yine de bir kulübe yapmakdan daha karmaşık bir yapı olduğu için hemen malzemeyi kapıp inşaaya başlayamıyacak ve planlama yapma gereği duyacaksınız. Bu görevi de düşünmeden gerçekleştirdiğinizi kabul edelim. Sizden bir spor solonu inşaa etmenizi istersek ne yapıcaksınız. Bu durumda kesinlikle probleminizi planlamak hatta bu planı belli aşamalara bölmek ve her aşamayı kendi içersinde belli bölümlere ayırarak incelemek zorunluluğu hissedeceksiniz. Yani yapıyı daha küçük parçalara ayıracak ve her parçayı ayrı ayrı modelleyerek gerçekleştirme çabasına girecek ve sonuçta tüm bunları yaptıktan onra işe koyulacaksınız. Tüm bunları yaptıktan sonra elinizde sonuç olarak bir spor solonunu nasıl inşaa ederim sorusuna cevap elde edeceksiniz.

Bu basit örneği yazılım dünyasına uydurmak oldukça kolaydır. Kaynak kod parçalarına bakmak ya da programın herhangi görsel arayüz parçalarını analiz etmek geliştiriciye projenin tümünü kapsayacak bir bakış açısı kazandırmaya yetmez. Bir model oluşturmak geliştriciye ya da programcıya proje çalışması içinde detay bilgisine boğulmadan temel konuyla ilgili zihninde bir resim oluşmasına olanak verir. Aksi taktirde, projenin karmaşıklılığı arttıkça, çok yanlış düşünce ve tahminler doğacaktır.

3.YAZILIM GERÇEKLEŞTİRME ARAÇ VE YÖNTEMLERİ

Yazılım alanında beklenenin aksine yaşanan bir başka gelişmede daha küçük, daha hızlı, ucuz ve basit yazılımların doğması yerine daha karmaşık ve daha fazla donanımsal ihtiyaçlara sebep olan, donanımlardan daha pahalı bir yazılım sektörünün doğmuş olmasıdır. Bunların önüne geçmek için çok fazla yazılım teknik ve yapısı ortaya çıkarılsa da bunun önüne geçememiştir. Buna en basit örnek olarak Nesne Yönelik (NY) Programlama Dilleri (Object Oriented Programming Languages) verilebilir. Tek başlarına hiç bir programlama teknik ve teknolojisi karmaşık yapıdaki projeleri gerçekleştirmeye yeterli olmamıştır.

Gelişen bu ortam içersinde literatüre giren ve yazılım piyasasına etki eden çok çeşitli yazılım gerçekleştirme araç ve yöntemleri çıkmıştır. Kabaca bu araçları iki katogoriye ayırmak mümkündür:

1) Programlama,
2) Tasarım ve Analiz,

Araçları.

Yazılım krizinin belirttiği gibi yazılım projelerinin geliştirmesinde, hem ürünün doğru ve kaliteli olarak üretilmesi hem de yeterli hızda üretime erişilip isteğin karşılanabilmesinde zorluklarla karşılaşılır. Bilgisayar destekli araçlarla bu eksiklikleri biraz da olsa gidermek mümkündür. Araçlar insanların yerini alamazlar, ancak geliştirme sürecinde yardımcı olurlar. Bu tür araçlar önce makina mühendislerine yönelik olarak 'Bilgisayar Destekli Tasarım (Computer Aided Design:CAD)' adı ile yaygınlaştı. Genellikle üç boyutlu katı maddelerin çizimi, değişik açılar ve kesitler tanımlanarak görüntünün oluşturulması temelinde, başarılı bir geçmişe sahip oldular. Bunu elektronik konusundaki araçlar izledi ve günümüzde birçok mühendislik alanında tasarım artık büyük ölçüde bir yazılım çabası şekline dönüştü. Bu araçlar üzerinde, geliştirilecek sistemler yazılan programlar ile tanımlanmaktadır. Gerektiği yerde grafik ortamda yapılan girdiler araç tarafından otomatik olarak bir programa çevirilmektedir. 'Mum dibine ışık vermez' sözünü doğrularcasına yazılım mühendileri bu gibi bir araç desteğine diğer mühendisliklerden daha geç kavuştular. Ancak hızla yetişip değişik boyutlarda gelişen Bilgisayar Destekli Yazılım Mühendisliği (CASE) teknolojisi, diğer alanlardaki benzer teknolojiler ile bir karşılıklı etkileşime girdi ve sonra yazılım konusunda bulunan teknikler diğer alanlara da uygulanmaya başlandı.

NY dillerin yaygınlaşmaya başladığı dönemlerde var olan sistem mühendisliği yaklaşımları daha çok yapısal programlama dillerine göre oluşturulmuş olduğundan, projelere sistem mühendisliği seviyesinde problemin çözümüne götürecek yaklaşımlardan uzak kalmışlar ya da sundukları imkanlar ile NY ortamlar arasında uyumsuzluk problemleri yaşanmasına neden olmuştur.

4.MODELLEME İLE GÖRSEL MODELLEME

İşte bu seviyede karşımıza "Görsel Modelleme (Visual Modelling)" tekniklerindeki değişimler ve gelişimler çıkmıştır. Görsel Modellemeler, var olan problemlerin gerçek dünya etrafında şekillenmiş halleridir. Modeller ;

1) Problemin kavranmasında,

2) Proje ile ilgili herhangi bir iletişimde (müşteri, analizci, tasarımcı, programcı...),

3) Döküman hazırlanmasında,

4) Tasarım ve problem çözümünde,

küçümsenmeyecek kolaylıklar sağlar. Yazılımın oluşturulması ve geliştirilmesi için büyük önem taşırlar.

Modeller, karmaşık sistem ya da yapıların gereksiz ayrıntılarından, yani bir nevi filtreleme yaparak, kurtarılmasını sağlayıp daha kolay anlaşılasını sağlarlar. Buna kısaca liratürde "Soyutlama (Abstraction) " denilmektedir. Bir mühendis herhangi bir karmaşık problem karşısında sistemi ypılandırmak için öncesinde ilgili sistemin değişik açılarını soyutlayan ve karşılayan modeller oluşturmalı ve adım adım bu modellere detayları ekleyerek istenen son şekle ulaşmalıdır.

5.NOTASYON KAVRAMI VE ÖNEMİ

Geliştirme sürecinde bir başka önemli husus da gösterim biçimi veya notasyon (Notation) dur. Her alanda notasyon işlemleri bir arada tutmaya yarayan birleştirici bir etkiye sahiptir. Yazılımın gerçekleştirilmesinde kullanılan notasyon daha sonra hep yazılımla birlikte yaşayacağından dolayı yazılımın kullanılması, bakımı ve geliştirilmesi hususlarında da büyük kolaylıklar sağlayacaktır. Asla unutulmamalıdır ki yazılımlar ölmezler. Her zaman belli isteklere karşılık verirler. Eksiklikleri bakım ve ek geliştirmeler ile tamamlanarak tekrar maksimum verimli hale getirilebilirler. Bu bakım ve geliştirme etkinlikleri hep yazılımı gerçekleyen kişi tarafından yapılmaya bilir. Hatta bu kişi tarafından yapılsa dahi belli bir zaman sonra bir takım unutulmalara ve yanlış anlaşılmalara sebebiyet verebilir. İşte bu evrede geliştirilme aşamasında kullanılmış olan evrensel bir notasyonun önemi çok büyüktür. Çünkü bu yazılım işiyle uğraşan herkesin ortak bir dili konuşması gibidir. İletişim her alanda olduğu gibi yazılım alanında da oldukça önemlidir. Başlıca görevleri;

1) Yazılımı kendisinden çıkarılmayacak ya da açık olmayan kararların iletiminde bir dil (iletişimsel) görevi yüklenmek,

2) Tüm önemli stratejik ve taktik kararların alınmasına olanak ve yardımcı olacak şekilde zengin bir içeriğe sahip olmak,

3) Araçların işleyebileceği insanların da anlayabileceği sağlam bir yapı oluşturmaktır.

6.UML(UNİFİED MODELLİNG LANGUAGE)

Yazılım geliştirme ve üretim alanında ortak bir notasyon gereksinimi özellikle 90'lı yıllarda iyiden iyiye bir ihtiyaç haline gelmiş ve bu ihtiyaca cevap olarak da ortaya çok sayıda metodlar çıkmıştı. Bunlardan üçü zamanla birbirleri ile uyumlu birleşme göstererek UML(Unified Modelling Language) modelini oluşturdular. Bunlar OMT, Boach ve DOSE metodlarıydı. Her bir metodun eksi ve artıları vardı. Örneğin OMT analiz aşamasında güçlü fakat tasarım aşamasında zayıf, Boach tam tersine tasarımda kuvvetli fakat analizde zayıf kalırken DOSE ise davranımsal analizde kuvvetli fakat diğer alanlarda zayıftı.

Zaman içerisinde Boach, Rumbough ve Jacobson bir çok analiz tekniğini içeren kitaplar yazdılar ve bir çok etkili tasarım tekniği ile ilgili makaleler yayımladılar. Bunlar içerisinde en önemlisi Rumbaugh tarafından ortaya konulan OMT-2 makaleleri oldu. Bütün bunlar yaşanırken piyasada tam bir karmaşa ortamı ortaya çıktı. Çünkü yöntemler her ne kadar birbirlerine yakınlaşmaya başlasa da hala kendilerine ait notasyonlara sahip idiler. Örneğin içi dolu bir daire OMT de "çokluk" belirtirken , Boach da "kapsama alanı" anlamında kullanılıyordu. Bu dönem "METOD SAVAŞLARI" dönemi olarak adlandırılır.

Peki bir sınıf bulutla mı yoksa dikdörtgen vasıtasıylamı ifade edilmeliydi? Hangisi daha iyiydi? Metod savaşlarının sonunu notasyon açısından UML'in ortaya konması getirdi . UML geliştirilen nesneye yönelik sistemlere ait birimlerin, özelliğini belirtmekte, görselleştirmekte ve dökümante etmekte kullanılır. Boach ,OMT ve nesnel notasyonların ve iyi fikirlere yöntemlere sahip diğer metodların bileşimini temsil eder. UML bu nesneye yönelik metodların notastonlarını birleştirmekle , nesneye yönelik analiz ve tasarım aşamalarında geniş bir kullanıcı deneyimi ile "de facto" bir standardı temsil eder.

UML analiz ve tasarımın birimlerini standarize etme girişim olarak da kabul edilebilir. (Semantik modeller, sentetik notasyon ve diyagramlar gibi...) İlk resmi sürümü 1995 Ekiminde (UML version 08) ilan edilmiştir. Kullanıcı ve Jacobson (Iuar)'dan

gelen geri beslemeler ile geliştirilerek Temmuz 1996 da (UML version 09), Ekim 1996'da (UML version 0,91) çıkarılmıştır. Sürüm 1,0 ile 1,1 ise Eylül 1997 OMG (Object Menagement Group) tarafından tarihinde standaralizasyon amacıyla çıkmıştır.

7.YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜ

UML Kullanımının geliştirme boyutu ötesinde proje yönetimi açısından da oldukça ve olumlu etkileri olduğu görülmüştür. Çok değişik yazılım projeleri, çok değişik nedenlerden dolayı başarısız olabilir. Bu başarısızlığın nedeni bazen çok kolay tespit edilirken bazen de asla tespit edilemez. Bu yüzden yazılım geliştirmesinde bir patern izlemek ve hata analizinde bu paterni takip ederek hatanın nedenini tespit etmeye çalışmak hem büyük bir kolaylık hem de tekrar etmemesini sağlayacak önlemlerin alınmasında uygun bir ortam yaratacaktır. Bu paterne "Yazılım Geliştirme Yaşam Döngüsü (Shlaer-Mellor Software Life Cycle)" denilir. Bu paternin başlıca aşamalarını şu şekilde sıralayabiliriz;

1) Planlama
2) Çözümleme
3) Tasarım
4) Gerçekleştirme
5) Bakım

Yazılım Yaşam Döngüsü, herhangi bir yazılımın, üretim aşaması ve kullanım aşaması birlikte olmak üzere geçirdiği tüm aşamalar biçiminde tanımlanır. Yazılım işlevleriyle ilgili gereksinimler sürekli olarak değiştiği ve genişlediği için söz konusu aşamalar bir döngü biçiminde ele alınır. Döngü içerisinde herhangi bir aşamada geriye dönmek ve ilerlemek söz konusudur. Bu nedenle yazılım yaşam döngüsünün tek yönlü ve doğrusal olduğu düşünülmemelidir. İşlevsel bağımsızlığı (İşlevsel Bağımsızlık; modüller arasındaki bağıntıların en aza indirgenmesi anlamına gelip, bir modülde bir hata olduğu taktirde diğer modüllerin etkilenmesini engellemek amacıyla yapılır) temin etmek için modüller arasında ilişkiliği olduğunca azaltmak ve bir modülün yalnızca bir işlev ile görevlendirilmesini sağlamak gerekir. Bu basamakların izleniş sırası ve geri dönüşlerin yapılması yöntemlerine göre bir kaç farklı yöntem geliştirilmiştir. Bunlara yaşam döngüsü modelleri adı verilir. (Örneğin V Modeli, Şelale Modeli, Evrimsel Model...)

8.YAZILIM KALİTESİ

Peki "Yazılım Kalitesi Nedir?" Yazılım mühendisliği çerçevesinde sözü geçen yöntem ve araçların amacı, üretilecek yazılımda yüksek kalite düzeylerine erişmektir. Kalite ise bir çok boyutu olan bir özelliktir. Program yazıldıktan sonra düşünülecek bir konu olmaktansa, bütün geliştirme evrelerine yönelik etkinlikler ile sağlanmalıdır. Yazılım kalite güvencesi için önemli yol ve süreçler aşağıdaki gibidir:

1) Planlama, yönetim ve organizasyon süreçleri
2) Çözümlemeci tasarım, sınama yöntem ve araçları
3) Resmi gözden geçirmeler
4) Çok aşamalı sınama stratejisi
5) Belgeleme ve değişikliklerinin kontrolü
6) Geliştirme standartlarına uyum süreci
7) Ölçme ve raporlama mekanizmaları

Genel anlamda üretilen ürünün kalitesi;

1) Belirtilmiş isteklere uygunluk,
2) Standartlara uygunluk,
3) Bunların yanısıra belirtilmeyen ve varsayılan isteklere uygunluk, (Örneğin bakım kolaylığı)

kriterlerini ne denli sağladığıyla ölçülür. Yazılım bir kişi ya da grubun ürettiği bir üründür. Yazılım kalitesine etki eden unsurlar, uygulamalara ve kullanıcılara göre değişir.

9.YAZILIM KALİTESİNE UML'İN ETKİSİ

Burada UML tanındıkça şu görülecektir. Yazılım Kalitesinde aranan her istekte UML kullanılmasının etkisi çok büyüktür. Bunları teker teker ele alırsak;

1) Belirtilmiş isteklere uygunluk;

Yukarıda da belittiğimiz gibi modellemek karmaşık bir problemin çözümlenmesinde çok büyük bir kolaylıktır. UML yardımıyla modelleme oldukça basittir. Aynı zamanda görsel modeller kullanmak problemin daha kolay anlaşılmasına olanak tanıyacaktır. UML' de tüm modeller görsel olarak tanımlanır. Bu modeller bilgisayar programlama bilgisinden yoksun bir kişi tarafından bile kolayca anlaşılabilir şekillerle ifade edilirler. Böylece kullanıcı ile programcı arasında teknik bir iletişim köprüsü kurulmuş olur. Kullanıcının isteklerinin tam ve net olarak tanımlanması her zaman için çok zor olmuştur. Çünkü kullanıcılar genel programlama konseptinden uzak kişilerdir ya da programcı kullanıcının yazılımı kullanacağı alan hakkında teknik bilgiye sahip değildir. Buna şöyle bir örnek verebiliriz: Bir süpermarket sahibi mağazasında kullanılmak üzere bir stok programını bir programcıdan sipariş etmiş olsun. Mağaza sahibinin programlama hakkında bilgi sahibi olması veya programcının süpermarketin stok yapısı hakkında bilgi sahibi olması çok düşük ihtimallerdir. İşte bu durum karşısında kullanıcının isteklerinin yanlış ifade edilmesi veya yanlış

anlaşılması olasılığı oldukça yüksektir. Fakat yazılım gerçekleştirme aşamasından önce yapılan model üzerinde kullanıcı ve programlamacının bir araya gelerek modeli kullanıp gereksinimlere ve bunlara verilen çözüm yollarına beraber bakmaları olası hataları ortadan kaldıracaktır. Böylece belirtilmiş isteklere uygunluk açısından doğabilecek olası hatalar kolaylıkla ve zamanında ortadan kaldırılacaktır. Yani üretilen yazılım kalite açısından belirtilmiş isteklere neredeyse bire bir uygunluk gösterecektir.

2) Standartlara uygunluk;
Daha önce de söylediğimiz gibi standardizasyon bir çok açıdan oldukça önemlidir. Burada standardizasyon ile belirtilmek istenen, üretilen ürünün yazılım olduğu düşünülürse, sonsuz yaşantısında ürünle beraber yaşayabilecek ve bu alanda çalışan herkes tarafından kolaylıkla anlaşılabilecek bir notasyondur. UML ile bu ihtiyaca da cevap verilmektedir. UML başlı başına bir yazılım geliştirme notasyonudur. UML ile modellenerek oluşturulan yazılım bütün UML bilgisine sahip programlamacılar tarafından kolaylıkla anlaşılabilir. Her türlü bakımı veya geliştirilmesi kolaylıkla gerçeklenebilir.

3) Bunların yanında belirtilmeyen veya varsayılan isteklere uygunluk;
Kullanıcı gereksinimlerini belirtirken, kendi gereksinimi dışında beraberinde farklı gereksinimleride getirebilir. Buna yukarıdaki örnek ile bağıntılı bir örnek verirsek; süpermarket sahibi stok bilgisini hem depodaki bilgisayardan hem de aynı anda kendi odasından eş zamanlı olarak görmek istediğini gereksinim olarak belirtmiş olsun. Bu gereksinim beraberinde yazılımın bir network ortamında eş zamanlı olarak çalışmasını gerektirecek bir kaç gereksinim daha doğuracaktır. (Server ve Clientlerin kurulması, her birisi için yazılım ve ekran çıktılarının tasarlanması, networkün fiziksel yapısının oluşturulması...) Tabi ki bilgisayar teknik bilgisinden yoksun bir kullanıcıdan bunları belirtmesi beklenemez. Ancak kesinlikle bunların programcı tarafından görülerek gereksinimlere eklenmesi gerekir. Ya da kullanıcı günümüz teknolojisi ile gerçellenemeyecek gereksinimler ya da birbiriyle çelişen ya da ihtiyaç olarak görünmese de gereksinim olarak belirttiği isteklerde bulunabilir. UML ile modelleme esnasında kullanılan bir yapı örneğin Network yapısı ayrı bir yapı olarak ele alınr ve modelin tamamlanması için muhakkak tanımlanması beklenir. Yani kullanıcı tarafından belirtilmeyen gereksinimler UML tabanlı bir modelleme aracı ile belirtilir. Aynı zamanda birbirleriyle çelişen durumlar modal üzerinde kolayca göze çarpacaktır. Gereksiz istekler rahatlıkla modellden çıkarılabilecektir. UML tüm bunlara imkan tanır.

Aynı zamanda kalite kriterleri içersinde yer alan bakım ve güncelleme olanakları da kullanıcı tarafından belirtilmeyebilir. Fakat bunlar her yazılımın doğal gereksinimleri haline gelmiştir. Üretilen yazılımın geliştirilebilme imkanlarının UML ile belirtilmesi daha kolaylaştrılmıştır. Çünkü sistemin bütün yapısı yanında geliştirilmek istenen modülün yapısı ve diğer mevcut modüller ile bağıntıları ayrıntılı olarak incelenmiş bir şekilde UML de görülebilir. Tüm bunlar referans alınarak güncellenmek, geliştirilmek ya da bakımı yapılmak istenen modülün veri kaybına yol açmadan tüm gerekli işlemlerden geçmesi daha kolay sağlayacaktır. Aynı zaman da UML ortaya koyduğu notasyon sayesinde programcıya bağımlılıktan yazılımı kurtarmış olur. Yani yazılımı oluşturan kişi dışında bir kişi yazılım ile ilgilenirse(Bakım, güncelleme veya geliştirme amaçlı gibi...) mevcut yazlım modelini daha kolay kavrayacak ve yanlış anlaşılmaları minimize ederken zaman açısından da büyük avantajlar sağlayacaktır.

10.YAZILIMDA ARANAN KALİTE ÖZELLİKLERİ

Bunların her birisi içinde farklı kriterler ele alınır; Bu kalite etkenlerini ve kabaca tanımlarını sıralarsak;

1. **Doğruluk:** Yazılımın belirtilmiş gereksinimleri karşılaması
2. **Güvenirlilik:** Üretilen yazılımın gereksinim duyulan işlevleri ne hassasliyetle yerine getirebileceği beklentisi
3. **Verimlilik:** İşlevlerin gerçekleştirilmesi için kullanılması gereken bilgisayar kaynakları, zaman ve kod miktarı
4. **Bütünlük:** Yazılım ve bilgilerine istenmeyen insanlarca ulaşımın ne derece engellenebildiği, yazılımın değişmeden etkilenmeden, amacı için kullanılabilmesi
5. **Kullanılabilirlik:** Programın öğrenilmesi, çalıştırılması, girdi ve çıktı işlemleri için gerekli çaba miktarı, hatasız verimli kullanılabilirliği
6. **Bakım Kolaylığı:** Bir programda hata bulma ve onarma için gereken çaba
7. **Esneklik:** Çalışan bir programda değişiklik yapma kolaylığı
8. **Test Kolaylığı:** İstenen işlevin gerçekleştirildiğine dair yapılacak testin gerektirdiği çaba
9. **Taşınabilirlik:** Bir progamın belli bir yazılım/donanım ortamından diğerine taşınması için gereken çaba
10. **Yeniden Kullanılabilirlik:** Bir programın veya parçalarının diğer bir uygulamada veya diğer bir programın geliştirilmesinde ne derecede kullanılabileceği

11. **Birlikte Çalışabilirlilik**: Bir yazılım sisteminin diğerleri ile bağlantı sağlaması kolaylığı
12. **Denetlenebilirlik:** Standartlara uygunluğun kontrol edilmesi kolaylığı
13. **Doğruluk:**Sistemin fiziksel yapısının oluşturulan mantıksal modele ne derece uygun olduğu
14. **Haberleşme Paylaşımı:** Standart arayüzler, protokoller ve kanal kapasiteleri kullanım miktarı
15. **Tamlık:** Gereken işlevlerin tam olarak tanımlanma derecesi
16. **Kesinlik:** Programın tasarımda satır sayısı olarak küçüklüğü
17. **Tutarlılık:** Proje süresince tasarım ve belgeleme tekniklerinde uygulanan biçem benzerliği
18. **Veri Paylaşımı:** Program boyunca standart veri yapılarının kullanılması
19. **Hataya Karşı Duyarlılık:** Bir hata sonucunda programın oluşturacağı zarar ve hataya karşı alınan önlemlerin tanımlanması
20. **Çalışma Verimi:** Programın koşturulduğundaki başarımının hesaplanması
21. **Genişletilebilirlik:** Tasarımın veri, yapı ve işlev boyutlarında büyütülebilme açısından tasarımda gösterilen kolaylığı
22. **Genellik:** Programın veya bileşenlerinin değişik sahalarda uygulama olasılığı ve imkanı
23. **Donanım Bağımsızlığı:** Programın üzerinde çalıştığı donanım ile organik bağlarının bulunmaması için tasarım aşamasında alınan önlemler
24. **Modülerlik:** Programın bileşenlerinin işlevsel bağımsızlığının model üzerinde tanımlanması
25. **Çalışabilirlik:** Programın çalışma kolaylığı
26. **Güvenlik:** Program ve verinin korunması için gerekli önlemlerin tasarım aşamasında ve programlamaya geçişte ne derece göz önünde bulundurulduğu
27. **Belgeleme:** Kaynak kodu içerisinde anlaşılabilir belgelemenin bulunması
28. **Basitlik:** Programın kolayca anlaşılabilmesi, karmaşık modelleme yapılarından soyutlama
29. **Yazılım ve sistem bağımsızlığı:** Programın standart olmayan komutlar, işletim sistemi işlevleri ve diğer çevresel öğeler kullanmaması için tasarımda göz önünde bulundurulan durumlar
30. **İzlenebilirlik:** Program öğeleri ya da tasarım yapılarının gereksinimleri karşılayabilirliğinin izlenme kolaylığı.
31. **Eğitim :** Yeni kullanıcıların sistemi kolayca uygulaması için programda bulunan kolaylıklar.

Yazılımda kaliteye etki eden bir başka husus da yazılım geliştirilmesinde görevli kişi ve programcı sayısıdır. Bu kişi sayılarını da iki ayrı konuda ele almamız gerekir;

1) Programın gerçeklenmesi (Kodlama) aşamasında çalışan kişi sayısı

2) Programa ait modelleme ve dökümantasyonun gerçekleştirilmesi aşamasında çalışan kişi sayısı.

11.YAZILIM KALİTE KRİTERLERİNİN DENEYSEL İNCELENMESİ

Yukarıda yazdığımız kalite kriterlerinin ve izlenecek olan yöntem ile araçların kaliteye olan etkisini deneysel olarak incelemek amacıyla aşağıda belirtilen programlama isteğini örnek problem alarak bir deney düzenledik.

11.1 Deney Probleminin Tanımlanması

Bir mağazamız ve bu mağazada daha önceden kullanılmakta olan bir otomasyon programı bulunmaktadır. Müşteriler mağazaya gelerek dükkanda bulunan 20 çeşit üründen istedikleri mikterda sipariş vererek ayrılmaktadırlar. Her müşteri mağazaya geldikten sonra ilk alışverşinde müşteri veritabanına kasiyer tarafından kaydedilir. Mağazada bulunan ürünlerin toplamı 200 parçayı geçmemelidir. Mağaza ürünleri stok adlı bir tabloda izlemektedir. Sipariş verilmesine müteakip istek sipariş tablosuna kaydedilir. Eğer istenen ürünler dükkanda mevcut değilse depodan kasiyerin yapacağı istek üzerine getirilir. Depoda ki ürün miktarı sınırsız olarak düşünülebilir. Kasiyer siparişlerin hazırlanıp gönderilmesinden sonra sipariş tablosunun gerekli alanını günceller. Müşterilerin ödeme şekli de dikkate alınmalıdır.

Bu koşullar altında mevcut sistemin yetersizliğine kanaat getirilmiş ve yenilenmesine karar verilmiştir. Mevcut tüm kayıtların korunacağı ve veri bütünlüğünün bozulmamasına dikkat ederek aşağıda belirtilen istekler karşılanacaktır.

Uygulama Kısıtları

- Kasiyer tarafından kullanılacak ekranların tasarlanması
- Veritabanlaının oluşturulması ve eski verilerin korunması
- Projenin gerçekleştirim aşamalarının belgelenmesi

Ana Veritabanı

Müşteri tablosu	**Stok tablosu**	**Sirapis tablosu**
- musteri_no	- urun_kodu	-siparis_no
- musteri_adsoyad	- urun_mıktarı	-siparis_tarih
- musteri_adresi	- urun_fıyatı	-siparis_edilen_urunler (fk)
- musterı_tel		-gonderildimi (boolean)
- musterı_kredı_kart_no		-odeme sekli
		-musteri_no (fk)

11.2. Deneyin yapılışı;

Maltepe Üniversitesi Bilgisayar Mühendisliği 4ncü sınıf öğrencilerine yukarıda tanımlanan problem çözülmesi için verilmiştir. Problemin çözümü için öğrencileri laboratuar ortamında 4 saat süre tanınmış ve bir birleriyle iletişimleri engellenmiştir. Böylelikle daha objektif sonuçlar elde edilmeye çalışılmıştır. Sürenin kısıtlı olma sebebi problemin basitliği ve etkileşimin minimuma indirilmesini sağlamaktadır.

Deneye 19 öğrenci katılmıştır. Bu öğrenciler öncelikle A, B ve C gruplarına ayrılmış ve daha sonra her grup içerisinde birerli, ikişerli ve üçerli olmak üzere tekrar bölünmüşlerdir.

A Grupları Free Form (Kısıt ve Plan gözetmeksizin) metodunu izlemeleri söylenmiş ve bu nedenle Yazılım Geliştirme Temel Adımlarını geçerek kod üretecekleri beklenmiştir.

B Gruplarından basit modelleme basamaklarının kullanılması istenmiş ve daha sonra kurulan bu yapı üzerine kod üretmeleri beklenmiştir.

C Gruplarının ise ayrıntılı modelleme çalışmasını yapmaları ve belgeleme üzerine itina göstermeleri istenmiştir ve daha sonra kod üretmeleri beklenmiştir.

UML ile de aynı problem tek kişilik bir grup tarafından 5 saat içersinde çözülmüştür.

Grupların çalışan öğrenci sayısı aşağıdaki tabloda (Tablo-1) verilmiştir

Tablo 1. Gruplarda çalışan öğrenci sayıları

	Tek Kişi	**İki Kişi**	**Üç Kişi**
Grup A	A1-1 / A1-2	A2	A3
Grup B	B1	B2	B3
Grup C	C1	C2	C3

11.3 DENEYİN DEĞERLENDİRİLMESİ

Üretilen programın kalite kriterlerine uygunluğu açısından incelenmiştir

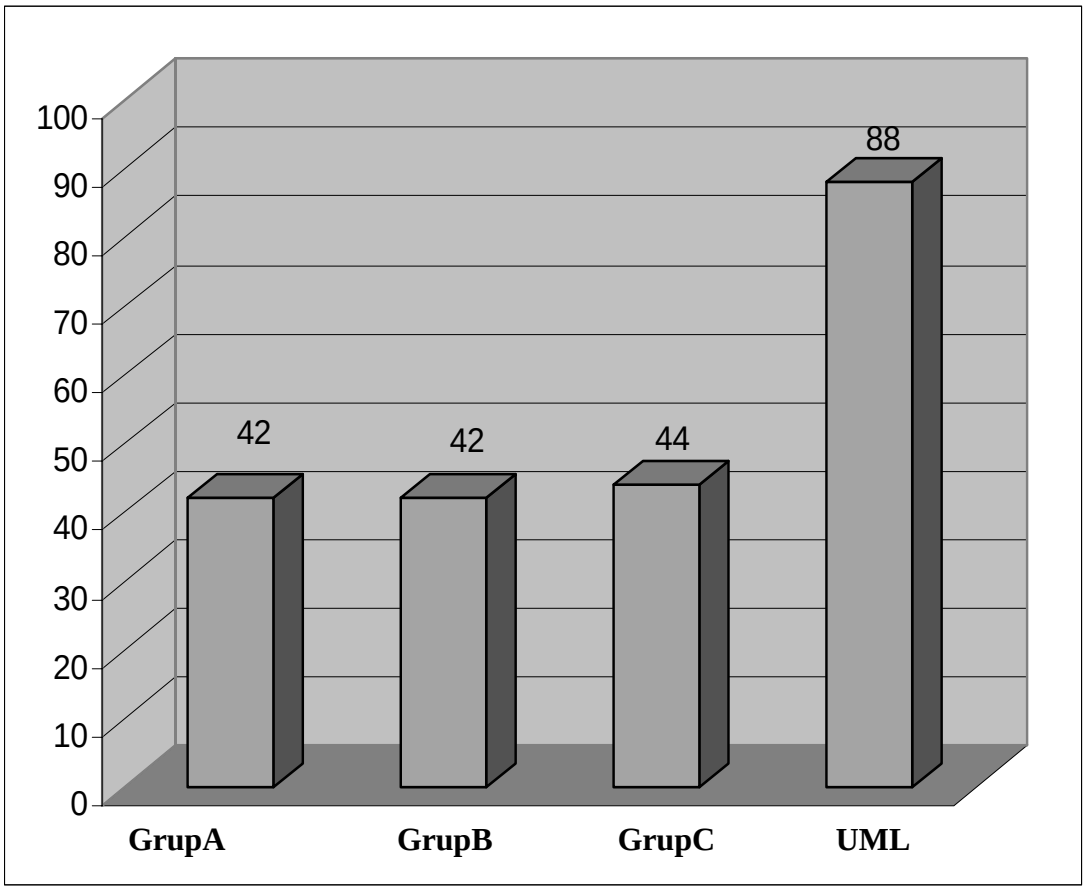
Tablo 2. Deney Değerlendirme Tablosu

	A1-1	A1-2	A2	A3	B1	B2	B3	C1	C2	C3	UML
Doğruluk	2	3	3	4	3	1	1	2	4	1	3
Güvenilirlik	1	2	3	3	3	1	2	2	4	1	4
Verimlilik	1	3	3	3	3	1	2	2	2	1	5
Bütünlük	1	1	1	4	1	1	1	1	1	1	3
Kullanılabilirlik	1	2	3	3	3	1	1	2	4	1	4
Bakım Kolaylığı	1	2	2	2	3	1	1	3	3	2	5
Esneklik	1	1	1	4	1	1	1	1	4	1	4
Test Kolaylığı	1	2	3	3	2	1	1	1	3	1	5
Taşınabilirlik	1	1	2	4	2	1	1	1	4	1	5
Yeniden Kullanılabilirlik	1	1	1	3	1	1	3	2	3	1	5
Birlikte Çalışabilirlik	1	3	3	4	3	3	2	1	4	1	5
Denetlenebilirlik	1	2	1	2	3	2	3	4	2	2	5
Doğruluk	1	3	3	3	3	3	3	3	4	3	5
Haberleşme Paylaşımı	2	2	3	3	3	2	3	1	4	3	3
Tamlık	1	2	2	3	2	2	4	4	3	2	5
Kesinlik	1	4	2	3	2	2	2	1	3	1	3
Tutarlılık	1	2	1	2	3	2	5	5	3	3	5
Veri Paylaşımı	1	3	3	2	3	2	2	1	2	1	4
Hataya Karşı Duyarlılık	1	1	1	1	1	1	1	1	1	1	1
Çalışma Verimi	1	2	2	4	2	1	2	1	4	1	5
Genişletilebilirlik	3	3	3	4	3	3	3	3	4	3	5
Genellik	1	2	2	3	2	2	3	1	3	1	5
Donanım Bağımsızlığı	1	3	3	3	3	1	2	1	2	1	5
Modülerlik	1	3	3	4	3	3	2	1	4	1	5
Çalışabilirlik	1	1	1	5	1	1	1	1	5	1	5
Güvenlik	1	1	1	4	1	1	1	1	1	1	4
Belgeleme	2	2	1	2	3	4	4	5	3	3	5
Basitlik	1	3	3	3	4	4	4	3	3	3	5
Yazılım Ve Sistem Bağımsızlığı	1	1	1	3	1	1	1	1	3	1	5
İzlenebilirlik	1	2	2	3	2	2	4	4	3	2	5
Eğitim	1	2	2	3	2	2	2	2	3	2	3
Toplam	**36**	**65**	**65**	**97**	**72**	**54**	**68**	**62**	**96**	**48**	**136**

Yukarıdaki tabloda (TABLO-2), incelenen deneylere Hava Harp Okulu Bilgisayar Mühendisliği 4ncü Sınıf Bilgisayar Mühendisliği Bitirme Tezi Öğrencileri Ömer ÇETİN ve Özgür DEREBAŞI tarafından not verilmiştir. Yazılım Mühendisliği Kalite Kriterlerine göre 5 (Beş) üzerinden verilen notlarda 1(bir) hiç uygun değil, 5(Beş) en uygun anlamındadır.

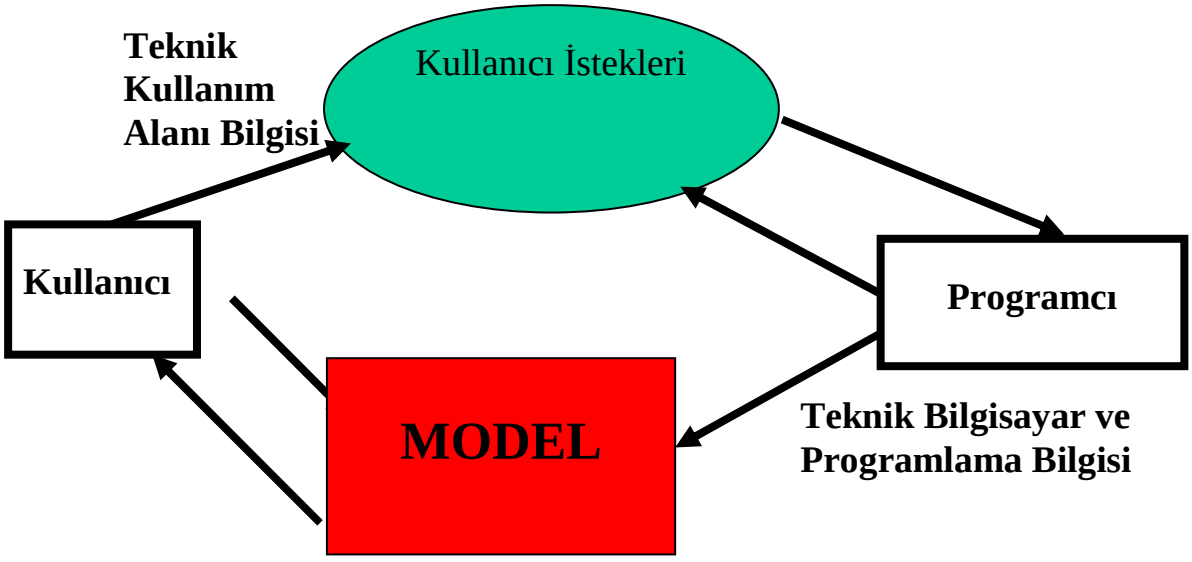
Tablo değerleri incelendiğinde UML yapısının sağladığı kolaylıklar ve standardizasyon işlemleri programcıları daha kaliteli yazılımlar üretmeye yönlendirdiği gözlemlenmektedir.

İstatiksel sonuçlar da aşağıda Grafik 1 ile sunulmuştur.



Grafik 1. İstatiksel sonuçlar

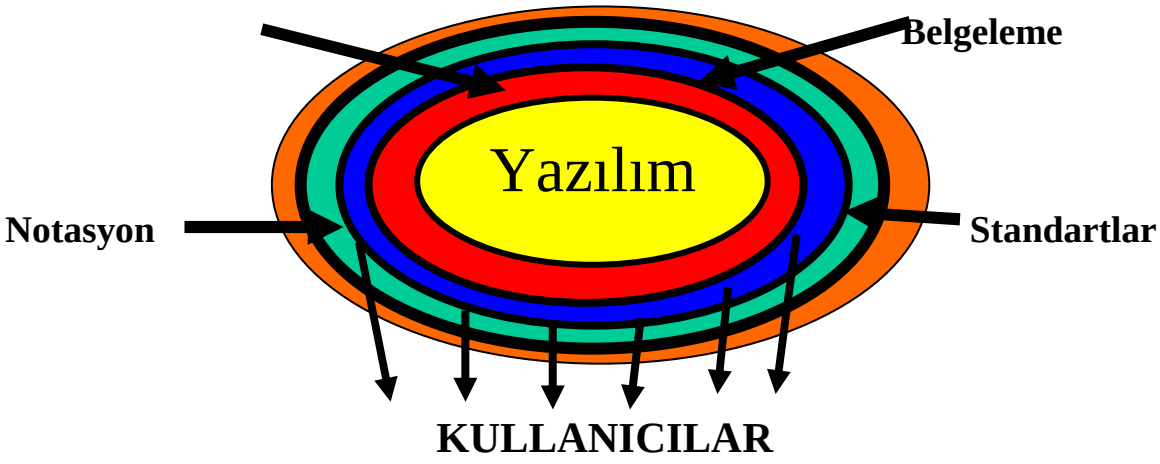
11.4 SONUÇ



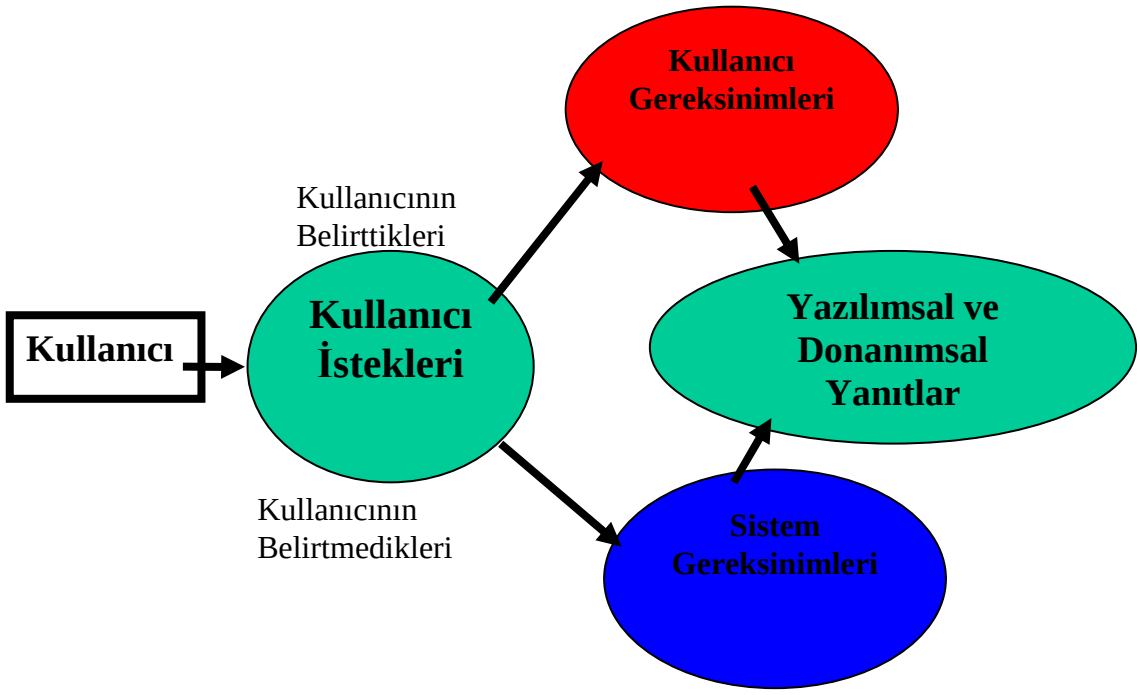
Şekil 1. Gereksinim oluşumları ve kullanıcı-programcı iletişimi

Yazılımda kalitenin sağlanması için gerekenlerin başında kullanıcının belirttiği isteklerin ve gereksinimlerin doğru anlaşılarak bunlara gereken cevabın verilebilmesidir. Bunun içinde kapsamlı bir kullanıcı-programcı iletişimine (Şekil 1) ihtiyaç duyulmaktadır. Bunun sağlamanın yolu ise Görsel Modelleme ve ortak Notasyon kullanımından geçmektedir. Yapılan deneysel uygulamadan ve UML'in temel kaliteye olan katkılarından da anlaşılacağı gibi bu alanda kalite de verimliliği arttıran ortam ve yöntem UML olmuştur.

Yazılımda kalitenin sağlanmasında diğer önemli bir hususta ortak bir notasyonun (ŞL-2) yaratılarak, tüm dünyada kolay anlaşıla bilir yaygın bir yapının modelleme yapısı olarak kullanılarak üretilen sistemin ya da yazılımın üretici konumundaki programcı ya da şirketten bağımsız kılmaktır. Bir başka deyişle, ürünün ilerki aşamalarda kullanım ve sonrası (üretimden sonra) herhengi bir kişiye ya da kuruluşa bağımlılığı kalmamalıdır. Evrensel bir yapı ile oluşturulduğundan konuyla ilgisi olan herkes tarafından kolayca anlaşılabilir şekilde oluşturulmuş olması gerekir.



Şekil 2. Ortak notasyon kullanımının yazılım içersindeki yeri ve etkileşimleri



Şekil 3. Kullanıcı tarafından belirtilen ve belirtilmeyen gereksinimlerin ortaya çıkışı

Son olarak yazılım kalitesinin oluşturulmasında aranan bir diğer hususda programcının kullanıcının belirtmediği fakat sistemin ihtiyaç duyduğu diğer gereksinimleride tespit ederek (ŞEKİL-3) bunlara çözüm yaratması beklentisidir. Yazılım kalitesi buna paralel olarak artacaktır. Mesela sistemin gereksinimleri tespit edilirken bakım, güncelleme veya kullanım sonrası değişiklik yapma imkanlarından bahsedilmeyebilir. Kullanıcı tüm bunlardan o an için habersiz olabilir. Ancak üretilen ürün tüm ihtiyaçlara cevap verebilecek dökümantasyon ve sistem yapısına sahip olmalıdır ki kaliteden söz edilebilsin. Aksi taktirde yazılım kalitesi kesinlikle sağlanamaz. UML in bu alanda da yazılıma katkıları oldukça büyüktür. UML sistemin oluşturulmasındaki her aşamada dökümantasyonu mecbur kılar. Böylece yapılan tüm işlemler belirli bir notasyon ile yani herkesin anlayabileceği bir dil ile kayıt altına alınmış ve istendiğinde kronolojik olarak sunulabilen bir yapı şeklindedir. Aynı zamanda kullanıcının belirttiği gereksinimlerin yaratmış olduğu fakat belirtilmeyen gereksinimlerin (Örneğin Network ortamına uyumlu yazılım üretilmesi ya da veri tabanı kullanımı gereksinimi gibi...) daha kolaylıkla görülmesi sağlanabilir.

KAYNAKLAR

[1] Software Engineerıng Theory Nad Practice Shari Lawrence Pfleeger 1998 Prentıce Hall Inc.

[2] Software Qualıty Producıng Practıcal, Consıstent Software Mordechaı Ben-Menachem And Garry S. Marlıss Internatıonal Thomson Computer Press

[3] V. R. Basili et.al. “A Validation of . . . Quality Indicators”, IEEE Transactions on Software Engineering, Oct 1996, sayfalar 751-761

[4] T. M. Khosgoftaar et. al. “Early QualityPrediction: . . .” IEEE Software, Jan 1996 sayfa 65-71

[5] Software Quality Journal dergileri

[6] J. Staunstrup, “Practical Verification of Embedded Software” IEEE Computer, May 2000, sayfalar 68-75

[7] T. Bull ve J. Larus, “Using Paths to Measure, Explain and Enhance Program Behaviour”, IEEE Computer, July 2000, sayfalar, 57-65

[8] F. Shull et.al. “How PBR Can Improve Requirements Inspections”, IEEE Computer, July 2000, sayfalar 73-79

[9] Ali Arifoğlu, Ali Doğru “Yazılım Mühendisliği”, SAS Bilişim Yayınları

[10] Rational Rose Enterprise Edition for Windows 2002.05.20.468.000, User Manuel Rose Help Terminology, Rational Rose Release Notes

[11] Microsoft Visio 2000 SR1 6.0.2072 User Manuel

[12] Poseidon For Uml 1.6.1_01 Users Guide, Code Generation Documantation

[13] “Mastering UML With Rational Rose 2002 “Wendy BOGGS, Michael BOGGS Fully Updated Cover To Rose 2002, as well as 2001 and 2001A, SYBEX Publishing

[14] UML (Unified Modelling Language) Course Book -Air Force Institute Of Technologies Electrical And Computer Engineering 2003

[15] Marc Lorens, “Object Oriented Software” A prictical Quide, Prantice Hall 1993

[16] BİLDEM IBM Rational Rose 2003 Semineri , Conrad Otel İstanbul

[17] BİLDEM IBM Rational Rose Eğitim Seti

[18] IBM Rational Rose Enterprice Edition Tanıtım CD leri

[19] UML Distilled: A Brief Guide to the Standard Object Modeling Language (2nd Edition) -- by Martin Fowler, Kendall Scott; Paperback

ÖZGEÇMİŞLER

Sefer KURNAZ

1978 Yılında Hava HarpOkulu Elektronik Mühendisliği Bölümünden lisans, Ege Üniversitesi Bilgisayar Mühendisliği Bölümünden yüksek lisans, İstanbul Üniversitesi Bilgisayar Mühendisliği Bölümünden doktora derecesi aldı. Halen Hava Harp Okulu Komutanlığı Havacılık ve Uzay Teknolojileri Enstitüsü Müdürü olarak görev yapmaktadır.

Ömer ÇETİN

1999 Yılından İzmir Maltepe Askeri Lisesini bitirmesine mütakip Hava Harp Okulu Bilgisayar Mühendisliği Lisans Eğitimine başlamış olup, halen Hava Harp Okulu Bilgisayar mühendisliği Lisans seviyesi 4ncü sınıf öğrncisidir.

Fuat İNCE

Akademik Unvanı : Prof. Dr.
Aldığı Dereceler : BS,Elektrik Mühendisliğinde, Boğaziçi Üniversitesi, 1968
MS, Elektrik Mühendisliğinde, Illinois Üniversitesi, 1969
PhD, Elektrik Mühendisliğinde, Illinois Üniversitesi, 1973
İlgi Alanları : Yazılım Mühendisliği, Bilgi Teknolojileri, Görntü İşleme, Uzay Teknolojileri
Verdiği Dersler : (HUTEN'de)

- Yazılım Mühendisliği İleri Konular
- Uzay Teknolojilerine Giriş

Deneyimler :

- Halen Maltepe Üniversitesi, İstanbul, Bilgisayar Mühendisliği Bölüm Başkanı.
- Bilgi Teknolojileri ve Araştırma Enstitüsü Kurucu Başkanı (erken emeklilik nedeniyle görevden ayrıldığında, 120 çalışanıyla enstitü, milyonlarca dolarlık birçok uluslararası araştırma projesini yürütüp başarıyla tamamlamıştı).
- NATO Bilgi Teknolojileri Paneli üyesi (1998-2001).
- Birçok ulusal ve uluslararası komite görevlerinde bulunmuştur.
- Birçok profesyonel topluma üye.