İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIMDA KOD GÖZDEN GEÇİRME SÜRECİNDE KOD KALİTESİ ÖLÇÜMÜNÜN SÜRECE VE YAZILIM KALİTESİNE ETKİSİNİN İNCELENMESİ

YÜKSEK LİSANS TEZİ

Mustafa Turgay Alptekin

Anabilim Dalı : ENDÜSTRİ MÜHENDİSLİĞİ

Programı : MÜHENDİSLİK YÖNETİMİ

EYLÜL 2008

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIMDA KOD GÖZDEN GEÇİRME SÜRECİNDE KOD KALİTESİ ÖLÇÜMÜNÜN SÜRECE VE YAZILIM KALİTESİNE ETKİSİNİN İNCELENMESİ

YÜKSEK LİSANS TEZİ

Mustafa Turgay Alptekin
507051218

Tezin Enstitüye Verildiği Tarih : 5 Ağustos 2008
Tezin Savunulduğu Tarih : 2 Eylül 2008

Tez Danışmanı: Doç.Dr. Cengiz GÜNGÖR
Diğer Jüri Üyeleri Doç.Dr. Mehmet Mutlu YENİSEY
Doç.Dr. Ferhan ÇEBİ

EYLÜL 2008

ÖNSÖZ

Tez çalışmamda emeği geçen başta danışmanım Doç. Dr. Cengiz GÜNGÖR olmak üzere, tez sırasında desteğini esirgemeyen aileme, çalışmayı gerçekleştirdiğim Veripark şirketindeki yöneticilerime ve çalışma arkadaşlarıma teşekkür etmeyi bir borç bilirim.

EYLÜL,2008 MUSTAFA TURGAY ALPTEKİN

İÇİNDEKİLER

KISALTMALAR

CMMI	: Capability Maturity Model Integration
CMM	: Capability Maturity Model
IEEE	: The Institute of Electrical and Electronics Engineers, Inc
ISO	: International Standards Organization
CASE	: Computer aided software engineering
NOC	: Number Of Children
DIT	: Depth of Inheritance
CBO	: Coupling by Object
CC	: Cyclomatic Complexity
LCOM	: Lack of Cohesion in Methods
Ce	: Efferent Coupling
Ca	: Afferent Coupling
WMC	: Weighted Methods Per Class
ROI	: Return On Investment
SEI	: Software Engineering Institute
FURPS	: Functionality, Usability, Reliability, Performance, Supportability
CK	: Chidamber&Kemerer

TABLO LİSTESİ

ŞEKİL LİSTESİ

Sayfa No

Yazılımda Kod Gözden Geçirme Sürecinde Kod Kalitesi Ölçümünün Sürece Ve Yazılım Kalitesine Etkisinin İncelenmesi

ÖZET

Yazılım sistemleri modern dünyada bir çok iş alanında önemli roller oynamaktadır. Alınan her üründe olduğu gibi yazılımın kalitesinin iyi olması da yazılımı alacak kişi ve kurumlar için önemli bir faktördür. Fakat yazılım üretimi diğer alanlara kıyasla oldukça farklı özellikler göstermektedir. Yazılımın kalitesinin değerlendirilmesi, yazılımın sanal olması, teknolojilerin hızla değişmesi ve standartların uygulamasının zor olması sebebiyle kolay değildir.

Kaliteli yazılımlar; kabul edilebilir düzeyde hatasız, planlanan bütçe ile zamanında bitirilip dağıtılabilen, gereksinimleri veya beklentileri karşılayabilen ve sürdürülebilir özelliklere sahip yazılımlar olarak değerlendirilebilir. Tüm hataları oluşmadan önlemek, proje koşulları ve maliyetleri içerisinde olanaklı değildir. Yazılımda ürünün kalitesini, ağırlıkla onu üreten sürecin kalitesi belirlemektedir.

Bu bakımdan isinde yazılımın tasarlanmasından, geliştirmesine ve test edilmesine kadar geçen süreç içerisinde bir takım standartların belirlenmesi ve uyulması gerekmektedir. Standartların yazılım kodunun kalite kontrolünü de içermesi gerekir.

Kod kalitesinin takibi için kod gözden geçirmeleri yapılmaktadır. Bir çok araştırmalarda kod gözden geçirmeleri sayesinde yazılım kalitesinde artışlar meydana geldiği kanıtlanmıştır.

Kod gözden geçirmeleri sırasında kodun neye göre inceleneceği çok önemlidir. Yanlış değerlendirmeler maliyet ve yazılım kalitesindeki düşüş olarak yazılım şirketlerine geri döner. Bu yüzden kod gözden geçirmelerinde yazılım ölçütlerini kullanmak yazılım kalitesi yönetiminde subjektif kod gözden geçirmelerine objektif bir perspektif kazandırır.

Çalışmanın amacı, orta çaplı bir yazılım firmasında, hali hazırda subjektif bir biçimde uygulanan kod gözden geçirme sürecini, bazı yazılım ölçütlerinin ölçülmesiyle desteklenerek kod kalitesinde artış sağlanması ve çalışanların kod gözden geçirme sürecine alakalarının arttırılmasıdır.

Çalışma sonucunda yazılım ölçütlerinin ölçülmesinin hem kod kalitesine hem de yazılım geliştiricilerin bunlara olan ilgisine pozitif yönde etki ettiğini söyleyebiliriz.

Examining the effect of code quality measurement to code review process and software quality

SUMMARY

Software systems have important roles through various fields in different business areas. The quality of software product should be high for customers like in any other type of product.

However, manufacturing of software have different attributes than other production areas. Its virtuality, rapidly changing technology and hard adaptation of standards, hardening evaluation of software quality. Error free, on time finished, able to met requirements and expectations and continious software can be defined as high quality software.

Avoiding all the errors before occurance can not be possible when considering project conditions and expenses. Mostly, product quality is affected by the processes. Therefore, there should be defined standards, which is obeyed in software design, development and test stages. These standards should include software code quality control in order to provide high software quality.

Evaluation criteria of code reviews is crucial. Incorrect assessments have devastating consequences in expenses, customer satisfaction. Software metrics provides objective perspective to subjective code review process, should be used for reliable code review. They have been proved to reflect software quality, thus have been widely used in software quality evaluation methods.

The aim of this work is to improve code review process in medium sized software company by supporting software metrics in recent subjective code review process and follow up increase of quality in software code and progressive concern in developers.

At the end of this work, the results are showing that using software metrics in code evaluation is affecting code quality and the concern for code review in a positive way.

1. GİRİŞ

1.1 Giriş ve Çalışmanın Amacı

Yazılımlar çeşitli süreçlerde kullanılmak üzere her sektörden şirketler için bir rekabet aracı olarak kullanılmaktadır. Bu durumda yazılım şirketleri müşterilerine rekabette üstünlük sağlayacak olan kaliteli ürünleri geliştirmeli ve bunları sunmalıdır. Kaliteli ürünü sağlamak için yapılması gereken önemli bir işlem, kaynak kodunun kalitesinin takip edilmesi ve arttırılmasıdır.

Çalışmada orta halli bir şirket için kod kalitesinin takibi sürecinin yazılım ölçütleriyle iyileştirilmesi ve bunun hem kod kalitesine, hem de yazılım geliştiriciler açısından süreçle ilgilenme seviyesine verdiği etkiler incelenmiştir.

Yazılım Kalitesi Kavramları kısmında kod kalite takibinin, yazılım ölçütlerinin, ve kullanılan diğer kavramların kaliteyle, kalite standartlarıyla ve kalite alanında ortaya sunulmuş çalışmalarla ne gibi ilgisi olduğu incelendi.

Bu aşamayı takiben uygulamanın yapılacağı organizasyon tanıtıldı, kullanılan yöntemler belirtildi.

Çalışmanın son bölümünde ise çıkan sonuçlar verilip yorumları girildi.

2. YAZILIM KALİTESİ KAVRAMLARI

2.1 Kalite

Kalite kavramı insanların ve sistemlerin "hata yapması" ve "mükemmele ulaşma isteği" gerçeğinden ortaya çıkmıştır. Latince bir şeyin nasıl oluştuğu anlamına gelen "Qualis" kelimesinden türemiş ve "Qualitas" kelimesiyle ifade edilmiştir.

Kalite'nin değişik tanımları bulunmaktadır:

- Kalite; belirlenen şartlar altında ve belirlenen bir zaman süresi içinde istenilen fonksiyonları yerine getirebilme kabiliyetidir.
- Kalite, bir ürünün kulanım uygunluğunu belirleyen özelliklerinin tümüdür.
- Kalite, herhangi bir ürün sınıfının özelliklerinin insan topluluklarının istek potansiyelini karşılayabilme derecesidir.
- . Kalite, önceden tespit edilmiş olan spesifikasyonlara ya da standartlara göre üretim yapma olgusudur.

Alıcı tarafından aranılan belirli şartları en iyi karşılayan anlamında kullanılan "Kalite" kısaca "amaçlara uygunluk derecesi" olarak tanımlanabilmektedir. Buradaki amaç kullanıcı kimsenin veya tüketicinin istek ve gereksinimleri olmaktadır. Kalite sınırları devamlı genişleyen bir kavramdır. Teknoloji, değişen koşullar, ihtiyaçlar kaliteye değişik boyutlar getirmektedir.

Kalite niteliği bakımından dinamik bir özellik taşımakta, tüketici ihtiyaçlarına paralel olarak gelişmekte ve değişmektedir. Veri toplamak suretiyle üretici, yeni teknikler ve yeni örgütlenme yolları geliştirerek aynı maliyetle daha yüksek kalitede üretmek ve tüketicinin kaliteye yönelik taleplerini yerine getirmek durumundadır. Üreticilerin birçoğu için düşük kalitenin karlılık üzerine olumsuz etki yapması gerçeği ortadadır.

Düşük kalite, imalatçı için hataları bulma ve düzeltmedeki maliyet demektir. Bazen bu maliyetler büyük boyutlara ulaşabilmektedir. Ayrıca düşük kalitenin alıcılardaki güven kaybından dolayı ürünün piyasa payının azalmasına neden olacağı da açıktır. Bir malın kalitesi, Kalite Parametreleri olarak nitelenebilen unsurlardan

oluşmaktadır. Bu unsurlar malın çeşidine göre değişmektedir. Mekanik ve elektronik mallarda performans, güvenlik ve görünümle ilgili olabilirken, kimyasal ürünler için fiziksel ve kimyevi özellikler, tıbbi etki, zehirlilik, tat gibi parametreler önemli olabilmektedir.

Tanımlar ve açıklamalardan yola çıkarak kalitenin esas olarak başlıca iki faktörü içerdiği görülmektedir.

1) Objektif özellikler
2) Subjektif özellikler

Objektif özellikler insan unsurunun dışında kalan özelliklerdir.

Subjektif özellikler ise objektif özellikleri görmekten hissetmekten ve düşünmekten kaynaklanan özelliklerdir.

Kalitenin subjektif özellikleri ne ölçüde objektif hale getirilebiliniyor ise o ölçüde kontrol etme olanağı ortaya çıkmaktadır. Standardizasyon objektif ölçüler esasına göre çalışan bir yöntemdir. Diğer bir anlamda kalite dediğimiz kavramın iskeletini teşkil etmektedir. Objektif esaslara göre yapılan bu iskelete subjektif bazı özelliklerin de ilave edilmesiyle kalite kavramı ortaya çıkmaktadır. Standarda uygun mal asgari kaliteye sahip demektir. Fakat her standart mal mutlaka en yüksek kalitede mal demek değildir. Bununla beraber kaliteli malda standarda uygun anlamına gelmemektedir. Kalite bir güven duygusu yaratmaktadır. Güven kriterinin oluşumu kalitenin değerlendirilebilme imkanına bağlıdır.

Değerlendirilebilme ise ölçülebilme imkanına bağlıdır. Ölçülebilme kalite ile ilgili ölçüm çalışmaları kapsamına girmekte ve çeşitli test ve incelemeleri içermektedir. Madde veya mamulün kalitesinin, imalatçıları tarafından kullanıcı veya tüketiciye güvenilir ve tarafsız kuruluşlarca verilmiş bazı belgelerle garantiye alınması da önem taşımaktadır. Belge bir güven aracı olacağından kalitesi belgelenmiş bir mamulün tercih edileceği de açıktır.

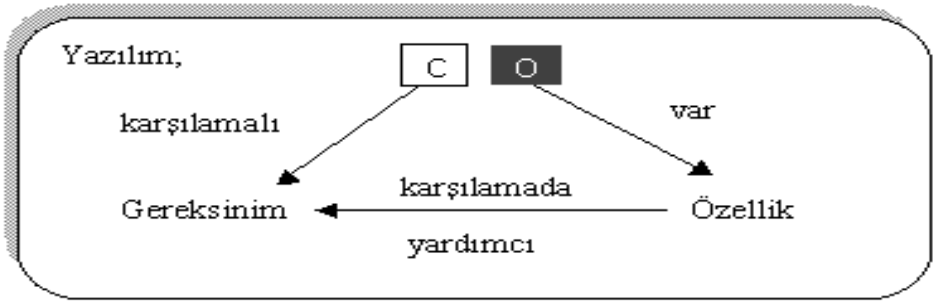
Kısaca ifade edilecek olursa, standartlar, kalite kontrolü, belgelendirme ve ölçüm çalışmalarının düzenlenmesinin kaliteyi artırma yönünde etkili araçlar oldukları ortaya çıkmaktadır.

2.2 Yazılımda Kalite

Yazılım mühendisliğinde yazılım kalitesi farklı tanımlamalar yapılmasına rağmen, bir yazılımın ne kadar iyi tasarlandığını (tasarım kalitesi) ve bu tasarıma ne kadar uyduğunu (uygunluk kalitesi) gösterir [1].

Uygunluk kalitesi yazılımın yürürlüğe koyulması ile ilgilenirken, tasarım kalitesi, kayda değer yazılım ürününün yaratılması için kullanılan tasarımın ve gereksinimlerin geçerliliğini gösterir [2].

Şekil 2-1 Özellikleriyle gereksinimleri karşılayan bir yazılım ürününü göstermektedir. Gereksinimler ve özellikler arasındaki ilişkinin incelenmesi yazılım kalitesini görmemizi sağlar.



Şekil 2.1 : Yazılım Gereksinim Özellik İlişkisi [3]

Yazılım kalitesi gereksinimleri karşılayacak özelliklerin varolmasıdır. Buna ek olarak, gereksinimlerle alakalı olmayan, yazılım kalitesini düşürücü nitelikte özelliklere de bakmamız gerekir. Gereksinimleri karşılayan özelliklerin var olmasının yanında verimliliği engelleyen özelliklerin de olmaması yazılım kalitesi sağlanması adına önemlidir.

Yazılım modern uygarlıkta tüm alanlara yayılmıştır. Sanal olarak tüm işler faturalama, ödeme, stok kontrolü, satın alma, süreç kontrolü iş tahmini ve bir çok kritik fonksiyonda yazılımlardan faydalanırlar. Amerika'da kişisel bilgisayarar ev giderlerinin yönetimi, vergilerin saklanması ve internet servislerine erişim için çok büyük oranda kullanımdadırlar. Bu görünür bilgisayar sistemleri aslında, otomobil sektöründen tutun aksesuar sektörüne kadar kullanılan içsel görünmez sistemler karşısında sayıca ezilirler. Bu sistemlerin kötü performans göstermeleri müşteri memnuniyetsizliği, müşteri kaybı, üretkenlik azalması ve hatta yaşam kaybına yol açabilir. Bu sistemlerde yer alan yazılımların kalitesinin kontrolü çok önemlidir.

2.2.1 Yazılımla ilgili terimler

Yazılım Ürünü**:** Müşteriye veya son kullanıcıya verilen, bilgisayar programları, prosedürleri ve bunlara bağlı dokümantasyondan oluşan tamamlanmış birimler kümesidir (IEEE 610.12-1990).

Hata**:** Aşağıdaki dört tanımın oluşturduğu kümedir.

Yanlış: Ölçülen, gözlenilen, kabul edilen ve hesap edilen durumlar arasında farklılık gözlenmesi.

Kusur: Doğru olmayan veri tanımlaması veya süreçte yanlış çalışan bir adım.

Arıza: Belirlenen performans kriterleri içerisinde istenen fonksiyonelitenin sağlanamaması.

Yanılgı: İnsan tarafından başlatılan bir süreçte oluşan yanlış sonuç (IEEE 610.12-1990).

Ölçüm: Bir şeyi ölçmek için gereken süreç veya eylem.

Ölçüt: Bir sistemin, öğenin veya sürecin verilen bir özelliği derecelendirmesinde kullandığı ölçüm birimi (IEEE 610.12-1990).

Nesne**:** Bir sınıftan oluşturulmuş, o sınıfın özelliklerini taşıyan yapı. Öğrenci bir sınıf ise Mahmut Aktaş adlı öğrenci bu sınıftan türemiş bir öğrenci nesnesi olabilir.

Sınıf: Objelerin şemalarının verildiği yapıdır. İçerisinde davranışı belirleyen metodlar ve veri alanları bulundurur.

Bağlama: Bir varlıktan diğer varlığa olan bağlantıya kurulan kuvvetli ilişki. Sınıflar ve böylece nesneler aşağıdaki koşullar oluştuğunda birbirlerine bağlanırlar.

· Nesneler arasında bir mesaj iletimi olduğunda.
· Bir sınıfta diğer sınıfın metotlarını veya değişkenlerini kullanan bir metot tanımlanmışsa.
· Bir sınıf başka bir sınıftan kalıtım yoluyla türetilmişse.

Kalıtım: Nesneye dayalı sistemlerde soyutlama tasarımı kalıtımın kullanılmasıyla gerçekleşir. Kalıtım, yazılımcıların değişkenleri ve operatörleri de içeren, daha önceden tanımladıkları nesneleri kullanmalarına yarayan ilişki tipidir. Kalıtımın doğru kullanılması karmaşıklığı ve tekrarı azaltır.

Birleştirilmiş Modelleme Dili (UML) : Nesne modelleme belirtimi için kullanılan görsel dil.

Microsoft.NET: Microsoft tarafından geliştirilen, açık İnternet protokolleri ve standartları üzerine kurulmuş komple bir uygulama geliştirme platformudur. Taşınabilir araçlar, Internet ve istemci taraflı yazılımlar ortak olan bu platformda bulunurlar.

2.2.2 Bilgisayar destekli yazılım mühendisliği

Bilgisayar destekli yazılım mühendisliği (CASE) yazılımın geliştirmenin tüm fazlarında yazılım araçlarından yararlanmaya dayalı disiplindir. Burada kullanılan yazılım araçlarına CASE araçları denir. Bazı CASE araçları aşağıdaki gibidir.

- Kod yaratma araçları
- Veri modelleme araçları
- Birleştirilmiş Modelleme Dili (UML)
- Kod takibi, yeniden düzenleme araçları
- Yapılandırma yönetimi araçları

CASE araçları gereksinim yönetimi, dokümantasyon kontrolü, proje ve sistem doğrulama ve süreç yönetimi başta olmak üzere, yazılım geliştirme takımları tarafından artarak kullanılmaya devam etmektedir. Bu tip araçların kullanımının üretkenliğin arttırdığı bir çok organizasyon tarafından raporlanmıştır. Ancak, esas beklenen kazanım kalite konusunda olmalıdır. Çünkü hatalar ve tutarsızlıklar yaşam döngüsünün ilk safhalarında yakalanıp müşterinin isteklerine göre uygun bir şekilde düzenlenebilir. Bir sistemin güvenilirliği bu araçların daha yürürlüğe konmadan kullanılmasıyla sağlanabilir. Değişik fazlar için, kafa karıştıran bir çok CASE aracı ticari olarak mevcuttur. Bu araçlarda uygulamadaki ihtiyacı karşılayacak olan doğru olanı bulmak zor bir görevdir [4].

CASE araçlarına örnek vermek gerekirse,

- Razor (Visible şirketi tarafından geliştirilmiş, süreç yönetimi, sorun takibi, veri modelleme konularında hizmet veriyor)
- Visual Paradigm (UML aracı, otomatik kod oluşturma, kodların geri dönüşümü gibi fonksiyonları mevcut)

- Sure Step (Microsoft'un kendi ürünlerinin ERP projelerindeki tüm fazları için önerdiği metodoloji aracı)
- NDepend (Kod kalitesi takibi ve erken uyarı aracı)

2.2.3 Yazılım Kalite Standartları

Yazılımda kalite olgusu kendine has özellikler ve ayrı bir odaklanma gerektirmekle birlikte; kuruluşta yürütülen tüm kaliteye ilişkin faaliyetlerle bütünsel olarak ele alınmayı gerektirir. Kuruluşta kaliteye ilişkin çalışmalar pazarın gereklerini karşılayacak şekilde, zaman içerisinde, sürekli geliştirilmek zorundadır. Yazılım kalite stratejilerinin belirlenmesi geçmiş deneyimler ışığında gelecekteki değişim hakkında yapılan uz görülere dayalı olarak belirlenmelidir.

1980.li yıllarda yönetim sistemlerinde önemli değişiklikler olmaya başlamıştır. Kalite yönetim sistemleri Toplam Kalite Yönetimi felsefesine dayalı olarak önemli evrim geçirmiştir. Bu dönemlerde, bilişim teknolojilerinin kuruluşlarda katma değeri artmaya başlamış, yazılımın kendine has yapısı kaliteli yazılımın geliştirilmesi konusu üzerinde önemle durulması gereksinimini açıkça ortaya çıkarmıştır. Bilişim teknolojilerinde kalite aşağıdaki özellikler nedeni ile ayrıca ele alınmayı gerektirir:

- Bilişim teknolojilerinin çok hızlı gelişmesi
- Elle tutulamayan yapı
- Üretim sürecinin olmaması (kopyalama)
- Mühendislik/geliştirme aşamasının önem kazanması
- Yazılım gereklerinin belirlenmesindeki güçlükler
- Yazılımda nitelikli insan gücü gereksiniminin daha fazla olması
- Ölçme ve değerlendirmedeki güçlükler
- Yazılım güvenilirliğinin donanım güvenilirliğine göre farklılıklar içermesi
- Maliyet yapısındaki değişiklikler (geliştirilen bir yazılımın çoğaltılmasının bir maliyetinin olmaması, maliyetin sabit olmaması, vb.)
- Yazılım güvenliliğinin kuruluşlar için çok önemli olması
- Entellektüel haklar

Yazılımda kalitede Toplam Kalite Yönetiminin temel ilkelerinin hepsi geçerlidir. TKY ve ISO 9001:2000.in en önemli temel ilkelerinden biri olan süreç yaklaşımı

yazılımın elle tutulamayan yapısı nedeniyle daha fazla önem kazanmıştır. Bu nedenle, kuruluş düzeyinde kurulan bir kalite yönetim sistemine, yazılımda kalite süreçlerini mercek altına alacak ek yazılım yaşam çevrimi süreç modellerine ve standartlarına gereksinim duyulur.

Yazılımda kalite yönetim sistemi kuruluşun yapısına ve bağlı olduğu sektöre göre farklılıklar gösterebilir. Ancak etkin yazılımda kalite yönetim sistemi için kuruluş düzeyinde bir kalite yönetim sisteminin olması önemlidir.

2.2.3.1 CMMI

CMM ilk olarak askeri bir araştırmayla desteklenip bulunmuştur. Amerika Birleşik Devletleri Hava güçleri, Carnegie-Mellon Yazılım Mühendisliği Enstitüsünündeki bir çalışmayı finanse etmiştir. Bu çalışmayla amaç yazılım projelerinin ihalesinde yazılım şirketlerinin değerlendirmesi için bir takım kriterler belirlemekti. Sonuç olarak CMM 1989 yılında yazılım süreci yönetimi alanında yayınlanmıştır. Eski olan çalışma (CMM), yazılım mühendisliği enstitüsü tarafından desteklenmemekle beraber, daha geliştirilmiş olan versiyon olan CMMI (Capability Maturity Model Integration) günümüzde 1.2 versiyonuyla yazılım dünyası tarafından geniş çapta kullanılmaktadır.

CMMI'nin kelime anlamı "Yeterlilik Olgunluk Modeli Entegrasyonudur". CMMI , kurumlara süreçlerini iyileştirme konusunda gereken temel adımları gösteren bir süreç iyileştirme yaklaşımıdır. Ne yapılması gerektiğini açıklar. Nasıl sorusuna kesin bir cevap vermez. Bu yaklaşım bir projeye, bir kuruma ait departmana ya da büyük bir organizasyona süreçlerini iyileştirme ve geliştirme konusunda rehberlik eder. Olgun olmayan bir süreçten olgun ve disiplinli bir surece giden evrimsel bir yol çizer. Ortaya konulan seviyelere göre olgunluk gelişimi sağlanır. Her olgunluk modelinin kendine özel süreçleri bulunur. Böylece sürekli bir iyileşme söz konusu olur. CMMI, geleneksel yapıda ayrı olan organizasyonel fonksiyonların entegre edilmesine, süreç iyileştirme için gerekli hedef ve önceliklerinin belirlenmesine, kalite süreçleri için kılavuz oluşturulmasına ve mevcut süreçlerin değerlendirilmesi için bir referans noktası oluşturulmasına yardım eder.

CMMI, dünyada ve Türkiye'de daha çok IT sektöründe ve özellikle yazılım geliştirme yapılan kurum, departman ve projelerde, yazılım kalitesini arttırmak için kullanılan bir referans model olarak uygulanmaktadır. Donanım ve network

alanlarında da uygulanan model, özellikle savunma sanayi başta olmak üzere; Ar-Ge ve yeni ürün geliştirme konusunda hizmet veren kurumların aradıkları bir standarttır.

CMMI Seviyeleri

Seviye 1: Başlangıç

Yazılım süreci gelişi güzel ve kaotiktir. Başarı kişisel gayretlere bağlıdır.

Kriz durumunda varolan planlar terk edilir. Yazılım süreç yetenekleri önceden kestirilemez

Seviye 2: Tekrarlanabilen

Proje yönetimi kontrolleri kuruludur, projeler dokümante edilen planlarına uygun şekilde gerçekleştirilir ve yönetilir. Yazılım gereksinimleri yönetilir, gereksinimler ile ilişkili ürünler oluşturulur ve denetlenir.

Proje yönetim sistemi proje surecini etkin bir şekilde kontrol eder, projeler planlı, başarılmış, ölçülmüş ve kontrol edilmiştir. Gereksinimler, süreçler, çıktı iş ürünleri ve hizmetler yönetilir. İş ürünlerinin ve sağlanacak hizmetlerin durumu tanımlandığı noktada yönetimin görüsüne açıktır.

Taahhütler, ilgili paydaşların istek ve revizyonlarına uygun şekilde tesis edilmiştir. İş ürünleri taraflarla gözden geçirilir ve kontrol edilir. İş ürünü ve hizmetler; onlara özel gereksinim, standart ve hedefleri sağlar.

Seviye 3: Tanımlı

Süreçler iyi tanımlanmış ve iyi anlaşılmıştır, standartlar, prosedürler, araçlar ve metotlar ile anlatılmıştır. Standart süreç tanımları ile organizasyonel boyutta tutarlılık sağlanır.

Projeler standart sureci uyarlama rehberlerine uygun olarak uyarlayarak, kendi süreçlerini oluşturur. Tüm projelerde standart yazılım geliştirme sürecinin ayarlanmış hali kullanılır. Süreçler İkinci seviyeye göre daha özenli ve detaylı olarak tanımlanmıştır.

Seviye 4: Yönetilen

Sürecin başarımı için önemli alt süreçler seçilir ve bu alt süreçler istatistiksel ve diğer nicel ölçüm teknikleri kullanılarak kontrol edilir. (Yazılım sürecinin ve ürün kalitesinin ölçümleri toplanır ve kullanılır).

Kalite ve süreç performansının nicel hedefleri tesisi edilir ve süreçlerin yönetiminde bir kriter olarak kullanılır. Nicel hedefler müşterinin ihtiyaçları/istekleri, son

kullanıcı, organizasyon ve süreçleri implemente edenler baz alınarak belirlenir. Süreçler ölçülür ve ölçülebilen sınırlar içinde isler gerçek veriler baz alınarak tahmin edilebilir.

Üçüncü seviye ile temel farklılık süreç performansının öngörülebilmesidir. Dördüncü seviyede süreçlerin performansı istatistiksel ve diğer nicel teknikler ile kontrol edilebilir ve nicel olarak öngörülebilir.

Seviye 5: İyileştirilen

Süreçten gelen sayısal geri beslemeler ve teknolojilerden yararlanılarak sürekli süreç iyileştirilmesi yapılır. Tüm organizasyon süreç iyileştirmeye odaklanmıştır.

Yazılım süreç yeteneği sürekli iyileşen olarak tanımlanabilir.

5.ve 4.seviye arasındaki önemli bir ayrım, adreslenen süreçlerin değişme derecesidir. 4.seviyede süreçler, süreç değişkenliğinin özel nedenleri ve sonuçların istatistiksel öngörüsüne odaklanmıştır. 5.seviyede ise süreçler, süreç değişkenliğinin genel nedenleri ve surecin değişimini (süreç başarımından istatistiksel öngörüye) adreslemekle ilgilenir.

SEI'in Aralık-2005 rakamlarına göre 25 büyük organizasyondan elde ettiği faydaların listesi şöyledir:

Tablo 2-1 : SEI Aralık 2005 Verim Raporu [18]

Performans Kategorisi	Ortalama	Veri kaynağı sayısı	En az	En çok
Maliyet	20%	21	3%	87%
Proje Takvimi	37%	19	2%	90%
Üretkenlik	62%	17	9%	255%
Kalite	50%	20	7%	132%
Müşteri Tatmini	14%	6	-4%	55%
ROI (Geri dönen fayda)	4.7: 1	16	2: 1	27.7: 1

2.2.3.2 ISO 9000-3:2004

Neredeyse tüm iş sektörlerinde yüksek bilişim teknolojilerinin kullanımı, ISO 9001'deki kalite standardının yazılım mühendisliğindeki kullanım potansiyelini ortaya çıkarmıştır.

ISO/IEC 90003:2004, ISO 9001:2000'in bilgisayar dünyasına nasıl uyarlanacağıyla ilgili yönergeler içermektedir. İçerisinde yazlım geliştirmeden, operasyona ve bakıma kadar geniş kapsamdaki konuları barındırır.

ISO/IEC 90003 gereksinimler ve yönergelerden oluşmaktadır. Gereksinimler direk olarak ISO 9001 standardından gelmiş olup standarda uyum sağlanabilmesi için gerçeklenmesi gerekenleri belirler. Yönergeler ise ISO 9001 deki gereksinimlerin bilgisayar dünyasına uyarlanmasıyla ilgilidir.Yönergeler kendi arasında ikiye ayrılır: tavsiyeler ve öneriler. Tavsiyeler kesinlikle gerçeklenmeli, önerilerin gerçeklenmesi ise zaruri değildir.

Temel olarak özellik yönetimi, değişim kontrolü, geliştirme planlaması, kalite planlaması, tasarım ve uygulama, test ve doğrulama ve bakım konularıyla ilgilenir.

Özellik Yönetimi

Amacı, ürünün tüm parçalarının doğru kaynak kodlarından oluşturulmasının sağlanmasıdır. Yazılım parçalarının (dokümantasyonlar dahil) hangi projede nasıl kullanıldığının yönetilmesi gerekmektedir.

Değişim Kontrolü

Yazılım geliştirme döngüsü içerisinde oluşabilecek değişikliklerin önceden ön görülmesi,ve engellenmesi, mecburi değişikliklerin ise maliyetsiz ve sorunsuz bir biçimde gerçekleştirilmesini sağlayan yapının yönetilmesi değişim kontrolüne girer. Değişiklik önerilerinin kabulü, alınacak aksiyonların belirli olması, öncelik verilmesi gerekmektedir.

Geliştirme Planlaması

Geliştirme planı bir projenin başından sonuna kadar gerekli tüm aktivitelerin tek bir pencereden görülmesini sağlar. Geliştirme içerisindeki her fazda girdilerin çıktıların takibi ve onaylanması mümkün kılınır. İş atamaları, görevlerin programlanması, süreç izleme bir geliştirme planında olması gereken adımlardır.

Kalite Planlama

Kalite planlaması bir projenin daha başlamadan kalite hedeflerine karar verilmesini sağlar. Projenin müşteriye verilmeden istenen kalite seviyesine ulaşıp ulaşmadığını kontrol edecek test ve doğrulama yapılarının kurulmasını içerir.

Tasarım ve Uygulama

Tüm projelerin tasarlanmasında ortak dilin kullanılması ve organizasyondaki tüm projelerde tutarlılığın sağlanması tasarım ve uygulama kapsamına girer. Sistematik bir tasarım sürecinin tüm projelerde kullanılması elzemdir. Projenin gereksinimlerinin karşılanıp karşılanmadığı tasarım gözden geçirmeleriyle takip edilmelidir.

Test ve Doğrulama

Bir test planı, test tiplerini ve seviyelerini belirlemekle kalmaz, test için gerekli kaynak kullanımını girdileri çıktıları programlar. Test ortamında kullanılacak araçlar, test koşulları ve beklenen sonuçlar, belirgin olmalı, yazılı bir şekilde bulundurulmalıdır.

Bakım

Bakım safhası genellikle tasarım ve geliştirmeden daha uzun süreceği için dikkatli planlanmış aktiviteler zincirini içermelidir. Bir bakım planı, destekleyen kaynakları, plan içerisindeki aktiviteleri , bakımla ilgili olan raporları, kayıtları ve müşteri güncellemelerinin nasıl dağıtılması ve kurulumu ile ilgili bilgileri barındırmalıdır. Bakım planı içerisinde sorunun hangi modifikasyonlarla nasıl çözüldüğüne dair bilgiler bulunmalıdır.

2.2.3.3 IEEE Standartları

IEEE yazılımda kaliteyi sağlamak adına ayrı ayrı bazı standartlar çıkarmıştır. Bunlardan bazıları :

• IEEE. 1220-1998: IEEE Standard for Application and Management of the Systems Engineering Process (Sistemlerin mühendislik sürecinde uygulama ve yönetim standartı)

• IEEE 730-1998: IEEE Standard for Software Quality Assurance Plans (Yazılım kalitesini teminat altına alan planların standartları)

• IEEE 828-1998: IEEE Standard for Software Configuration Management Plans – Description Yazılım yapılandırma yönetim planları standartı)

• IEEE 829-1998: IEEE Standard For Software Test Documentation(Yazılım test belgeleme standartı)

• IEEE 830-1998: Yazılım gereksinim belirlenmesi için tavsiye edilen uygulamalar (IEEE recommended practice for software requirements specifications)

• IEEE 1012-1998: Yazılım doğrulama ve onaylama standartı (IEEE standard for software verification and validation plans)

• IEEE 1016-1998: Yazılım tasarım tavsiyeleri (IEEE recommended practice for software design descriptions)

• IEEE 1028-1997: Yazılım gözden geçirmeleri standartı (IEEE Standard for Software Reviews)

• IEEE 1058-1998: Yazılım projesi yönetim plan standartı (IEEE standard for software project management plans)

• IEEE 1061-1998: Yazılım kalite ölçütleri yöntem standartı (IEEE standard for a software quality metrics methodology)

• IEEE 1063-2001: Yazılım kullanıcısı belgeleme standartı (IEEE standard for software user documentation)

• IEEE 1074-1997: Yazılım yaşam döngüsü süreçlerini geliştirme standartı (IEEE standard for developing software life cycle processes)

• IEEE/EIA 12207.0-1996: Yazılım endüstrisinin uluslararası platformda kullanım standartı (Standard Industry Implementation of International)

Bu standartlar genel olarak aşağıdakileri kapsar.

• Birincil süreçler: Kazanım, Destek, Geliştirme, Operasyon ve Bakım

• Destekleyici Süreçler: Dokümantasyon, Özellik Yönetimi, Kalite Sağlaması, Doğrulama, Geçerli Kılma, Gözden Geçirme, Denetleme ve Problem Çözme

• Örgütsel Süreçler: Yönetim, Yapı, Geliştirme ve Eğitim

2.2.2 Yazılım Kalite Faktörleri ve Modelleri

Müşterinin yazılım kalitesiyle ilgili anladıkları genellikle aşağıdaki maddelerden oluşmaktadır.

1. Performans : Birincil müşteri gereksinimlerinin karşılanması.
2. Özellikler : Sağlanan ek özelliklerin müşteri gereksinimleri aşması.

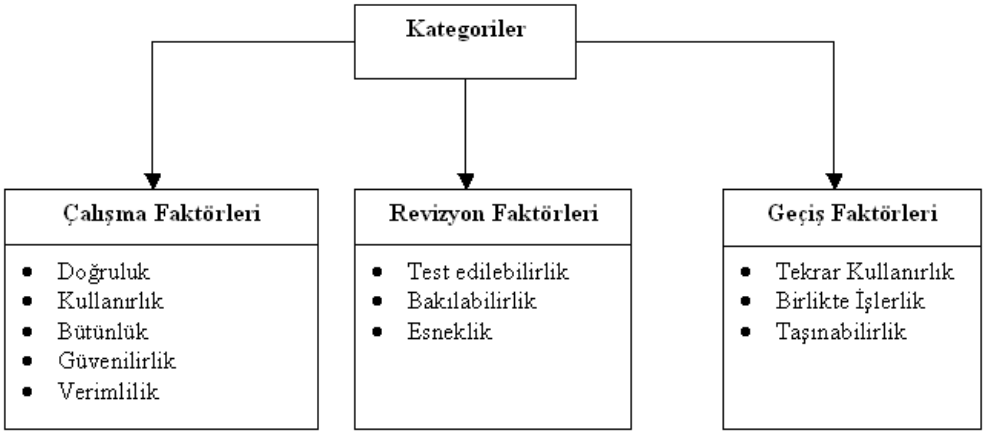
3. Güvenilirlik : Müşteri tarafından belirlenmiş koşullarda ve belirli periyotta yazılımın sürekliliği.
4. Uygunluk : Performansın ve fiziksel karakteristiklerin müşterinin daha önce belirlediği standartlara uyma derecesi.
5. Dayanıklılık : Yazılımın atıl olana kadar yararlı olduğu süre.
6. Kullanışlı olması : Yazılımın tamiri ve bakımının kolaylığı.
7. Estetik : Ürünün genel görünüşü ve hissettirdikleri.
8. Algılanan : Müşterinin beklediği kalite ve sağlanan kalite arasındaki fark [5].

Müşteri kalite kriterlerini etkileyen yazılım kalite faktörleri ile ilgili yıllarda çeşitli modeller üretilmiş ve sınıflandırmalar yapılmıştır.

2.2.4 McCall Kalite Modeli

Jim McCall tarafından 1977 yılında sunulmuştur.(General Electric Model olarak da bilinmektedir. Amacı kullanıcılar ve yazılım geliştiricilerin yazılım kalitesine olan bakış açıları arasındaki uçurumu kapatmaktır.

McCall 3 ana grup içerisinde 11 tane kalite faktörü belirlemiştir. Yazılım ürününün değişim yeteneği, yeni ortamlara uyumu ve çalışma özellikleri ana gruplardır.



Şekil 2.2 : McCall Kalite Faktörleri [6]

2.2.4.1 Yazılım Ürünü Çalışma Faktörleri

Doğruluk (Correctness): Doğruluk, gereksinim listesiyle yazılımın ilgili çıktılarının ne kadar uyuştuğuyla belirlenir. Doğruluğu oluşturan alt faktörler:

- Bütünlük
- Kullanırlık(cevap süresi)

- Kesinlik
- Güncellik
- Tutarlılık

Kullanılabilirlik (Usability): Yazılımı kullanacak yeni bir çalışanın eğitimi ve yazılımın işletilmesi için gereken kaynakların çokluğu veya azlığı kullanabilirliği belirler. Kullanabilirlik, işletme ve eğitim faktörlerinden oluşur.

Bütünlük (Integrity): Bütünlük yazılımın güvenliği ile ilgilenir. Örnek olarak yetkisiz bir kullanıcının erişimini engellemek ve bu işlemi çoğunluğa verileri okuma hakkını ve sınırlı sayıda kullanıcıya Bütünlük erişim kontrolü ve erişim takibi faktörlerinden oluşmaktadır.

Güvenilirlik (Reliability): Güvenilirlik gereksinimleri, yazılımın uygun bir seviyede servis vermesi sırasında çıkan hatalarla ilgilenir. Bununla beraber, yazılım sisteminin genelinde veya fonksiyonlarında oluşabilecek maksimum hata oranı konusunda fikir verir. Sistem güvenilirliği, donanım hatasını telafi etme, uygulama güvenilirliği ve hesaplama hatasını telafi etme, güvenilirliğin dört alt başlığıdır.

Verimlilik (Efficiency): Verimlilik gereksinimleri, tüm diğer gereksinimler için ne kadar donanım harcaması gerektirdiği konusuyla ilgilenir. İşlem verimliliği, hafızaya saklama verimliliği, iletişim verimliliği ve güç kullanımı verimliliği (taşınabilir üniteler için), verimliliğin dört alt başlığıdır.

2.2.4.2 Yazılım Ürünü Revizyon Faktörleri

Test Edilebilirlik (Testability): Test edilebilirlik gereksinimleri bilgi sisteminin ve belirtilen özel bir operasyonun testi için harcanması gereken efor ile ilgilenir. Takip edilebilirlik, Kullanıcı test edebilirliği ve bakım hatası test edilebilirliği kalemlerinden oluşur.

Bakım Kolaylığı (Maintainability): Bakım kolaylığı, yazılım hatalarının nedenlerinin araştırılması, giderilmesi ve hatanın giderildiğine dair doğrulama yapılması işlerinin tüm potansiyel kullanıcılar ve bakım çalışanları açısından ne kadar efor harcama gerektirdiğiyle ilgilenir. Alt başlıkları aşağıdaki gibidir.

- Modülerlik
- Basitlik
- Tutarlılık

- Dokümanlara erişim
- Kodlama ve belgeleme rehberleri
- Kendini açıklama

Esneklik (Flexibility): Esneklik gereksinimleri, yeni bakım aktivitelerinin adaptasyonunda ne kadar yetenekli olunduğu ve ne kadar efor harcanması gerektiği konularıyla ilgilenir. Basitlik, modülerlik, genelleme ve kendini açıklama, alt başlıklarıdır.

2.2.4.3 Yazılım Ürünü Geçiş Faktörleri

Yeniden Kullanılabilirlik (Reusability): Yeniden kullanılabilirlik, başka bir proje için özellikle geliştirilen yazılım modüllerinin yeni geliştirilen veya geliştirilecek projeler de kullanılmasıyla ilgilenir.

- Basitlik
- Genelleme
- Modülerlik
- Doküman erişim kolaylığı
- Kendini açıklama
- Uygulama bağımsızlığı
- Yazılım sistem bağımsızlığı

Başlıklarından oluşur.

Birlikte İşlerlilik (Interoperability): Birlikte işlerlilik, geliştirilen yazılımın başka yazılımlarla veya başka ekipmanlarla bağlantıyı sağlayan arayüzlerin yaratılmasıyla ilgilenir. Modülerlik, ortaklık, sistem uyumluluğu ve yazılım sistem bağımsızlığı alt başlıklarıdır.

Taşınabilirlik (Portability): Taşınabilirlik, yazılımın farklı işletim sistemleri ve farklı donanım gibi farklı platform bileşenleri üzerinde çalışması için gerekli adaptasyon eforu ile ilgilenir. Modülerlik, kendini açıklama ve yazılım sistem bağımsızlığı, taşınabilirliği oluşturur [6].

2.2.5 Boehm's Quality Model

Günümüzdeki modellerin temeli olan ikinci kalite modeli Barry W. Boehm tarafından 1978 yılında geliştirilen kalite modelidir. Üç üst seviye, yedi orta seviye ve kalite metriklerine baz oluşturan alt seviye faktörlerinden oluşmaktadır. Üst

seviye faktörleri yazılım kalitesini değerlendirmede akla gelecek üç soruyu cevaplamak üzere oluşturulmuştur.

• Olduğu gibi kullanılabilen (As-is utility): Üründe herhangi değişiklik yapmadan, bu ürünü ne kadar iyi(kolaylık, güvenilirlik, efektiflik anlamında) kullanabilirim?

• Bakım Kolaylığı (Maintainability): Yazılım ürününü anlama, değişiklik yapma ve test etme ne kadar kolay?

• Taşınabilirlik (Portability): Yazılım ortamını değiştirirsem ürünü hala kullanabilirmiyim?

Yazılım kalitesi dendiği zaman beklenen yedi orta seviye faktör aşağıdaki gibidir.

• Bilgisayarlar arası taşınabilirlik (Portability) : Kodun, başka bir bilgisayar yapılandırmasına geçiş esnasında yazılımın yeniden işlerliğini sağlayabilmesi için ne kadar taşınabilirliğe sahip olduğunu gösterir.

• Güvenilirlik (Reliability) : Kodun, yazılım ürününün amaçladığı fonksiyonu başarıyla yerine getirmesini sağlayan özelliklere ne kadar sahip olduğunu gösterir.

• Efektiflik (Efficiency): Kodun, atıl kaynak kullanmadan görevini yerine getirdiğini gösterir

• Kullanırlık (Usability): Kodun, insan mühendisliği düşünülerek güvenilirliği ne kadar sağladığını gösterir.

• Test edilebilirlik (Testability): Kodun kriterlerin doğrulanmasını ve performans ölçümünün değerlendirmesini ne kadar desteklediğini gösterir.

• Anlaşılabilirlik (Understandability): Kodun sahip olduğu özelliklerin, kodu inceleyen kişiye amacını ne kadar iyi açıklayabildiğini gösterir.

• Esneklik (Flexibility): Kodun sahip olduğu özelliklerin değişime ne kadar elverişli olduğunu gösterir.

McCall'un kalite modeline benzer bir yapısı vardır. Aralarındaki fark, "olduğu gibi kullanılabilen" faktörlerin McCall modelinde ayrı ayrı ana faktör şeklinde ele alınmış olmasıdır.

2.2.6 Diğer Modeller

Önceki modelleri içeren ve bazı değişiklikleri içeren Dromey Modeli, FURPS, ISO 9126 gibi bir çok model mevcuttur [7].

2.3 Nesneye Yönelik Yazılımlarda Kalite Ölçütleri

Düzgün forme edilmiş bir nesneye dayalı metotun aşağıdaki özelliklere sahip olması gerektiği savunulur.

- Kabul edilir derecede yapışık
- Az karmaşık
- Uygun büyüklükte
- Bağımsız test edilebilir
- İyi dokümante edilmiş

Bu özellikleri sağlayan, araştırma çevreleri tarafından nesneye yönelik programlama dahilinde bir çok ölçüt geliştirilmiştir (F. Brite e Abreu ve Carapuca, 1994; Benlarbi ve Melo, 1999; Briand, Devanbu, ve Melo, 1997; Cartwright ve Shepperd, 2000; Chidamber ve Kemerer, 1994; Henderson-Sellers, 1996; Li and Henry, 1993; Lorenz ve Kidd, 1994; Tang, Kao, ve Chen, 1999).
Uzak arayla bu ölçütlerin en popüler olanı Chidamber ve Kemerer'in 1994 yılında bulmuş oldukları ölçütlerdir. Yeni yapılan çalışmalarda da CK ölçütleri en fazla referans gösterilen ölçütlerdir(Briand, Arisholm, Counsell, Houdek, and Thevenod-Fosse, 1999). Ayrıca çoğu ticari ölçüt araçları bu ölçütleri toplamak için yazılmışlardır [8].

2.3.1 Chidamber&Kemerer Nesneye Yönelik Ölçütleri

Nesneye yönelik programlamada kullanılan ve yazılım kalite faktörlerini etkileyen temel metrikleri içerir.(Chidamber & Kemerer object-oriented metrics suite)

2.3.1.1 Ağırlıklı Metod Sayısı (WMC)

S.R Chidamber ve C.F Kemerer WMC ölçütünü ve bir nesnenin karmaşıklığının tanımını ilk olarak 1991'de önermişlerdir. Öneriye göre kullanılan metod sayısı ve bu metodların karmaşıklıkları nesnenin geliştirilmesinde ne kadar zaman ve efor harcandığının göstergesidir.

Bir alt sınıf, ana sınıfın tüm metodlarını kalıtım yoluyla içerdiği için, ana nesnenin metod sayısı ne kadar büyürse, nesneden türetilmiş sınıflara etkisi o kadar çok olur. Çok fazla metod kullanılan sınıf o uygulamaya çok bağlı olacağından başka uygulamalarda tekrar kullanım olasılığı kısıtlanmış olur.

WMC, CC ölçütünün bir uzantısı olduğundan (eğer WMC ölçülürken CC kullanılıyorsa), çıkan eşik değerinin CC nin üst limiti ile karşılaştırılması önerilir. Hesaplanan WMC değeri alınır ve metod sayısına bölünür. Çıkan sonuç CC'nin üst değeri ile karşılaştırılabilir. CC'yi kullanmanın bir dezavantajı, metodların tamamen uygulamaya geçmeden karmaşıklığının bulunamayacağı ve dolayısıyla henüz tasarım aşamasında olan, içi boş metodlarla bu ölçümün gerçekleştirilemeyeceğidir. Bu tip durumlarda WMC ölçütüne karmaşıklık katılmaz ve sadece metod sayısını ölçmek yeterli olacaktır. Böylece WMC ölçütü karmaşıklık ölçen bir ölçüt olmaktan çıkıp büyüklük ölçen bir ölçüt haline gelir.

Tanım: C1 sınıfın olduğunu, bu sınıf içerisinde M1, M2, .. Mn.adında metodların olduğunu ve metodların sırasıyla karmaşıklıklarının c1, c2, …cn olduklarını farzedelim.

Buna göre:

$$WMC = \sum_{i}^{n} C_i \qquad \textbf{(2.1)}$$

Eğer karmaşıklıkları gözardı edersek, WMC değeri metod sayısına eşit olur.

Statik karmaşıklık birden fazla yolla ölçülebilir. En yaygın ölçüm modeli döngüsel karmaşıklıktır.

2.3.1.2 Kalıtım Ağaç Derinliği (DIT)

Kalıtım bir sınıfın, daha önce tanımlanmış başka bir sınıfın yapısını ve davranışını kendisine almasıyla oluşur. Bir alt sınıf sadece bir üst sınıftan kalıtım yaparsa buna tekil kalıtım, birden fazla üst sınıf ile yaparsa çoklu kalıtım denir. Kalıtım yazılımdaki gereğinden fazla mantıksal yapıyı azaltarak efektifliği arttırır. Fakat kalıtım hiyerarşisinin derinleşmesi, yazılımın karmaşıklaşması ve davranışının tahmin edilememesi olasılığını arttırır.

Bu hiyerarşide derinliği ölçmek için Shyam R. Chidamber ve Chris F. Kemerer 1991 yılıda kalıtım ağaç derinliği ölçütünü önermişlerdir. J. Bloch, paketlerin farklı yazılım geliştiriciler tarafından yazıldığından ötürü paketler arası bir kalıtımın tehlikeleri barındırdığına parmak basmıştır.

Bir alt sınıf aslında üst sınıfa bağlı olduğundan, üst sınıfta yapılacak herhangi değişiklik alt sınıfa yansıyacaktır. Bloch'a göre, kalıtım sadece ve sadece alt sınıfın

üst sınıfın bir özelleşmiş hali olduğunun kesinliği neticesinde yapılmalıdır. Eğer bu durumda bir kesinlik yoksa, yeni bir sınıfın eski sınıfın bir özel değişkenine referans veren durum olan bileşim işlemi kullanılmalıdır.

Sınıfların genişletilmesinde tasarıma önem verilmişse, dokümantasyon yeterli bilgiyi veriyorsa veya sınıf genişletilmesi aynı programcıların kontrolü altındaysa kalıtım kullanılarak kod tekrar kullanımı büyük ölçüde sağlanır demektir.

DIT ölçütünün eşik değerinin ne olması gerektiği hakkında bir takım çalışmalar vardır, fakat net olarak bulunmuş herhangi bir değer yoktur. Değer yüksek oldukça tekrar kullanabilirlik artar lakin diğer tarafta karmaşıklık artar, yazılımın anlaşılması zorlaşır. Sistemin kendi özelliklerine göre eşik değeri bulmak yazılım geliştiricilere kalmıştır.

Tanım: Bir sınıfın kalıtım ağacındaki derinliği DIT değerini verir. Birden fazla kalıtım varsa, DIT değeri daldan köke en uzun uzaklıktır.

2.3.1.3 Alt Sınıf Sayısı (NOC)

Alt sınıf sayısı ölçütü, Shyam R. Chidamber ve Chris F. Kemerer tarafından tekrar kullanım ve buna bağlı olarak test etme seviyesini belirleme amacıyla 1991 yılında önerilmiştir. Kalıtım ağacında alt sınıf sayısı ne kadar çoksa, tekrar kullanabilirlik o kadar artıyor demektir. Ancak bir sınıfın veya paketin, aşırı derecede alt sınıfı varsa o sınıfın soyut sınıf olarak kullanılmasında ve alt sınıfların kalıtılmasında hata var demektir. Alt sınıf sayısı arttıkça, yapılması gereken test sayısı artar.

Tanım: NOC, Sınıf hiyerarşisinde bir sınıfın çocuğu olan alt sınıf sayısı.

2.3.1.4 Sınıfın Cevabı (RFC)

Eğer bir sınıf çok fazla sayıda metoda sahipse sınıfın karmaşık olması yüksek olasılıktadır. Sınıfa bir mesaj iletildiğinde, bu mesaja cevap verebilmek adına yapılan metod çağrılarının fazla olması bakım ve test işlerini komplike hale sokarlar.

Shyam R. Chidamber ve Chris F. Kemerer 1991 yılında sınıfın bir mesaja cevap vermek için yerel metodların yaptıkları çağrı sayısını ve o metodların sayısını hesaplayan bir ölçüt önermişlerdir. RFC ölçütünün eşik değeri şu ana kadar yapılan hiçbir çalışmada net olarak verilmemiştir. Chidamber ve RFC değerinin büyüdükçe test edilecek parçanın anlaşılırlığının düştüğünü öne sürmüşlerdir.

Tanım:

$RFC = |RS|.RS$ Bir sınıfın cevap veren kümesi olsun.

$RS = \{M\} \cup_i \{R_i\}$,

$\{R_i\} = i$ metodu tarafından çağırılan metod sayısı

{M} = Sınıftaki metod kümesi

Tanımı anlayabilmek için aşağıdaki örneğe bakılabilir:

A::f1() B::f2()'i çağırır

A::f2() C::f1()'i çağırır

A::f3() A::f4()'i çağırır

A::f4() hiçbir metod çağırmaz.

Buna göre RS = {A::f1, A::f2, A::f3, A::f4} U {B::f2} U {C::f1} U {A::f4}

= {A::f1, A::f2, A::f3, A::f4, B::f2, C::f1}

ve buradan da RFC = 6 çıkar.

2.3.1.5 Nesneler Arası Bağlılık (CBO)

Bir sınıf kendi içerisindeki bir metodun diğer sınıftan bir değişkene referans vermesiyle veya başka bir metodu çağırmasıyla bağlı olur. Bunun tersi de geçerlidir.

CBO değeri bir sınıfın içsel veya dışsal bağlı olduğu sınıf sayısını verir. R. Marinescu yaptığı çalışmada yüksek bağlılığın kalite değerleri üzerinde yarattığı etkileri listelemiştir;

• Bağlılığı yüksek olan sınıflarda, varlık içeriğinde yüksek bağımlılık olduğundan, başka bir içerikte bu varlığı kullanmak zordur ve tekrar kullanabilirliği düşürür.

• Normalde bir modülün diğer modüllerle arasındaki bağ düşük olmalıdır. Sistemin belli başlı parçalarıyla olan yüksek bağlılık, modülerlik üzerine negatif etki yaratır. Bu durum tüm bu parçaların sorumluluk alanlarının belirli olmadığı kötü bir tasarıma işaret eder.

• Kendine yetemeyen sınıflardan oluşan sistemde, sistemin anlaşılması oldukça güçleşir. Bir sınıfın kontrol akışının başka sınıfların çok büyük sayılarda kullanımına bağlı olması o sınıfın mantığını takip etmeyi zorlaştırır. Bunun nedeni sınıfa bağlı dış parçaların özyinelemeli bir şekilde anlaşılması gerektiğidir. Bu yüzden az bağlılık terci edilen durumdur.

Tanım:

CBO bağlı olunan diğer sınıfların sayısıdır [9].

2.3.1.6 Metodların Yapışma Yetersizliği (LCOM)

Bir sınıfın yapışıklığı yerel metodların yerel değişkenlere olan yakınlığı ile karakterize edilir. S.R Chidamber ve C.F Kemerer, LCOM ölçütünü ilk kez 1991 yılında tanımlamışlardır ve 1993 yılında geliştirmişlerdir. O zamandan bu zamana daha fazla tanım önerilmiştir ve halen bu alan içerisinde araştırmalar yapılmaktadır.

Yapılan bir çalışmada yazarlar birden fazla LCOM tanımını değerlendirmiş ve karşılaştırmış, bunun sonucunda CK LCOM tanımının Li ve Henry'nin LCOM tanımından daha esnek olduğu kanısına varmışlardır. Bunlarla beraber Henderson-Sellers'ın önerdiği LCOM ölçütü de popüler bir ölçüttür. LCOM ölçütü kim tanımlamış olursa olsun, bir sınıftaki metodlar arası farklılığı göstermektedir. Yüksek LCOM değerinde çıkan bir sınıfın iki veya daha çok sınıfa ayrıştırılması iyi bir fikir olacaktır. Bir sınıfın çok fazla farklı görevi yapması gerektiğinden, spesifik nesneleri kullanmak tasarım açısından akıllıca olacaktır.

LCOM metodlar arası farklılığı gösterdiğinden dolayı, sınıf tasarımlarındaki niteliksel kusurların da yakalanmasını sağlar. Metodların birbirine yapışık olması, kapsüllemeyi sağladığından ve nesnelerin karmaşıklığını azalttığından istenen bir durumdur.

Tanım:

C1 sınıfının M_1, M_2, .. M_n adında n metodu olsun.

$\{I_j\} = M_i$ metodu tarafından kullanılan değişkenler kümesi olsun. Böylece bu kümeden n adet olacaktır.

$P = \{(I_i, I_j) \mid I_i \cap I_j = \varphi\}$ *and* $Q = \{(I_i, I_j) \mid I_i \cap I_j \neq \varphi\}$

Eğer tüm n kümesi ($\{I_1\}$,..$\{I_n\}$) ø ise P = ø.

LCOM = |P| - |Q|, eğer |P| > |Q|

LCOM = 0, değilse.

LCOM üzerinde yapılan bir çalışma sonucunda iyi LCOM değerine sahip bir sınıfın iyi tasarlandığını, fakat kötü değere sahip bir sınıfın her zaman kötü tasarımı göstermediği ortaya çıkmıştır. Tipik olarak çok iyi sonuç veren LCOM değerli sınıflarda az değişken yer almaktadır. Kalıtım kullanılan alanlarda genellikle genişleme dikeysel veya kısmen dikeysel LCOM değeri genellikle yapışıklığı iyi bir şekilde ölçer.

Çıkan sonuçlar, büyük sınıflarda düşük, daha küçük sınıflarda yüksek yapışma değerlerinin ortaya çıktığını açıkça göstermiştir. Oldukça çok sınıfın LCOM değerleri kötü çıkmıştır, çünkü metodlar çok fazla değişkenden çok azını kullanmaktadırlar. Bu durumun çoğu zaman aslında bir tasarım hatası olmadığı bulunmuştur. Metodlar gelişen metod sayısının aslında bir dağılımı olarak form bulmuşlardır. Yazarlara göre bu tip durumlarda LCOM değerinin normalize edilmesi kullanışlı olacaktır. Bu durum özellikle çok fazla metodu olup az değişkene sahip büyük sınıflar için anlamlı olacaktır. Normalizasyon ile ilgili bir formülasyon ileri çalışmalar için bırakılmıştır.

Çok kötü sonuçların da başka açıklamaları vardır. Yazarlar, kötü tasarımın dışında, bu tip vakalara aşağıdaki konuların yol açtığını söylemektedirler.

• Sınıfın kalıtım ile kullanılması amacıyla tasarlanması
• Sınıfın kendi özellikleri yerine kalıtımdan gelmiş özellikleri kullanması
• Sınıfın iç sınıfları destekler nitelikte tasarlanması
• Sınıfın dış sınıf haline gelebilmesi amacıyla tasarlanması.

Bu tip durumlar için LCOM değeri kale alınmamalıdır [10].

Chidamber ve Kemerer tanımında iki problem vardır:

i. Ölçütün bir üst limiti yoktur. Bu yüzden ölçülen değerden bir anlam çıkarmak zordur
ii. LCOM=0 veren fakat aynı yapışıklığa sahip olmayan bir çok örnek vardır.

Bu dezavantajlarına rağmen, yapışkanlık değerlendirmesinde LCOM ölçütü hala en geniş çapta kullanılan ölçüt olma ünvanını korumaktadır [11].

NASA'da yapılan bir çalışma Chidamber ve Kemerer ölçütleri için ortalama aşağıdaki değerleri vermektedir. Çalışma üç sistemi ve onun kalite ölçütlerini ortaya çıkarmıştır.

Tablo 2-2 : NASA CK Ölçüt Karşılaştırmaları [19]

Sistemler	**Java**	**Java**	**C++**
Sınıflar	46	1000	1617
Satırlar	50,000	300,000	500,000
Kalite	"Düşük"	"Yüksek"	"Orta"
CBO	2.48	1.25	2.09
LCOM1	447.65	78.34	113.94
RFC	80.39	43.84	28.60
NOC	0.07	0.35	0.39
DIT	0.37	0.97	1.02
WMC	45.7	11.10	23.97

Çalışma sonucunda CBO ve WMC yüksek çıktığında kalitenin düştüğü gözlenmiştir. Diğer ölçütler herhangi bir kanaat getirilmesinde yardımcı olmamışlardır.

2.3.2 Henderson-Sellers LCOM tanımı

Henderson-Sellers 1996 yılında yapışıklık değerlendirmesi için yeni bir tanım önermişdir. Bu tanım 0 ile 1 arasındaki yapışıklığı verir. Burada 0 mükemmel yapışıklığı temsil etmektedir.(Tüm metotlar tüm değişkenlere erişir) Fakat bazı durumlarda 1'i geçer. Bu durum üç senaryoda meydana gelir:

1. Bir sınıfta hiçbir değişken tanımlanmadıysa (örneğin tüm değişkenlerini üst sınıftan kalıtım yoluyla aldıysa). Bu durum programlama açısından olabilir, bu yüzden LCOM değerini 0 olarak verebiliriz.
2. Bir takım değişkenler yaratılmış, fakat bunlara hiç referans verilmemiş olabilir. Bu durum kötü olmakla beraber yaratılan varlıkların ya hiç kullanılmadığını ya da alt sınıflar tarafından kullanıldığına işaret eder. Bu yüzden LCOM değerini hesaplarken örneklenen değişken sayısını erişilmiş değişken sayısı olarak belirleriz.
3. Değişkenlere erişim sayısı toplam değişken sayısından düşük olabilir. Bu durum ikinci durumun bir uç durumudur ve gene aynı şekilde değerlendirilebilir [12].

Formülü aşağıdaki gibidir;

$$LCOM^{*} = \frac{\left[\frac{1}{a}\sum_{j=1}^{a}\mu(A_j)\right] - m}{1 - m} \qquad \textbf{(2.2)}$$

a değişkeni sınıftaki değişken sayısını, m ise sınıftaki metot sayısını belirtir. $i(A_j), A_j(1.j.a)$ varlığına ulaşan metotların sayısını verir. Bu tanım 0 ile 1 arasına normalize edilmiştir. Bu değerin ölçülmesinin önceki LCOM tanımlarından daha rahat olduğunu iddia etmektedir.

Bu ölçütteki bir problem, sınıf içerisindeki yapışıklığın hangi bölgelerde ne kadar olduğunun(metotlar arası ilişki) verilmemesidir. Bunun sonucu olarak, hangi metotun hangi değişkenlere erişmesinden ziyade, bir değişkene ne kadar erişildiği ölçülmektedir.

2.3.3 McCabe Döngüsel Karmaşıklık Ölçütü

1976 yılında McCabe rakamsal döngüsel karmaşıklık ölçütünü tanımlamıştır. Ölçüt, bir yazılım modülünde birbirinden bağımsız yolların sayısını verir. McCabe bu ölçüt için üst limit olarak 10'u önermiştir ve bundan daha büyük bir değerin az yönetilebilir ve test edilebilir olduğunu savunmaktadır. Bu üst limit bir deneysel çalışmayla ortaya çıkarılmamıştır fakat dünyada birden fazla projede yapılan incelemeler yüksek döngüsel karmaşıklık olan modüllerin daha çok hataya yol açtığını ve anlaşılabilirliği düşürdüğünü onaylamıştır.

Döngüsel karmaşıklık geniş bir alanda kullanılmasına rağmen bu konuda eleştiriler de mevcuttur. Sheppard'ın 1988 yılında getirdiği eleştiri bu ölçütün kötü teorik temelden ve yetersiz yazılım geliştirme modelinden oluşturulduğunu söylemektedir. Ayrıca Sheppard, bu ölçütün koddaki satır sayısına vekil olduğunu eklemiştir.

Döngüsel karmaşıklık ölçütünün bu kadar geniş çapta kullanılmasının bir nedeni, ölçütün istatiksel analiz yardımıya çok rahat ölçülebilmesi ve endüstride zaten kalite kontrol amacıyla yaygın bir şekilde kullanılıyor olması.

Döngüsel karmaşıklık ölçütü araştırmalarda ve pratikte hala kullanılan en eski ölçütlerden biridir. Fakat bu konuda karışık görüşler mevcuttur. Bu ölçütün

müşterilere açıklaması kontrol akışının anlatılmasını beraberinde getirir. Birkaç örnekle bu ölçüt müşterilere anlatılabilir. Ölçüt hem prosedürel, hem de nesne odaklı programlama modüllerinde ölçülebilir. Birçok yazılım aracı bu ölçütü ölçer ve bu ölçüt üzerinde kıyaslamalar yapılmaktadır [13].

McCabe Döngüsel karmaşıklık ölçütü, doğrusal olarak kod içerisindeki bağımsız yol sayısını ölçer ki bu toplam karar noktaları + 1 şeklinde tanımlanmıştır. Burada karar noktaları if/else veya while gibi koşullu ifadelere denk düşer. Tablo 2-3'de McCabe'in karmaşıklık ölçeği gösterilmektedir. Hedef kodda az karmaşıklık sağlayarak riski düşük tutmaktır. Az seviyedeki karmaşıklık metodu daha anlaşılabilir ve bakılabilir yapar.

Tablo 2-3 : Döngüsel Karmaşıklık Yol-Risk Değerleri [13]

Yol sayısı	Kod karmaşıklığı	Risk
1-10	Çok karmaşık değil	Az
11-20	Orta derecede karmaşık	Orta
21-50	Yüksek karmaşıklığa sahip	Yüksek
51+	Kararlı değil	Çok yüksek

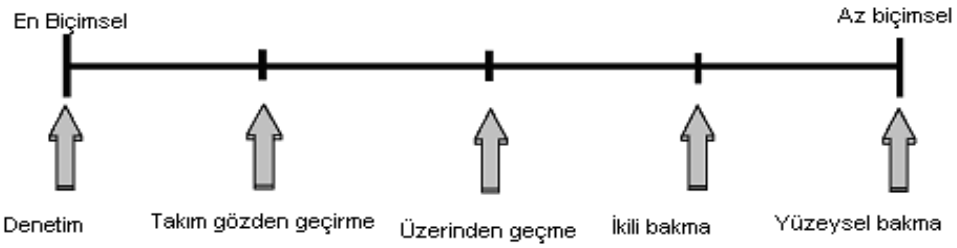
2.4 Kod Gözden Geçirmeleri

Kod gözden geçirmesi(birlikte gözden geçirme), hataların geliştirme aşamasında bulunması ve giderilmesi için bilgisayar kaynağında yapılan sistematik incelemedir. Yazılımın genel kalitesini arttırmasıyla beraber, yazılım geliştiricilerinde yeteneklerinde artış sağlar.

Çoğunlukla yaygın katar biçim yanlışlıkları, hafıza kaçakları, arabellek taşmaları ve yanlış çalışan koşulları yakalar ve bunların kaldırılmasına, dolayısıyla yazılım güvenilirliğini sağlamaya yarar.

Kod gözden geçirmeleri biçimsel kod gözden geçirmesi ve biçimsel olmayan kod gözden geçirmesi arasında yer alır.

Biçimsel kod gözden geçirmeleri daha çok vakit alır ve daha prosedürel bir yaklaşımdır. Birden fazla kişinin katılımıyla ve birden fazla faz içerisinde gerçekleştirilir. Uygulanırken dikkatli ve detaylı prosedürler izlenir. Geleneksel bir yapıdır ve eskidir.



Şekil 2.3 : Kod Gözden Geçirmeleri [20]

Çeşitli kod gözden geçirme yöntemleri aşağıdaki gibidir.

Denetim: En biçimsel kod gözden geçirme yöntemi denetimdir. İyi tanımlanmış ve dokümante edilmiş altı adımdan oluşur.(Planlama, genel bakış, kişisel hazırlık, denetim toplantıları, tekrar çalışma ve takip) Bu süreç içerisinde yer alan kişiler kodu yazan, değiştiren, okuyan, kaydeden rollerine sahip olabilirler. Kontrol listeleri süreç esnasında yardımcı olarak kullanılır.

Takım gözden geçirmesi: Denetimden daha az katı bir yapıdadır. Genel bakış ve takip aşamaları atlanır ve bazı roller birleştirilir.

Üzerinden geçme: Yazılımı geliştiren kişinin liderliğinde toplu olarak yapılır. Belirli bir prosedürü ve dokümantasyonu içermediğinden biçimsel değildir.

Birlikte bakma: Yazılım geliştiricinin bir kişiyle beraber kodu incelemesine denir.

Etrafa gönderme: Son ürünün kodunun birden fazla kişiye gönderilmesi ve yorumların alınması şeklinde olur. [20]

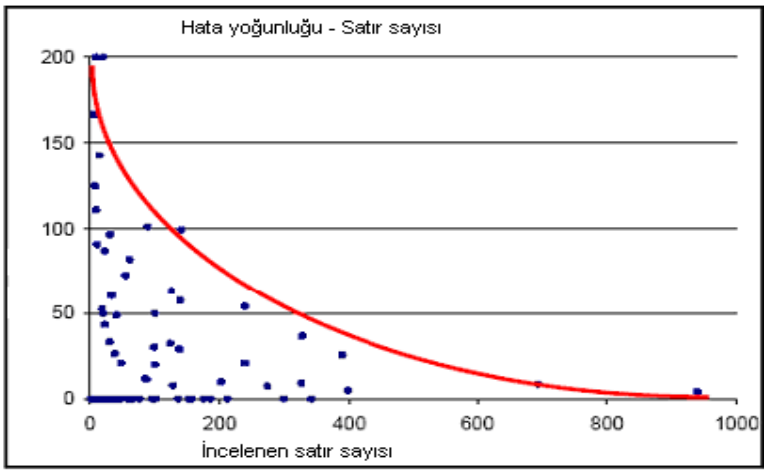
2.4.1 Kod Gözden Geçirmeleri ile ilgili yapılan çalışmalar

Bunlardan birincisi Cisco Systems'in dünya çapında yaptığı 3.2 milyon satırlık kodu inceleyen çalışmadır. 10 aylık bu çalışmada hafif kod gözden geçirmelerinin biçimsel kod gözden geçirmelere oranla daha çabuk yapıldığı ve aynı oranda efektif olduğu belirlenmiştir. Çıkan sonuçlar aşağıdaki gibi maddelenmiştir. Tüm çalışma [14] numaralı kaynaktan alıntıdır.

a. Bir seferde 200-400 satır kod parçası taranması gerekir.

Cisco çalışması, yazılım geliştiricilerin bir seferde 200 ila 400 satır kod taramaları gerektiğini göstermiştir. Bundan daha fazla sayıda bakılacak kodlarda hata bulma oranı düşmektedir.

Aşağıdaki grafik bu kuralı doğrular nitelikte olup, hata yoğunluğu ile kod satır sayısı arasındaki değerleri verir. Hata yoğunluğu 1000 satır başına rastlanan hata sayısını verir. Şekilden görüleceği gibi satır sayısı 300'ü aştığında hata yakalamada yüksek oranda düşüş yaşanır.

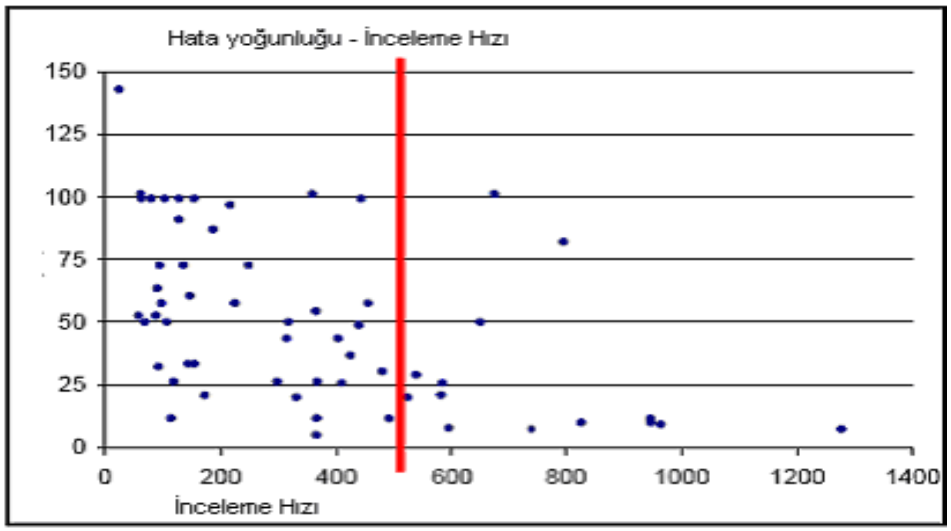


Şekil 2.4 : Hata Yoğunluğu – Satır Sayısı Analizi [14]

b. Saatte 300-500 satır inceleme hızına erişilmesi idealdir

Çalışmada ideal inceleme hızını bulmak için hata yoğunluğunun inceleme hızına oranı incelenmiştir. Sonuç beklendiği gibi, kod gözden geçirmelere fazla zaman ayrılmadığı takdirde hata bulma sayısının azaldığı yönünde çıkmıştır. Aynı zamanda kod gözden geçiren kişiye incelemesi için az zaman ve fazla kod verilmişse kişi tüm koda istediği dikkati vermekte zorlanır.

Şekil 2-5: 400-500 LOC/saat'den fazlasında verimlilik düşüyor, saatte 1000 ve sonrasında kod gözden geçiren için kodu incelemek bir hayli zorlaşıyor, belki koda bakmayı bir süre sonra bırakıyor.



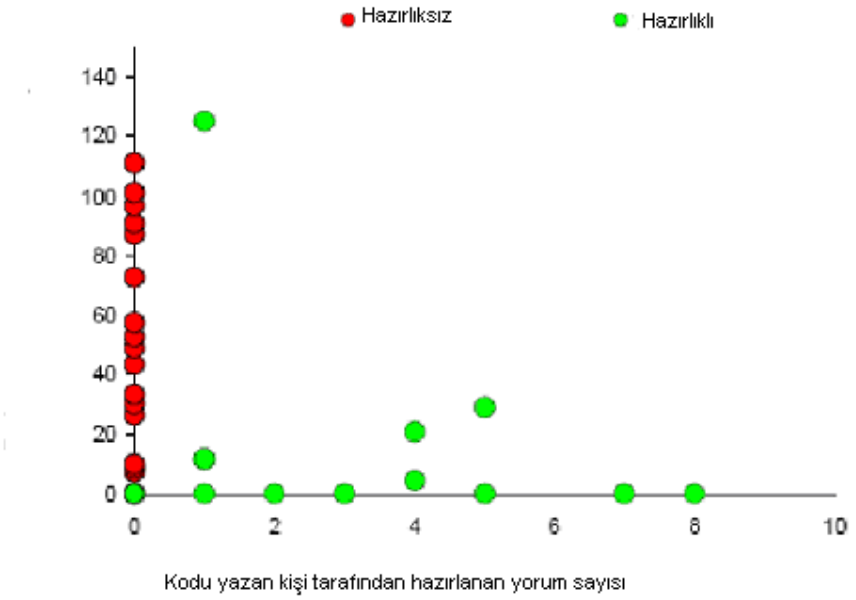
Şekil 2.5 : Hata Yoğunluğu – İnceleme Hızı Analizi [14]

c. Kod gözden geçirmesi 60-90 dakikadan daha fazla sürmemeli.

Bu kural genel olarak insanların herhangi aktiviteyle uğraşırken konsantrasyon kaybına uğramaya başladıkları süredir. Bu süre yapılan araştırmalarda 60-90 dakika arası çıkmıştır.

d. Gözden geçirme başlamadan, programlayıcı kodu notlarla açıklamalı.

Daha kod gözden geçirme başlamadan yazılım geliştirici koda bakacak olursa hatalar önceden yakalanabilmektedir.Kodu gözden geçiren kişiler yapılan açıklamalar sayesinde ve önceden giderilen hatalar neticesinde işlerini daha hızlı ve verimli şekilde yapabilmektedirler. Açıklamaların normal kod yorumlarından ziyade kodu inceleyen kişilere yapılması gerekir.



Şekil 2.6 : Hata Yorum Analizi [14]

e. Kod gözden geçirmeleri için hedef belirlenmeli, sürecin gelişimi için ölçütler takip edilmeli

Diğer tüm projelerde olduğu gibi, kod gözden geçirme sürecinin hedeflerinin ve verimliliğinin nasıl ölçüleceğinin belirlenmesi öneml teşkil eder. Hedefler belirliyse kod gözden geçirmenin doğru gidip gitmediğini görüp ona göre eylem uygulanabilir.

Başlangıç olarak "destek çağrılarının alınmasında %20 lik düşüş" veya "hataların %50 sinin geliştirme aşamasında yakalanması" gibi dış ölçütlerin kullanılması en iyisi olacaktır. Bu bilgi dış çerçevede kodun nasıl çalıştığını açık bir şekilde gösterecektir.

Burada ölçütün ölçülebilir olması bulanıklığı azaltacaktır. Fakat dış ölçütlerin ortaya çıkması biraz zaman alabilir. Örneğin destek çağrıları yeni bir versiyona kadar değişmeyebilir. Bu yüzden iç sürecin takibi için iç ölçütler kullanılması gerekmektedir. Bu ölçütler sayesinde ne kadar hata bulunmuş, yazılım geliştiriciler hatalar üzerinde ne kadar çalışma yapmışlar gibi soruların cevapları alınmış olur. Kod gözden geçirmeler için en genel iç ölçütler denetim hızı, hata hızı ve hata yoğunluğudur.

f. Kontrol Listeleri hem kodu yazan hem de gözden geçiren için idealdir. Kontrol listelerinin kullanılması, kontrol edilen noktaların unutulmaması için kod gözden geçiren ve yazılım geliştiriciye faydası olan bir araçtır. Bir kez yakalanmış ve sonradan atlanmış maddelerin yakalanması çok zordur. Kontrol listesi, yazılım geliştirici ve kod gözden geçirene tüm hataların yakalanıp yakalanmadığını, fonksiyon değişkenlerinin geçersiz değerlere sahip olup olmadığını ve buna bağlı olarak birim testler yaratılması gerekip gerekmediğini verir.

g. Bulunan hataların giderildiğini kontrol etmek gereklidir. Hataların giderilmesi kod gözden geçirmeler sonunda esas istenen durumdur.

h. Kod gözden geçirmeler sonunda çıkan hataların giderilmesi için kod yeniden düzenlenmelidir. Kod yeniden düzenlenmesinin takibinin yapılması biçimsel olmayan kod gözden geçirmelerde güçtür. Hataların takibi ve giderilmesi için uygun bir yazılım aracı kullanımı önerilmektedir.

i. Yöneticiler, hata bulmanın iyi bir şey olduğunu içeren kod gözden geçirme kültürünü yazılım geliştiricilere adapte etmeliler. Yöneticiler kod gözden geçirmenin öğretici ve iletişimi geliştirici yanını keşfederlerse, yazılım ekibinin motivasyonuna olumlu katkı sağlarlar. Kodda bir hata bulunmasının olumsuz bir hava yaratması çok kolaydır bu olumsuz havadan etkilenenler hata bulma sürecini baltalayabilirler. Yöneticiler hataların olumlu olduklarını, kodu geliştirmek için fırsat yarattığını ve bu sürecin kimseyi suçlamak için değil, kodu geliştirmek olduğunu empoze etmesi gerekmektedir. Yazılım geliştiriciler için ise kötü kod geliştirme alışkanlıklarından kurtulmak için bir fırsat ve eşsiz bir eğitim aracıdır. Hatalarını görebilen yazılım geliştiriciler kendilerini geliştirebilirler. Bu süreçten korkan geliştiricilerde kod gözden geçirme süreci verimli olmaz. Özellikle genç yazılım geliştiriciler, tecrübeli olanlarla kod gözden geçirme yaptıklarında onların tecrübelerini kendilerine alırlar ve iyi bir yazılım geliştirici olma yolunda hızlı mesafe katederler.

j. Motivasyon bozukluğuna dikkat etmek gereklidir. Kod gözden geçirmeler sonrası akla “Değişiklikler için çok mu zaman harcadım”, ”Çok fazla mı hata çıktı?”, “Bu sonuçlar performans görüşmemi nasıl etkileyecek?” gibi sorular gelebilir. Ölçütler kod gözden geçirme sürecinde temel rol oynarlar, fakat bu ölçütlerin kendilerine karşı kullanıldığını düşünen yazılım geliştiriciler sürece düşman

olurlar ayrıca iyi kod yazmadan ziyade bu ölçütlerin iyi çıkmasıyla ilgilenirler. Eğer ölçütler bir geliştirici ile ilgili genel bir problemi ortaya çıkarmışsa, o kişinin uyarılması tekil şekilde olmamalı, grup halinde ve herkese hitaben yönlendirmeler yapılmalıdır.

k. Kendi kodunu gözden geçirme: Yazılım geliştirici kodunun gözden geçirileceğini bilirse kodun hatasız olması için tekrar tekrar kontrol eder. Akabinde kendi farkettiği hataların düzeltilmesini son sıralara atmaz. Bu davranışın nedeni, yazılım geliştiricinin aslında yaptığı işlerle çevresinden kabul görmesi beklentisidir. Ego efekt sayesinde yazılım geliştirici diğerlerinin kodda aradığı ölçütleri ve tarzı kodunda yaratmaya çalışır, basit hatalar yapmaktan ve tecrübesiz durumuna düşmekten kaçınır.

l. Hafif kod gözden geçirmeleri formal kod gözden geçirmeleri kadar efektif, ve onlardan daha hızlıdır.

Biçimsel kod gözden geçirmeleri 30 yılı aşkın kullanılmaktadır ve geçerliliğini yitirmiştir.Ortalama biçimsel kod gözden geçirmelerde 9 saat başına ancak ve ancak 200 satır kod incelenebilmektedir. Bu süreç içerisinde birden fazla kişi bir çok toplantıda detaylara ve dokümanlara boğulur. Günümüz organizasyonları bu kadar çok kişinin bu iş için bağlı kalmasına olanak vermemektedir.

Cisco çalışmasında biçimsel olmayan sürecin biçimsel olana 1'e 5 oranla daha çabuk sonuçlandığı ve biçimsel süreçlerle aynı miktarda hata yakaladıkları görülmüştür.

2.4.2 Kod Gözden Geçirmeleri Yararları

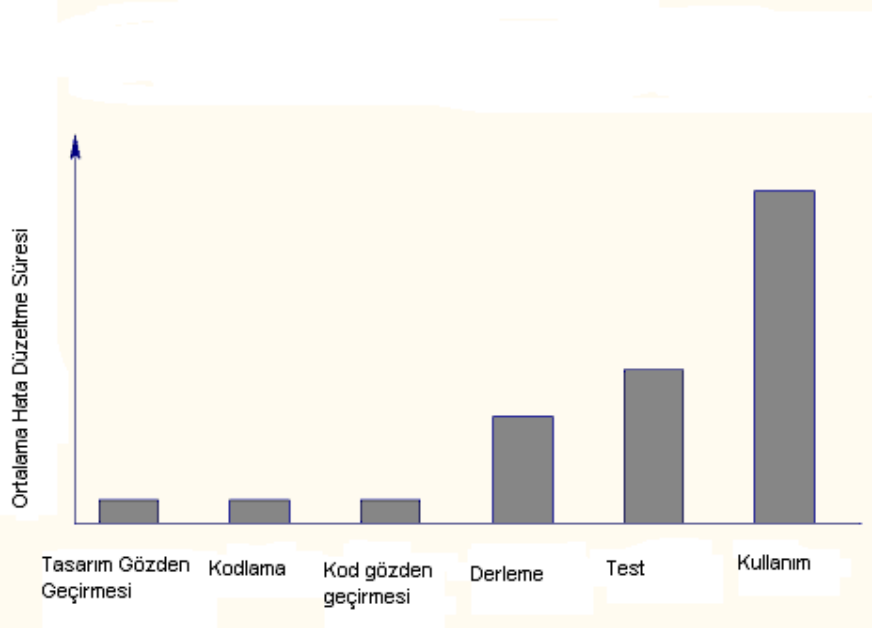
Çoğu şirket iyi ve pratik bir kod gözden geçirmenin özellikle denetimlere nazaran, sonuçlarının karşılanılan maliyete oranla ağır geldiğini öğrenmiştir.

Bu konudaki yapılan çalışmalardan bazılarının sonuçları aşağıdaki gibidir.

- Jet Propulsion Laboratory şirketinin NASA için geliştirdiği bir yazılım üzerinde yaptığı 300 kod gözden geçirmenin şirkete getirisinin 7.5 milyon dolar tahmin edilmektedir.
- Bir başka çalışma müşteri ortamında bulunan hataların 2900 dolara mal olduğu fakat kod gözden geçirmeler ile bulunan hataların 146 dolara mal olduğu ve toplamda 2.5 milyon dolarlık yıllık bir kazanç sağlanmıştır.
- Hewlett-Packard denetim programının bire on kazanç dönüşümü alınmıştır.

- Aetna Sigorta Şirketi ve AT&T Bell Laboratuarları kod gözden geçirmelere geçilmesiyle kod üretkenliğinin artmıştır [15].
- ModSoft'da 6 yıl süreyle yapılmış çalışmada proje maliyetine %1-%2 arasında bir maliyet eklenerek problem gözlenmesinde %40 düşüş yaşandığını görülmüştür [16].

Bu çalışmalardan da görüldüğü gibi üretkenlik kod gözden geçirmelerle beraber artmıştır. Bunun nedeni erken safhadan yakalanan hataların düzeltilmesinin maliyetinin, sonradan yakalanma maliyetinden çok daha az oluşudur.



Şekil 2.7 : Yazılım Geliştirme Aşamalarında Hata Maliyeti Seviyeleri

2.5 Kod Düzenlemeleri

Yazılım ölçütlerinin yazılım kalitesini gösterdikleri kanıtlanmış ve yazılım kalite değerlendirme metotları içerisinde geniş kapsamda kullanılmaktadır. Bu değerlendirme metotlarının sonuçları sonucunda yazılım sistemi içerisinde bazı parçaların tekrar oluşturulması gerektiği kanısına varılabilir. Tekrar oluşturma genellikle bizim tekrar düzenleme(refactoring) dediğimiz yöntemle sağlanır. Tekrar düzenleme, yazılım sisteminde, kodun dış davranışın değiştirilmeden iç yapısının geliştirilmesi süreci şeklinde tanımlanır.

Tekrar düzenleme, yazılım parçasının kalitesini arttırmak için kullanılan en önemli araçların başında gelir. Sistemin tasarım ve kaynak kodu seviyesinde karmaşıklığını

azaltmayı hedefler. Böylece yazılım geliştiricinin üretkenliğinde az maliyetle evrim sağlar ve tasarım hatalarına daha az yer bırakır. Tekrar düzenleme uygularken, yazılım mühendislerinin sorunları keşfettiği yer tekrar düzenlemenin uygulanma aşamasıdır. Fowler, koddaki problemli ve kötü koku gelen alanların bulunmasını insan sezgisine bağlar. Yazılım ölçütlerinin kullanılması ise bu sezginin desteklenmesi açısından gereklidir [17].

3. UYGULAMA

Şirketteki kod kalitesini arttırmak, kod kalite alanına olan ilginin arttırılması ve bunların takip edilmesi amacıyla bir dizi aksiyon alınmıştır.

Şirkette geliştirilmekte olan bazı projelerde yazılım ölçüt toplama yazılımı kullanılarak

- Henderson'un LCOM ölçütü
- Sınıftaki metot sayısı(Chidamber&Kemerer)
- Döngüsel karmaşıklık(McCall's Cyclomatic Complexity)
- Dışarıya Bağlılık
- İçeriye Bağlılık
- Sınıfın çocuk sayısı
- Sınıfın kalıtım ağacındaki derinliği

Ölçütlerinin değerleri alınmıştır. Ayrıca yazılım geliştiricilerin kod gözden geçirmelerine olan alakalarının ölçülmesi için İlgi anketi (Stages of Concern) şirket genelindeki yazılımcılara gönderilmiştir.

Veri toplama aşamaları tamamlandığında şirket tarafından yayınlanan yazılım kalitesi eğitimleri düzenlenmiştir. Katılımcılar eğitimlerde;

- Yazılım kalitesi
- Yazılım kalite faktörleri
- Kod kalitesinin önemi
- McCall, Henderson ve Chidamber&Kemerer ölçütleri
- Kod ölçütü toplama ve değerlendirme programı ile kaliteli kod yazımında kullanılan araçlar

Konularında bilgilendirilmiştir. Bununla beraber yapılan kod gözden geçirme çalışmalarında yazılım ölçüt toplama aracının kullanılması sağlanmıştır.

Eğitim ve kod gözden geçirmeleri sonucunda ilk safhada yapılan ölçümler tekrar yapılarak hem kod kalitesinin hem de yazılım geliştiricilere sağladığı ilgi artışı değerlendirilebildi.

3.1 Uygulamanın Yapılacağı Organizasyonun Tanıtılması

Veripark, e-business, ERP ve mobil çözümlerde uzmanlaşmış olan bir danışmanlık ve teknoloji şirketidir. Şirket, farklı sektörlerdeki kurumsal şirketler için yazılım çözümleri sunar. Veripark, çevrimiçi finans, özel B2B değişimleri ve B2C vitrinlerden mobil çözümlere kadar uzanan geniş bir ürün yelpazesi vardır. Finans, perakende, medya, telekom, otomotiv, tıp ve fabrikasyon alanlarında kullanılmak üzere B2C,B2B,B2E çözümler sunar. Şirketin şu anda çalışmakta olduğu sektörlerden bazıları şunlardır:

- Finans

Online Finans, İnternet Bankacılığı, Çağrı Merkezi, GPRS Bankacılığı, Mobil Servisler, Dosya Transfer Otomasyonu, İş Akış Otomasyonu

- Üretim/Perakende

Tedarik Zinciri Entegrasyonu, Bankacılık Entegrasyonu, Sipariş/Dağıtım Otomasyonu, Tedarikçi/Müşteri Portalleri, E-Business

- Otomotiv

CRM, ERP

- Medya

Mobil Oyun Gösterileri, Mobil Yarışmalar, Mobil Oylama, SMS uygulamaları, Medya İş Akışı Çözümleri

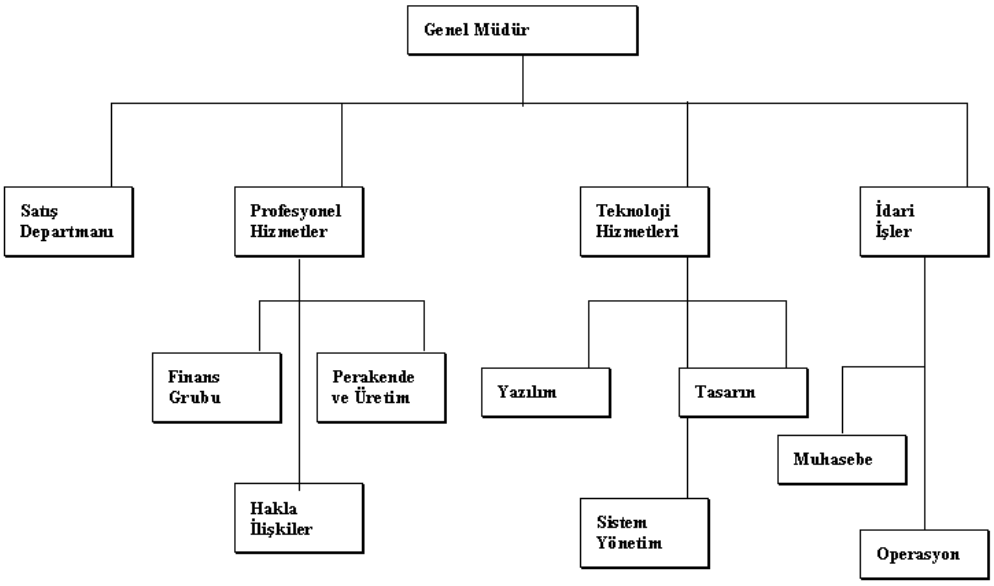
- Telekom

Logo ve Melodi Servisleri, Taşeronluk

3.1.1 Şirket Tarihçesi

Veripark A.Ş 1998 yılında Boğaziçi Üniversitesi mezunu üç ortak tarafından kuruldu. İlk olarak KOSGEB'in desteğiyle Boğaziçi Üniversitesi kampüsü içinde faaliyetlerine başladı. PamukBank İnternet Bankacılığı projesi ile sektör içinde yer buldu ve 6 yılda çevrimiçi çözümler alanında Türkiye'nin lider kuruluşları arasında yerini aldı. Misyonu; doğru yöntemlerle, doğru kalitede, doğru araçlar, metodolojiler ve teknolojiler kullanılarak yazılım geliştirmek ve zamanında müşterilere ulaştırmaktır.

3.1.2 Şirket Organizasyon Şeması



Şekil 3.1 : Şirket Organizasyon Şeması

Veripark'da çalışanlar:

2008 yılı itibariyle 60 çalışan

- 16 uzman yazılım geliştirici
- 24 yazılım geliştirici
- 4 proje/ürün yöneticisi
- 5 destek mühendisi
- 1 satış direktörü, 2 satış temsilcisi
- Taşeron elemanlar (Outsource)

3.1.3 Yazılım Kalite Çalışmaları

İki ayda bir yazılım geliştirici ve codewalk uygulanacak projedeki teknik bilgiye sahip proje yöneticisi toplantı yaparlar. Takriben 1-2 saat süren bu toplantıda aşağıdaki işlemler yapılır.

2. Developer tarafından projede daha önce codewalk yapılmamış iş akışı seçilir. Aşağıdaki gibi işler bu sınıfa girer.
    - Internet bankacılığında para transferi işi

- Gayrimenkul satışında ev rezervasyonu
- Internet üzerinden satış esnasında yaşanan stok işlemleri

3. Proje yöneticisi ile akış üzerinden gidilir ve bu akış üzerindeki kodlar incelenir. Kodlar incelenirken genel yazım kuralları ile yazılan kodun efektif bir biçimde yazılıp yazılmadığı kontrol edilir, önerilerde bulunulur.
4. Kod gözden geçirme sonucunda proje yöneticisinin görüşü doğrultusunda çeşitli konularda puanlama yapılır.
5. Yazılım geliştirici kendisine söylenen noktalarda gerekli düzenlemeleri yapar ve bundan sonra yazdığı kodlarda bu noktalara dikkat eder.

3.2 Kod kalitesi ölçümü

Eski yapıdan farklı olarak kod gözden geçirmeye başlamadan önce ve projede yazılım geliştirme safhasında projenin ölçülebilen yazılım kriterleri bir yazılım uygulaması vasıtasıyla ölçülür şekile getirilmiştir. Bu yöntemle hem kodu gözden geçiren kişinin hem de kodu yazan kişinin kendi belirledikleri alanlar dışında nerelere bakmaları gerektiği, hangi bölgelerin riskli olduğu ve yeniden düzenlenmesi gerektiği yakalanır.

Yazılım ölçüm aracının kullanılmasıyla subjektif ölçümde iyileştirmesi tasarlanan konular aşağıdaki gibidir.

1. Subjektif ölçümde, özellikle zaman ve kaynak kısıtı olan kod gözden geçirmelerde, hataları atlanan yerler çok olmaktadır. Ölçütleri toplayan araç bize bakmamız gereken veya atladığımız başka alanlara da işaret eder.
2. Subjektif değerlendirmede tüm projeler o projenin proje yöneticisi tarafından değerlendirildiğinden hangi projenin daha kaliteli olduğu ve buna bağlı olarak kalite iyileştirmesi için nereye yüklenilmesi gerektiği belirlenemez. Objektif ölçüm projeleri ortak bir paydada değerlendirmemize olanak tanır.
3. Subjektif ölçüm yapan kişi o anki ruh haline bağlı olarak veya kod gözden geçirme yaptığı kişiye olan önyargısından dolayı ölçümde hatalı davranabilir.
4. Subjektif ölçüm yapıldıktan sonra iki ay beklenmektedir. Bu süre zarfı içerisinde yazılım aracını kullanan yazılım geliştirici istediği zaman kodunun kalite ölçütlerini toplayıp onları düzenleme arayışına girebilir.

5. Objektif ölçümde sonuçları daha net görebilen yazılım geliştiricinin ve proje yöneticisinin yazılım kalitesi alanına daha sıcak bakması ve bu konuda motive olmaları sağlanabilir.

3.2.1 Ölçüm araçları

Yazılım ölçüm aracı olarak;

- Kolay kullanımı
- Akademik lisanslamasının bedava olması
- Yaygın kullanımı
- Neredeyse tüm projelerin kullandığı C# dilini desteklemesi
- İncelecek ölçütleri ölçen yapısı
- Ölçüm sınırı olmaması

Nedeniyle ölçümlerde NDepend programı kullanıldı. İlk ölçümlerde 2.6.1, ikinci ölçümlerde 2.8.1 sürümü kullanıldı Vil, FxCop, PreFast gibi kod ölçüm araçları yukarıdaki maddelerden bir veya birkaçını karşılayamadığından kullanılmadılar.

NDepend, yazılım geliştiricilerin. NET tabanlı projelerinde kod yapısını analiz etme, tasarım kurallarını uygulama, büyük yapılandırma planlarına yardımcı olma ve efektif kod yazma alanlarında kullanılması için geliştirilmiş bir yazılımdır.

Programa verilen. NET proje Devingen Bağlı Kitaplıkları(DLL)'lerini analiz ederek bunlarla ilgili çeşitli ölçümleri kullanıcıya rapor olarak sunar. Projede bu raporlardan faydalanılarak değerlendirmeler yapılmıştır. Projelerde kale alınan sınıflar yazılım geliştiricinin o an ilgilendiği ve geliştirme yaptığı sınıflar ile o sınıfın kullandığı diğer sınıflardır.

Ndepend ölçüm aracı Chidamber&Kemerer ölçütlerinden sınıfın cevabı (response for a class) ölçütünü ölçmediğinden bu ölçüt kalite değerlendirmesinden çıkartıldı. Ayrıca Chidamber Kemerer'in nesne bağlılık ölçütü NDepend tarafından iki taraflı değerlendirilecek şekilde bölünmüştür. Çalışmada bu ölçütler dikkate alınmıştır.

İçeriye bağlılık (Afferent Coupling): Bir sınıfa bağlı olan (kullanıldığı) sınıf sayısı, o sınıfın içeri bağlılığını verir.

Dışarıya bağlılık (Efferent Coupling): Bir sınıfın bağlı olduğu (kullandığı) sınıf sayısı, o sınıfın dışarı bağlılığını verir.

Çalışma sonucunda ölçüt değerlerinin hepsinde düşüş beklenmekedir.

3.3 Kod gözden geçirme ilgi ölçümü

Şirketteki ilgi ölçümleri, ilgi aşamaları anketiyle yapılmıştır. İlgi aşamaları anketi Hall, George ve Rutherford tarafından 1974 yılında oluşturulmuştur. Anketin amacı organizasyonda kullanılan bir innovasyona kişilerin ilgilenme düzeylerini belirlemek, ilginin arttırılması amacıyla hangi konuya parmak basılması gerektiği konusunda yol göstermektir. Bu anketin soruları, kod gözden geçirmeleri konusundaki ilgiyi ölçmek amacıyla ilgi aşamaları anketindeki öneriler doğrultusunda değiştirilmiştir. Orijinal anket fakültedeki öğretmenlere gönderilen 195 öğeden geliştirilmiştir. Üç yüz elli dokuz anket geri, dönmüştür. Öğe korelasyon ve faktör analizlerinin gerçeklemesinden sonra %60 ortak varyans ile yedi ayrı faktör açıklanmıştır. Bu yedi faktörü belirleyen 35 öğe seçilmiştir. Anket iki yıl boyunca çapraz bölümlerde boylamasına çalışmalarda 11 çeşitli yenilik üzerinde kullanılmıştır. Anketin geçerliliği 1974 yılında 830 öğretim görevlisi ve profesör ile kanıtlamıştır. İçsel güvenirlik katsayısı ilk yönetimde 0,6 -0,83 ikincisinde 0,65 – 0,86 arasında çıkmıştır.

Anket sorularının anlaşılması açısından sorular anlamını yitirmeyecek şekilde kod gözden geçirmelerine (şirket içerisinde bilinen adıyla codewalk) çevrildi ve bu şekilde katılımcılarla paylaşıldı. Anket sonucunda katılımcıların tüm seviyelerde kod gözden geçirmelerine olan ilgilerinin artacağı düşünülmüştür.

Bir yenilik hakkındaki ilginin yedi aşaması aşağıdaki gibidir.

3.3.1 Farkındalık

Yeniliğin ne olduğu konusunda bilgi sahibi olma aşamasına farkındalık(awareness) aşaması denir, yeniliğin kendisine olan ilgiyi ölçer. Kişinin yenilik konusundaki kulak dolgunluğu farkındalık seviyesini verir. Kişinin yenilik hakkında “bu nedir?” sorusunu yöneltme isteğini ölçer. Kişinin yenilikle ilgili bilgi seviyesiyle alakalıdır.

3.3.2 Bilgisel

Bu aşama kişinin yenilik hakkındaki detaylı bilgiye olan ilgisini ortaya çıkartır. Kişinin yenilik hakkında “Bu nasıl oluyor?” sorusunu yöneltme isteğini ölçer. Kişinin yenilikle ilgili bilgi seviyesiyle alakalıdır.

3.3.3 Kişisel

Yeniliğin uygulanmasında kendisinden beklenenler konusunda ne istendiğini, istenenleri yapıp yapamayacağı, rolünün ne olacağı gibi konuları ölçer. Kişinin bu yeniliği uygulamasına olan ilgisini ölçer. "Bunu kullanmak beni nasıl etkiler? Buradaki rolüm nedir?" sorularının yöneltme isteği ölçülür.

3.3.4 Yönetim

Kişinin yeniliği uygulamak için gereken verimlilik, düzenleme, yönetim, zamanlama ve kaynak planlaması gibi olan konulara olan ilgisini ölçer. Yeniliğin nasıl uygulanacağıyla ilgilenir. Yönetim seviyesi kişinin yeniliği öğrenme görevine odaklanmıştır. "Bunu nasıl öğrenirim? Ne kadar zaman/kaynak gerekir?" sorularını yöneltme isteği yönetim aşamasında ölçülür.

3.3.5 Netice

Kişinin bu yenilik sonucunda beklentilerini ortaya çıkarır. Yeniliğin sonuçlarının o kişi için önemli olup olmadığı ölçülür. Odaklanan nokta teknolojinin kişiler için getirdikleridir. "Bu gerekli midir? Bunu kullanmam işimde faydalı olacak mı?" gibi soruların cevapları netice aşamasında verilmektedir.

3.3.6 İş Birliği

Yenilik konusunda kişilerin beraber çalışma, koordinasyon ve araştırma isteklerini ölçer. Kişilerin yenilikle ilgili dış sistemlerde neler yapıldığıyla ne kadar ilgilendiklerini ve yeniliği şirket içinde ne kadar yaymak istediklerini belirler. Kişilerin "Başkaları bu konuda neler yapıyor?" sorusunu yöneltme isteğini, iş birliği aşaması ölçer. Odaklanan nokta yeniliğin getirdikleridir.

3.3.7 Tekrar Odaklanma

Yenilik konusunda alternatif arama ve sorgulama yönelimlerini ölçer. Kişinin tekrar odaklanma seviyesi ne kadar fazla ise, yeniliği geliştirmedeki isteği o kadar fazladır. Tekrar odaklanması yüksek olan kişi "Bu işin yapılması için daha iyi bir yol var mı?" sorusunu sorar.

Ankette bu 7 faktörü (aşamayı) oluşturan kullanılan 35 soru ekte verilmiştir.

4. UYGULAMA SONUCU ELDE EDİLEN BULGULAR

4.1 Kod kalitesi ölçüm bulguları

İlk kalite ölçümleri Aralık 2007'de yedi proje içerisindeki 2211 sınıfta yapılmıştır. İkinci kalite ölçümleri Nisan 2008 tarihinde dört proje içerisindeki 1932 sınıf içerisinde yapılmıştır. İlk ölçümde olup, ikinci ölçümde olmayan üç projeden biri iptal edilmiş, diğer ikisi üzerinde kayda değer herhangi çalışma yapılmamış olduğundan değerlendirilmemiştir. Diğer dört projede bulunan sınıfların toplu değerlendirilmesi sonucunda kod ölçütleri bazında ve her iki ölçümde de yer alan projeler için sonuçlar aşağıdaki gibi çıkmıştır. Bu dört projeye ait ölçüt değerlerinin değişim grafikleri ekler kısmında belirtilmiştir.

4.1.1 Henderson yapışma yetersizliği(LCOM)

LCOM sınırı Henderson'un belirlediği 0-1 aralığında seçilmiştir. LCOM değeri çok küçük bir miktarda artmıştır (0,5616'dan 0,5753'e). Bu artma tek yönlü varyans analizi uygulandığında bu artış anlamlı çıkmamıştır. Azalışın gerçekleşmemesinde LCOM ölçütünün yazılım geliştiriciler arasında kavram olarak tam oturmaması, anlayanların da bu ölçüte müdahelenin zorluğu nedeniyle odaklanmamış olmaları gösterilebilir.

Descriptives

		N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean		Minimum	Maximum
						Lower Bound	Upper Bound		
LCOM	1	927	,5616	,39246	,01289	,5363	,5869	,00	1,00
	2	811	,5753	,39509	,01387	,5481	,6026	,00	1,00
	Total	1738	,5680	,39363	,00944	,5495	,5865	,00	1,00

Test of Homogeneity of Variances

	Levene Statistic	df1	df2	Sig.
LCOM	,072	1	1736	,789

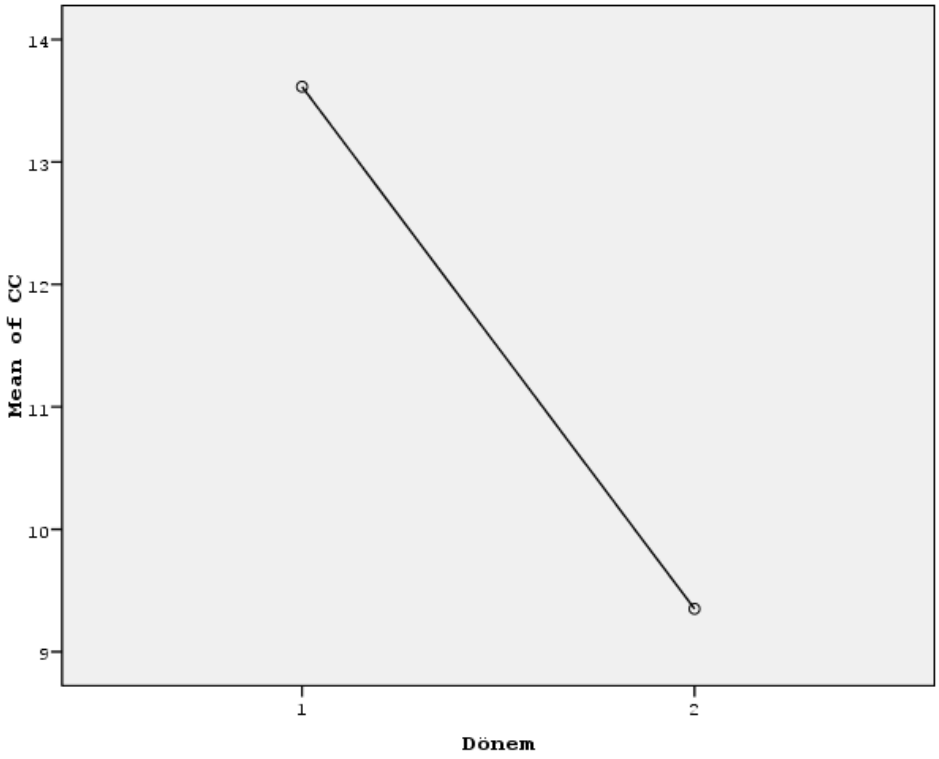
ANOVA

		Sum of Squares	df	Mean Square	F	Sig.
LCOM	Between Groups	,082	1	,082	,528	,468
	Within Groups	269,062	1736	,155		
	Total	269,144	1737			

Şekil 4.1 : Henderson LCOM Analizi

4.1.2 Döngüsel karmaşıklık(Cyclomatic complexity)

Değiştirilebilmesinin nispeten kolay olması özelliği sebebiyle projelerde yazılım geliştiriciler tarafından en çok iyileştirilmeye çalışılan ölçüt olmuş ve bunda başarı sağlanmıştır (13,61 olan karmaşıklık 9,35'e indirgenmiştir).



Şekil 4.2 : Döngüsel Karmaşıklık Değerleri

Bu düşüşe ait tek yönlü varyans analizi yaptığımızda sonucun aşağıdaki gibi anlamlı olduğu görülür.

Descriptives

CC

	N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean		Minimum	Maximum
					Lower Bound	Upper Bound		
1	1910	13,61	64,928	1,486	10,70	16,53	0	1967
2	1923	9,35	46,386	1,058	7,28	11,42	0	1839
Total	3833	11,48	56,426	,911	9,69	13,26	0	1967

Test of Homogeneity of Variances

CC

Levene Statistic	df1	df2	Sig.
10,636	1	3831	,001

ANOVA

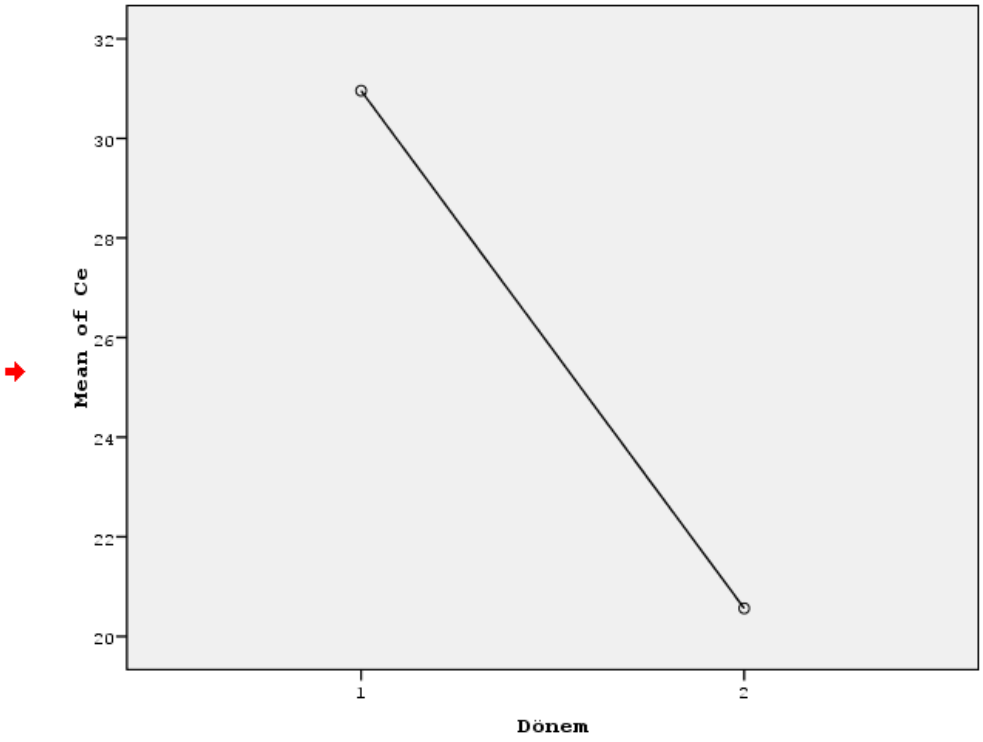
CC

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	17428,014	1	17428,014	5,480	,019
Within Groups	12183214	3831	3180,165		
Total	12200642	3832			

Şekil 4.3 : Döngüsel Karmaşıklık Analizi

4.1.3 Dışarıya bağlılık

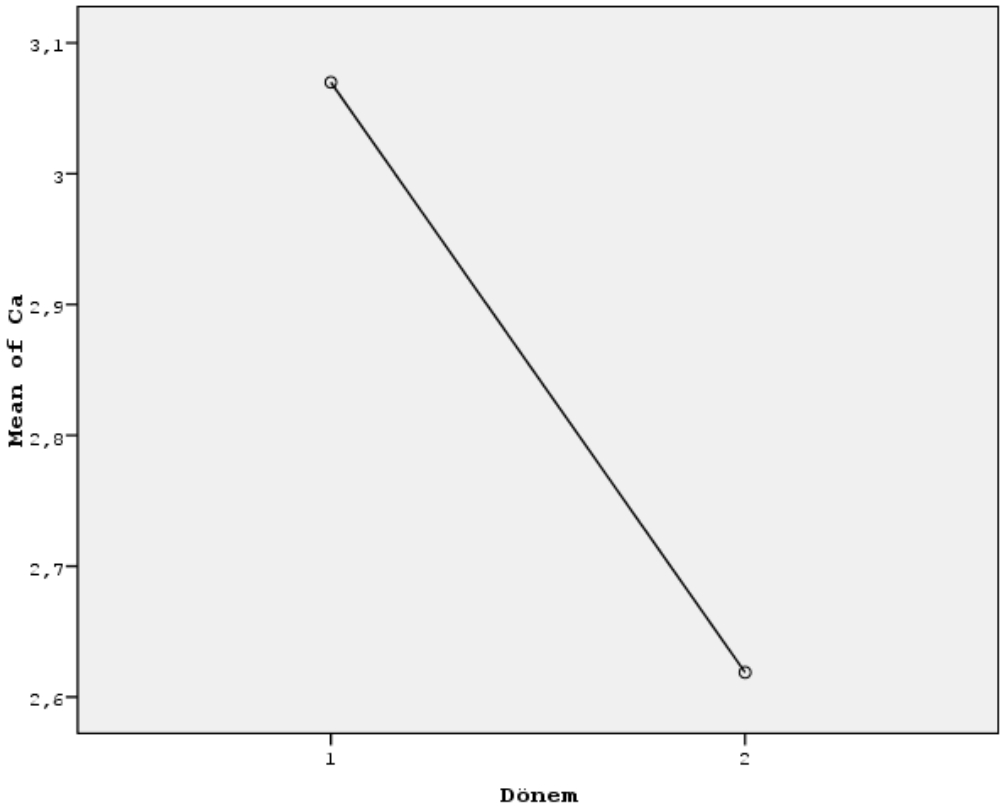
Dışarıya bağlılık (kullandığı sınıf sayısı) ölçütünde büyük oranda iyileşme görülmektedir (30,96'dan 20,56). Azalma istatiksel olarak da anlamlı çıkmıştır. Yapılan eğitimlerde önerilen ortak işleri yapan sınıfların yeniden yapılandırılması ve yeni yazılan kodlarda bunlara dikkat edilmesi, bu ölçütte iyileşme olmasının nedenlerindendir. İlgili istatiksel analiz Şekil 4.7'den takip edilebilir.



Şekil 4.4 : Dışarıya Bağlılık Değerleri

4.1.4 İçeriye bağlılık

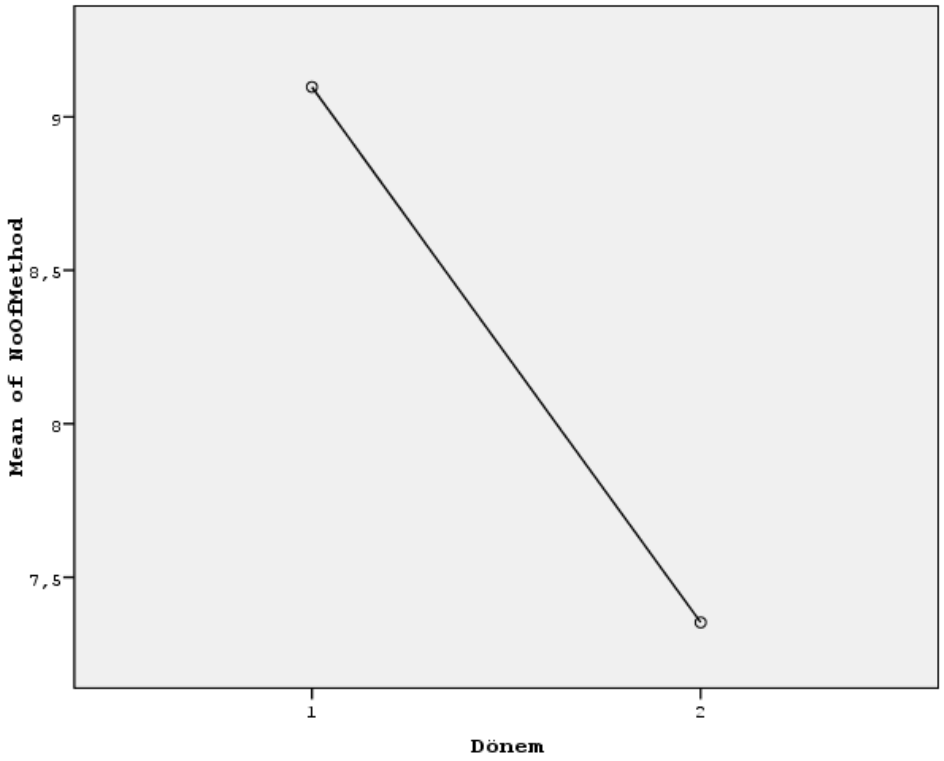
Üzerinde çalışılan projelerin sınıf ve metot yapıları daha önceden belli olduğundan, sadece yeni sınıflarda bu ölçüte dikkat edilmesi az bir etki yaratmıştır (3,07'den 2,62'ye düşüş). Düşüş tek yönlü varyans analizi uygulandığında anlamlı çıkmıştır. İlgili istatiksel analiz Şekil 4.7'den takip edilebilir.



Şekil 4.5 : İçeriye Bağlılık Değerleri

4.1.5 Toplam metot sayısı

Metot sayısında az bir azalma olmasına karşın bu istatiksel bir anlama sahip değildir (9,1'den 7,35'e düşüş). Bunun nedeni daha önceki durumda zaten kod geliştirmesi yapılırken metot sayısının takip edilmesi ve böylece önceki durumda zaten ideal maksimum değerinin (20) çok altında olması, daha fazla gelişecek yanının kalmamasıdır. İlgili istatiksel analiz Şekil 4.7'den takip edilebilir.



Şekil 4.6 : Toplam Metot Sayısı Değerleri

Descriptives

		N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean		Minimum	Maximum
						Lower Bound	Upper Bound		
Ca	1	1975	3,07	6,840	,154	2,77	3,37	0	91
	2	1928	2,62	7,208	,164	2,30	2,94	0	146
	Total	3903	2,85	7,027	,112	2,63	3,07	0	146
Ce	1	1975	30,96	46,515	1,047	28,90	33,01	1	427
	2	1928	20,56	21,965	,500	19,58	21,54	1	569
	Total	3903	25,82	36,876	,590	24,67	26,98	1	569
NoOfMethod	1	1975	9,10	41,894	,943	7,25	10,95	0	1213
	2	1928	7,35	42,821	,975	5,44	9,27	0	1831
	Total	3903	8,24	42,358	,678	6,91	9,56	0	1831

Test of Homogeneity of Variances

	Levene Statistic	df1	df2	Sig.
Ca	,522	1	3901	,470
Ce	394,255	1	3901	,000
NoOfMethod	2,895	1	3901	,089

ANOVA

		Sum of Squares	df	Mean Square	F	Sig.
Ca	Between Groups	198,526	1	198,526	4,024	,045
	Within Groups	192481,2	3901	49,341		
	Total	192679,7	3902			
Ce	Between Groups	105383,4	1	105383,419	79,046	,000
	Within Groups	5200749	3901	1333,183		
	Total	5306132	3902			
NoOfMethod	Between Groups	2969,112	1	2969,112	1,655	,198
	Within Groups	6998046	3901	1793,911		
	Total	7001015	3902			

Şekil 4.7 : Dışarıya Bağlılık, İçeriye Bağlılık ve Metot Sayısı Ölçütlerinin Analizi

4.1.6 Sınıfın çocuk sayısı

Projelerde genellikle bir sınıftan az sayıda kalıtım yapılıp, diğer kalıtımın kalıtılmış sınıftan yapılması nedeniyle çocuk sayısı değeri önceki durumda ve sonraki durumda çok düşük çıkmış(0,3276 ve 0,3261) ve herhangi bir değişme olmamıştır.

Descriptives

NOC

	N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean		Minimum	Maximum
					Lower Bound	Upper Bound		
1	1749	,3276	2,82025	,06744	,1954	,4599	,00	54,00
2	1797	,3261	4,03968	,09530	,1392	,5130	,00	145,00
Total	3546	,3268	3,49136	,05863	,2119	,4418	,00	145,00

Test of Homogeneity of Variances

NOC

Levene Statistic	df1	df2	Sig.
,019	1	3544	,892

ANOVA

NOC

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	,002	1	,002	,000	,990
Within Groups	43212,182	3544	12,193		
Total	43212,184	3545			

Şekil 4.8 : Sınıfın Çocuk Sayısı Analizi

4.1.7 Kalıtım ağacındaki derinliği

Yazılım geliştiricilerin eğitimlerden sonra yapacakları kalıtımı iki defa düşünmeleri ve varolan kalıtım ağaçlarında bölmelere gidilmesi kalıtım ağaç derinliği ölçütünü düşürmüştür(2,87,'den 2,65'e). Sonuçları istatiksel olarak da anlamlı çıkmıştır.

Descriptives

DIT

	N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean		Minimum	Maximum
					Lower Bound	Upper Bound		
1	1949	2,87	1,860	,042	2,79	2,95	1	11
2	1909	2,65	1,412	,032	2,59	2,71	1	6
Total	3858	2,76	1,657	,027	2,71	2,81	1	11

Test of Homogeneity of Variances

DIT

Levene Statistic	df1	df2	Sig.
2,555	1	3856	,110

ANOVA

DIT

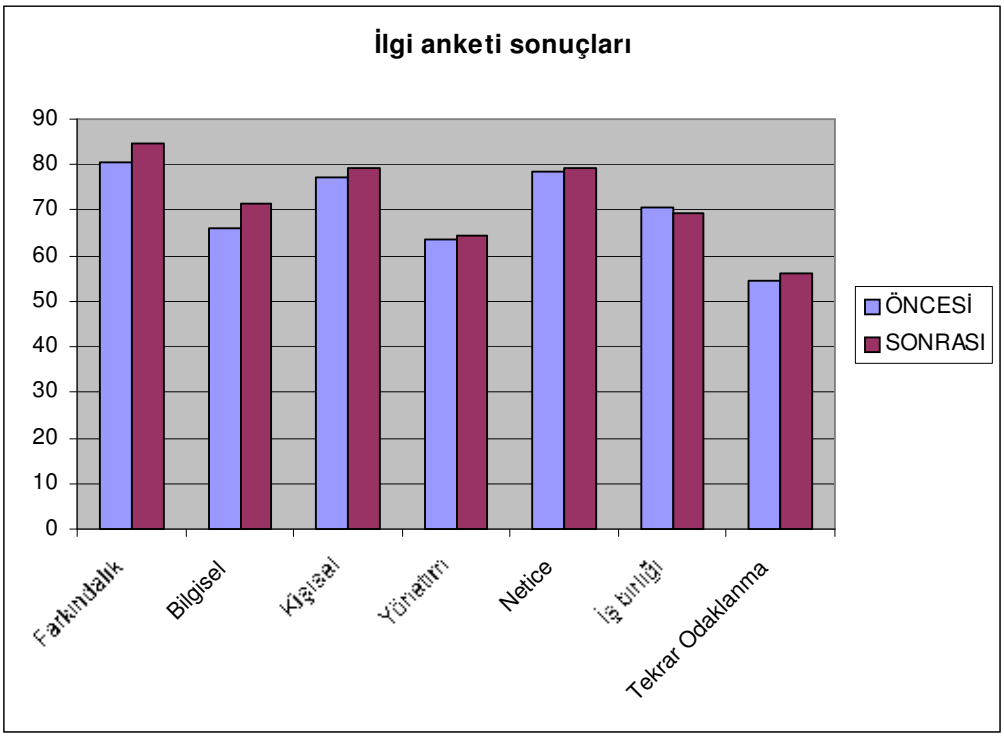
	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	45,848	1	45,848	16,771	,000
Within Groups	10541,287	3856	2,734		
Total	10587,135	3857			

Şekil 4.9 : Kalıtım Ağacı Derinliği Analizi

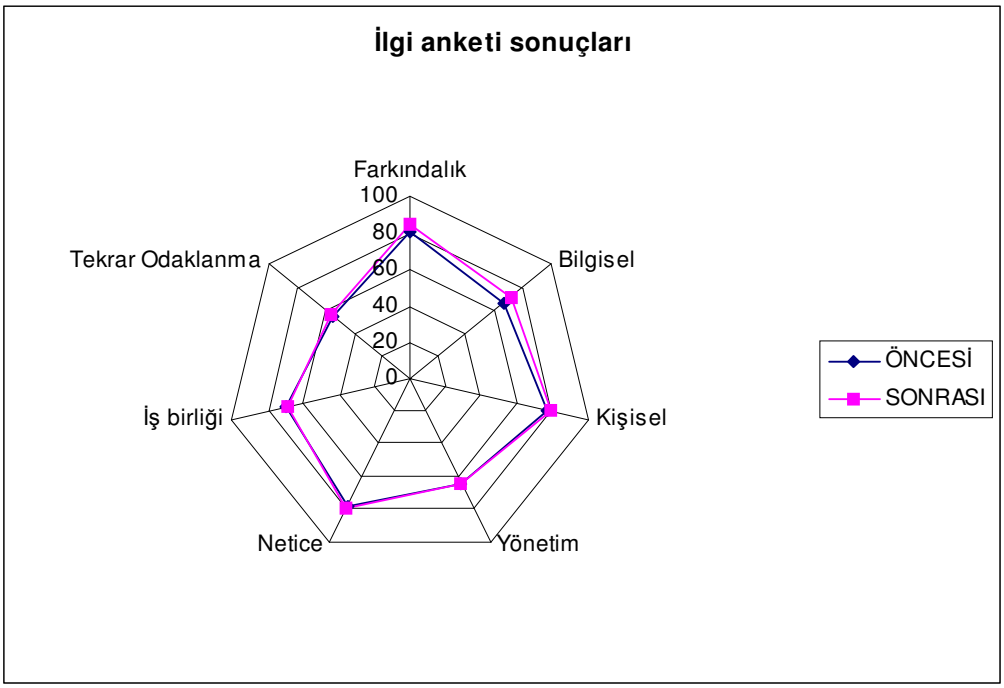
4.2 Kod gözden geçirme ilgi alaka ölçüm bulguları

İlgi anketi uyarlanarak 30 yazılım geliştirici ve proje yöneticisine gönderilmiştir. İlk Ölçümde bunlardan %66lık bir oranda, 20 katılımcı sonuç dönmüşlerdir. İlk ölçümde cevap veren 20 kişiden dördü ayrıldığından dolayı ikinci ölçüme 16 katılımcı katılmış ve hepsi ankete cevap vermişlerdir.

Çıkan bulgularda yazılım geliştirici ve proje yöneticilerinin kod gözden geçirmelerine olan ilgileri farkındalık ve bilgisellik aşamalarında arttırmış, diğer aşamalar için değiştirmemiştir. Çalışanlar kod ölçütlerinin ve bunları ölçen CASE araçlarının kod gözden geçirmelerinde kullanılmasıyla kendilerini bu konuda daha çok geliştirebileceklerini anlamış ve detay bilgi almaya sürüklemiştir. Aynı zamanda kod gözden geçirmeleriyle ilgili en sorunlu bölge olan yönetim aşamasında kayda değer bir gelişme olmamıştır. Bunun nedeni yapılan çalışmanın insanların kod gözden geçirmelerini nasıl öğrenmeleri gerektiğiyle veya buna nasıl zaman hazırlayacaklarıyla ilgili olmamasıdır.



Şekil 4.10 İlgi Anketi Sonuçları Raporu (Sütun)



Şekil 4.11 İlgi Anketi Sonuçları Raporu (Radar)

5. SONUÇLAR VE TARTIŞMA

Kod ölçütlerinin incelenmesiyle kod kalitesinde genel olarak bir artış sağlanıldığı söylenebilir. Döngüsel karmaşıklık, dışarıya bağlılık, içeriye bağlılık ölçütlerinde istatiksel anlamda ilerleme katedilirken, diğer ölçütlerde yaşanan az seviyedeki olumlu düşüş tek yönlü varyans analizinden geçememiştir.

Aynı zamanda kod ölçütlerinin kod gözden geçirme süreçlerinde kullanımı ve bu konuda verilen eğitim, yazılım geliştirici ve proje yöneticilerinin kod gözden geçirmelerine olan ilgileri farkındalık ve bilgisellik aşamalarında arttırmış, diğer aşamalar için değiştirmemiştir. Bu sonuç bize verilen teknik eğitimin kişiler arası paylaşım, yönetim ve süreci geliştirme isteğini arttırmadığını fakat kod gözden geçirmeyle bilgilenme ve keşfetme isteğini arttığını göstermektedir.

Şirket yapılan bu çalışmada gördüğü fayda sayesinde kod kalitesini daha da çok geliştirme amacıyla yeni kod ölçütlerinin ve CASE araçlarının araştırılması için bir araştırma geliştirme çalışması başlatmıştır.

Tez kapsamında yapılan çalışmalar, kod kalite ölçütlerinin kod kalitesi iyileştirme sürecine nasıl etki ettiği konusunda bir örnek teşkil etmiştir.

5.1 Sonraki Çalışma

Tez kapsamındaki çalışmada kod gözden geçirme sürecine kod kalite ölçütlerinin izlenmesi eklenmiştir. Buna benzer bir çalışma daha önce kod gözden geçirmelerin yapılmadığı bir organizasyonda gerçekleştirilirse, kalite artırımı konusunda çok daha fazla verim alınması söz konusu olacaktır. Bu çalışmayı destekler nitelikteki çalışmalar diğer kod ölçütleri ve CASE araçlarının kullanımıyla da yapılabilir.

KAYNAKLAR

[1] **Pressman, S.,** 2005. Software Engineering: A Practitioner's Approach Sixth Edition, pp.746, McGraw-Hill Education.

[2] **Pressman, S.,** 2005. Software Engineering: A Practitioner's Approach Sixth Edition, pp.388, McGraw-Hill Education.

[3] **Petrasch, R.,** 1999. The Definition of ‚Software Quality': A Practical Approach, *10th International Symposium on Software Reliability Engineering (ISSRE)*, Kasım.

[4] **William A., Ward, Jr., Venkatamaran, B**., 1999. Some observations on software quality, *37th annual Southeast regional conference (ACM)*, Nisan.

[5] **Peskin I, Myron., Hart, J, James**., 1996. Measuring the quality of computer systems development, Benchmarking for Quality Management & Technology 3(2), pp. 68-82, MCB University Press.

[6] **Dhillon, B, S.,** 2007. Applied Reliabilty and Quality, pp.151-164, Springer London.

.

[7] **Millcic, D.,** "Software Quality Models and Philosophies". Blekinge Institute Of Technology. 4 Kasım 2007. <http://www.bth.se/tek/besq.nsf/(WebFiles)/CF1C3230DB425EDCC125706900317C44/$FILE/chapter_1.pdf>.

[8] **El-Amam, K.,** 2001. Object Oriented Metrics: A Review of Theory And Practice, *National Research Council Canada*, **1085,** NRC 44190.

[9] **Anderson, M., Vestergren, P.,** 2004. Object Oriented Design Quality Metrics, *PhD Thesis,* Upsalla University, Information Technology, Sweden.

[10] **Mäkelä, S., Leppänen, V .,** 2006. Observations on lack of cohesion metrics, *International Conference on Computer Systems and Technologies (CompSysTech)*, Haziran.

[11] **Fernandez, L., Pena, R .,** 2006. A Sensitive Metric Of Class Cohesion, *International Journal "Information Theories & Applications" Vol.13* , pp.82, Institute for Information Theories and Applications.

[12] **Assassa, G.,** "Literature Review in Object-Oriented Design Metrics". 3 Nisan 2008. <http://faculty.ksu.edu.sa/ghazy/CBD/ch0.pdf >.

[13] **Quenel, G., Lövdahl H.,**"Object oriented software quality models". 22 Şubat 2008. <http://web.abo.fi/~kaisa/LQ.pdf >.

[14] **Cohen, J.,** 2006. Best Kept Secrets of Peer Code Review, Smartbearsoftware.com.

[15] **Wiegers, K., Moore P.,**" Lightweight Tool Support For Effective Code Reviews". 18 Mart 2008. <http://www.atlassian.com/software/crucible/learn/codereviewwhitepaper.pdf>.

[16] **Baker, R, A.,** 1997. Code review enhance software quality, *19th International Conference on Software Engineering,* Boston, Massachusetts, United States, 17-23 Mayıs, s.570-571.

[17] **Stroggylos, K., Spinellis D.,** 2007. Refactoring- Does it improve software quality, *Fifth International Workshop on Software Quality,* Minneapolis, MN, 20-26 Mayıs, s.10.

[18] **Gibson, D., Goldenson, D.. Kost, K.,**"Performance Results of CMMI-Based Process Improvement". Ağustos 2006. <http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tr004.pdf>.

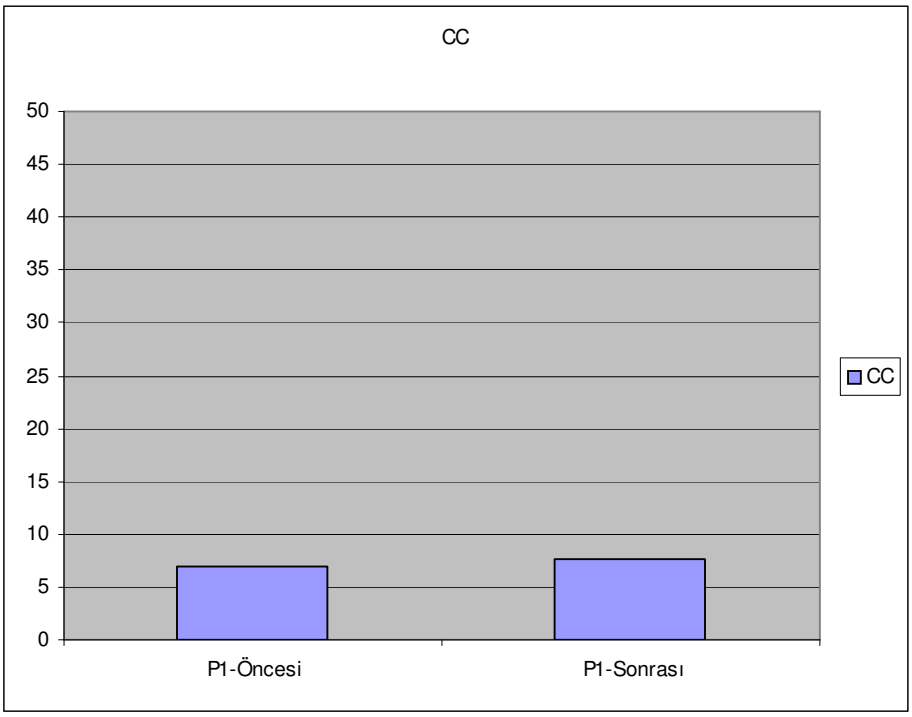
[19] **Laing, V.. Coleman, C.,** 2001. Principal Components of Orthogonal Object-Oriented Metrics, White Paper Analyzing Results of NASA Object-Oriented Data, SATC NASA.

[20] **Wiegers, K.,** "When Two Eyes Aren't Enough". 18 Mart 2008. <http://www.processimpact.com/articles/two_eyes.html>.

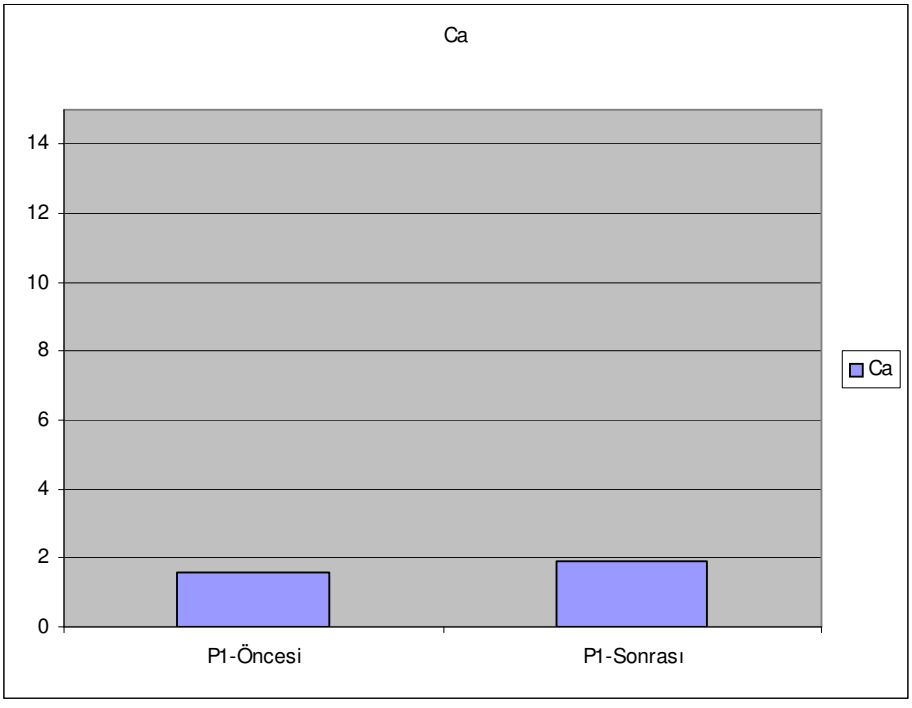
EKLER

EK.A
P1 Projesi Değerleri

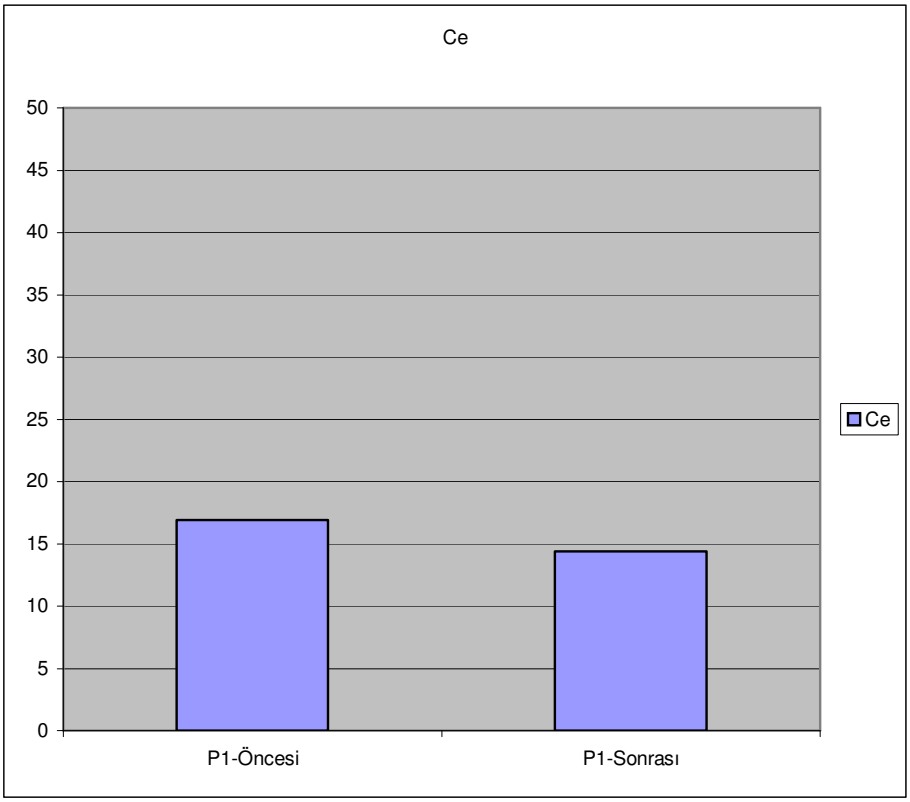
Şekil A.1 P1-Henderson Yapışma Yetersizliği Değerleri



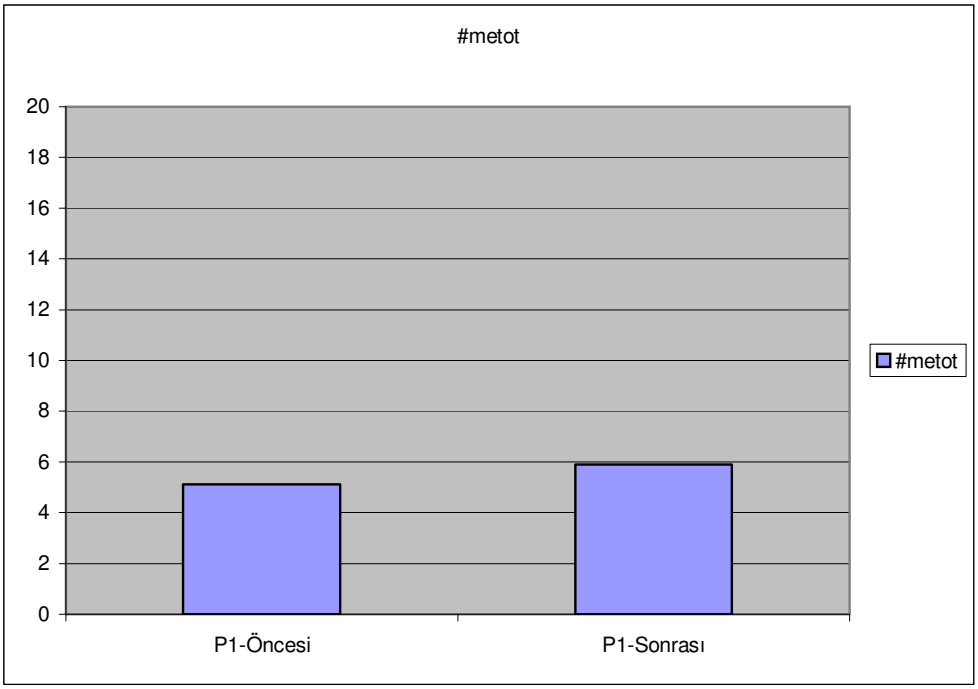
Şekil A.2 P1-Döngüsel Karmaşıklık Değerleri



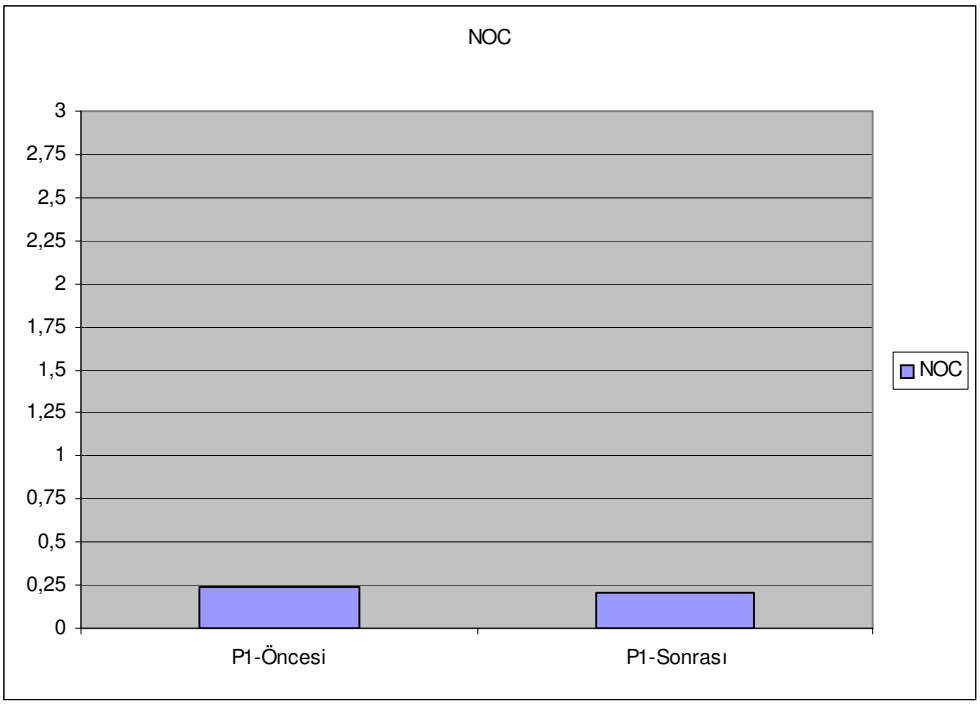
Şekil A.3 P1-İçeriye Bağlılık Değerleri



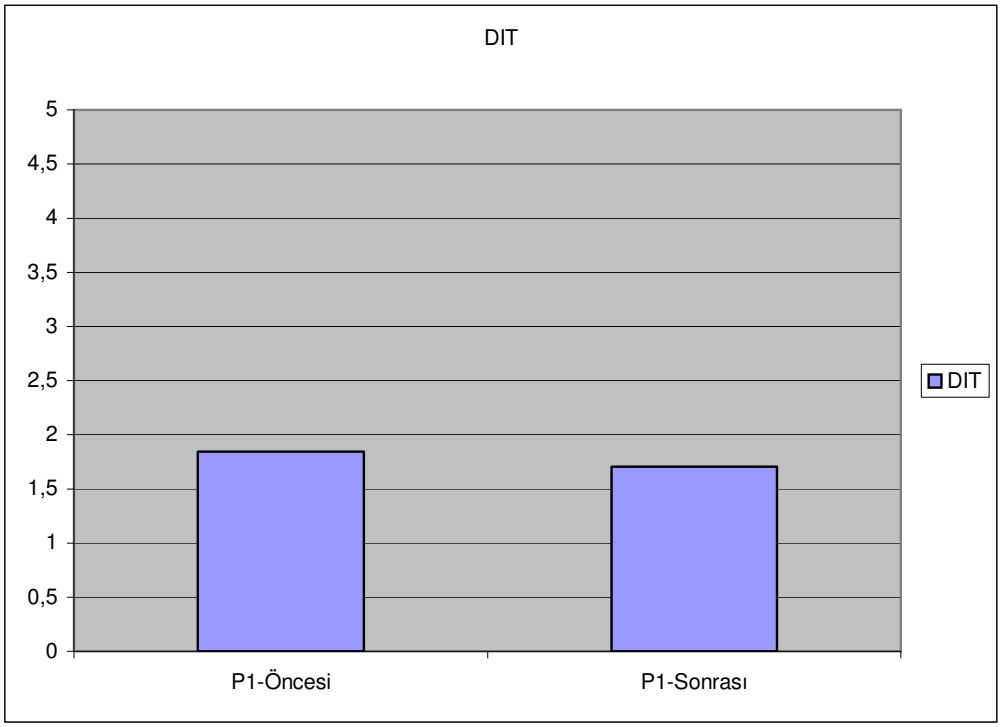
Şekil A.4 P1-Dışarıya Bağlılık Değerleri



Şekil A.5 P1-Metot Sayısı Değerleri



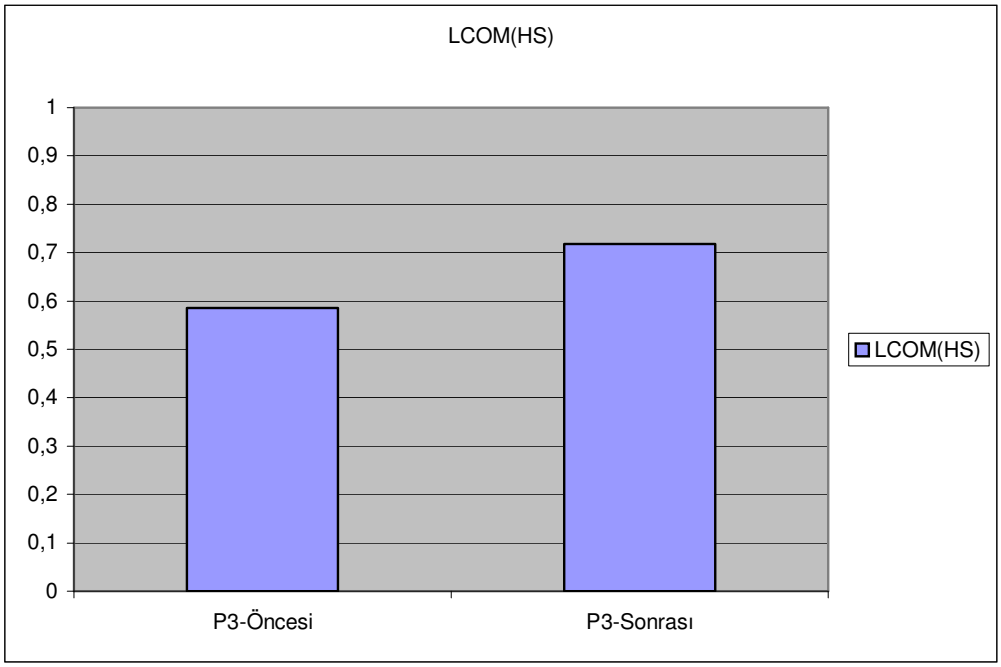
Şekil A.6 P1-Sınıfın Çocuk Sayısı Değerleri



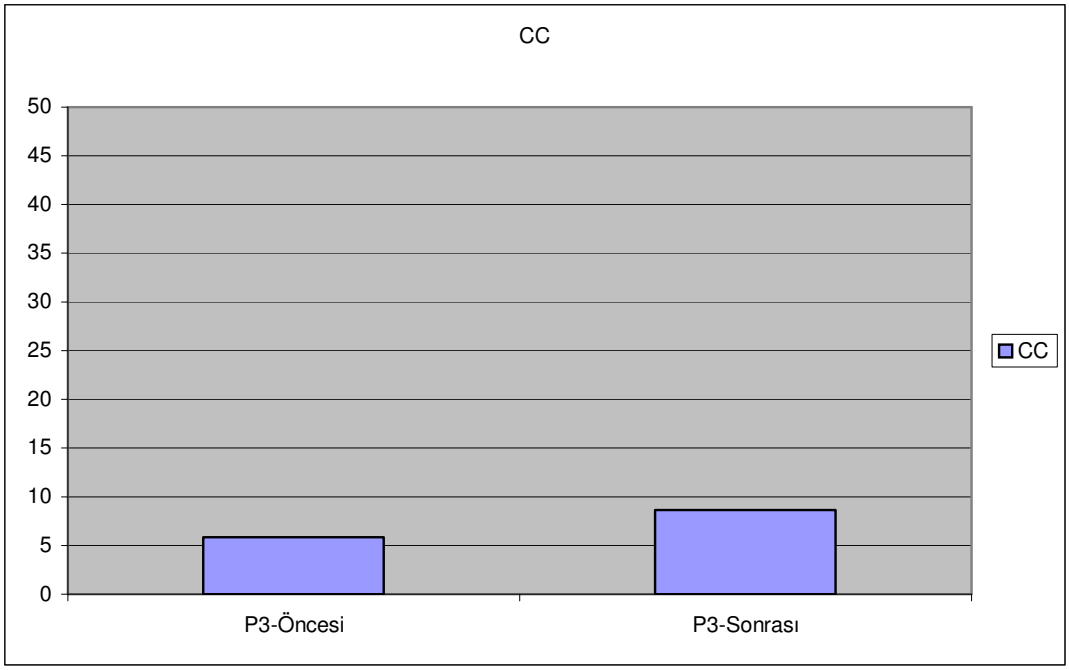
Şekil A.7 P1-Kalıtım Ağacındaki Yeri Değerleri

EK-B

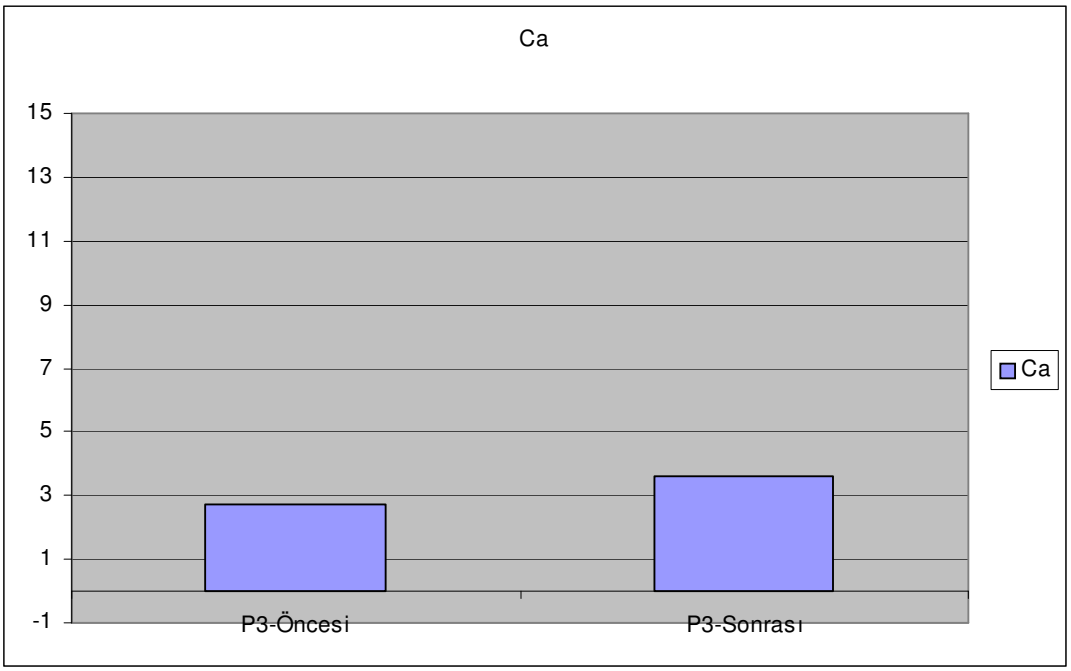
P3 Projesi Değerleri:



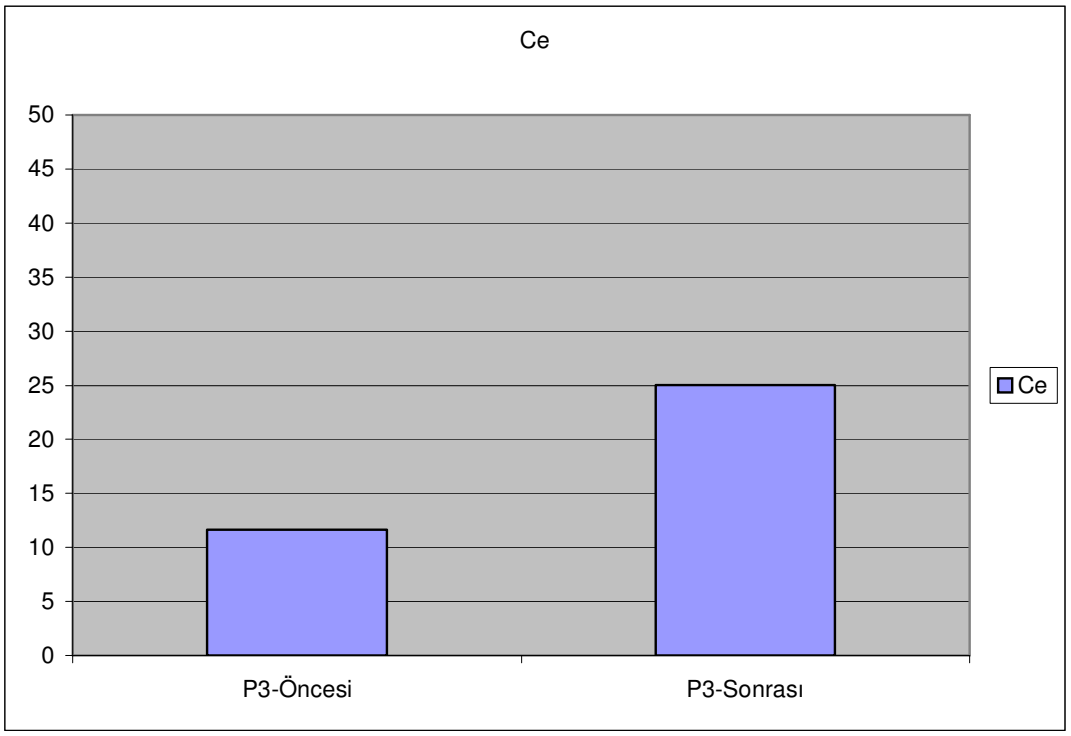
Şekil B.1 P3-Henderson Yapışma Yetersizliği Değerleri



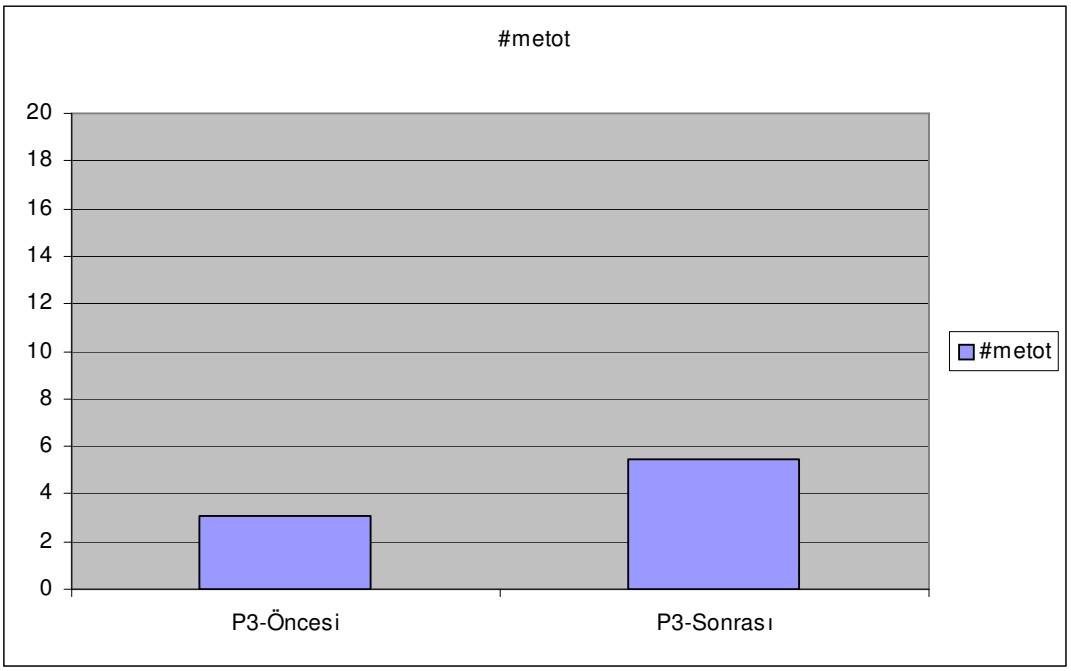
Şekil B.2 P3-Döngüsel Karmaşıklık Değerleri



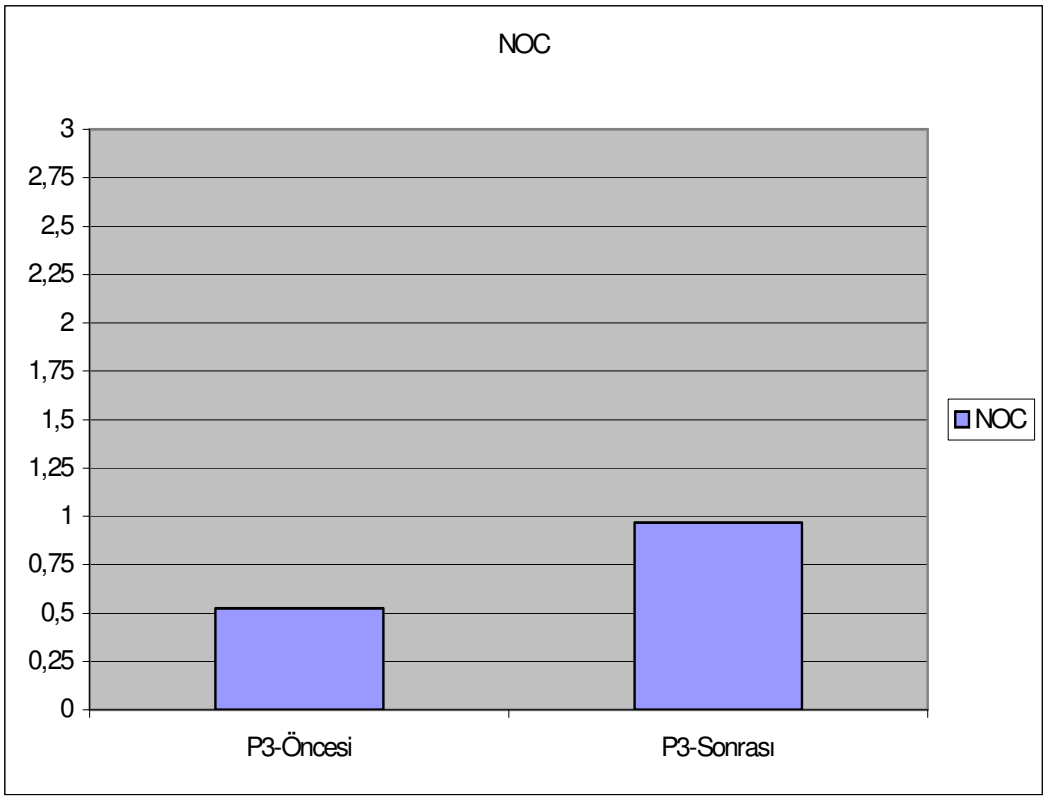
Şekil B.3 P3-İçeriye Bağlılık Değerleri



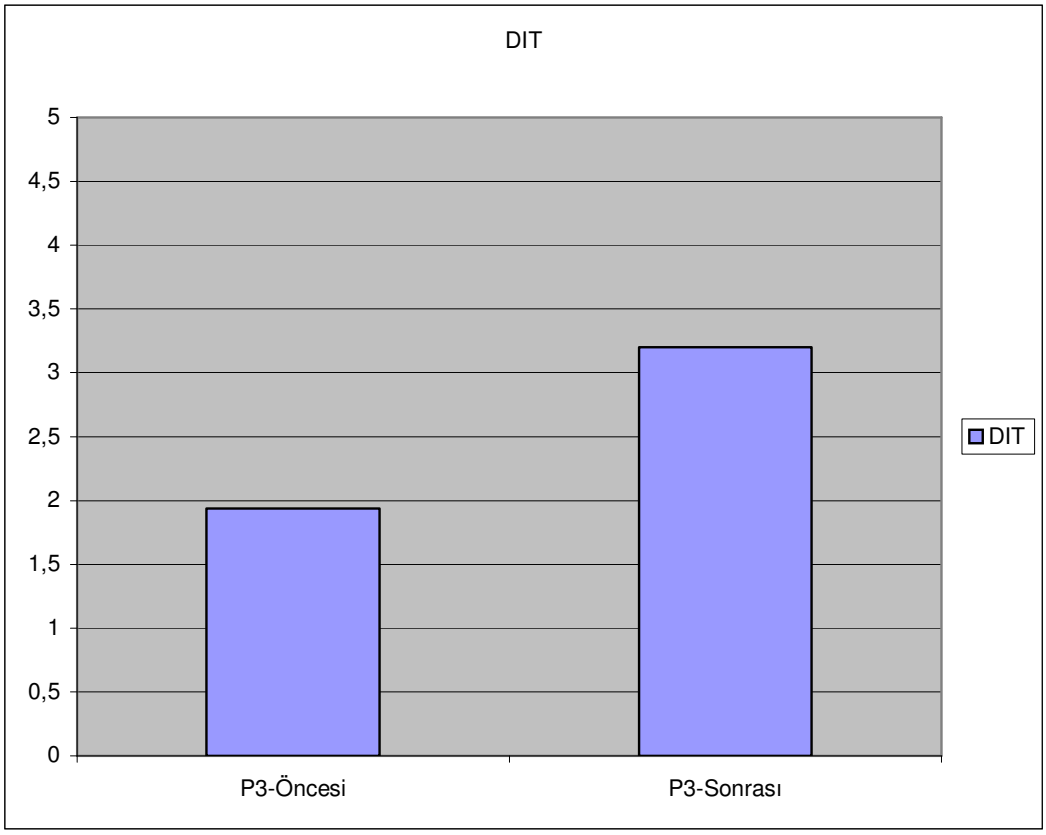
Şekil B.4 P3-Dışarıya Bağlılık Değerleri



Şekil B.5 P3-Metot Sayısı Değerleri



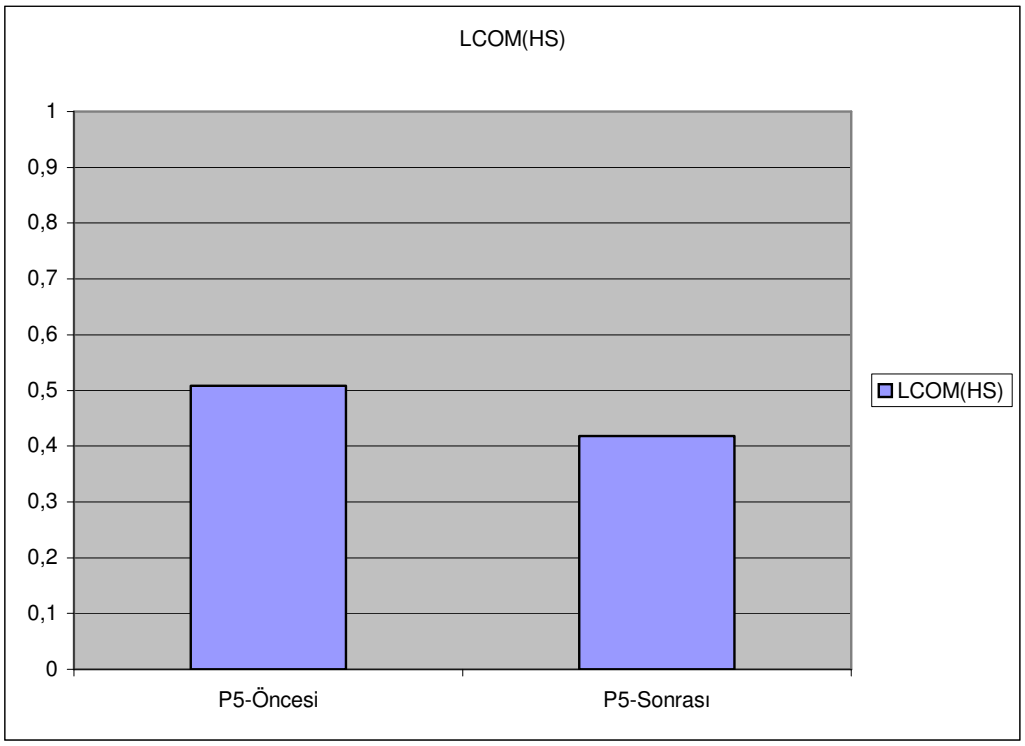
Şekil B.6 P3-Sınıfın Çocuk Sayısı Değerleri



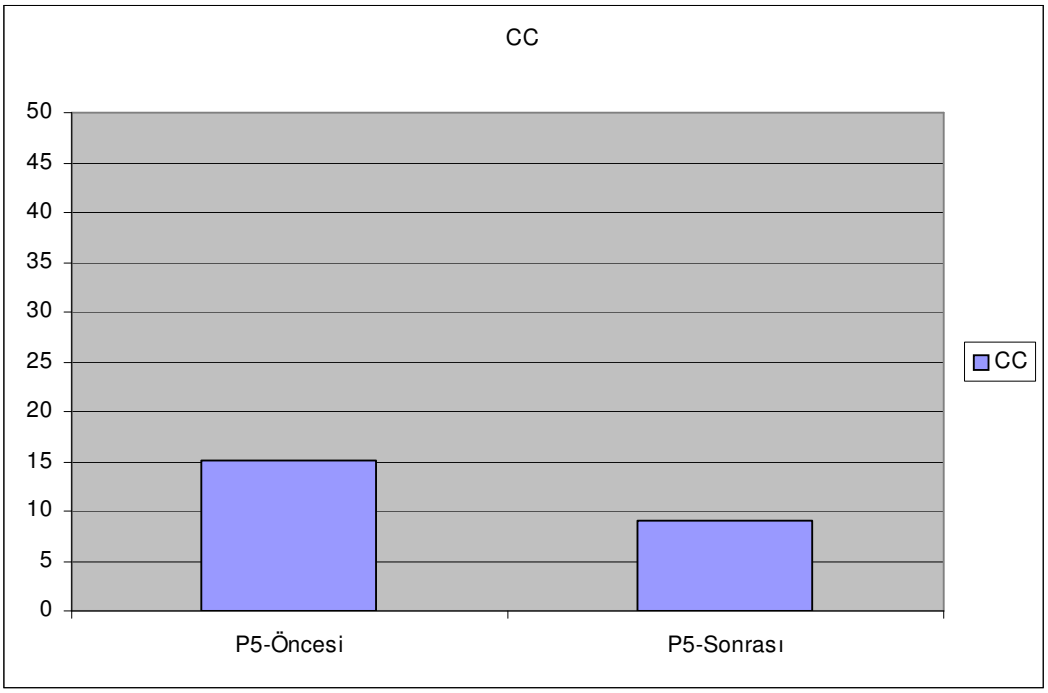
Şekil B.7 P3-Kalıtım Ağacındaki Derinliği Değerleri

EK-C

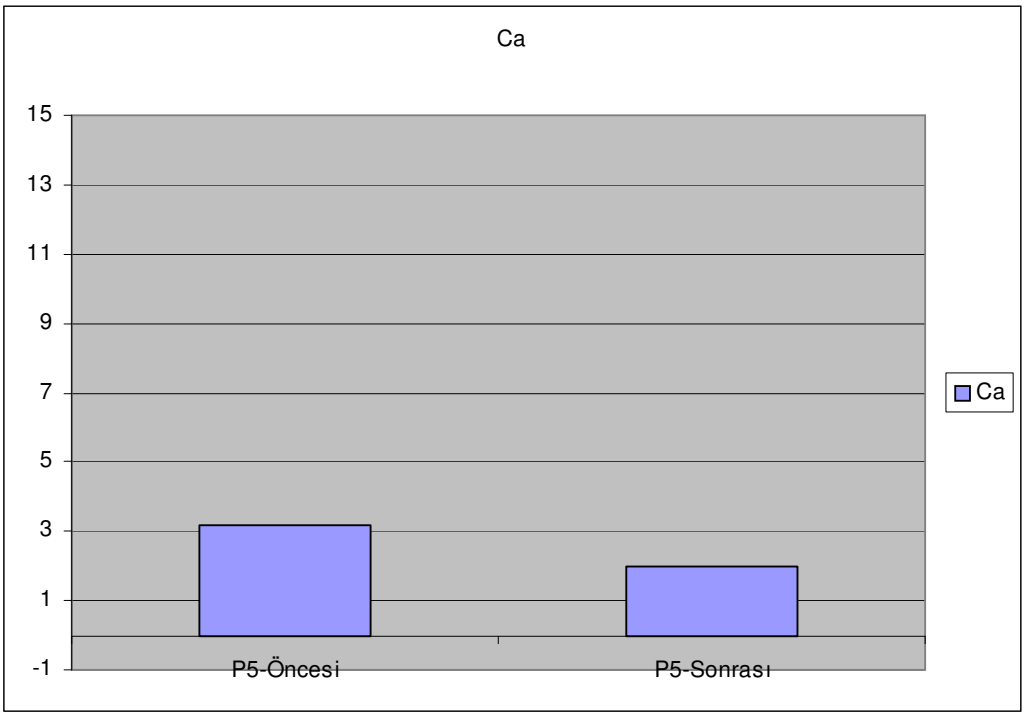
P5 Projesi Değerleri:



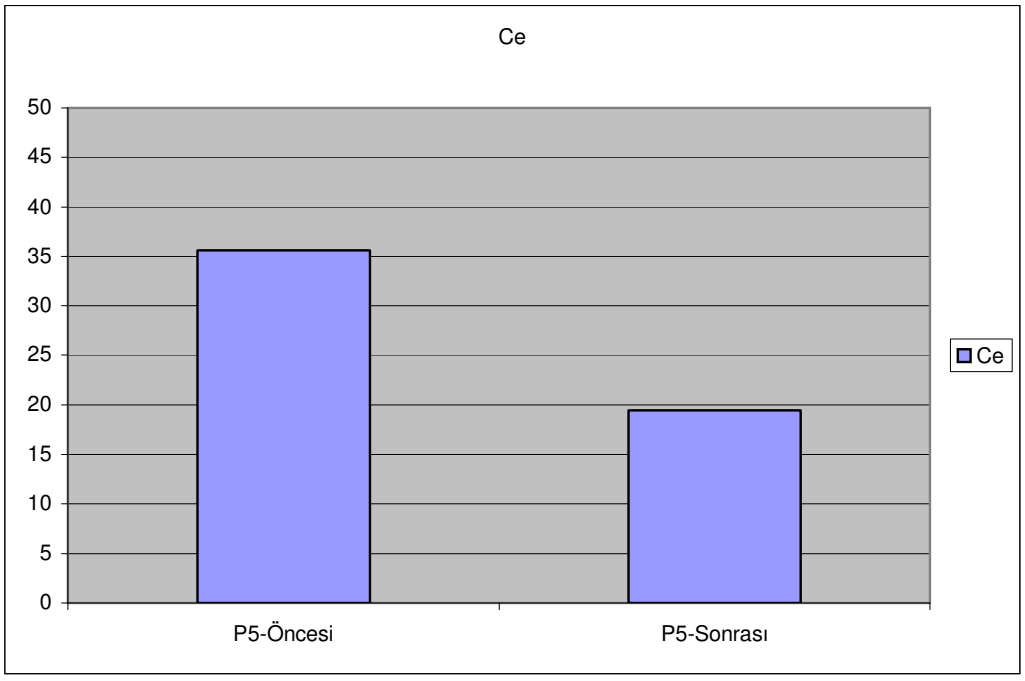
Şekil C.1 P5-Henderson Yapışma Yetersizliği Değerleri



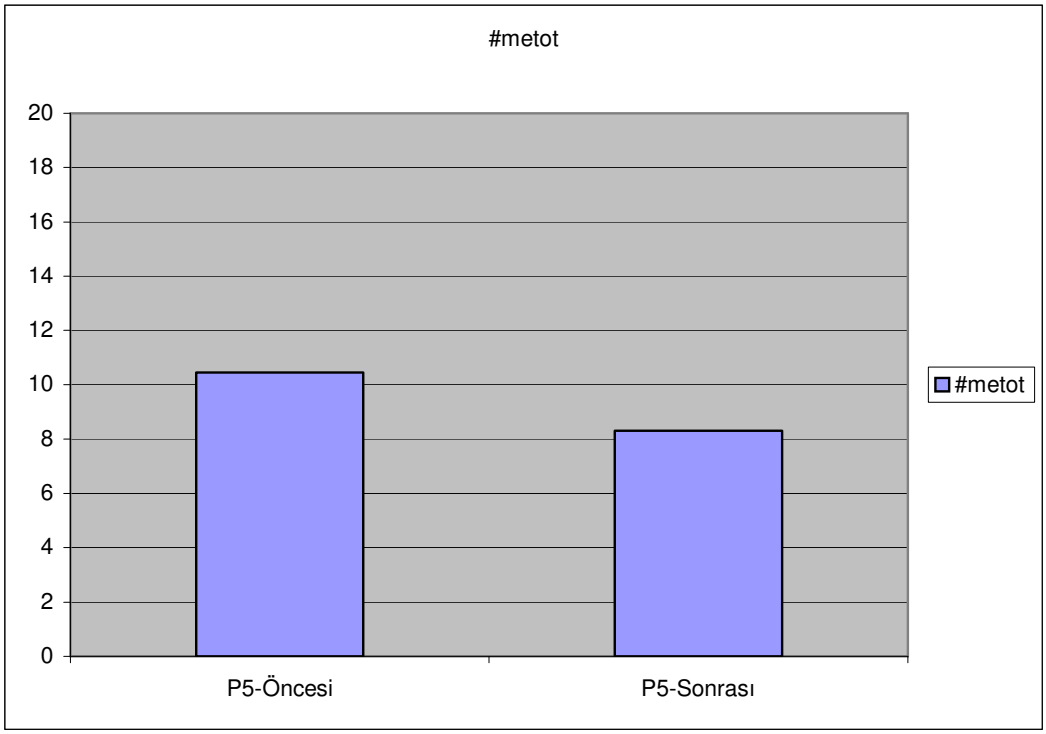
Şekil C.2 P5-Döngüsel Karmaşıklık Değerleri



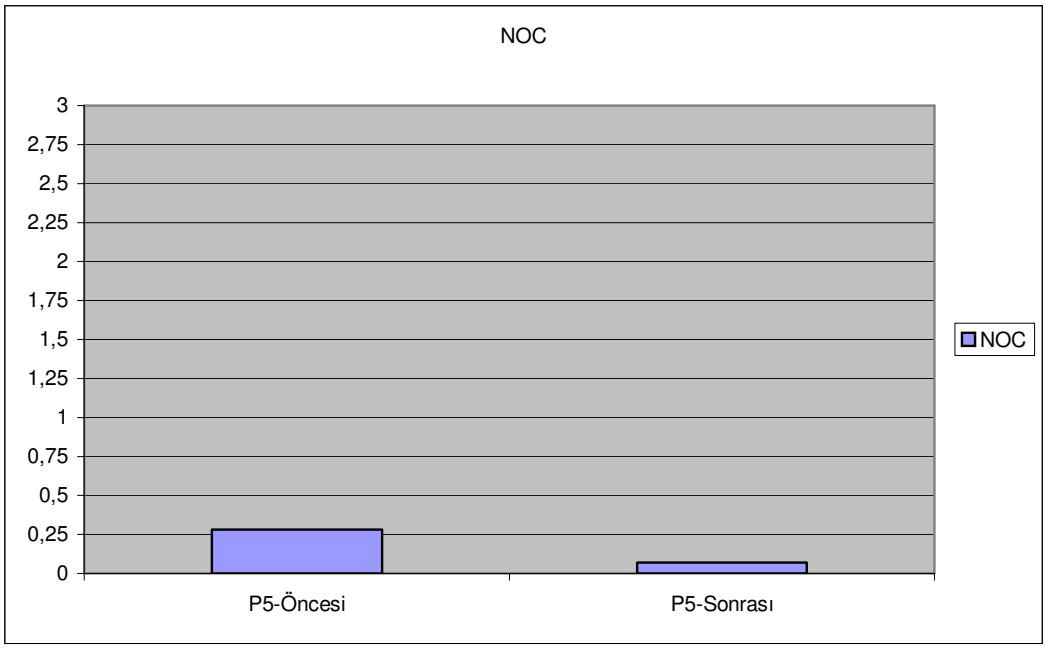
Şekil C.3 P5-İçeriye Bağlılık Değerleri



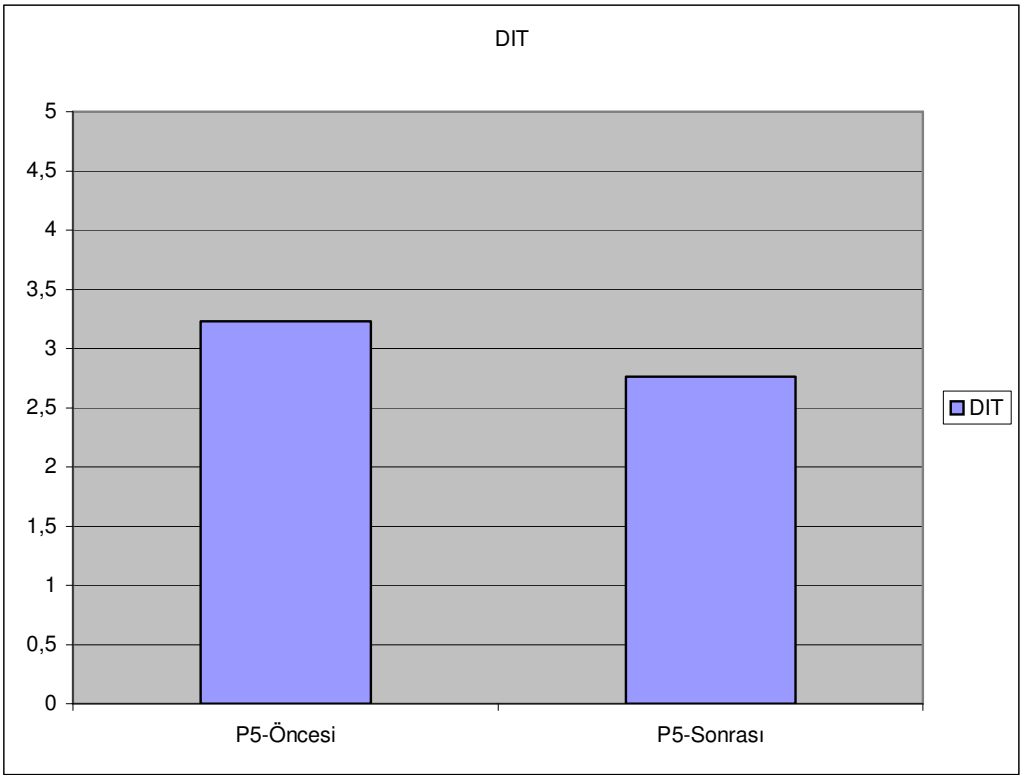
Şekil C.4 P5-Dışarıya Bağlılık Değerleri



Şekil C.5 P5-Metot Sayısı Değerleri

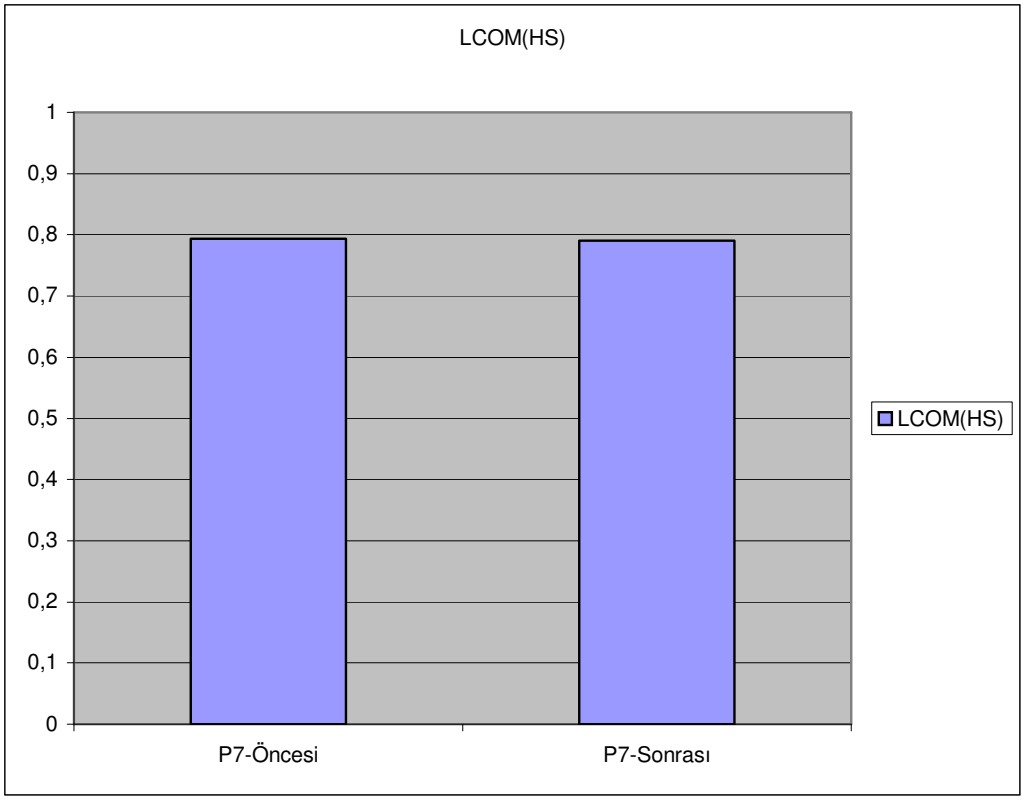


Şekil C.6 P5-Sınıfın Çocuk Sayısı Değerleri

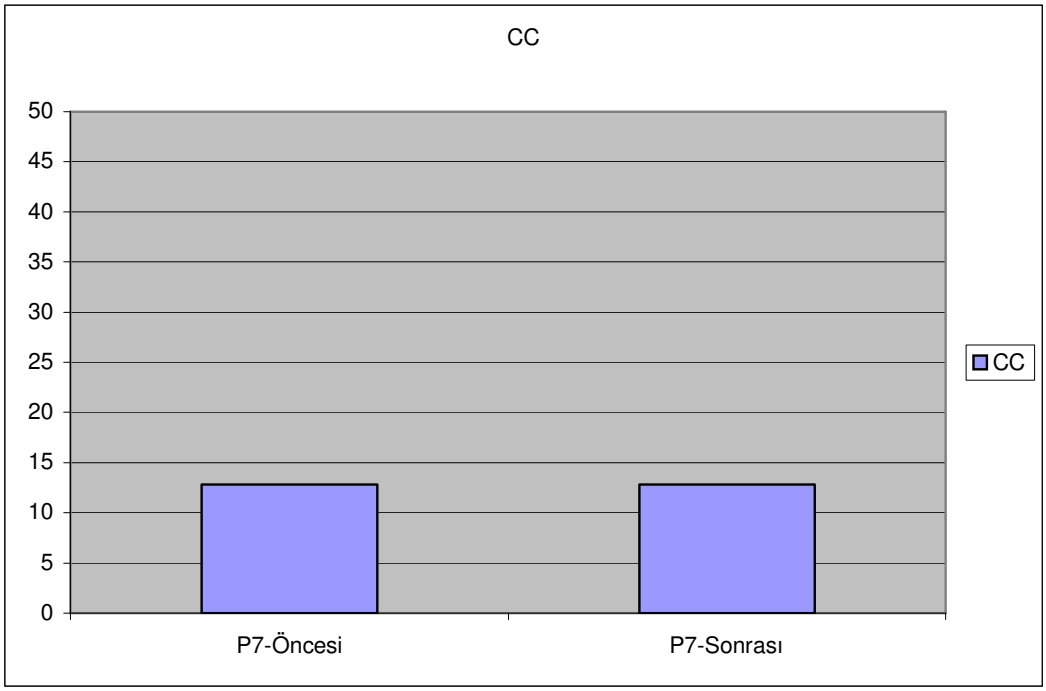


Şekil C.7 P5-Kalıtım Ağacı Derinliği Değerleri

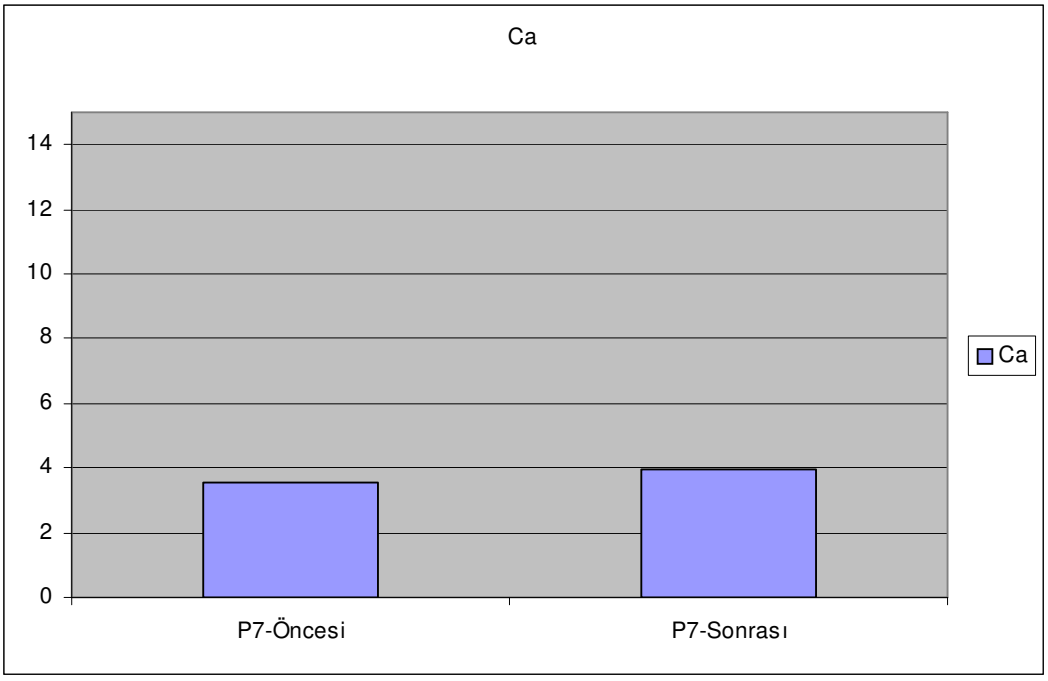
EK-D
P7 Projesi Değerleri:



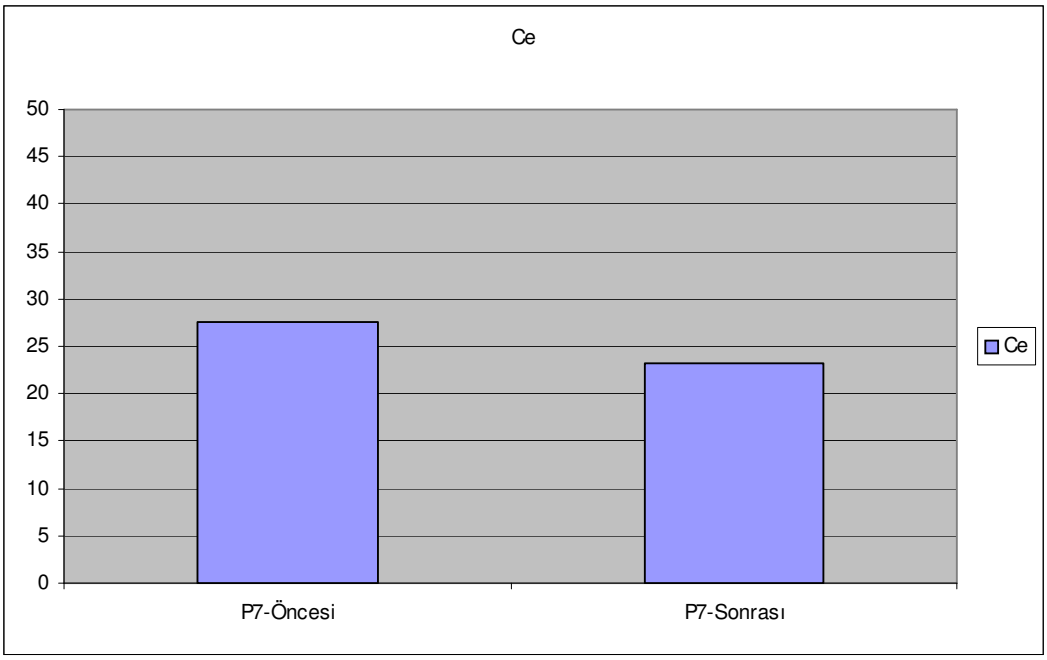
Şekil D.1 P7-Henderson Yapışma Yetersizliği Değerleri



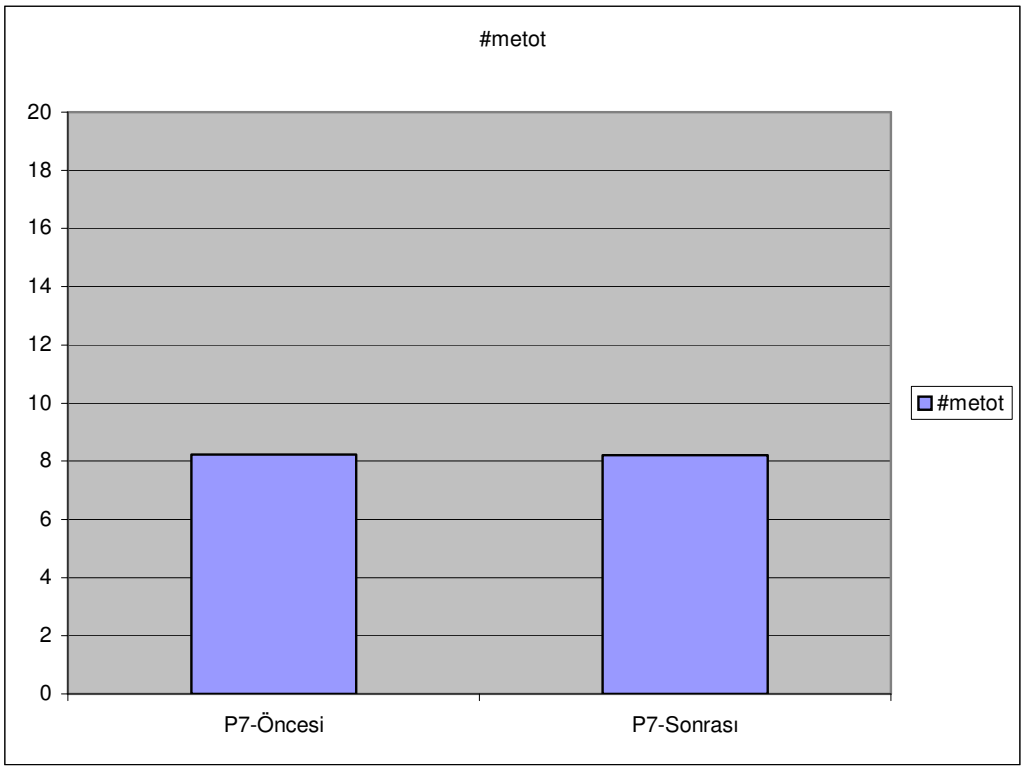
Şekil D.2 P7-Döngüsel Karmaşıklık Değerleri



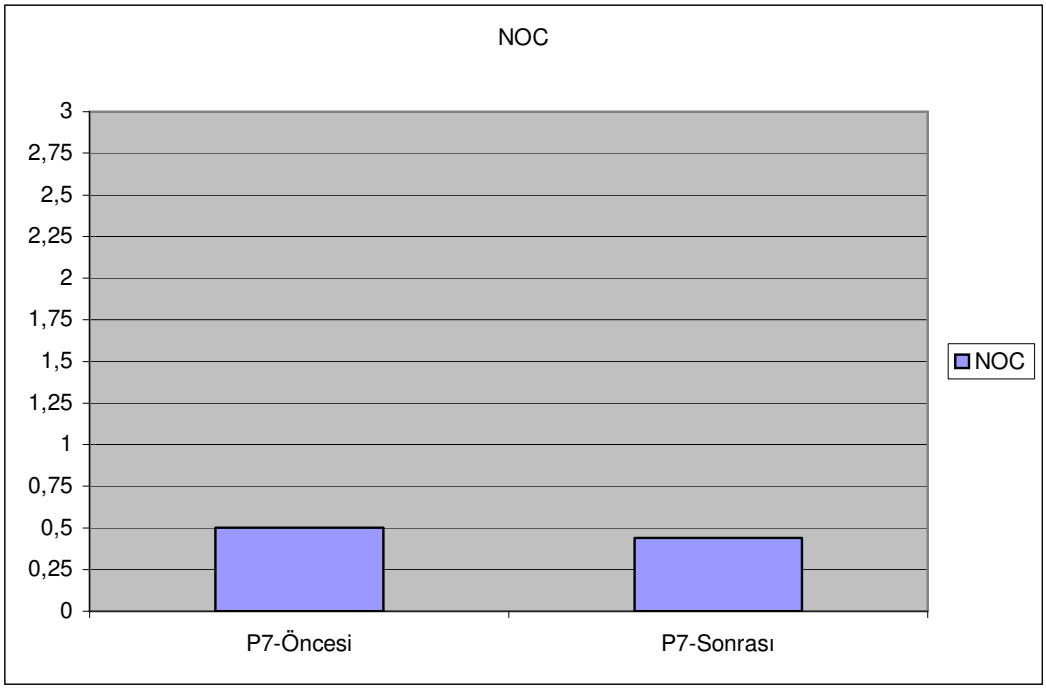
Şekil D.3 P7-İçeriye Bağlılık Değerleri



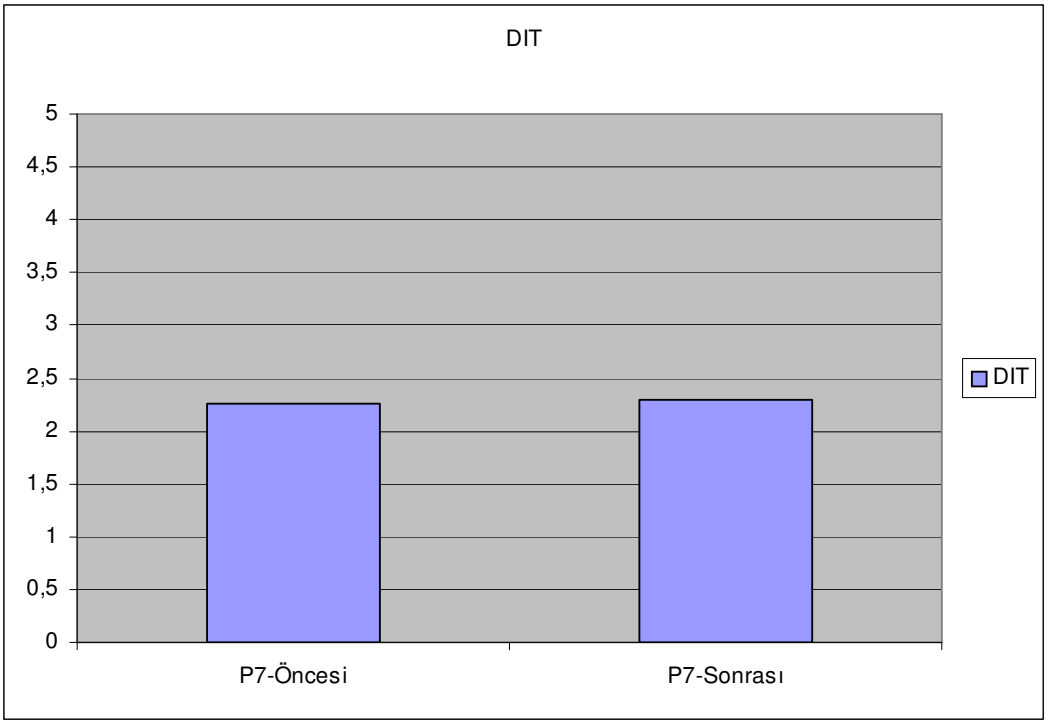
Şekil D.4 P7-Dışarıya Bağlılık Değerleri



Şekil D.5 P7-Metot Sayısı Değerleri



Şekil D.6 P7-Sınıfın Çocuk Sayısı Değerleri



Şekil D.7 P7-Kalıtım Ağacı Derinliği Değerleri

EK-E

BİR YENİLİK HAKKINDAKİ İLGİNİN AŞAMALARI

İLGİ AŞAMALARI ANKETİ

Değerlendirme:
0-7 arası skala :
0-1: Şu anda benim için doğru değil(İlgisiz)
2-3: Şu anda benim için doğru değil ama ileride olabilir
4-5: Şu anda benim için doğru sayılabilir
6-7: Şu anda benim için çok doğru

SORULAR

1. Kod gözden geçirmesinin programlama davranışımla ilgili getirecekleri konusunda endişeliyim.
2. Kod kalitesi geliştirmesinde, kodu gözden geçirmeden daha iyi sonuç verecek yaklaşımlar biliyorum.
3. Kod gözden geçirmenin ne olduğu konusunda fikrim yok.
4. Kodu gözden geçirme işini organize etme konusunda yeterince zaman bulamayacağımdan endişeliyim.
5. Arkadaşlarıma kod gözden geçirme konusunda yardımcı olmak isterim.
6. Kod gözden geçirme hakkındaki bilgim oldukça sınırlı.
7. Kod gözden geçirmenin kariyer planıma nasıl bir etkisi olacağını bilmek isterim.
8. İlgi alanlarım ve sorumluluklarım arasında ikileme düşmek istemem.
9. Kod gözden geçirme uygulama şeklini revize etmek isterim.
10. Kod gözden geçirme ile ilgili şirket içi ve şirket dışı aktif bağlantılar geliştirmek isterim.
11. Kod gözden geçirmenin genel iş yapış şekilimizi nasıl etkileyeceği konusunda endişeliyim.
12. Kod gözden geçirmeye önem veriyorum.
13. Kod gözden geçirme ile ilgili sistemde kimlerin karar alma mekanizmasında yer alacağını bilmek isterim.
14. Kod gözden geçirmenin kullanılması ile ilgili tartışmak isterim.
15. Kod gözden geçirmenin kullanılması durumunda, hangi kaynakların kullanılabileceğini öğrenmek isterim.
16. Kod gözden geçirmenin gerektirdiği kabiliyeti gösteremeyeceğimden endişeliyim.
17. Kod gözden geçirme sonrası görevimin nasıl değişeceği konusunda endişeliyim.
18. Kod gözden geçirmeyi kullanacak departmanların veya kişilerin sürece tanıdık olmalarını isterim.
19. Kod gözden geçirmenin, yazdığım kodun kalitesinin değerlendirmesine etkisi konusunda endişeliyim.
20. Kod gözden geçirmenin eğitim süreci konusunda gözden geçirme yapmak isterim.
21. Tamamen başka işlerle meşgulüm
22. Elde edilen tecrübeye göre kod gözden geçirme kullanımında bir takım değişiklikler yapmak isterim.
23. Kod gözden geçirme hakkında bilgi sahibi olmamama rağmen, kod kalitesi geliştirme alanına ilgiliyim.

24. Takım arkadaşlarımı kod gözden geçirme hakkında teşvik etmek isterim.
25. Kod gözden geçirmenin gerektireceği yan işlerle uğraşmaktan, diğer işlerime zaman kalmayacağı konusunda endişeliyim.
26. Yakın gelecekte kod gözden geçirmenin gerektireceği işleri merak ediyorum.
27. Kod gözden geçirmesinden maksimum yarar elde etmek için arkadaşlarımla işbirliği yapmak isterim.
28. Kod gözden geçirmenin uygulamasının ne kadar zaman ve emek gerektireceğini bilmek isterim.
29. Kod kalitesi yönetimi konusunda başka şirketlerin neler yaptıklarını bilmek isterim.
30. Şu anda kod gözden geçirme ile ilgili birşey öğrenmek istemiyorum.
31. Kod gözden geçirmenin nasıl destekleneceği, geliştirileceği ve değiştirileceği konusunda karar vermek isterim.
32. Kod gözden geçirmeyle ilgili değişikliklerde geri dönüş almak isterim.
33. Kod gözden geçirmesi ile beraber şirketteki rolümün nasıl değişeceğiyle ilgili bilgi sahibi olmak isterim.
34. Kod gözden geçirmesi ile ilgili görevlerin(kodu hazırlama vb.) ve insanların koordinasyonu(toplanılacak kişiyi ayarlama vb.) çok fazla vaktimi almakta.
35. Kod gözden geçirmesinin eski sistemden neden daha iyi olduğunu öğrenmek isterim.

ÖZGEÇMİŞ

Mustafa Turgay Alptekin, 15 Kasım 1982 yılında Ankara'da doğdu. Lise öğrenimini Cağaloğlu Anadolu Lisesinde, lisansı ise Yıldız Teknik Üniversitesi Bilgisayar Mühendisliğinda tamamladı. Lisans tezi olarak Yıldız Teknik Üniversitesi ÖREM Burs Otomasyon Sistemini geliştirdi. Lisans öğreniminden sonra İ.T.Ü teknokent bölgesinde çalışmalarını sürdüren yazılım şirketi Veripark'da yazılım geliştirme uzmanı olarak çalışmaktadır. İyi derecede İngilizce ve Almanca bilgisine sahiptir.