KOD DENETLEME SÜRECİNİ ETKİLEYEN FAKTÖRLERİN TAHMİNİNİN YAPILMASINA İLİŞKİN İSTATİSTİKÎ ANALİZ ÇATISININ OLUŞTURULMASI VE SONUÇLARIN DEĞERLENDİRİLMESİ

HİLAL AYIK

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ

TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

NİSAN, 2013

ANKARA

Fen Bilimleri Enstitü onayı

_____

Prof. Dr. Ünver KAYNAK

Müdür

Bu tezin Yüksek Lisans derecesinin tüm gereksinimlerini sağladığını onaylarım.

_____

Doç. Dr. Erdoğan DOĞDU

Anabilim Dalı Başkanı

Hilal AYIK tarafından hazırlanan **KOD DENETLEME SÜRECİNİ ETKİLEYEN FAKTÖRLERİN TAHMİNİNİN YAPILMASINA İLİŞKİN İSTATİSTİKÎ ANALİZ ÇATISININ OLUŞTURULMASI VE SONUÇLARIN DEĞERLENDİRİLMESİ** adlı bu tezin Yüksek Lisans tezi olarak uygun olduğunu onaylarım.

_____

Yrd. Doç. Dr. Tansel ÖZYER

Tez Danışmanı

Tez Jüri Üyeleri

Başkan : Yrd. Doç. Dr. Tansel ÖZYER _____

Üye : Doç. Dr. Bülent TAVLI _____

Üye : Yrd. Doç. Dr. Esra KADIOĞLU URTİŞ _____

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada orijinal olmayan her türlü kaynağa eksiksiz atıf yapıldığını bildiririm.

Hilal AYIK

Üniversitesi : **TOBB Ekonomi ve Teknoloji Üniversitesi**
Enstitüsü : **Fen Bilimleri**
Anabilim Dalı : **Bilgisayar Mühendisliği**
Tez Danışmanı : **Yrd. Doç. Dr. Tansel ÖZYER**
Tez Türü ve Tarihi : **Yüksek Lisans – Nisan 2013**

HİLAL AYIK

KOD DENETLEME SÜRECİNİ ETKİLEYEN FAKTÖRLERİN TAHMİNİNİN YAPILMASINA İLİŞKİN İSTATİSTİKÎ ANALİZ ÇATISININ OLUŞTURULMASI VE SONUÇLARIN DEĞERLENDİRİLMESİ

ÖZET

Yazılım endüstrisinde rekabet gün geçtikçe artmaktadır. Firmaların bu rekabet ortamında var olabilmek için müşteri memnuniyetini en üst seviyede tutmaları gerekmektedir. Müşteri memnuniyeti kaliteli ürünlerin sunulmasıyla kazanılır. Yazılım ürünlerinin hatalardan arındırılmış olması kalite açısından önemli bir ölçüttür. Kod denetleme yazılım ürününün içerdiği hataların tespit edilmesi ve giderilmesini sağlayan test işlemleri öncesinde gerçekleştirilen etkili bir süreçtir. Bu sürecin verimliliğinin ölçülmesi hem sürecin hem de ürünün kalitesinin artmasını sağlayacaktır. Kod denetleme verimliliğinin ölçülmesinde yazılım kalite metrikleri kullanılmaktadır. Denetleme verimliliğini ölçmeye yönelik çalışmalar incelendiğinde denetleme süresi, hazırlanma süresi, denetleyici sayısı ve denetleyici tecrübesi gibi faktörlerin denetleme süreci üzerinde önemli etkileri olduğu gözlemlenmiştir. Verimli bir kod denetleme işlemi gerçekleştirilmek isteniyorsa sürecin başında bu dört değişkene ait değerler en doğru şekilde belirlenmelidir.

Bu tez çalışmasında denetleme sürecinin değerlendirilmesinde kullanılan metrikler incelenmiş olup, Denetleme Derinliği, Denetleyici Performans Metriği ve Hata Giderme Etkinliği metriklerinin çalışmada kullanılmasına karar verilmiştir. Bu metrikler ve tahmin algoritmaları kullanılarak sürece en uygun denetleme süresi, hazırlanma süresi, denetleyici sayısı ve denetleyici tecrübesi değerlerinin denetleme sürecinin başında belirlenmesini sağlayan bir sistem geliştirilmiştir. Tahmin işlemini gerçekleştirmek için lineer regresyon, Eğri Uydurma ve Gauss-Newton algoritmaları kullanılmıştır.

Anahtar Kelimeler: Kod denetleme verimliliği, kod denetleme metrikleri, kod denetleme süresi, hata giderme etkinliği

University : **TOBB Economics and Technology University**
Institute : **Institute of Natural and Applied Sciences**
Science Programme : **Computer Engineering**
Supervisor : **Assistant Prof. Dr. Tansel ÖZYER**
Degree Awarded and Date : **M.Sc. – Nisan,2013**

HİLAL AYIK

CONSTRUCTION OF STATISTICAL ANALYSIS FRAMEWORK FOR PREDICTING FACTORS THAT AFFECT CODE INSPECTION

ABSTRACT

Competition in the software industry increases every passing day. Companies need to be kept at the highest level of customer satisfaction to be in this competition. The customer satisfaction is gained with the introduction of high-quality products. Error free software product is important criteria for software quality. Code inspection is an effective process that detects defects early and removes them from software products before testing. Measurement of effectiveness of the product increases quality of the software product and the process. Software quality metrics are used to measure the efficiency of the code inspection. The factors like preparation for the inspection, preparation time, number of inspectors, and the experience level of the inspectors affect the inspection productivity. These four are really important for the inspection and should be clearly defined at the beginning of the process.

This thesis includes the metrics used for evaluation of inspection process, Depth of Inspection (DI), Inspector Performance Metric (IPM), and Defect Reduction Technique. A new system is developed by using these metrics and the estimation algorithms to have the best inspection time, preparation time, and inspector number and level of experience for the process at the beginning of the project. To compete the estimation operation linear regration, curve fitting and Gauss-Newton algorithm is being used.

Key words: Productivity for code inspection, code inspections metrics, code inspection time, defect removal efficiency

TEŞEKKÜR

Bu tez çalışmasının hazırlanmasında bilgi ve birikimiyle beni yönlendiren danışman hocam Sayın Yrd. Doç. Dr. Tansel ÖZYER'e, yüksek lisans eğitimim boyunca katkılarını esirgemeyen TOBB Ekonomi ve Teknoloji Üniversitesi Bilgisayar Mühendisliği bölümü hocalarıma, hayatım boyunca beni destekleyen ve her zaman yanımda olan kıymetli aileme çok teşekkür ederim.

İÇİNDEKİLER

ÖZET ..... İİİ
ABSTRACT ..... İV
TEŞEKKÜR V
İÇİNDEKİLER ..... Vİ
TABLOLARIN LİSTESİ ..... Vİİİ
ŞEKİLLERİN LİSTESİ ..... İX
KISALTMALAR ..... Xİ
1. GİRİŞ 1
2. KOD DENETLEME 4
2.1 YAZILIM DENETLEME NEDİR? 4
2.1.1 Gereksinim Denetleme 5
2.1.2 Dizayn Denetleme 5
2.1.3 Kod Denetleme 5
2.2 KOD DENETLEME ELEMENTLERİ 6
2.2.1 Giriş ve Çıkış Kriterleri 6
2.2.2 Süreçler 7
2.2.3 Denetleme Takımının Oluşturulması ve Roller 13
2.2.4 Denetleme Süreçlerinde Kullanılan Okuma Teknikleri 15
2.3 KOD DENETLEME SÜRECİNİN FAYDALARI 20
2.3.1 Kod Denetleme Kaynak Kodun Kalitesini Arttırır 20
2.3.2 Kod Denetleme Verimliliği Arttırır 21
2.3.3 Kod Denetleme Maliyeti Düşürür 21
2.3.4 Kod Denetleme Ekip İlişkisini ve Eğitimini Teşvik Eder 22
2.3.5 Kod Denetleme Kalite Kültürü Oluşmasını Teşvik Eder 22
3. KOD DENETLEME SÜRECİNİ ETKİLEYEN DEĞİŞKENLER VE DENETLEME METRİKLERİ 23
3.1 KOD DENETLEME VERİMLİLİĞİ VE ÖLÇÜLMESİNİN ÖNEMİ 23
3.2 METRİK KAVRAMI 24
3.2.1 Hedef Soru Metrik Paradigması Kullanılarak Metriklerin Oluşturulması 25
3.2.2 Kod Denetleme Sürecinin Değerlendirilmesi İçin Kullanılan Metrik Örnekleri 27
3.3 DENETLEME SÜRECİNİ ETKİLEYEN DEĞİŞKENLERİN VE BU DEĞİŞKENLER İÇİN OPTİMAL DEĞERLERLERİN BELİRLENMESİNE YÖNELİK ÇALIŞMALAR 34
3.3.1 AT & Bell Laboratuarlarında Yapılan Çalışmalar 34

3.3.2 650 NASA SEL Denetleme Kayıtları Kullanılarak Yapılan Çalışmalar 36
3.4 DRE, DI VE IPM METRİKLERİ KULLANILARAK DENETLEMEYİ ETKİLEYEN DEĞİŞKENLERİN TAHMİN EDİLMESİ 39
3.4.1 Denetlemeyi Etkileyen Değişkenler....39
3.4.2 Tahmin İşleminde Kullanılacak Metrikler ve Tercih Edilme Sebepleri ..43
3.4.3 Denetleme Sürecini Etkileyen Değişkenlerin Tahmin Edilmesi İşleminde Kullanılan Algoritmalar 49
3.4.4 IPM, DI ve DRE Metrikleri Kullanılarak Denetleme Sürecini Etkileyen Değişkenlerin Tahmin Edilmesi 54
4. SONUÇ 60
KAYNAKLAR....62
ÖZGEÇMİŞ....68

Tabloların Listesi

Tablo	Sayfa
Tablo 2.1 Denetleme hata dağılım oranları	8
Tablo 2.2 Denetleme yöntemlerinin süreç bazlı karşılaştırılması	13
Tablo 2.3 Okuma tekniklerinin kıyaslanması	20
Tablo 3.1 AT & Bell laboratuarları hedef soru metrik uygulaması	26
Tablo 3.2 Denetleme metrikleri için optimal değerler	31
Tablo 3.3 Denetleme verimliliği ve denetleyici verimliliği hesaplamasında kullanılan değerler	32
Tablo 3.4 Güncel kod denetleme işlemine ait veriler	35
Tablo 3.5 Güncel kod denetleme işlemine ait veriler	35
Tablo 3.6 DI değer aralıkları	46

Şekillerin Listesi

Şekil **Sayfa**

Şekil 2.1 Denetlemenin yazılım yaşam döngüsünde bulunduğu aşamalar 6

Şekil 2.2 Derleme adımının denetleme sürecindeki yeri 12

Şekil 2.3 Yazılım Kalitesizliğinin proje zamanına etkisi 21

Şekil 3.1 Hedef-Soru-Metrik yapısı 26

Şekil 3.2 Yeni geliştirilmiş programlarda denetçi sayısı-etkinlik ilişkisi 37

Şekil 3.3 Bakımı yapılan programlarda denetçi sayısı-etkinlik ilişkisi 37

Şekil 3.4 Yeni geliştirilmiş programlarda denetçi sayısı-denetleme etkinliği ilişkisi 38

Şekil 3.5 Bakımı yapılan programlarda denetçi sayısı-denetleme etkinliği ilişkisi 38

Şekil 3.6 Açılış sayfası 54

Şekil 3.7 Grafiksiz tahmin edilen değerler ekranı 55

Şekil 3.8 Grafikli tahmin edilen değerler ekranı 56

Şekil 3.9 Algoritmalar ve tahmin edilen değerler 56

Şekil 3.10 Orta ölçekli bir proje için değişken tahmini 57

Şekil 3.11 Orta ölçek, DRE, DI ve IPM değerleri kullanılarak analiz yapılması 57

Şekil 3.12 Orta ölçek, DI ve IPM değerleri kullanılarak analiz yapılması 58

Şekil 3.13 Orta ölçek, DI ve IPM değerleri kullanılarak analiz yapılması 58

Şekil 3.14 Orta ölçek, DI değerleri kullanılarak analiz yapılması 59

Kısaltmalar

Kısaltmalar	Açıklama
AT & Bell	Bell Telefon Laboratuarı (Bell Telephone Laboratories)
AT & T	Amerikan Telefon ve Telgraf Şirketi (American Telephone and Telegraph Company)
CMMI	Yetenek Olgunluk Model Entegrasyonu (Capability Maturity Model Integration)
DI	Denetleme Derinliği (Depth of Inspection)
DRE	Hata Giderme Etkinliği (Defect Removal Efficiency)
GQM	Hedef Soru Metrik Paradigması (Goal Question Metric Paradigm)
IBM	Uluslararası İş Makineleri (International Business Machines)
IEEE	Elektrik Elektronik Mühendisleri Enstitüsü (Institute of Electrical and Electronics Engineers)
IEEE STD-2008	IEEE Yazılım Gözden Geçirme Standartları (IEEE Standard for Software Reviews)
IPM	Denetleme Performans Metriği (Inspection Performance Metric)
KLOC	Kod Satır Sayısı / 1000 (One Thousand Lines of Code)
LOC	Kod Satır Sayısı (Lines of Code)
NASA	Ulusal Havacılık ve Uzay Dairesi (National Aeronautics and Space Administration)

1. Giriş

Karmaşık bir projenin başarı ile tamamlanmış olması demek belirlenen bütçeye sadık kalınarak, öngörülen zamanda ve yüksek kalite ile tamamlanması demektir. Bu sebeple bir projenin başarı ile tamamlanması isteniliyorsa yazılım geliştirme yaşam döngüsünün gereksinim analizi, yazılım tasarımı, kodlama gibi aşamalarında karşılaşılan hataların ortadan kaldırılması gerekmektedir. Yazılım geliştirme firmaları bu tür hataların tespit edilerek ortadan kaldırılması için birçok yönteme başvurmaktadırlar. Bu yöntemlerden en geçerli olanı yazılım denetimleridir. Yazılım denetimleri sırasında üründen bağımsız kişiler yazılım ürünü kapsamında bulunan hataların ve üzerinde yapılabilecek iyileştirmelerin tespiti için çalışırlar. Bu çalışmalar sırasında denetleme prosedür ve tekniklerinden faydalanırlar. Yazılım denetimlerinin amacı yazılım projesinin maliyet, takvim, ve kalite bakış açılarına göre ortaya çıkan performansını kötü etkileyecek unsurları tespit edilip engellemek ve belirlenen zamanda belirlenen bütçeye uygun kaliteli ürünler ortaya koymaktır. Yazılım denetim süreci 3 aşamadan oluşmaktadır: gereksinimlerin tespit edilmesi, tasarımın denetlenmesi ve kodun denetlenmesi. Kod denetleme aşamasında denetçiler hataları tespit etmek ve önlemek amacıyla kaynak kodu incelerler [5]. Kod denetleme test ve bakım maliyetlerini düşürür, yazılım kalitesi ve üretkenliği arttırır [6].

Son yıllarda denetleme sürecinin verimliliği ve nasıl yönetilmesi gerektiği tartışılmaya başlanmıştır. Denetleme işlemi iyi yönetilmelidir çünkü sürecin verimliliği ürünü ve ürün kalitesini etkilemektedir [7]. Hewlett-Packard firmasında program yöneticisi olan Tom Kendrik David Packard'ın ölçemediğini yönetemezsin sözünü hatırlatarak metriklerin kalitenin iyileştirmesinde önemli bir rolü olduğunu söylemiştir [8]. Yazılım denetleme işleminin daha disiplinli bir mühendislik tekniği haline getirilebilmesi için düzenli bir performansa, hedeflerin iyi bir şekilde belirlenmesine ve metriklere ihtiyaç vardır. Metrikler süreci ve ürünü ölçmek için kullanılan nümerik değerlerdir [9]. Yazılım metrikleri, geliştirilen yazılım ürününü ölçekleyebilmek ve ürünün kalitesini kontrol altında tutabilmek için nicel gözlem yapabilmeyi mümkün kılacak ölçü birimlerini temsil eder [10]. Denetleme süreci ve denetleyicilerin performansını ölçmek için yazılım kalite metrikleri kullanılmaktadır.

Denetleme ve denetleyici verimliliğini ölçmeye yönelik araştırmalar ile birlikte bu süreçlerde kullanılan metrikler incelendiğinde denetleme verimliliğini etkileyen parametreler hakkında bilgi sahibi olmanın denetleme tekniğinden fayda sağlamak açısından önemli olduğu gözlemlenmiştir [11]. Uygun denetleme zamanı, denetleme ekibi kişi sayısı, denetleyici tecrübe seviyesi ve hazırlanma zamanı denetlemenin verimliliğini etkileyen en önemli parametrelerdir. Bu tez çalışmasının amacı denetleme sürecini doğrudan etkileyen bu dört parametrenin denetleme sürecinin başlangıcında doğru belirlenmesine yardımcı olmaktır. Bu sebeple hata giderme etkinliği, denetleme performans metriği ve denetleme derinliği metriklerinin sahip olduğu değerler doğrultusunda geçmiş kod denetleme verileri kullanılarak tahmin algoritmaları yardımıyla optimal değişken değerlerini bulan bir çatı oluşturulmuştur. Daha sonra algoritmaların parametreleri hesaplama konusunda ne kadar yetenekli olduğunun kıyaslaması denetlenen yazılım ürününün büyüklüğüne göre yapılmıştır.

Tez çalışması sırasıyla giriş, kod denetleme, kod denetleme sürecini etkileyen değişkenler ve denetleme metrikleri ve sonuç bölümleri olmak üzere dört bölümden meydana gelmektedir.

Tez çalışmasının ikinci bölümünde yazılım denetleme kavramı ve kod denetleme süreci elementleri anlatılacaktır. Sürecin başlaması ve bitmesi için gerekli olan giriş-çıkış kriterlerinden bahsedilecektir. İlk kod denetleme yöntemi olan Fagan denetleme yöntemi ve geçmişten günümüze en çok kullanılan yöntemler anlatılacak ve süreçler baz alınarak kıyaslamasına yer verilecektir. Denetleme takımının oluşturulmasında dikkat edilmesi gereken unsurlar, takım üyelerinin rolleri ve sorumluluklarından bahsedilecektir. Kod denetleme sürecinin gerçekleştirilmesinde en çok tercih edilen okuma teknikleri anlatılacak ve bu teknikler uygulama içeriği, kullanışlılık, tekrarlanabilirlik, adapte olabilirlik, kapsam, eğitim ve doğrulama ölçütleri dikkate alınarak kıyaslanacaktır. Kod denetleme sürecinin faydalarından bahsedilecek ve kod denetleme ve diğer hata bulma yöntemlerinin özellik karşılaştırmasına yer verilecektir.

Tez çalışmasının üçüncü bölümünde kod denetleme sürecinin verimliliğinin yazılım ürününe etkisi ve denetleme sürecinin ölçülmesinin önemi anlatılacaktır. Bu

ölçümlemenin yapılabilmesi için metriklere ihtiyaç duyulmaktadır. Üçüncü bölümde yazılım metrikleri ve hedef soru metrik paradigması kullanılarak metriklerin belirlenmesi işleminden bahsedilecektir. Buna ek olarak kod denetleme sürecinin ölçülmesinde kullanılan metrikler anlatılacaktır. Metriklerin incelenmesi ve akademik çalışmalar doğrultusunda belirlenmiş denetlemeyi etkileyen değişkenler ve bu değişkenlerin sürece etkisinden bahsedilecektir. Bu değişkenlerin tahmin edilmesi sırasında kullanılan denetleme performans metriği, hata giderme etkinliği ve denetleme derinliği metrikleri anlatılacaktır. Denetleme sürecini etkileyen değişkenlerin doğru belirlenmesi önemli bir husustur. Tez çalışmasının son bölümünde değişkenlerin denetleme performans metriği, hata giderme etkinliği ve denetleme derinliği metriklerini kullanarak tahmin edilmesini sağlayan çatının yapısından ve tahmin işleminin gerçekleştirilmesinde kullanılan algoritmalardan bahsedilecektir.

Dördüncü yani son bölümde tez çalışması ile ilgili olarak değerlendirmelere yer verilecektir. Gelecek çalışmalardan bahsedilip tez çalışması sonuçlandırılacaktır.

2. Kod Denetleme

2.1 Yazılım Denetleme Nedir?

Yazılım inceleme/denetleme yazılımın gereksinimleri karşılayıp karşılamadığını doğrulamak için gerçekleştirilen statik test yöntemidir. Bahsi geçen inceleme süreci yazılım geliştirenlerin yani insanların makine testlerinden daha fazla hatayı daha az maliyetle tespit edeceği garantisini verir. Bu yöntemi uygulayanlar kalite açısından çok önemli gelişmeler kaydetmişlerdir.

Yazılım denetleme süreci yazılım kalitesini ve yazılımcı üretkenliğini arttırmak maksadıyla ilk kez 1972 yılında IBM'de ortaya çıkmıştır [12]. Daha sonra IBM çalışanı ve yazılım denetleme fikrinin sahibi Michael Fagan, Inspection yani denetleme sürecini ilk kez resmi olarak 1976 yılında yayınladığı makale ile duyurmuştur. Bu makalenin yayınlanmasından sonra Fagan Denetleme süreci pek çok makale, kitap ve raporda yer bulmuştur. Tüm bu raporlar yazılım denetleme sürecinin ürün kalitesini ve gelişim sürecinde verimliliği ve idare edilebilirliği arttırdığını iddia etmiştir [13]. Yazılım geliştirme ve bakım endüstrisi genelinde denetleme sürecinin hızla benimsenmesi hedeflerini karşılamadaki etkinliğinin bildirimi olmuştur. Denetim sürecinin açık yapısı kalkınma sürecini beraberinde getirmiştir. Gelişme, tasarım ve kod denetleme çalışmalarının simültane olması denetleme işlemi ilkelerini gereksinimlerin, kullanıcı bilgilerinin, dokümantasyonun, test planlarının ve test safhalarının denetlenmesi için tetiklemiş bu sayede yazılım denetleme sürecinin etkinliği ve kalitesi artmıştır [12].

Fagan yazılım denetlemeyi dizayn ve kodda yer alan hataları usulüne uygun olarak, etkili ve ekonomik bir şekilde bulma yöntemi olarak tanımlamıştır [14]. Frank H. Lewski ise Yazılım denetlemeyi gereksinimlerin karşılanıp karşılanmadığının denetlenmesi, dizaynın denetlenmesi ve kodun denetlenmesi olarak üç ana başlıkta toplamıştır [12].

2.1.1 Gereksinim Denetleme

Gereksinim denetleme zincirin ilk ve en önemli halkasıdır. Ürün mükemmel bir kod ile yazılmış da olsa eğer ihtiyaçları karşılamıyorsa hiçbir kıymeti kalmaz. Bu süreç tecrübelerden faydalanılarak büyük hataları erkenden keşfetmek adına yapılır. En az zaman harcanılan denetleme adımıdır çünkü okunması ya da araştırılması gereken sayfalarca doküman yoktur. Gereksinimleri denetlenmesinde amaç strateji ve hedeflerin netleştirilmesidir.

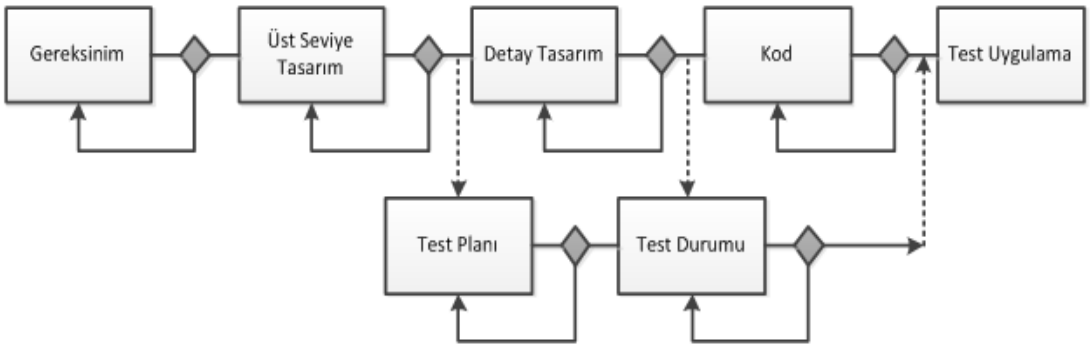
2.1.2 Dizayn Denetleme

Dizayn denetleme gerçekleştirilmesi en zor denetleme sürecidir. Zor bulunmasının sebebi kuralları iyi bir dizi şeklinde tanımlama sıkıntısıdır çünkü bu kurallar göz ardı edilemez katı kurallardır. Tasarım şablonlarının kullanılması süreci kolaylaştırabilir.

2.1.3 Kod Denetleme

Kod denetleme en çok vakit harcanılan denetleme sürecidir. Konu ile ilgili doküman sayısı bir hayli fazladır. Kod içinde tutarlı bir şekilde belirlenen standartlara göre kontrol edilmelidir. Kullanıcı rehberleri ve uygulama dokümanları mutlaka denetlenmelidir. Kod insanlar (yazılımcılar, proje yöneticileri, denetleyiciler vb.) tarafından kontrol edilirken bunun yanında otomatik denetleme araçlarından da faydalanılabilinir [15].

Frank H. Lewski yazılım denetlemeyi yukarıdaki üç başlık altında toplamıştır fakat bunun dışında yazılım denetleme işlemi yazılım geliştirme süreçlerinin diğer aşamalarında da kullanılabilir. Şekil 2.1 de denetlemenin yazılım yaşam döngüsünde bulunduğu aşamalara yer verilmiştir [16].



Şekil 2.1 Denetlemenin yazılım yaşam döngüsünde bulunduğu aşamalar [16]

2.2 Kod Denetleme Elementleri

Kod Denetleme süreci temelde aşağıdaki dört elementin etkileşimini içermektedir.

2.2.1 Giriş ve Çıkış Kriterleri

Denetleme adımları öncesinde giriş kriteri ve çıkış kriteri kavramlarını bilmek gerekmektedir.

2.2.1.1 Giriş Kriteri

Giriş kriterleri ürünün denetlemeye hazır olup olmadığını kontrol eden eylem kümeleridir [17]. Denetleme süreci için ölçülebilir bir eylem kümesi belirtmeli ve bu eylem kümesi denetleme sürecinden önce tamamlanmalıdır. Eylemlerin tamamlanması yazılım yaşam döngüsünün önceki safhaları ile ilgili tüm aktivitelerin ilgili denetlemeden önce tamamlandığını garanti eder [18].

Giriş kriterleri ürüne bağlı olarak farklılıklar gösterebilir ve kriterler sağlanana kadar denetlemeye başlanmaz. Aşağıda bir dizi örnek giriş kriterine yer verilmiştir [19].

- Denetleme yapılacak yazılım ürünü tamamlanmış olmalıdır.
- İçerik ve biçim bakımından belirlenen standartlara uygun olmalıdır.
- Eğer otomatik bir hata denetleme aracı kullanılıyor ise bu aracın mevcut olup olmadığı kontrol edilmelidir.
- Gerekli destekleyici dokümanların mevcudiyeti kontrol edilmelidir.

- Denetlenen kaynak kodun derleme hatası olup olmadığının kontrolü yapılmalıdır. Bulunan hatalar düzeltilmelidir [20].

2.2.1.2 Çıkış Kriteri

Çıkış kriteri denetleme sürecinin başarıyla tamamlanıp tamamlanmadığını kontrol eder. Genellikle denetleme esnasında bulunan hataların düzeltildiğini garanti etmek maksadı ile kullanılır. Programın çalışmasını engelleyecek düzeyde olmayan göz ardı edilebilecek hataların düzeltilmesi çıkış kriteri olarak belirtilmeyebilir [19]. Aşağıda bir dizi örnek çıkış kriterine yer verilmiştir.

- Tüm majör hatalar düzeltilip düzeltilmediği kontrol edilmelidir.
- Değişiklik yapılan ürün projenin konfigürasyon yönetim sistemine kaydedilmelidir.
- Moderatör denetleme verilerini bir araya getirmeli ve kayıt altında tutmalıdır [21].

2.2.2 Süreçler

2.2.2.1 Fagan Denetleme Süreçleri

Michael Fagan tarafından altı adet denetleme adımı tanımlanmıştır. Bu adımlar; Planlama, Gözden geçirme, Hazırlanma, Toplantı, Yeniden ele alma ve Takip adımlarıdır [13]. Aşağıda Fagan denetleme adımlarının temel hedefleri listelenmiştir.

Planlama

- Denetleme yapılacak materyaller mutlaka denetleme giriş kriterleri ile uyuşmalıdır.
- Denetleme planlanmalıdır.
- Denetleme katılımcılarının ulaşılabilinir olduğu zaman belirlenmelidir.
- Denetleme için uygun toplantı mekânı ve zaman dilimi belirlenmelidir.

Gözden Geçirme

- Grup çalışması olarak yapılmalıdır.

- Denetleme katılımcılarına neyin denetleneceği konusunda gurup olarak eğitim verilmelidir.
- Katılımcılara rolleri tayin edilmelidir.

Hazırlanma

- Bu adım bireysel olarak gerçekleştirilmelidir.
- Katılımcılar materyalleri anlamalı ve rollerinin gereğini yerine getirebilmek için hazırlanmalıdır.
- Hata belirleme oranının artması için denetleme takımı daha önce gerçekleştirilmiş denetleme işlemlerinde bulunan hata türlerini ve bu hataların dağılımını incelemelidir. Tablo 2.1 de hata dağılım oranları gösteren bir örnek tablo yer almaktadır.

	Denetleme	Dosyası			
BİREYSEL AD	EKSİK	YANLIŞ	FAZLA	HATA	HATA YÜZDESİ
CC Kod Yorumları	5	17	1	23	6.6
CB Kullanım	3	21	1	25	7.2
DE Tasarım Hataları	31	32	14	77	22.1
F1		8		8	2.3
IR Bağıntılı Çağrılar	7	9	3	19	5.5
LO Mantık	33	49	10	92	26.4
MN Korunabilirlik	5	7	2	14	4.0
OT Diğer					
PE Performans	3	2	5	10	2.9
PR Başlangıç	25	24	3	52	14.9
PU PL/S ya da BAL Kullanımı	4	9	1	14	4.0
RU Kayıt Kullanımı	4	2		6	1.7
SU Hafıza Kullanımı	1			1	3
TB Test	2	5		7	2.0
	123	185	40	348	100.0

Tablo 2.1 Denetleme hata dağılım oranları[14]

Toplantı

- Gurup çalışması olarak yapılmalıdır.
- Hata bulma işlemi gerçekleştirilmelidir. (Bu adımda çözüm arama ve alternatif bulma işlemleri önerilmez.)

- Öncelikle yazılımcı tasarımı nasıl koda uyguladığını anlatır. Dizayn anlaşıldıktan sonra hedef hataları bulmak olmalıdır.
- Hata bulma işlemi tamamlandıktan sonra denetleme sürecini yöneten kişi (Moderator) hataları ciddiyetine göre sınıflandırmalı ve yazılı bir denetleme raporu oluşturmalıdır.

Yeniden Ele Alma

- Author rolü verilen katılımcı belirlenen hatalı kısımları düzeltmelidir. (Author rolü tasarımı gerçekleştiren kişiye ya da kodlama işlemini gerçekleştiren kişiye verilir)

Takip

- Denetleme işlemini yöneten katılımcı denetleme sürecinde gerçekleştirilen düzeltme işlemlerinin verimli olduğunu ve hiçbir ikincil hata ile karşılaşılmadığını doğrulamalıdır [12],[14].

NASA yayınladığı Denetleme Rehberinde 7 adet denetleme süreci belirlemiştir. Planlama, Gözden geçirme, Hazırlanma, Toplantı, Yeniden ele alma ve Takip adımları Fagan Denetleme yöntemi neredeyse aynıdır. Bu denetleme adımlarına ek olarak Toplantı adımı sonrasına üçüncü saat adımını eklemiştir.

Üçüncü Saat: İsteğe bağlı ilave zaman dilimidir. Denetleme toplantılarından farklı olarak bu zaman diliminde olası çözümler konuşulabilir ya da toplantı adımında açık kalmış konuların kapatılması üzerinde durulabilir [19].

2.2.2.2 Genel Denetleme Süreçleri

Denetleme sürecinde yer alan adımlar denetleme yöntemlerine göre farklılıklar gösterebilirler. Bazı kod denetleme katılımcıları zaman içersinde Fagan denetleme yönteminde farklılıklara giderek bu altı denetleme aşamasını aşağıdaki üç adıma indirmişlerdir [22],[23].

Hata Tespiti

Hata tespiti aşaması kod denetlemenin çekirdeği yani merkezdeki bölümüdür [22]. Denetleme kavramının ortaya çıkışından günümüze kadar geçen sürede pek çok hata tespit etme yöntemi önerilmiştir fakat tüm bu yöntemlerin temel amacı belirlenen kod modüllerinin olası hatalara karşı incelemesidir. Yazılım mühendisleri kod modüllerinde önemli değişiklikler yalıtıklarında ya da yeni bir kod modülü geliştirdiklerinde mutlaka hata-tespit bazlı denetleme yöntemlerini kullanmalıdır [17],[23]. Saptanan tüm potansiyel hatalar hata tespit raporu oluşturulmalı ve bu rapora kaydedilmelidir.

Hata tespiti adımın nasıl organize edileceği halen literatürde tartışılmakta olan bir konudur. Hata tespitinin bireysel olarak yapılması mı daha verimlidir? Yoksa gurup olarak mı yapılmalıdır? Sorularının cevabı hala netlik kazanmamıştır. Michael Fagan hazırladığı "Fagan Denetleme" yönteminde gurup toplantılarına odaklanmıştır. Denetleme takımının üyelerinin bir araya geldiklerinde bireysel çalışmalarından daha fazla ve daha çabuk hata bulduklarını savunmuştur [22]. Denetleme toplantılarının amacı denetleme takımına sağladığı sinerjidir. Fagan farklı bakış açılarının, yeteneklerin ve birikimlerin gurup olarak bir araya gelinen toplantılarda ortaya çıkardığı sinerji ile hataların daha çabuk belirlendiğini iddia etmiştir [24]. Ayrıca bu sinerji sayesinde yazılım geliştirme süreçlerinde ortaya çıkan pek çok problemin üstesinden rahatlıkla gelinebileceğini söylemiştir [25]. Ebenau ve Strauss "Software Inspection Process" isimli çalışmalarında Fagan'ın sinerji fikrini desteklemişlerdir [27],[26].

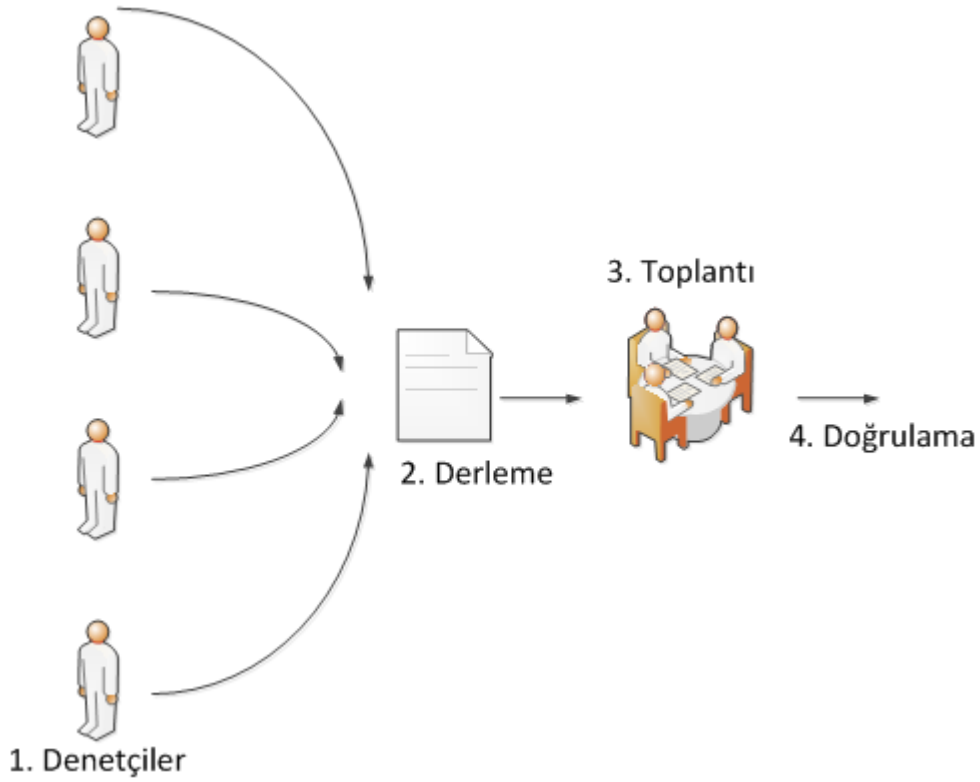
Yapılan son çalışmalarda yazılım mühendisleri hata tespiti aşamasında gurup toplantısı yapılması fikrine zıt düşen sonuçlar elde etmişlerdir. Denetleme toplantılarının yazılım ürününe ait hataları tespit etme konusunda minimal etkiye sahip olduğunu iddia etmişlerdir [27],[28]. McCarthy hata tespit etme işleminin bireysel olması gerektiğini savunmuştur. Hataların hazırlanma aşamasında bireysel olarak tespit edilmesini ve daha sonra denetleme toplantılarında bir araya getirilmesini savunmuştur [27]. Yapılan deneysel çalışmalar hataların %90'ının hazırlanma aşamasında bireysel çalışmaların sonucunda bulunduğunu göstermiştir. Buna karşılık olarak hataların sadece %10'luk bölümü denetleme toplantılarında

saptanmıştır [29]. Hata bulma oranları dışında toplantıların maliyete olan etkisi araştırılmıştır. Denetleme toplantılarının bireysel çalışmalara oranla daha maliyetli olduğunu iddia etmiştir [30].

Hataların Derlenmesi

Hata tespit aşamasında kaydedilen hataların bazılarının daha sonra incelendiğinde gerçek hata olmadıkları görülmüştür. Hataların derlenmesi adımında yapılan toplantıların temel amacı potansiyel hataların gerçek birer tehdit olup olmadığının belirlenmesidir. Bu duruma ek olarak derleme toplantılarında yapılan incelemelerde hata tespiti adımında bulunamamış yeni hatalar tespit edilebilir fakat hata tespit etmek kesinlikle hata derleme adımının birincil hedefi değildir. Daha önce incelenmiş modüllerin yeniden denetlemeye tâbi tutulup tutulmayacağına bu toplantılarda karar verilir. Yeniden denetleme yapılabilmesi için kod denetleme faaliyetlerine ayrılan proje kaynaklarının arttırılması gerekir. Bu yüzden, proje yönetimi yazılım ürününün yeniden incelenmesi kararını akılcı bir şekilde almalıdır. Literatürde bu önemli kararın verilmesine yönelik nesnel (objektif) ve öznel (sübjektif) metotlar önerilmektedir.

Yeniden denetleme sürecinde genellikle Ele Geçirme/Yeniden Ele Geçirme metodu kullanılır. Bu metoda göre yeniden denetleme kararı verilirken hata tespiti yapıldıktan sonra derleme adımında bulunan hatalara yani kalan hata sayısına bakılır. Öncelikle tüm denetleyiciler bireysel olarak hata tespit adımını gerçekleştirirler. Daha sonra hata derleme adımında bir araya gelirler. Tüm denetleyiciler rapora kaydettikleri hataları belirtirler. Herkesin bulduğu ortak hatalar dışında bulunan farklı hatalar belirlenir. Ürünün yapısına göre bir eşik değeri belirlenir ve eğer farklı hata sayısı bu eşiğin üzerinde ise hata tespiti yeniden yapılır. Hata derleme adımında belirlenen tüm gerçek hatalar ve yeniden hata tespiti yapılacak modüller toplantı raporuna kaydedilmelidir [23],[22],[1],[31]. Şekil 2.2 de derleme adımının denetleme sürecindeki yeri belirtilmiştir.



Şekil 2.2 Derleme adımının denetleme sürecindeki yeri [31]

Hata Düzeltme

Hata düzeltme adımı boyunca Author rolünün sorumluluğunu yerine getirerek derleme toplantılarında raporlanan hataları çözümler. Bu adım yapısal test işlemi öncesinde gerçekleştirilir. Literatürde hata düzeltme adımı ile ilgili tartışmaların sayısı yok denecek kadar azdır [23],[13].

Tablo 2.2 de Fagan Denetleme yöntemi sonrasında ortaya çıkan denetleme yöntemleri ve bu yöntemler ile Fagan denetleme yöntemi süreçleri arasındaki karşılaştırmaya yer verilmiştir.

		Planlama	Gözden Geçirme	Hazırlanma	Denetleme Toplantısı	Yeniden Ele Alma	Takip
1976	Fagan Denetleme Yöntemi						
1985	Active Design Denetleme Yöntemi						
1989	Two Person Denetleme Yöntemi						
1990	N-Fold Denetleme Yöntemi						
1993	Fazlı Denetim Yöntemi						
1993	Toplantısız Denetim Yöntemi				Toplama		
1993	Gilb Denetim Yöntemi				Süreç		
		Beyin fırtınası					

Tablo 2.2 Denetleme yöntemlerinin süreç bazlı karşılaştırılması [44]

2.2.3 Denetleme Takımının Oluşturulması ve Roller

Yazılım oluşturma büyük ölçüde bireysel entelektüel bir etkinliktir. Yazılımın her bir parçası bir takımda yer alan bireyler gibidir çünkü her bir parçanın kendine özgü bir dünyası vardır. Bir yazılım ürününü verimli bir şekilde denetleyebilmek için denetleme takımı bu dünyayı iyi tanımalı ve sırlarını keşfetmiş olmalıdır [13]. Denetleme takımı aynı seviye ve kabiliyette ürüne hâkim personellerden oluşmuş bir guruptur [19]. Takım üyeleri birden fazla role sahip olabilirler. Her bir rol için özel kabiliyet ve bilgi birikimi gerekir. Michael Fagan Author, Moderator, Reader ve Tester olmak üzere dört adet denetleme rolü tanımlamıştır [24]. Denetleme takımının büyüklüğü 3 ile 7 kişi aralığında bir değerdir. Büyük takımlar üst düzey belgeleri incelemek için tercih edilirken, küçük takımlar ise daha detaylı teknik seviyelerde tercih edilir.

Denetleme takımının her bir üyesi rolünün getirdiği özel sorumluluklara sahip olmalıdır [19]. Katılımcılar rollerinin gereğini yerine getirirlerse en iyi hizmeti sunmayı başarır [14].

2.2.3.1 Denetleme Takımının Büyüklüğü

Kod denetleme yazılım kalitesini arttıran ve maliyeti azaltan bir yazılım mühendisliği çalışması olarak kabul edilmektedir. Ancak ideal gurup sayısı ve maliyet konusu halen tartışılmaktadır [25]. Fagan'a göre ideal denetleme gurubu dört kişilik olmalıdır. Weller 1993 yılında yaptığı çalışmada bu konu üzerine eğilmiş ve dört kişilik gurubun 3 kişilik guruptan daha başarılı olduğunu iddia etmiştir. IEEE 1997 yılında yazılım denetleme için yayınladığı IEEE STD-1028 standartlarında takım sayısının üç ile altı arasında olması gerektiğini savunmuştur [20]. D.A. Wheeler 1993 yılında gerçekleştirdiği endüstriyel çalışmada dört ya da beş kişiyle daha başarılı olunacağını iddia etmiştir [32].

Yakın geçmişte yapılan bazı çalışmalarda yazılım mühendisleri bu geleneksel görüşe karşı çıkmışlardır. Bu konu ile ilgili olarak en çok referans gösterilen çalışma L.Votta'nın çalışmasıdır. L. Votta kalabalık ekiplerle yapılan denetlemelerin hata bulma hususunda kayda değer bir etkisi olmadığını, aksine bu tarz denetleme toplantıları üç ile yedi arasında değişen sayılarda insan barındırdığı için maliyeti ciddi oranda arttırdığını iddia etmiştir. Bisant and Lyle 1989 yılında yayınladıkları makalede kod denetleme için iki kişinin yeterli olacağını ve bu iki kişilik denetleme ile denetleme hızının arttığını söylemişlerdir [25].

2.2.3.2 Fagan Denetleme Yöntemi Rolleri ve Sorumlulukları

Moderatör

Fagan moderatörü anahtar oyuncu olarak belirlemiştir. Fagan'a göre moderatör;

- Tecrübeli bir yazılımcı olmalıdır fakat denetlenen program hakkında teknik uzmanlığa sahip olmasına gerek yoktur.
- Liderlik becerilerine sahip olmalıdır. Dengeli ve ölçülü olmalıdır. Havayı koklamalı ve ona göre davranmalıdır. Kişisel duyarlılığa sahip olmalıdır.

- Sinerji oluşturabilmek için takım elemanlarının güçlü yönlerini kullanmalıdır.
- Uygun toplantı yeri ve zamanını belirlemelidir.
- Denetleme sonuçlarının yer aldığı rapor denetleme işleminin yapıldığı gün oluşturulmalı ve hata düzeltme işlemini gerçekleştirecek takım elemanlarına iletilmelidir.
- Denetleme sürecinden en iyi şekilde faydalanabilmek için özel olarak eğitilmiş olmalıdır [14].

Author

Fagan'ın yaptığı tanımlamaya göre Author,

- Tasarımcı veya yazılımcı olmalıdır. Tasarımcı program dizaynını oluşturur. Yazılımcı ise tasarımı koda dönüştürür [14].
- Hata düzeltme adımında belirlenen hataları düzeltmelidir [12].

Reader

Fagan'ın yaptığı tanımlamaya göre Reader,

- Yazılımcı olmalıdır.
- Denetleme toplantıları esnasında dizayn veya kodu ana hatlarıyla anlaşılır bir şekilde anlatarak denetleme takımına yol göstermelidir [19].

Tester

Fagan'ın yaptığı tanımlamaya göre Tester,

- Yazılımcı olmalıdır.
- Denetleme sırasında sürece test aşamasındaki gibi bir bakış açısıyla yaklaşmalıdır [12].

2.2.4 Denetleme Süreçlerinde Kullanılan Okuma Teknikleri

2.2.4.1 Okuma Teknikleri

Okuma teknikleri denetleyicilere ürünü derinlemesine anlayabilmeleri için rehberlik eden yöntemlerdir. Denetlenen ürünün anlaşılması hemen göze çarpmayan ya da karmaşık olan hataları saptayabilmeleri için ön koşuldur. İncelenen ürünün

kusurlarını tespit etmek için kullanılan bir mekanizma ya da strateji olarak düşünülebilir. Okuma tekniklerinin kullanımı sayesinde hata tespiti yapılırken insan faktörüne bağlı değişikliklerden daha az etkilenilir [22]. Araştırmacılar okuma tekniği seçiminin denetleme performansını doğrudan etkilediği görüşünde hemfikirdir. Okuma teknikleri Simetrik okuma teknikleri ve Asimetrik okuma teknikleri olmak üzere iki guruba ayrılmıştır. Simetrik okuma teknikleri sürece bir hayli açık ve yapısal bir yaklaşım uygular. Denetleyicilere yazılım dokümanını nasıl okuyacaklarını anlatan talimatlar seti sağlar. Asimetrik okuma teknikleri sezgisel bir yaklaşım uygular ve denetleyicilere kısıtlı destek sağlar [25].

2.2.4.1.1 Ad Hoc

Denetleyiciler için açık ya da belli bir yöntem yoktur. Eğitim gerekli değildir. Denetleyiciler kendi bilgi birikimlerine, kabiliyetlerine ve tecrübelerin dayanarak dokümanlardaki hataları belirlerler. Ad Hoc okuma tekniğinde denetleyiciler için herhangi bir destek önerilmez [24].

2.2.4.1.2 Kontrol Listesi (Checklist)

Kontrol Listesi okuma tekniği en popüler okuma tekniğidir [22]. Kontrol Listesi okuma tekniği Ad Hoc okuma tekniğinden daha sistematiktir. Denetleyici listede yer alan önceden tanımlanmış soruları cevaplar ya da önceden tanımlanmış kontrol edilmesi gereken sorunları işaretler. Belirlenen sorular denetleme süreci boyunca denetleyicilere rehberlik etmelidir. Kontrol Listesi okuma tekniğinde amaç denetleyicilerin sorumluluklarını tanımlamak ve onlara hataları tanımlayabilmeleri için rehberlik etmektir. Örneğin Gilb ve Graham doğrulama listeleri her bir rol, ürün ve doküman için özel olarak hazırlanmaktadır [24],[33]. Doğrulama listeleri oluşturulurken aşağıdaki prensiplere dikkat edilmelidir.

- Doğrulama listesinin uzunluğu bir sayfayı geçmemelidir.
- Doğrulma listesinde yer alan sorular olabildiğince kesin ifade edilmelidir.
- Doğrulama listeleri iyi yapılandırılmalıdır. Sorular kalite özelliklerinin nasıl güvenceye alınabileceği konusunda ipuçları vermelidir [22].

2.2.4.1.3 Adım Adım Ayırma Okuma Tekniği (Stepwise Abstraction)

Adım Adım Ayırma okuma tekniği yetmişlerin sonunda doğrulamaya dayalı denetleme yaklaşımı bağlamında kod okuma tekniği olarak geliştirilmiştir. Cleanroom yazılım geliştirme metodu ile birlikte kullanılmaktadır. Her denetleyici yazılımda yer alan program parçacıklarını inceler ve bu parçacıklarda yer alan fonksiyonları belirler. Daha sonra denetleyiciler belirledikleri fonksiyonları bir araya getirerek programın tamamında yer alan fonksiyonları saptarlar. Bu karşılaştırma sayesinde uyumsuzluklar belirlenir ve bir sonraki aşamada bu uyumsuzluklar üzerinde çalışılır [24]. Adım Adım Ayırma okuma tekniği denetleyicilere fonksiyonel doğruluğun kontrol edilmesinde daha resmi bir yaklaşım sunar [22].

2.2.4.1.4 Hata Bazlı Okuma (Scenario Based Reading - Defect Based Reading)

Hata Bazlı okuma tekniği hataları gereksinim dokümanında belirlemek için geliştirilmiştir. Hatalar sınıflandırılmıştır ve her bir hata sınıfı için soru gurupları hazırlanmıştır. Bu sınıflandırılmış hataları saptayabilmek için senaryolar oluşturulur. Denetleyiciler bu özel senaryoları takip ederek soruları cevaplarlar [24]. Senaryolar özel olarak tanımlanan hata tespit işlemi yapılırken denetleyicileri belirli sınırlar içersinde görevini gerçekleştirmeye zorlar çünkü her denetleyici farklı birer senaryo kullanır ve her senaryo farklı bir hata tipi üzerine yoğunlaşmıştır. Bu okuma tekniği uygulanırken denetleme ekibi bir arada çalışır ise süreç daha verimli ilerler. Scenario Based okuma yönteminin verimliliği senaryonun içeriğine ve planlanma şekline göre farklılıklar gösterir. Kısaca bu okuma tekniğinin amacı denetleyicilere nasıl hata saptayacaklarını anlatan özel bir rehber sunmaktır [22].

2.2.4.1.5 Perspektif Tabanlı Okuma (Perspective Based Reading)

Perspektif Tabanlı Okuma NASA tarafından geliştirilmiştir. Bu teknikte senaryo kavramı genişletilmiştir. Senaryolar denetleyicilerin bakış açılarına ve ihtiyaçlarına odaklanmıştır. Her senaryo soru listeleri içermektedir. Denetleme süreci boyunca denetleyiciler özel bakış açıları ile dokümanları inceler ve fiziksel bir model oluştururlar. Bu fiziksel model denetleyicilerin görüşleri esas alınarak hazırlanan soruların cevaplanması ile analiz edilir [24].

NASA'nın çalışmalarından önce Fagan yayınladığı dizayn ve kod denetleme makalesinde kodun test uzmanı gözü ile de incelenmesi gerektiğini savunmuştur. Farklı bakış açılarından faydalanabilmek için farklı özelliklere sahip roller belirlemiştir.

Bu okuma tekniğinde amaç yazılım ürününün farklı bakış açılarına sahip denetleyiciler tarafından incelenmesidir çünkü yazılım kalitesi kavramının tek bir tanımı yoktur. Sadece anahtar kalite parametrelerinin (örn: doğruluk, sürdürülebilirlik, sınanabilirlik) nasıl belirleneceği konusunda genel bir kabul söz konusudur. Perspektif Tabanlı Okuma tekniği gereksinim dokümanları, obje tabanlı dizayn modelleri ve kod dokümanları denetlenirken kullanılabilir [22].

2.2.4.1.6 Aktif Tasarım Denetimi (Active Design Review)

Aktif Tasarım Denetimi okuma tekniği Parnas ve Weiss tarafından seksenli yılların ortasında tanıtılmıştır [34]. Yazarlar geleneksel denetleme yöntemlerinin ürün geliştirme sürecinde hata bulma işlemlerinde başarısız olduğunu iddia etmiştir. Bunun sebebini aşağıdaki üç madde ile ilişkilendirmişlerdir.

- Hazırlanma sürecinde denetleyicilere aşırı bilgi yüklenmektedir ve konu ile ilgili bilgileri bulmak zordur.
- Çoğu durumda denetleyiciler ürün ve amacına pek aşina değillerdir.
- Büyük toplantılarda ortaya çıkan sosyal etkileşim değerlendirme korkusu gibi bazı sorunları da beraberinde getirmiştir.

Bu sorunları üstesinden gelebilmek için denetleyiciler kabiliyetleri ve tecrübeleri seviyesine göre seçilmeli ve her bir denetleyiciye özel sorumluluklar verilmelidir [24]. Denetleyicilere rehberlik etmesi için soru serileri verilmelidir [22]. Denetleyiciler aynı kontrol listesini kullanmamalıdır. Her bir süreç için ayrı listeler oluşturulmalıdır [35]. İlk olarak denetleyicilere ürün hakkında bilgi verilmelidir. Daha sonra denetleyiciler verilen soru listeleri ile beraber materyal üzerinde çalışmalıdır ve son olarak küçük toplantılarda yaptıkları çalışmalar ve sonuçlarını tartışmalılardır [24].

2.2.4.2 Okuma Teknikleri ve Karşılaştırmalı Analizi

Hangi okuma tekniğinin kullanılması ya seçilmesi gerektiği konusunda genel bir yöntem pek yoktur. Aşağıda yer alan çalışmada okuma teknikleri Uygulama İçeriği, Kullanışlılık, Tekrarlanabilirlik, Uyum Sağlama, Kapsam, Eğitim ve Doğrulama ölçütleri kullanılarak analiz edilmiştir. Bu ölçütler aşağıdaki sorulara cevap sağlamaktadır.

Uygulama İçeriği: Yazılım ürününe hangi okuma tekniği uygulanabilir ve yazılım ürününe hangi okuma tekniği uygulanmıştır?

Kullanışlılık: Okuma tekniği yazılım ürününün hata bulmak için nasıl incelenmesi gerektiğini anlatan kurallı bir rehber sunuyor mu?

Tekrarlanabilirlik: Denetleyicinin yaptığı çalışmanın sonuçları tekrarlanabilir mi ve denetleme sonuçları hata buluyor mu?

Uyum Sağlama: Okuma tekniği ortamdaki tipik hata profilleri gibi belirli durumlara adapte olabiliyor mu?

Kapsam: Okuma tekniği yazılım ürününün gerekli tüm kalite niteliklerini denetliyor mu?

Eğitim: Denetleyicilere bu okuma tekniğini kullanabilmeleri için eğitim verilmesi gerekiyor mu?

Doğrulama: Okuma tekniği bugüne kadar enine boyuna nasıl uygulanmış, okuma tekniği nasıl doğrulanmış? [22].

Tablo 2.3 de okuma teknikleri bu ölçütlere göre analiz edilmiştir.

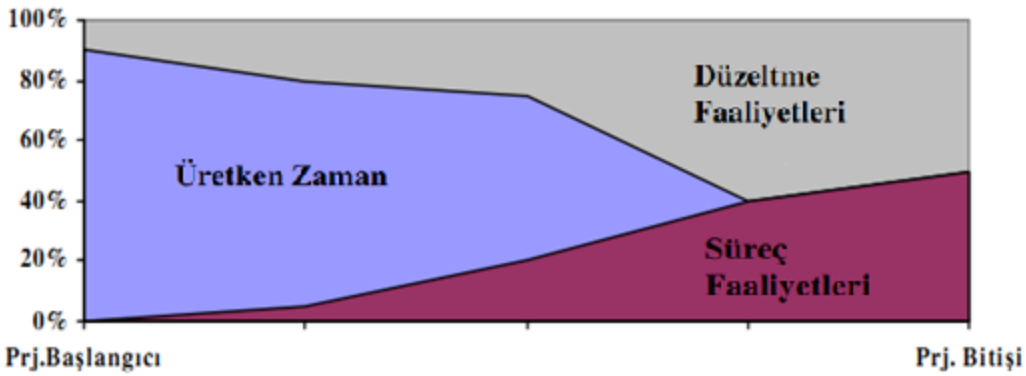
Okuma Tekniği	Karakteristik Özellikler						
	Uygulama İçeriği	Kullanılabilirlik	Tekrarlanabilirlik	Uyum Sağlama	Kapsam	Eğitim	Doğrulama
Ad-hoc	Bütün Ürünler	X	X	X	X	X	Endüstriyel Uygulama
Kontrol Listesi	Bütün Ürünler	X	X	√		X	Endüstriyel Uygulama
Adım Adım Ayırma Yöntemi ile Okuma	Bütün Ürünler	√	√	X	Yüksek	√	Projeler
Aktif Tasarım Denetim	Tasarım	√	√	√	?	?	Başlangıç Çalışması
Hata Bazlı Okuma	Bütün Ürünler, Gereksinimler	√	Duruma Bağlı	√	Yüksek	√	Deneysel Doğrulama
Perspektif tabanlı Okuma	Bütün Ürünler, Gereksinimler, Tasarım Kodu	√	√	√	Yüksek	√	Deneysel Doğrulama ve Başlangıç Endüstriyel Uygulama

Tablo 2.3 Okuma tekniklerinin kıyaslanması [22]

2.3 Kod Denetleme Sürecinin Faydaları

2.3.1 Kod Denetleme Kaynak Kodun Kalitesini Arttırır

Don O'Neill kod denetlemenin yazılım geliştirme safhasında %50 oranında hataların erkenden tespit edilmesini ve giderilmesini sağladığını iddia etmiştir. IBM de yapılan denetleme işlemlerinin sonucunda hataların erkenden tespit edilmesi oranının %83 olduğu görülmüştür. Aynı oran AT&T laboratuarlarında %92 olarak tespit edilmiştir. Hataların erken tespiti ve giderilmesi kod kalitesini attırmaktadır. Kod bir sonraki safhaya daha az hata oranı ile geçmektedir. Şekil 2.3 de yazılım kalitesizliğinin proje zamanına etkisi gösterilmiştir.



Şekil 2.3 Yazılım Kalitesizliğinin proje zamanına etkisi [67]

Kod denetleme yeni geliştirmenin yanı sıra bakım çalışmaları içinde verimli bir süreçtir. Watts S. Humphrey bakım yapılan bir yazılım ürününde kod denetleme sayesinde hata bulma oranının %55 olduğunu belirtmiştir. Bunun yanında hata oranı %2 azalmıştır.

Kod denetleme kod kalitesini doğrudan etkilemektedir. Denetleme esnasında mantık hataları, kullanılmayan değişkenler, kod ile ilgisi olamayan yorumlar, kullanılmayan kod parçaları gibi kısımlar belirlenmekte ve düzeltme safhasında koddan çıkarılmaktadır [36],[37].

2.3.2 Kod Denetleme Verimliliği Arttırır

Humphrey AT&T Bell laboratuarlarında yapılan çalışmaların sonucunda kod denetleme sayesinde yazılım geliştirme verimliliğinin %14 oranında arttığını belirtmiştir. Ayrıca laboratuarda yapılan çalışmalarda kod denetlemenin testten 20 kat daha etkili olduğu görülmüştür [36]. IBM kod denetleme sürecinin gerçekleştirilmesi sonucunda kod geliştirme verimliliğinin %23 oranında arttığını gözlemlemiştir. Bununla birlikte bütünlük testlerinde hata belirleme oranı %38 oranında düşmüştür [14],[38].

2.3.3 Kod Denetleme Maliyeti Düşürür

Hataların erkenden belirlenmesi ve düzeltilmesi kodun yeniden ele alınması sayısını ve bu sürece bağlı olan maliyetleri düşürür. Don O'Neill yaptığı çalışmalar sonucunda denetleme yapılmadan hataların test aşamasına aktarılmasının denetleme

aşamasında bulunup düzeltilmesinden 9 kat daha maliyetli olduğunu söylemiştir [37].

2.3.4 Kod Denetleme Ekip İlişkisini ve Eğitimini Teşvik Eder

Kod denetleme süreci yazılım geliştirme personeline yaptığı çalışmaları ve tecrübelerini paylaşma imkânı sunar. Bunun yanında yazılımcıları birlikte çalışmaya ve kaliteli kod geliştirmeye teşvik eder. Hataların incelenmesi ve tartışılması yazılımcıların kendilerini geliştirmelerine yardımcı olur.

Kod denetleme toplantılarına acemi yazılım geliştirme personelleri dâhil edilmelidir. Burada tecrübeli yazılımcılardan daha iyi nasıl kod yazacakları hakkında bilgi edinir ve yazılım ürünün yapısını daha iyi anlarlar [36].

Kod denetleme verileri personel performans değerlendirmesine dâhil edilmemelidir. Yazılım geliştirme personelinin psikolojisi yazılım kalitesini etkileyen bir faktördür. Denetleme verilerinin performans göstergesi olarak kullanılması insanlar üzerinde olumsuz bir baskı oluşturacak ve psikolojilerini olumsuz yönde etkileyecektir [12],[39].

2.3.5 Kod Denetleme Kalite Kültürü Oluşmasını Teşvik Eder

Kod denetleme kalite odaklı bir yapıdır. Hata kayıtlarının tutulması sayesinde yazılım geliştirme personeli aynı hataları tekrar yapmamak için çaba gösterir [12]. Standartlar ve denetlemenin varlığı yazılımcıları daha kaliteli kod yazmaya teşvik eder [36].

3. Kod Denetleme Sürecini Etkileyen Değişkenler ve Denetleme Metrikleri

3.1 Kod Denetleme Verimliliği ve Ölçülmesinin Önemi

Son zamanlarda teknoloji pazarında büyük yarışlar yaşanmaktadır. Yazılım endüstrisinde maliyet, yazılan kodun kalitesi, zaman ve hizmet elverişliliği gibi faktörler bu yarışlarda belirleyici rol üstlenmektedir. Bu yüzden yazılım firmalarının kaliteli yazılım üretme konularına eğilmeleri onların faydalarına olacaktır.

Yazılım firmaları iş dünyasında başarılı olabilmek için ciddi rakamlara ulaşan yatırımlar yapmaktadır. Kaliteli ürünler elde edebilmek için ölçülebilir hata yönetim süreçleri gerçekleştirmelilerdir [6].

Shu, Boehm ve Wang yazılım firmalarının kalitelerini kalite güvence süreçlerini iyi bir şekilde özümseyip uygulayabildiklerinde üst seviyeye taşıyacaklarını iddia etmektedirler [41]. Bir ürünün sürekliliği yani sürdürülebilirliği kalitesine bağlıdır [42]. Hata tespiti ve hatanın giderilmesi kaliteli ürünlerin oluşturulmasındaki en önemli problemlerdendir. Bu sebeple müşteri memnuniyetini sağlamak ve uzun vadede başarıyı elde edebilmek için iyi organize edilmiş hata yönetim sistemi oluşturmak birincil hedefler arasında yer almalıdır. Etkili hata yönetimi yazılım endüstrisindeki yarışta avantaj sağlayacaktır [6].

Denetleme ve test etme kendini ispatlamış iki etkin hata yönetim metodudur [6]. Yazılım denetleme işlemi yazılım kalitesini geliştirmek için etkili, verimli bir metot olarak düşünülmüş ve tercih edilmiştir. Denetleme işleminin amacı test aşamasından önce hataları saptamaktır [43]. Hata saptama ve korunma kaliteli ürün sağlamayı hedefler. Hatalara zamanında müdahale edilmesiyle beraber maliyet azalır, üretkenlik artar ve müşteri memnuniyeti üst seviyelere çıkar [6].

Otoriteler genel olarak kod denetlemenin maliyeti düşürdüğü ve üretkenliği arttırdığı konusunda hem fikirlerdir. Denetleme işleminin iyi yönetilmesi ve verimliliğinin artması ürüne yansıyacak ve ürünün kalitesini arttıracaktır. Proje yöneticileri aşağıdaki işlemleri yaparak denetleme enformasyonlarını etkili bir şekilde toparlar ve kullanabilirler [44].

- Kaynakların bölüştürülmesi
- Prosedürlere uygunluğun kontrolü
- Denetlenen ürünün kalitesinin belirlenmesi
- Denetleme sürecinin verimliliğinin ölçülmesi ve iyileştirilmesi

Hata yönetim sistemleri verimliliği, tercih edilen süreçlere ve bu süreçleri gerçekleştiren insanlara bağlı olarak farklılıklar gösterebilir. Kaliteli bir ürün için hata yönetimi verimliliği değerlendirilmelidir [6]. Hewlett-Packard firmasında program yöneticisi olan Tom Kendrik David Packard'ın ölçemediğini yönetemezsin sözünü hatırlatarak metriklerin kalitenin iyileştirmesinde önemli bir rolü olduğunu söylemiştir [8].

3.2 Metrik Kavramı

Metrikler süreci, ürünü ve kişi performanslarını ölçen nümerik değerlerdir.[45] Ölçme standardı olarak da tanımlanabilirler [46]. Metrikler sürecin, ürünün ve kişilerin verimliliğini ölçer, tanımlar, yönetir, gözlemler ve geliştirirler [47]. Ölçüm yapılmadan dünyadaki değişiklerin farkına varılamaz ve gidişatın iyi ya da kötü olduğu anlaşılamaz [46]. Goodman projelerin metrikler ile değerlendirilmesinin bilgi teknolojileri yönetiminde başarıyı getiren en önemli uygulamalardan biri olduğunu iddia etmiştir [47].

Bir yazılım geliştirme projesi süresince metriklerin dört temel kullanımı söz konusudur.

- Yöneticiye projenin ne aşamada olduğunu gösterir.
- Karar alma aşamasında bilgi sunar.
- Gerçekleştirilecek projeler öncesinde projeye yönelik tahmin bilgileri sunar.
- Ürünün kalitesi ve güvenilirliği hakkında bilgi sağlar [46].

Metrikler süreç ve kişilerin performans ölçümünde kalite göstergesi olarak kullanılabilirler. Bunun yanında ürünün kalite seviyesinin belirlenmesi ya da tahmin edilmesi işlemlerinde yer alabilirler [6].

H. Stephan yazılım kalite metriklerini üç büyük kategoride sınıflandırmıştır. Bunlar ürün metrikleri, süreç metrikleri ve proje metrikleridir.

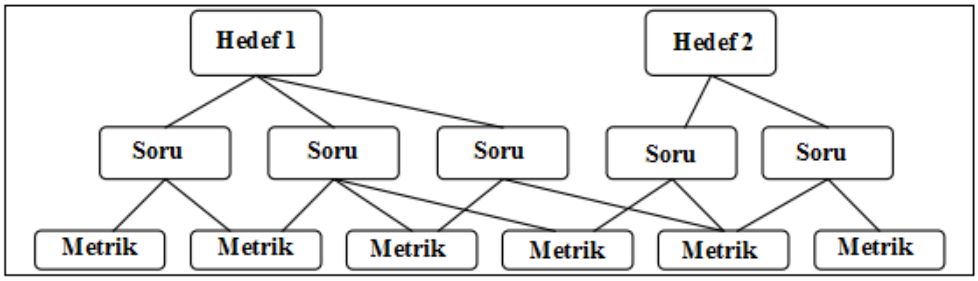
1. Ürün Kalite Metrikleri: Ürünün karakteristik özelliklerini tarif eden metriklerdir. Ürün kalitesi ürünün boyutu, karmaşıklığı, performansı ve dizayn özellikleri dikkate alınarak ölçülür.
2. Süreç Metrikleri: Ürünün oluşturulması aşamasında gerçekleştirilen yazılım geliştirme süreçlerinin ve yazılım bakım süreçlerinin iyileştirilmesine katkıda bulunan metriklerdir. Hata giderme verimliliği sık kullanılan popüler bir süreç metriğidir.
3. Proje Metrikleri: Proje özelliklerinin yazılım maliyeti ve personel yapısı gibi değerler kullanılarak nicel ölçülmesidir [6].

Ampirik projeler üzerinde yapılan stratejik araştırmalar yazılım endüstrisinde hata profilini ölçmek için kalite metriklerinin kullanıldığını göstermiştir [48]. Yazılım denetleme metriklerinin kullanılmasının ana hedefi hata tespiti işlemini iyileştirmek ve yeniden ele almanın maliyetini düşürmektir. Ürünün dağıtılması sürecinde hata ile karşılaşılmasının maliyeti oldukça yükseltmektedir [11].

Kalite ölçümünde kullanılacak metrikler seçilirken akıllıca davranılmalıdır. Westfall hedef odaklı metriklerin tercih edilmesi gerektiğini iddia etmiştir [47].

3.2.1 Hedef Soru Metrik Paradigması Kullanılarak Metriklerin Oluşturulması

Öncelikle iyi bir ölçüm planı oluşturmalıdır. Bu plan kullanılacak metrikleri ve ne amaçla kullanılacaklarını açıklamalıdır. Eğer plan hazırlanmaz ise çok fazla ya da çok az, hatalı ya da gereksiz bilgi toparlanabilir. Plan oluşturmadaki genel problemlerden biri neyin ölçümleneceğinin bilinmemesidir. Hedef Soru Metrik (GQM) paradigması bu sorunun çözümü için etkili bir tekniktir. GQM ölçümleme ihtiyaçlarını metriklere dönüştüren bir yaklaşımdır. Öncelikle ölçme işleminin amaçları listelenmeli ve sorular ile ilişkilendirilmelidir. Her bir sorunun tek bir ölçüm hedefi olmalıdır. Şekil 3.1 de bu paradigmanın genel yapısı tasvir edilmiştir [44].



Şekil 3.1 Hedef-Soru-Metrik yapısı [68]

Bu yöntem AT & Bell laboratuarlarında uygulanmıştır. Hedefler dikkate alınarak hazırlanan sorular neticesinde dokuz metrik belirlenmiştir. Laboratuarda yapılan çalışma Tablo 3.1 yer almaktadır. Tablo incelenerek GQM yapısını daha iyi anlaşılabilir [44].

HEDEFLER, SORULAR, METRİKLER		
Hedef	**Sorular**	**Metrikler**
Plan	Denetim Süreçlerinin maliyeti ne kadardır?	KLOC Başına Düşen Ortalama Çaba Yeniden Denetleme Oranı
	Denetim Süreçleri ne kadar sürer?	KLOC Başına Düşen Ortalama Çaba Denetlenen toplam kaynak kod satırı
Gözlem ve Kontrol	Denetlenen yazılımın kalitesi nedir?	KLOC Başına Düşen Ortalama Hata Tespiti Ortalama Denetleme Oranı Ortalama Hazırlanma Oranı
	Çalışanlar prosedürlere hangi seviyede uygunluk göstermişlerdir?	Ortalama Denetleme Oranı Ortalama Hazırlanma Oranı Denetlenen ortalama kod satırı Yeniden Denetleme Oranı
	Denetim sürecinin durumu nedir?	Denetlenen toplam kaynak kod satırı
Geliştirme	Denetim süreci ne kadar etkilidir?	Hata Giderme Etkinliği KLOC Başına Düşen Ortalama Hata Tespiti Ortalama Denetleme Oranı Ortalama Hazırlanma Oranı Denetlenen ortalama kod satırı
	Denetim sürecinin üretkenliği nedir?	Tespit edilen hata başına düşen ortalama çaba Ortalama Denetleme Oranı Ortalama Hazırlanma Oranı Denetlenen ortalama kod satırı

Tablo 3.1 AT & Bell laboratuarları hedef soru metrik uygulaması [44]

3.2.2 Kod Denetleme Sürecinin Değerlendirilmesi İçin Kullanılan Metrik Örnekleri

Denetleme verilerini toparlamak ve değerlendirmek oldukça zor bir süreçtir. AT & Bell Laboratuarlarında uzmanlar GQM paradigması kullanılarak dokuz anahtar metrik tanımı yapmıştır

1. Denetlenen Toplam Kaynak Kod Satırı

   Denetlenen kaynak kod satırı genellikle denetleyicilerin performansını değerlendirmek için kullanılan metriklerde hesaplamaya dâhil edilir. Bunun yanında denetlenecek kod miktarının belirlenmesinde ölçüt olarak kullanılır.

   $$\text{Denetlenen Total KLOC} = \frac{\sum_{i=1}^{N} LOC_i}{1{,}000}$$

   N: Total denetleme sayısı

   LOC: Denetlenen kod satırı miktarı

2. Denetlenen Ortalama Kod Satırı

   $$\text{Denetlenen ortalama kod satırı} = \frac{\text{Total KLOC} \times 1{,}000}{N}$$

   N: Total denetleme sayısı

3. Ortalama Hazırlanma Oranı

   Kod denetleme işleminin gerçekleştirildiği yazılım ürününün kalitesini ölçmek maksadıyla kullanılır. Ürün kalitesinden taviz vermemek için ortalama hazırlanma oranı değeri bir saat için 150 kod satırı civarında olmalıdır. Hazırlanma sürecinin verimliliğini ölçmek amacıyla da kullanılır. Eğer hazırlanma süreci bireysel olarak gerçekleştiriliyorsa denetleyicilerin hazırlanma safhasındaki performansını değerlendirmek için kullanılabilir. Genel olarak denetleme sürecinin ne kadar etkili ve verimli olduğunun belirlenmesine yardımcı olur.

   $$\text{Ortalama Hazırlanma oranı} = \frac{\text{Total KLOC} \times 1{,}000}{\sum_{i=1}^{N} \frac{P_i}{I_i}}$$

   P: Hazırlanma Süresi

I: Denetleyici Sayısı

N: Total denetleme sayısı

4. Ortalama Denetleme Oranı

   Ortalama denetleme oranı metriği değerlendirme süreçlerinde ortalama hazırlanma metriği ile birlikte kullanılır. Ürün kalitesinin ölçülmesinde, denetleme sürecinin verimliliğinin ve denetleyicilerin performansının değerlendirilmesinde kullanılır.

   Ortalama denetleme oranı = $\frac{\textbf{Total KLOC x 1,000}}{\sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{ID_i}}$

   ID = Denetleme Süresi

   N: Total denetleme sayısı

5. KLOC Başına Düşen Ortalama Çaba

   Denetleme işlemi için gerekli olan eforun belirlenmesi önemli bir konudur. Bu noktada KLOC başına düşen ortalama çaba metriği kullanılır. Saatte ortalama 150 kaynak kod satırı denetlendiği varsayılır ise KLOC başına düşen ortalama çaba değeri ortalama 50 saat olarak alınabilir. Bunun dışında denetleme işlemi için gerekli olan zamanın belirlenmesinde kullanılır. Denetleme işlemine tabi tutulan yazılım ürünün kalite değerlendirmesinde KLOC başına düşen ortalama çaba değeri dikkate alınır.

   KLOC başına düşen ortalama çaba = $\frac{\sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{IE_i}}{\mathbf{1,000}}$

   $\mathbf{IE_i = P_i + (NP_i \times ID_i) + R_i}$

   N: Total denetleme sayısı

   P: Hazırlanma Süresi

   IE: Denetleme eforu

   NP: Denetlemeye katılanların sayısı

   ID = Denetleme Süresi

   R = Hataların düzeltilmesi için harcanan süre

6. Tespit Edilen Hata Başına Düşen Ortalama Çaba

   Genellikle denetleme sürecinin sonunda yöneticiler sürecin verimliliğini değerlendirir ve iyileştirmek için neler yapılabileceğini araştırır. Kod

denetleme verimliliğinin analiz edilmesi için tespit edilen hata başına düşen ortalama çaba metriği kullanılır.

Tespit edilen hata başına düşen ortalama çaba = $\frac{\sum_{i=1}^{N} \mathbf{IE_i}}{\sum_{i=1}^{N} \mathbf{TFD_i}}$

N: Total denetleme sayısı

IE = Denetleme eforu

TFD: Denetleme sürecinde tespit edilen hata sayısı

7. KLOC Başına Düşen Ortalama Hata

   KLOC başına düşen ortalama hata metriği denetlenen ürünün kalitesinin ölçülmesinde kullanılır. Bunun dışında denetleyicilerin performansının değerlendirilmesinde ve denetleme sürecinin ne kadar etkili olduğunun belirlenmesinde bu metrikten faydalanılır. Örneğin ilk denetlemede KLOC başına düşen ortalama hata 90'dan fazla ise kaynak kod düzeltme işleminin ardından yeniden denetlenmelidir.

   KLOC başına düşen ortalama hata = $\frac{\sum_{i=1}^{N} \mathbf{TFD_i}}{\mathbf{KLOC}}$

   N: Total denetleme sayısı

   TFD: Denetleme sürecinde tespit edilen hata sayısı

8. Yeniden Denetleme Oranı

   Denetleme işlemi için gerekli olan çabanın belirlenmesinde ve denetleyicilerin performansının değerlendirilmesinde yeniden denetleme oranı metriği kullanılır.

   Yeniden denetleme oranı = $\frac{\boldsymbol{RD}}{\boldsymbol{N}} \times \mathbf{100}$

   RD = Yeniden denetleme eğilimi

   N: Total denetleme sayısı

9. Hata Giderme Etkinliği

   Kod denetlemenin verimliliği ölçülmeden önce sürecin hangi yönlerinin daha önemli olduğuna karar verilmelidir. Kod denetleme test ve bakım maliyetlerini düşüren ve kaliteyi arttıran bir süreçtir. Bu faydalar ürün için önemlidir. Denetleme sürecinin etkinliğini ölçmek için hata giderme etkinliği esas alınmıştır. Bu metrik denetleme süreci sayesinde giderilen hataların oranını test esnasında bulunan hatalar ve müşteri tarafından bulunan hatalar

ile kıyaslayarak ölçer. Denetleme sürecinde bulunan hata sayısını tek başına kullanmak verimliliği ölçmek için yeterli değildir çünkü ürünün yapısına karmaşıklığına göre içerdiği hata sayısı farklılıklar gösterir. Hata giderme etkinliği kod yapısından bağımsızdır.

Hata giderme etkinliği geliştirme süreci sona erene kadar hesaplanamaz. Eğer sürecin o an nasıl ilerlediği görülmek isteniyorsa KLOC başına düşen ortalama hata metriği kullanılmalıdır.

Hata giderme etkinliği = $\frac{\sum_{i=1}^{N} \mathbf{TFD_i}}{\mathbf{FDD}} \times \mathbf{100}$

N: Total denetleme sayısı

TFD: Denetleme sürecinde tespit edilen hata sayısı

FDD: Toplam hata sayısı. Denetleme sürecinde tespit edilen hatalar, test sürecinde tespit edilen hatalar ve müşteri tarafından 90 gün içerisinde tespit edilen hataları içerir [44].

Barnard ve Price yazılım kalitesini ve denetleme sürecini değerlendirmek için altı adet metrik tanımlamıştır.

1. Denetlenen LOC (lines of code) miktarı.
2. Denetleme işlemini gerçekleştiren denetleyici sayısı.
3. Hazırlanma süreci için ayrılan zaman.
4. Denetleme işlemi için ayrılan zaman.
5. Denetleme süreci esnasında tespit edilen hata sayısı
6. Kod karmaşıklığı bilgisi

Tervonen ve Iisakka Barnard ve Price'ın tanımladığı metrikler ve AT & Bell laboratuarlarında üretilen metriklerden yola çıkarak averaj kalite metriklerini üretmişleridir. Averaj kalite metrikleri aşağıda listelenmişlerdir.

1. Denetlenen materyale ait averaj büyüklük değeri
2. Hazırlanma süreci için ayrılan averaj zaman değeri
3. Denetleme işlemi için ayrılan averaj zaman değeri
4. KLOC başına düşen ortalama çaba değeri
5. Denetleme esnasında belirlenen hata başına düşen çaba değeri

6. KLOC başına düşen ortalama hata değeri
7. Yeniden denetleme oranı
8. Averaj kod karmaşıklığı bilgisi ve bulunan hata sayısı
9. Hata giderme etkinliği

Tervonen ve Iisakka belirledikleri bu averaj metrikleri kullanılarak elde edilen sonuçların değerlendirilmesi için tavsiye edilen değerler belirlemişlerdir. Bu değerler Tablo 3.2 de listelenmiştir [49],[50].

Ortalama denetlenen kod satır sayısı	< 40 / 500 LOC
Diğer karmaşa metrikleri	CC < 15 / 100
Ortalama hazırlanma oranı	< 150 - 200 LOC
Ortalama denetim oranı	< 150 - 200 LOC
Hazırlık Zamanı / Denetim Zaman Oranı	1.25 - 1,4
Ortalama Çaba / KLOC	50 saat / KLOC
Ortalama Çaba / Bulunan Hata	0.5 - 1 saat/hata
Denetlenen Hata Toplamı / KLOC	40 hata / KLOC
Tekrar Denetleme veya Çıkış	0.25 branş / sayfa
Hata giderme etkinliği	% 74, % 61, % 55
Kabul Etme genişliği	26

Tablo 3.2 Denetleme metrikleri için optimal değerler [9]

AT & Bell laboratuarlarında oluşturulan dokuz kilit metriğinden farklı olarak Batzinger ve Ramaswamy tarafından iki metrik tanımlanıştır. Bu metriklerden birincisi denetleme toplantısının verimliliğini ölçmek için kullanılan denetim etkinliği (Em) metriği, ikincisi ise denetleyicilerin performansını ölçmek için kullanılan denetleyici etkinliği (Ei) metriğidir [51].

1. Denetleme Etkinliği
   Em = SLOC / (Hazırlanma süresi + Denetleme Süresi) * Denetleyici Sayısı
2. Denetleyici Etkinliği
   Ei = SLOC / (Hazırlanma süresi + Denetleme Süresi)

Bu metriklerin hesaplamasında kullanılan değerler ve tanımlarına Tablo 3.3 de yer verilmiştir.

Ölçüm	Tanımlama
Hazırlanma_Süresi	Denetim toplantısına hazırlanmak için saat cinsinden zaman
Denetleme_Süresi	Denetim toplantısının saat cinsinden uzaklığı
Denetleyici Sayısı	Denetim toplantısında bulunan denetleyicilerin sayısı
SLOC	Denetim toplantısında denetlenen kaynak kodun satır sayısı

Tablo 3.3 Denetleme verimliliği ve denetleyici verimliliği hesaplamasında kullanılan değerler [51].

Christenson, Huang, ve Lamperez Denetleme işlemini verimliliğini etkileyen en önemli faktörlerin hazırlanma sürecinde sarf edilen çaba (PE), denetlenen kod oranı (IR) ve denetlenen kod miktarı olduğunu iddia etmişlerdir.

Hazırlanma Süreci Eforu

Kod denetleme toplantısından önce yapılır. Denetleyiciler kodu inceleyerek denetleme süreci için hazırlık yaparlar. Hazırlanma eforu ile hata yoğunluğu doğru orantılıdır. Verimli bir hazırlanma süreci geçirildiğinde daha çok hata tespit edildiği görülmüştür. Hazırlanma eforu aşağıdaki formül ile hesaplanmaktadır.

Hazırlanma Eforu = Toplam hazırlanma zamanı / KNCSL

Denetleme Oranı

Denetleyiciler denetleme toplantılarında bir araya gelerek hataları belirlemek için kodu incelerler. Denetleme oranı hata yoğunluğu ve hazırlanma eforu verilerine göre farklılıklar gösterebilirler. Denetleme oranı ve hata yoğunluğu ters orantılıdır. Denetleme oranı aşağıdaki formül ile hesaplanmaktadır.

Denetleme Oranı = Denetlenen KNCSL / Denetleme toplantıları süresi

Kod Büyüklüğü

Denetlenecek kodun büyüklüğü denetlemeyi doğrudan etkileyen bir faktördür. İncelenecek kod miktarı fazla olduğunda hazırlanma süreci için ayrılan süresi kısalır ve denetleme oranı artar. Hazırlanma süresinin kısa olması hata bulma oranını düşürür. Eğer incelenecek kodun büyüklüğü yeterli hazırlanma süresini kısıtlamıyor ise hata bulma oranı yüksek olacaktır [52].

Denetleme ekibinin ne kadar efor sarf edeceği hakkında bilgi sahibi olması proje kalitesini ve başarısı etkileyen en önemli faktörlerdendir. Projede görev alan kişilerin bireysel performansı uygulama alanı hakkında bilgi sahibi oldukça ve becerilerini arttırdıkça çoğalır [6]. Milicic ve Wohlin yazılım geliştirme projelerinde efor tahmininin başarıyı getiren önemli faktörlerden biri olduğunu iddia etmektedir [53]. Kanabar ise katılımcıların tecrübelerinin ve yeteneklerinin seviyesinin daha önemli olduğunu savunmuştur [54]. Nair ve Suma bu bilgiler doğrultusunda Denetleme sürecini ölçmek için iki yeni metrik tanımlanmıştır. Bu metrikler Denetleme Derinliği (DI) metriği ve Denetleme Performans metriğidir (IPM).

Denetleme Derinliği(DI)

Denetleme süreci boyunca saptanan hata sayısı (Ni) ve denetleme süreci ve test süreci boyunca tespit edilen hata sayısı (Td) değerleri kullanılarak denetleme derinliği hesaplanır.

DI = Ni / Td

Denetleme Performans Metriği (IPM)

Denetleyici sayısı, denetleyicilerin tecrübe seviyesi, denetleme süresi ve hazırlanma süresi değerlerini kullanarak denetleme ekibinin performansını ölçen metriktir [6].

IPM = Denetleme sürecinde bulunan hata sayısı (Ni) / Denetleme eforu (IE)

IE = denetleyici sayısı (n) * total denetleme süresi (T)

T = Denetleme zamanı + hazırlanma zamanı

$$\mathrm{IPM} = \frac{\sum_{i=1}^{n} N_i}{\sum_{i=1}^{n} (I_{ti} + P_{ti})}$$

3.3 Denetleme Sürecini Etkileyen Değişkenlerin ve Bu Değişkenler için Optimal Değerlerlerin Belirlenmesine Yönelik Çalışmalar

3.3.1 AT & Bell Laboratuarlarında Yapılan Çalışmalar

Denetleme eforu denetleme sürecini ve bu sürecin maliyetini etkileyen en önemli faktördür. Denetleme işlemi ve hataların düzeltilmesi işlemi için gereken kişi-saat değerleri hesaplanarak denetleme işlemi için gerekli olan efor belirlenir. KLOC başına düşen ortalama çaba ve yeniden denetleme oranı metrikleri daha önce deneteme işlemi gerçekleştirilmiş projelere ait verileri kullanarak yöneticinin gerekli eforu belirlemesine yardımcı olur. Eğer önceki projelere ait veriler yoksa KLOC başına düşen ortalama çaba KLOC başı 50 saat olarak alınabilir.

Denetleme işlemi için gerekli olan sürenin belirlenmesi zor bir süreçtir fakat metrikler yardımıyla makul değerler tahmin etmek mümkündür. Daha önce denetleme işlemi yapılmış benzer projelere ait veriler dikkate alınarak değerlendirme yapılabilir. Bu işlem yapılırken kullanılan eski projeler ve yeni proje arasındaki farklılıklar mutlaka belirlenmeli ve bu farklılıklarda dikkate alınarak tahmini süre belirlenmelidir.

Denetleme sürecinin ne kadar etkili olduğunun belirlenmesinde KLOC başına düşen ortalama hata metriği kullanılabilinir. Eski veriler baz alınarak hesaplanan KLOC başına düşen ortalama hata miktarı ile güncel veriler kullanılarak hesaplanan KLOC başına düşen ortalama hata miktarı değerleri kıyaslanır. Eğer güncel değer eski değerin çok altında ise denetleme işleminin beklenenin arlında hata belirlediği kabul edilir ve buda ürünün kalitesini düşürür. AT&T Bell laboratuarlarında yapılan çalışmalarda eski denetleme verileri kullanılarak elde edilen KLOC başına düşen ortalama hata miktarı 118,5 çıkmıştır. Ortalama denetleme ve hazırlanma oranı değerleri bir saat için 150 kod satırı civarında çıkmıştır. Güncel denetleme işlemine

ait veriler aşağıdaki tablo 3.4 de verilmiştir. Güncel KLOC başına düşen ortalama hata miktarı 106'dır. Güncel değer eski değerden %10 daha azdır. Bu veriler doğrultusunda denetleme ürünü kaliteli olarak kabul edilmiştir.

KOD DENETLEME İSTATİSTİKLERİ	
Örnek denetleme süreci sayısı	27
Denetlenen toplam KLOC miktarı	9,30
Denetlenen ortalama LOC miktarı	343
Ortalama hazırlanma oranı	194
Ortalama Denetleme oranı	172
KLOC başına düşen toplam hata	106
Yeniden denetleme oranı	11

Tablo 3.4 Güncel kod denetleme işlemine ait veriler [44]

Hata saptama işlemini değişkenlerin nasıl etkilediği görülmek isteniyorsa hazırlanma oranı, denetleme oranı ve denetlenen kod büyüklüğü gibi değerlere bakılmalıdır. Bunun için ortalama hazırlanma oranı, ortalama denetleme oranı ve denetlenen ortalama kod satırı metrikleri kullanılmalıdır. Bu değerlerin optimal olup olmadığına bakılıp denetleme süreci hakkında yorumlar yapılabilir. Tablo 3.5 de örnek bir projeden alınan değerlere yer verilmiştir. Hata giderme etkinliği %74'tür. Ortalama denetleme oranı 154,8 ve ortalama hazırlanma oranı 121,9 olarak belirlenmiştir. Önerilen 150 LOC değerine yakındır. Denetlenecek ürün büyüklüğü 409 LOC dur. Önerilen 500 LOC değerinin altındadır yani risk teşkil etmemektedir.

KOD DENETLEME İSTATİSTİKLERİ	
Örnek denetleme süreci sayısı	55
Denetlenen toplam KLOC miktarı	22,50
Denetlenen ortalama LOC miktarı	409
Ortalama hazırlanma oranı	121,9
Ortalama Denetleme oranı	154,8
KLOC başına düşen toplam hata	89,7
Yeniden denetleme oranı	0,5

Tablo 3.5 Güncel kod denetleme işlemine ait veriler [44]

Verimliliği arttırmak için süreçte denetleme oranı, hazırlanma oranı gibi değişkenlerde düzeltmeler yapmak, denetleyici sayısını arttırmak ya da azaltmak gibi işlemler yapılabilir [44].

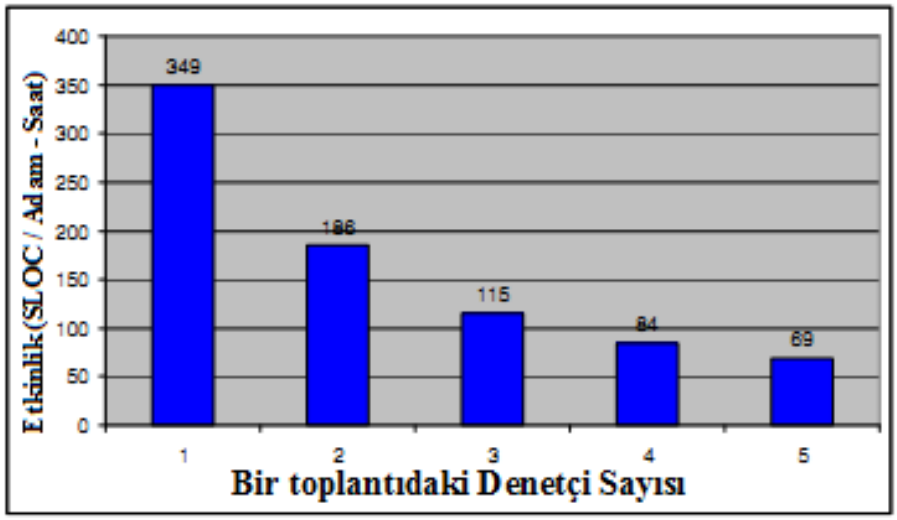
3.3.2 650 NASA SEL Denetleme Kayıtları Kullanılarak Yapılan Çalışmalar

Kod denetleme kaynak kod üzerinde hata bulmak amaçlı gerçekleştiren özel bir yazılım denetleme çeşididir. Kod denetleme yazılım projelerinde kullanılan en yaygın muayene yöntemleri arasında yer almaktadır fakat son yıllarda denetleme sürecinin verimliliği tartışılmaya başlanmıştır. Pek çok makalede kod denetleme verimliliği incelenmiştir. Bu makalede kod denetleme verimliliği üzerinde durulmuştur. Yeni geliştiriliş yazılımlar ve bakımı yapılan yazılımlar üzerinde değerlendirmeler yapılmıştır. Aşağıdaki iki sorunun cevabı aranmıştır.

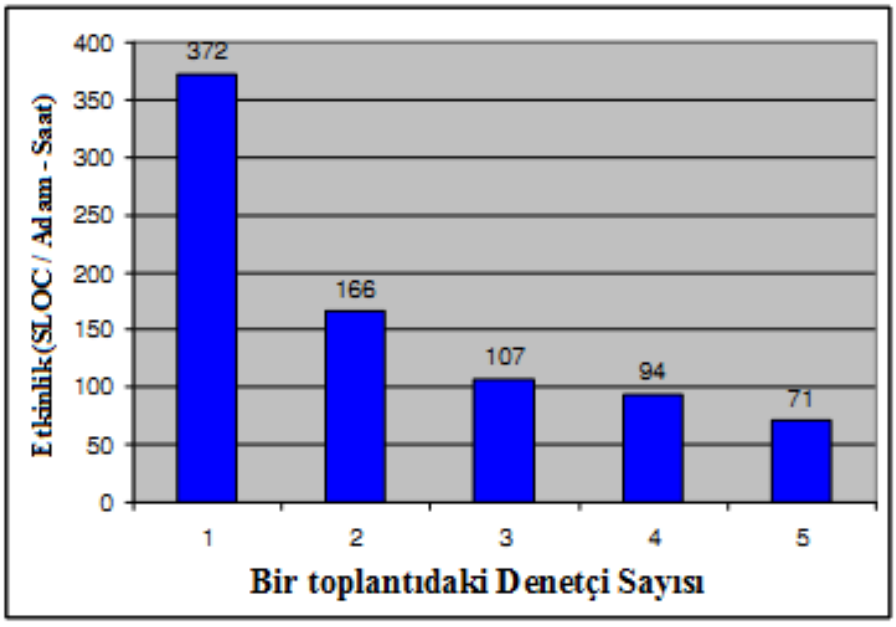
1. Denetleyici sayısı kod denetleme işleminin verimliliğini etkiler mi?
2. Hazırlanma sürecinde harcanılan zaman ve denetleme esnasında harcanılan zaman kod denetleme işleminin verimliliğini etkiler mi?

Bu çalışmada yeni geliştirilmiş kaynak kodlar ve bakımı yapılan kaynak kodlar için ayrı ayrı değerlendirme yapılmış ve denetleyici etkinliği (Ei) ve denetleme etkinliği (Em) metrikleri formülleştirilmiştir.

İlk olarak denetleyici etkinliği metriği ele alınmıştır. Şekil 3.2 de yeni geliştirilen bir yazılımda yapılan denetleme işlemi sonrasında ortalama denetleyici etkinliği ve denetleyici sayısı arasındaki ilişki gösterilmiştir. Şekil 3.3 de ise bakım yapılan bir yazılımda yapılan denetleme işlemi sonrasında ortalama denetleyici etkinliği ve denetleyici sayısı arasındaki ilişki gösterilmiştir.



Şekil 3.2 Yeni geliştirilmiş programlarda denetçi sayısı-etkinlik ilişkisi [51]

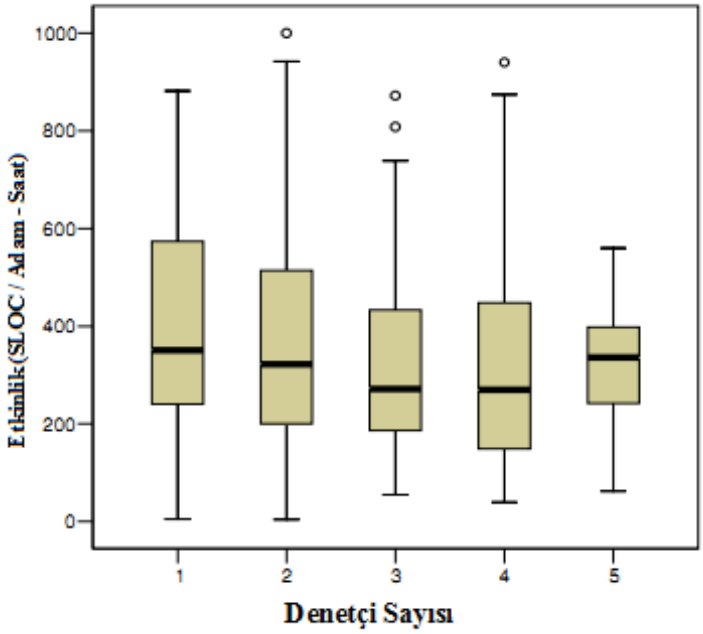


Şekil 3.3 Bakımı yapılan programlarda denetçi sayısı-etkinlik ilişkisi [51]

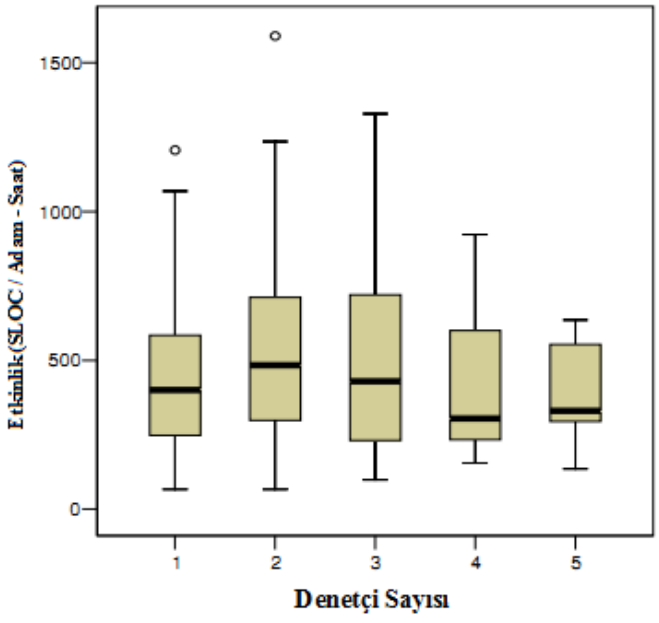
Yapılan çalışmalar sonucunda denetleyicilerin verimliliği denetleme yapan denetleyici sayısı arttıkça azalmaktadır iddiasında bulunulmuştur.

Denetleyici etkinliğinin incelemesinin ardından denetleme etkinliği metriği ele alınmıştır. Şekil 3.4 de yeni geliştirilen bir yazılımda yapılan denetleme işlemi sonrasında denetleme etkinliği ve denetleyici sayısı arasındaki ilişki gösterilmiştir. Şekil 3.5 de ise bakım yapılan bir yazılımda gerçekleştirilen denetleme etkinliği ve

denetleyici sayısı arasındaki ilişki gösterilmiştir. (Kutuların ortasında yer alan kalın siyah çizgiler medyanı göstermektedir. Kutulara iliştirilmiş çizgiler veriler için standart aralığı göstermektedir. Çubukların dışında görülen çemberler standart aralığın dışında kalan veri noktalarını göstermektedir.) [51].



Şekil 3.4 Yeni geliştirilmiş programlarda denetçi sayısı-denetleme etkinliği ilişkisi [51]



Şekil 3.5 Bakımı yapılan programlarda denetçi sayısı-denetleme etkinliği ilişkisi [51]

Yapılan araştırmalar sonucunda denetleyici sayısının değişmesinin denetleme etkinliğini etkilemediği iddia edilmiştir

Şekil 3.2, Şekil 3.3, Şekil 3.4 ve Şekil 3.5'te yer alan sonuçlardan yola çıkılarak denetleyici sayısının denetleyicilerin performansını etkilerken denetleme performansını etkilemediği belirtilmiştir. Buna sebep olarak denetleme toplantılarında incelenen kod satırı sayısının denetleyici sayısına bağlı bir değer olmaması gösterilmiştir. Denetleme etkinliği ve denetleyici etkinliği metrikleri hesaplanırken esas göz ardı edilemeyecek unsurların hazırlanma zamanı ve denetleme zamanı olduğu belirtilmiştir. SLOC değeri baz alınarak yapılan bir araştırma olduğu için denetlenen kod satır miktarının denetleme süresi ve hazırlanma süresine bağlı olarak değişiklik göstereceğinin altı çizilmiştir [51].

3.4 DRE, DI ve IPM Metrikleri Kullanılarak Denetlemeyi Etkileyen Değişkenlerin Tahmin Edilmesi

3.4.1 Denetlemeyi Etkileyen Değişkenler

Deneysel birçok çalışmaya göre hizmet ve ürün tabanlı birçok firmanın gerçekleştirdiği projenin sonuçları Yazılım Geliştirme Yaşam Döngüsünün bütün evrelerinde denetimin etkisini değiştiren birçok değişken olduğunu meydana çıkarmıştır. Bu değişkenler denetim zamanı, denetime katılan denetçilerin sayısı ve yazılım geliştirme süreçlerine ve hazırlık zamanına dâhil olan bütün denetçilerin deneyim seviyesidir [56].

3.4.1.1 Denetlemeyi Etkileyen Değişkenlerin Belirlenmesi

Yukarıda 2.2.2 numaralı bölümde kod denetleme sürecini değerlendirmede kullanılan metrikler anlatılmıştır. Bu metriklerin hesaplanmasında denetleme sürecinde etki sahibi olan parametreler kullanılmıştır. Hesaplamalar incelendiğinde en çok kullanılan değişkenlerin proje büyüklüğü, hazırlanma süresi, denetleme süresi ve denetleyici sayısı olduğu belirlenmiştir. Aşağıda değişkenlere ve bu değişkenler kullanılarak hesaplanan metriklere yer verilmiştir. IPM metriği hesaplanırken bu dört değişkenin tamamı hesaplamaya dâhil edilmektedir. Buna ek olarak denetleyicilerin tecrübe seviyesi de IPM hesaplamada yer almıştır. IPM metriğini anlatan akademik

çalışmalar incelendiğinde denetleyicilerin tecrübe seviyelerinin süreç üzerinde büyük etkisi olduğundan bahsedilmiştir. Bu sebeple denetleme sürecini etkileyen faktörlerin arasına denetleyicilerin tecrübe seviyesi de dâhil edilmiştir [44].

3.4.1.2 Denetleyici Tecrübe Seviyesi

Denetime katılan denetçilerin deneyim seviyeleri de denetim için önemli bir faktördür. Gerçekleştirilmiş projeler yenilikçi projelerin gereksinimlerinin daha kısa sürede karşılanmasını gerektirir. Bu yüzden yenilikçi projeler deneyimli denetçileri gerektirir. Tümleşik projeler yazılım geliştirme yaşam döngüsü evrelerinden tasarım aşamasında daha fazla zaman gerektirir. Bu projelerde daha önce en az üç projede tasarım verileri ile ilgili denetimlere katılmış denetçiler tercih edilir. Bu denetimlerde önemli olan güvenlik, sürdürülebilirlik, tekrar kullanılabilirlik, tasarımın bütünlüğü gibi konularla uyumun sağlanmasıdır. Uygulama seviyesinde denetimler gerçekleştiren bir denetçi güvenlik gereksinimleri ile ilgili beklentiler ve organizasyonun kuralları ile ilgili bilgiye sahip olmalıdır. Yazılım kodunda fazlalık olup olmadığını, satır sayısını, kodun verimliliğini, tekrar kullanılabilirliğini, güvenliğini, tasarım politikalarını, sürdürülebilirliğini ve diğer kalite gereksinimlerini tespit edebilmek için kodlama ile ilgili beceriye sahip olmalıdır. Hata bulmada insan faktörü etkilidir. Organizasyonel ve kişisel faktörler de aynı şekilde. Denetimin etkili olabilmesi için denetçilerin yetenekleri önemli bir etkendir [56].

3.4.1.3 Projenin Büyüklüğü: Denetlenen projenin büyüklüğünün göstermek amacıyla kullanılır. Denetlenen toplam kaynak kod satırı, denetlenen koddaki işlev puanı sayısı, modül sayısı ya da denetlenen kod parçacığı geliştirilirken harcanılan süre (adam/saat) proje büyüklüğünü göstermek amacıyla kullanılabilir. Aşağıda proje büyüklüğü değeri kullanılarak hesaplanan metrik örneklerine yer verilmiştir.

- Denetlenen Toplam Kaynak Kod Satırı (KLOC)
- Ortalama denetleme oranı
- Ortalama Hazırlanma oranı
- KLOC Başına Düşen Ortalama Hata
- Denetleme Oranı
- Hazırlanma Eforu

- IPM

3.4.1.4 Denetleyici Sayısı: Denetleme sürecinde yer alan denetleyicilerin sayısıdır. Aşağıda denetleyici sayısı değeri kullanılarak hesaplanan metrik örneklerine yer verilmiştir.

- Ortalama Hazırlanma Oranı
- KLOC başına düşen ortalama çaba
- Tespit Edilen Hata Başına Düşen Ortalama Çaba
- Denetleme Etkinliği
- Denetleyici Etkinliği
- IPM

Denetleme takımının büyüklüğünün belirlenmesine yönelik çalışmalardan bölüm 2.2.3.1'de bahsedilmiştir. Denetçi sayısı denetim sırasındaki bütün gelişim fazlarının etkilemektedir. Denetime katılacak denetçi sayısı organizasyonun denetim için ayırdığı bütçe ile bağlantılıdır [56].

3.4.1.5 Hazırlanma Süresi: Denetleyicilerin hazırlanma safhasında harcadıkları süredir. Hazırlanma süreci bireysel ya da grup olarak gerçekleştirilebilir. Aşağıda hazırlanma süresi değeri kullanılarak hesaplanan metrik örneklerine yer verilmiştir.

- Ortalama Hazırlanma Oranı
- KLOC başına düşen ortalama çaba
- Tespit Edilen Hata Başına Düşen Ortalama Çaba
- Denetleme Etkinliği
- Denetleyici Etkinliği
- Hazırlanma Eforu
- IPM

Hazırlık zamanı denetçiler için denetimin etkisini artıran önemli bir faktördür. Denetim için hazırlık zamanı projenin kapsamına bağlıdır. Denetimin başkanı diğer denetçilerle talimatları paylaşır. Bu sayede denetim ekibi hataların tespit edilebilmesi için yardımcı olur. Ön çalışma sırasında yapılan planlama insan gücüne ihtiyacın

azalmasını ve denetçilerin yeteneklerini geliştirmelerini sağlar. Bu hazırlık zamanları bireysel eğitimler, eğitim ve okuma tekniklerinin edinilmesi ile değerlendirilir. Bu sayede hangi bulguların nasıl tespit edileceğini öğrenmiş olurlar. Bu hazırlık aşaması denetim toplantılarının etkilerini artırır. Hazırlık aşamasında edinilen bilgiler sayesinde hata tespit oranını artırır [56].

1.4.1.6 Denetleme Süresi: Kaynak kodun denetlenmesi için ayrılan toplam süredir. Denetleme süreci genellikle grup toplantısı şeklinde gerçekleştirilir. Aşağıda denetleme süresi değeri kullanılarak hesaplanan metrik örneklerine yer verilmiştir.

- Ortalama denetleme oranı
- KLOC başına düşen ortalama çaba
- Tespit Edilen Hata Başına Düşen Ortalama Çaba
- Denetleme Etkinliği
- Denetleme Oranı
- IPM

Denetimi etkileyen en önemli değişkenlerden birisi denetim zamanıdır. Denetimin her evresi için bir zaman çerçevesi belirlemek önemlidir. Birçok örnek çalışma kanıtlar ki denetimin doğruluğu uygun bir şekilde takvimlendirilmesi ile doğru orantılıdır. Suma V. Ve Gopalkrishnan Nair'ın çalışmaları sonucunda %99 doğru sonuçlanmış bir ürün ortaya koymak ancak planlanan proje zamanından en fazla % 10 -15 sapma ile mümkün olur. Denetim zamanındaki azalma hataların gözden kaçırılmasına sebep olur. Öyle ki denetim yazılımının otomatikleştirilmesi manüel denetim zamanını azaltır ve hata bulma oranını artırır [56].

Denetimin daha etkin ve verimli olabilmesi için bütün bu değişkenlerin mümkün olan en uygun seviyede kullanılması ve bu uygunluk seviyesine de önceki denetim bulguları kontrol edilerek karar verilmesi gereklidir [44].

3.4.2 Tahmin İşleminde Kullanılacak Metrikler ve Tercih Edilme Sebepleri

3.4.2.1 Hata Giderme Etkinliği (DRE)

Hata giderme etkinliği ürün müşteriye dağıtılmadan önce giderilen yazılım denetleme veya test hatalarının oranıdır. 1960'lardan itibaren kullanılan güçlü bir kalite metriğidir. Bu sebeple yazılım dünyasında yer alan herkes tarafından anlaşılmalıdır. Hata giderme yazılım projelerinin üst seviye masrafları arasında yer almaktadır ve çalışma süresi üzerinde ciddi etkilere sahiptir. Etkili bir hata giderme ürün kalitesini arttırır ve geliştirme süresini kısaltır. Kalite ve üretkenlikte artma, süre ve maliyette azalma gibi avantajlar hata giderme süreçlerinin verimliliğinin maksimum seviyelerde tutulması sayesinde olur. Bu koşulu sağlayabilmek için sürecin verimliliği ölçülmelidir [57].

3.4.2.1.1 Hata Giderme Etkinliğinin Hesaplanması

Hata giderme etkinliğinin normal periyodu denetlemenin gerekliliği ile başlar ve yazılımın kullanıcılara veya müşterilere dağıtımından 90 gün sonra sona erer. Elbette yazılımda halen 90 günlük süreçte bulunamayan hatalar vardır fakat 90 günlük süreç hata giderme etkinliği için standart bir kıyaslama noktası sağlamaktadır. Bu periyodun 90 günden 6 ay yahut 12 aya çıkarılması düşünülebilir fakat güncellenenler ve yeni serbest bırakılanlar 90 gün sonrasında ortaya çıkar bu nedenle orijinal hata sayımında etki azalmış olur.

Saklı olan hatalar 90 günlük periyottan sonra yıllarca var olabilir ancak kalan saklı hataların ortalama %50 si her takvim yılı bulunmaktadır. Sonuçlar uygulamaların kullanıcı sayılarıyla değişmektedir. Kullanıcı ne kadar fazla olursa saklı hatalardan kalanların keşfedilmesi o kadar hızlı olur. Sonuçlar yazılımın kendi yapısıyla da değişmektedir. Askeri, gömülü ve sistem yazılımları virüs veya hata bulmakta bilişim sistemlerine göre daha çabuktur.

Hata giderme etkinliği hesaplanması kolay bir metriktir. Yazılım geliştirme döngüsü boyunca tespit edilen tüm hatalar kalite ekibi ya da geliştirme takımı tarafından kaydedilir. Bir yazılım projesinde geliştirildiği süre boyunca 90 hata tespit edildiğini varsayalım. Yazılım ürünü müşteriye dağıtıldıktan sonra ilk üç aylık kullanım süresi boyunca müşteri tarafından 10 adet hata tespit edilmiş olsun. Bu üç aylık sürenin

bitmesinin ardından ürün müşteriye dağıtılmadan önce ve dağıtıldıktan sonra tespit edilen hatalar bir araya getirilmelidir [59].

Geliştirme süreci boyunca tespit edilen hata sayısı (DD): 90

Müşteri tarafından bulunan hata sayısı (LD): 10

Toplam hata sayısı (TD): 100 (LD + DD)

Hata giderme oranı = $\frac{DD}{LD+DD} \times 100$

$= \frac{90}{100} \times 100 = \%90$

Çoğu test biçiminde ortalama sadece yaklaşık olarak % 30 ve %35 arasındadır. Hata giderme etkinliği seviyeleri nadiren %50'yi aşar. Diğer taraftan formal dizayn ve kod denetlemenin kullanıldığı yapılarda bu değer sıklıkla %85'i açar.

3.4.2.1.2 Kod Denetlemenin Hata Giderme Etkinliği Üzerindeki Etkisi

Yazılım tarihinde hata giderme ve önleme aktiviteleri içinde formal dizayn ve kod denetlemeleri en etkili aktivitelerdir. Bütün test aşamaları ortalama olarak sadece %30 verimlilik sağlayacağından, sadece testle %95 hata giderme etkinliğine ulaşılması mümkün değildir. Testten önceki formal denetleme hataların çoğunu belirleyecektir.

Ayrıca formal denetleme hata önleme açısından da çok etkilidir. Formal denetlemeye katılanlar denetleme işlemi boyunca bulunan benzer çeşitteki sorunları yapmaktan kaçınırlar

Testi tek başına kullanmaktansa test ve denetlemenin bir arada olması gelişme programının daha kısa sürede sonuçlanmasını sağlayacaktır. Bu durum test denetlemeden sonra başladığında hataların hemen hemen % 85 'inin zaten gitmiş olmasından kaynaklanır. Bu yüzden test programı % 45 'ten fazla oranda kısalmış olur.

IBM formal denetlemeyi geniş veri tabanı projelerinde kullandığında ilginçtir ki önceki sürümlere göre teslim edilen hatalar % 50 ‘ den daha fazla azalmıştır. Bütün programlar %15 civarında kısalmıştır. Kendini test etme 60 günlük 2 vardiyalık periyottan 40 günlük tek vardiyaya düşmüştür. Daha da önemlisi müşteri memnuniyeti çok zayıfken bu durum geliştirilerek iyi konumuna getirilmiştir. Kümülatif hata giderme etkinliği formal dizayn ve kod denetleme kullanılarak yaklaşık % 80 den % 95 in biraz üzerine çıkarılmıştır.

3.4.2.1.3 Hata Giderme Etkinliğinin Tercih Edilme Sebepleri

1. Toplam hata giderme etkinliği yaklaşık % 95 olan projeler birçok açıdan optimal olma özelliği gösterir. Bunlar en kısa geliştirme zamanına, en az geliştirme maliyetine, en yüksek müşteri memnuniyetine ve en yüksek derecede takım moraline sahiptir. Bu yüzden hata potansiyeli ve hata giderme etkinliği ölçümü sektörün bütününde önemlidir.
2. Ekonomik açıdan incelediğinde Amerika Birleşik Devletleri’nde ortalama % 85 - % 95 Aralığında hata giderme etkinliğine sahip olanların parasal yönden tasarruf edip aynı zamanda kısa geliştirme zamanına sahip oldukları belirlenmiştir [58].
3. Hata giderme etkinliği hesaplanırken yazılım ürününün dağıtımından 90 gün sonrasına kadar geçen sürede müşteri tarafından tespit edilen hata miktarı da dâhil edilir. Bu sebeple müşteriye hatasız ürün göndermek için harcanan efor artar.

3.4.2.2 Denetleme Derinliği (DI) Metriği

Denetleme Derinliği (DI) Metriği daha basit fakat daha etkili bir yöntemdir. N_i denetim esnasında, T_d ise hem denetim hem de testler sırasında tespit edilen hata sayısını temsil eder.

$DI = N_i / T_d$

DI ya faz faz ya da Denetleme Derinliği Metriği kullanılarak ürünün kullanımından önce ölçülebilir.

DI Değerlendirmesi 2 fazda gerçekleştirilir. İlk fazda belirlenen projelerde hatalar sayılarak DI ölçülür. Bu faz sayesinde yazılım firması denetleme süreci hangi

projede ve hangi derinlikte faz faz mı yoksa proje seviyesinde mi gerçekleştirilmiş tespit edebilir. Hizmet tabanlı ve Ürün tabanlı birçok projede gerçekleştirilen derin ve titiz araştırmalar sonucunda DI ölçümlerinin projeden projeye farklılık gösterdiği tespit edilmiştir. Ölçüm seviyeleri de projeden projeye farklılık gösterebilir. Denetleme derinliğinin 0 ile 1 arasında olması beklenir. 0 en düşük 1 ise &100 hata tespiti demektir. % 100 tespit etmek ise neredeyse imkânsızdır. Normal bir denetleme sürecinde 0.3 ile 0.5 arasında değişen değerler kabule dilebilir seviyelerdir.

Tablo 3.6 da DI aralıkları gösterilmektedir. Bu tablo yazılım ekibinin ve diğer iş ortaklarının o firmanın olgunluk seviyesini anlaması için yardımcı olur. Bu sayede firma o seviyede devam etmesi ya da daha iyi bir seviyeye yükselmesi gereksinimine dair planlama yapması sağlanır [6],[56].

DI	**Denetleme Performansı**
0 - 0.1	Kötü
0.1 – 0.2	Çok Düşük
0.2 – 0.3	Düşük
0.3 – 0.4	Normal
0.4 – 0,5	Normalin Üzerinde
0.5 – 0,6	Yüksek
0.6 – 0,7	Çok Yüksek
0.7 – 0,8	En iyi
0.8 – 0,9	Mükemmel
0.9 – 1	İdeal

Tablo 3.6 DI değer aralıkları [55]

3.4.2.2.1 Denetleme Derinliği (DI) Metriğinin Tercih Edilme Sebepleri

Nair ve Suma'nın denetleme derinliğinin faydalarını listelerken aşağıdaki maddelere de yer vermiştir.

1. DI denetleme işleminin derinliğini sayısallaştırmak için kullanılan bir metriktir. Denetleme derinliğinin hata belirleme metriği olarak tanımlanmasının sebebi, denetleme süreci boyunca hata yakalama durumunu analiz etmeye imkân tanımasıdır.

2. DI metriği hataların denetim süreci boyunca saptanma derinliğini test ekibine bildirir.
3. DI metriğinin uygulanması yazılım şirketleri için önemli bir tasarruf demektir. Bu kazancı statik olan bütün hataları görebilmesi ve kurtarması sayesinde sağlamaktadır. Test aşaması boyunca planlar dinamik hataların belirlenmesi ve ortadan kaldırılması yönünde olacaktır.
4. DI metrik sistemi fonksiyonel ve öngörülen denetlemenin kalite seviyesinde çeşitlilikler ortaya koymaktadır. Bu özellik geliştirme takımına yazılım sektöründeki rekabet ortamında büyüme planları boyunca kendilerini donatma imkânı da vermektedir.
5. Denetleme Derinli metriği denetim takımına, şirket yönetimine, dış kaynak temsilcisine ve diğer hissedarlara denetimin yapıldığı derinlikle ilgili görüş kazandırarak süreçte şeffaflığı sağlar [56].

Denetleme işleminin başlangıcında genellikle potansiyel hata tahmini yapılır. Tahmini hata sayısı ve denetleme süreci boyunca tespit edilen hata miktarı değerleri kullanılarak test süreci öncesinde DI değeri hesaplanabilir. Bu sayede test ekibi test süreci için ne kadar efor gerektiğini hesaplayabilir. Buna ek olarak denetim esnasında bu hesaplama yapılarak süreç değerlendirilebilir ve eğer gerekiyorsa müdahale edilebilir. Denetleme derinliği metriği anlaşılması kolay bir metriktir. Denetleme takımı dışında diğer yöneticilerde kolaylıkla sürecin verimliliği hakkında fikir sahibi olabilir.

3.4.2.3 Denetleme Performans Metriği (IPM)

Kalite yönetiminde en önemli parametrelerden biride denetleme ekibinin performansıdır. IPM Denetleme ekibinin ortaya koyduğu eforun değerlendirilebilmesi için kullanılabilecek bir efor analiz metriğidir. Denetleme performansı metriği hesaplanırken denetleme ekibinin performansı, denetleme süresi, denetleyici sayısı, hazırlanma süresi, denetleme ekibinin tecrübe seviyesi ve ürünün karmaşıklığı gibi performans üzerinde büyük etkisi olan parametreler kullanılmaktadır [6],[56].

IPM = Denetleme sürecinde bulunan hata sayısı (Ni) / Denetleme eforu (IE)

IE = denetleyici sayısı (n) * total denetleme süresi (T)

T = Denetleme zamanı + hazırlanma zamanı

$$\mathrm{IPM} = \frac{\sum_{i=1}^{n} N_i}{\sum_{i=1}^{n} (I_{ti} + P_{ti})}$$

3.4.2.3.1 Denetleme Performans Metriğinin (IPM) Tercih Edilme Sebepleri

Nair ve Suma'nın denetleme derinliğinin faydalarını listelerken aşağıdaki maddelere de yer vermiştir.

1. IPM Denetleme ekibinin ortaya koyduğu kalitenin seviyesinin ölçülebilmesi için bir kalite göstergesidir.
2. Müşterilere geliştirme masraflarının kontrol ve doğruluğunu sağlayabilecekleri saydam ve gözlemlenebilir bir ortam sağlar.
3. Yazılım endüstrisinde IPM kullanımı yönetimin eleman masrafları ile ilgili karar almasını ve doğrulamalar yapmasını kolaylaştırır.
4. Denetim performans ekibinin takım performansı ile ilgili farkındalıklarının artmasını ve performansın artması için yeni teknikler belirlemelerini sağlar.
5. Takım performansı ile ilgili derin bilgi edinim olanağı sağladığı için şirket ekonomisine katkıda bulunur [56].

IPM metriği denetleyici sayısı, denetleme süresi, hazırlanma süresi, denetleyicilerin tecrübe seviyesi ve denetlenen yazılım ürünü büyüklüğü değerleri kullanılarak hesaplanan bir metriktir. Bu değerler denetleme süreci üzerinde önemli etkilere sahip olan değerlerdir. Denetleme sürecinin verimliliğini ölçmek için kullanılan pek çok metrik hesaplamasında bu değerler kullanılmaktadır. Tez çalışmasında denetleme sürecini doğrudan etkileyen bu parametrelerin denetleme işleminin başlangıcında tahmin edilmesi amaçlanmıştır. IPM metriği tüm değişkenleri içerdiği ve denetleme ekibinin performansını derinlemesine ölçtüğü için tercih edilmiştir.

Denetlemenin kabul edilebilir bir seviyede tamamlanması için DI ve IPM tekniklerinin birbirleri ile uyumlu çalışması gerekmektedir [56]. Bu ilkeden yola çıkarak DI ve IPM metrikleri kullanılarak yapılan hesaplamalar tahmin işleminde kullanılacaktır.

3.4.3 Denetleme Sürecini Etkileyen Değişkenlerin Tahmin Edilmesi İşleminde Kullanılan Algoritmalar

3.4.3.1 Lineer Regresyon

Regresyon analizi, iki ya da daha çok değişken arasındaki ilişkiyi ölçmek için kullanılan analiz metodudur. Gerçek yaşamın çeşitli alanlarında herhangi bir uygulama ile toplanan veriler tablo şekline getirilerek incelenir ve toplanan veriyi modelleyen bir fonksiyon bulunmaya çalışılır. Çoğu zaman bu veri tablosuna tam olarak uyan bir fonksiyon bulmak mümkün olmaz; veri tablosuna en iyi uyan fonksiyon belirlenmeye çalışılır. Bir veri tablosuna en iyi uyan fonksiyonu bulma sürecine regresyon analizi denir. Bağımlı değişken sayısı tektir. Ancak bağımsız değişken sayısı birden fazla olabilir. Eğer tek bağımsız değişken var ise "Basit Doğrusal Regresyon" iki ve daha fazla bağımsız değişken var ise "Çoklu Doğrusal Regresyon" adı verilmektedir.

Regresyon Analizinde, değişkenler arasındaki ilişkiyi fonksiyonel olarak açıklamak ve bu ilişkiyi bir modelle tanımlayabilmek amaçlanmaktadır. Bir kitlede gözlenen X ve Y değişkenleri arasındaki doğrusal ilişki aşağıdaki "Doğrusal Regresyon Modeli**"** ile verilebilir;

$Y=\beta_0+ \beta_1X+r$

Burada;

X: Bağımsız (Açıklayıcı) Değişken

Y: Bağımlı (Açıklanan; Etkilenen; Cevap) Değişken

β_0: X=0 olduğunda bağımlı değişkenin alacağı değer
(kesim noktası)

β_1: Regresyon Katsayısı

r: Hata terimi (Ortalaması=0 ve Varyansı=σ^2'dir

Regresyon Katsayısı (β_1) :

Bağımsız değişkendeki bir birimlik değişimin, bağımlı değişkendeki yaratacağı ortalama değişimi göstermektedir.

r (Hata terimi):

Her bir gözlem çiftindeki bağımlı değişkene ilişkin gerçek değer ile modelden tahmin edilen değer arasındaki farktır.

$r_i = (\beta_0 + \beta_1 X) - Y_i$

Doğru ve güvenilir bir regresyon modelinde amaç, gerçek gözlem değeri ile tahmin değeri arasında fark olmaması ya da farkın minimum olmasıdır. Bunun için çeşitli tahmin yöntemleri geliştirilmiştir. Bu yöntemlerden biri "En Küçük Kareler" kriteridir.

Regresyon yapılırken en çok tercih edilen yöntem en küçük kareler yöntemidir.

En küçük kareler yöntemi çeşitli bilim dallarında çeşitli değişkenler arasındaki ilişkiler belirlenirken kullanılan en önemli araçlar arasındadır. Bu yöntemde aşağıdaki eşitlikteki farkın en küçük olması amaçlanır.

$$\sum_{i=1}^{n} e_i^2 \; {}_i = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Aşağıdaki yöntem ise en küçük kareler yöntemi ile bulunan tahminleri belirtmektedir

$$b_1 = \frac{\sum_{i=1}^{n} x_i y_i - (n\,\bar{x}\,\bar{y})}{\sum_{i=1}^{n} x_i^2 - (n\,\bar{x}^2)}$$

$$b_0 = \bar{y} - b_1 \bar{x}$$

Değişkenler birlikte artıyor ya da birlikte azalıyor ise "b_1 pozitif değerlidir "

Değişkenlerden biri artarken diğeri azalıyor ise "b_1 negatif değerli" dir [60],[61],[62].

3.4.3.2 Eğri Uydurma Yöntemi

Eğri uydurma bir eğri ya da matematiksel fonksiyon oluşturma işlemidir. Bu işlem bir fonksiyonun nokta nokta verilen değerlerinde, fonksiyona en yakın başka bir fonksiyonun belirlenmesi veya pratikte kolaylık sağlayabilecek yeni fonksiyonların araştırılması şeklinde yapılır. Eğri uydurma yapabilmek için ya tam bir uyuma sahip veri gerektiğinde interpolasyon kullanımı ya da veriye yaklaşık olarak uyum sağlamak için "düzgünleştirme" uygulanması gerektirebilir. Eğri uydurma, fonksiyonda kullanışlı bir veri yoksa fonksiyonun değerlerinden sonuç çıkararak veya iki ya da daha fazla değer arasındaki ilişkiyi toparlayarak veri görüntülemede yardımcı olur. Extrapolasyon gözlemlenen veri aralığı ötesinde oluşturulmuş eğriler için kullanılır.

3.4.3.2.1 Eğri Uydurma Çeşitleri

Doğru ve Polinomal Eğri Uydurma

Birinci dereceden polinom eşitliği ile başlarsak;

$y=ax+b$

Bu 'a' eğimine sahip bir doğruyu belirtir. Bu yüzden birinci dereceden bu eşitlik herhangi iki noktadan elde edilir.

Eğer eşitliğin derecesini artıracak olursak, aşağıdaki eşitlik elde edilir.

$y=ax^2+bx+c$

Bu üç noktadan oluşturulmuş basit bir eğri denklemini belirtir.

Eşitliğin derecesi bir daha artırıldığında;

$y=ax^3+bx^2+cx+d$

Elde edilir. Bu durumda bu denklem dört noktadan elde edilmiştir.

Eğer bu örnekleri genelleyecek olursak Düzlemde N+1 adet noktadan N ‘inci dereceden bir polinom geçirmek mümkündür. Bu durumda üçüncü dereceden bir polinom dört noktada, dördüncü derecen bir polinom beş noktada kısıtlanacaktır.

Eğri uydurmada en çok kullanılan en küçük kareler metodu kullanıldığında;

$$F(x) = a_0 + a_1 x + a_2 x^2 + \cdots . + a_k x^k$$

Hataların minimum olması için hataların karelerin türevleri alınır ve sıfıra eşitlenirse aşağıdaki denklem takımları kolayca bulunabilir.

$$\begin{aligned}
a_0 \sum_{i=1}^{n} 1 + a_1 \sum_{i=1}^{n} x_i^1 + a_2 \sum_{i=1}^{n} x_i^2 + \ldots + a_k \sum_{i=1}^{n} x_i^k &= \sum_{i=1}^{n} y_i \\
a_0 \sum_{i=1}^{n} x_i^1 + a_1 \sum_{i=1}^{n} x_i^2 + a_2 \sum_{i=1}^{n} x_i^3 + \ldots + a_k \sum_{i=1}^{n} x_i^{k+1} &= \sum_{i=1}^{n} y_i x_i \\
a_0 \sum_{i=1}^{n} x_i^2 + a_1 \sum_{i=1}^{n} x_i^3 + a_2 \sum_{i=1}^{n} x_i^4 + \ldots + a_k \sum_{i=1}^{n} x_i^{k+2} &= \sum_{i=1}^{n} y_i x_i^2
\end{aligned}$$

Matris formatında yazılacak olursa;

$$\begin{bmatrix}
n & \sum_{i=1}^{n} x_i^1 & \cdots & \sum_{i=1}^{n} x_i^k \\
\sum_{i=1}^{n} x_i^1 & \sum_{i=1}^{n} x_i^2 & \cdots & \sum_{i=1}^{n} x_i^{k+1} \\
\sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \cdots & \sum_{i=1}^{n} x_i^{k+2} \\
\cdots & \cdots & \cdots & \cdots \\
\sum_{i=1}^{n} x_i^{k+1} & \sum_{i=1}^{n} x_i^{k+2} & \cdots & \sum_{i=1}^{n} x_i^{k+k}
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \cdots \\ a_k \end{bmatrix}
=
\begin{bmatrix}
\sum_{i=1}^{n} y_i \\
\sum_{i=1}^{n} y_i x_i \\
\sum_{i=1}^{n} y_i x_i^2 \\
\cdots \\
\sum_{i=1}^{n} y_i x_i^k
\end{bmatrix}$$

Sonuç olarak k+1 tane bilinmeyen ’ a ‘ değeri vardır. Bilinmeyen 0-k arasındaki a değerleri denklem takımlarının çözüm metotlarından biri kullanılarak çözümlenebilir [63], [64],[65].

3.4.3.3 Gauss-Newton Algoritması

Gauss-Newton algoritması lineer olmayan en küçük kareleri çözümlemek için kullanılır. Bu yöntem en az bir fonksiyonu bulmak için kullanıldığından Newton metodunun biraz değişime uğramışı olarak düşünülebilir. Newton metodundan farklı

olarak, Gauss-Newton algoritması sadece kare fonksiyonu değeri toplamını azaltmak için kullanılır.

3.4.3.3.1 Tanımlama

Bir m fonksiyonunda r = ($r_{1, r2...}$, r_m), n tane değişkenle $\beta = (\beta_1, \beta_2, \ldots \beta_n)$ ve m≥n olmak şartıyla, Gauss- Newton algoritması hesap edilen parametrelerin hataları minimum yapma eğiliminde değişim göstereceği kabulü altında iterasyon yapar.

$$S(\beta) = \sum_{i=1}^{m} r_{i^2}(\beta)$$

Minimum olarak en başta $\beta^{(0)}$ noktasından başlayarak iterasyona devam edilir.

$$\beta^{(s+1)} = \beta^s - (J_r^T J_r)^{-1} J_r^T r(\beta^{(s)})$$

$$J_r = \frac{\partial_{r_i}}{\partial_{\beta_j}}(\beta^{(s)})$$

(J = Jacobi matrisi T = transpoz)

Veri uydurmada amaç β parametresini bulmaktır.

(x_i, yi) , $y = f(x_i, \beta)$ şeklindeki bir fonksiyondaki en uygun nokta, r_i= tahmini ve gerçek değer arasındaki fark olmak üzere;

$$r_i(\beta) = y_i - f(x_i, \beta)$$

Gauss- Newton metodu f fonksiyonun Jacobian'ı olarak ifade edilir

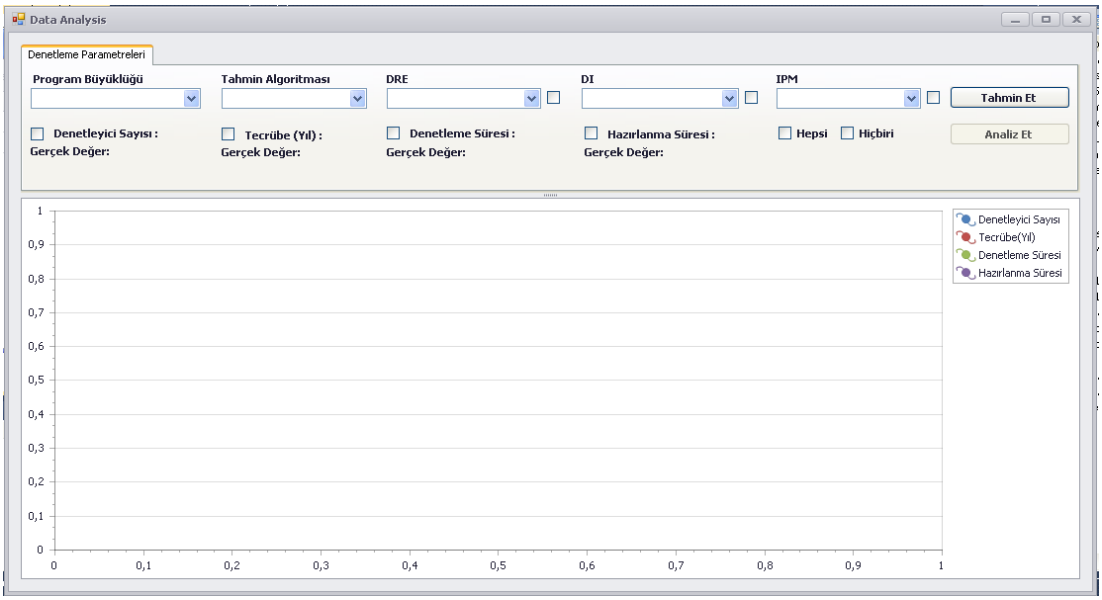
$$\beta^{(s+1)} = \beta^{(s)} + \left(J_f^T J_f\right)^{-1} J_f^T r(\beta^{(s)})$$

(m≥n kesinlikle olması gereken bir şarttır. Eğer bu şart sağlanmazsa Jacobi matrisinde transpoz alınamaz ve eşitlik çözülemez) [66].

3.4.4 IPM, DI ve DRE Metrikleri Kullanılarak Denetleme Sürecini Etkileyen Değişkenlerin Tahmin Edilmesi

Tez çalışmasının bu bölümünde denetleme verimliliğinin ölçülmesinde kullanılan DRE, DI ve IPM metrikleri ve tahmin algoritmaları yardımıyla denetleme süreci verimliliği üzerinde doğrudan etkisi bulunan değişkenleri tahmin etmeye yönelik çalışmanın işleyişinden bahsedilecektir.

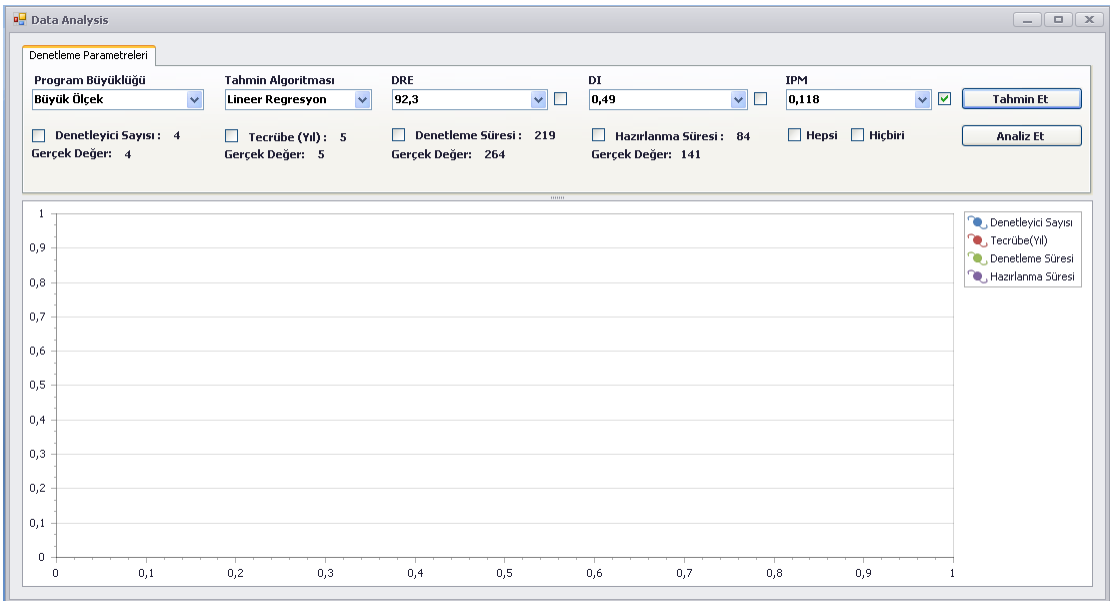
Şekil 3.6 da yer alan ekran görüntüsü tahmin çatısına ait açılış sayfadır. Tahmin işleminde kullanılan veriler aynı dilde yazılmış, aynı sektöre proje üreten ürün tabanlı yazılımlara aittir. Projeler büyük ölçekli, orta ölçekli ve küçük ölçekli olmak üzere grup altında toparlanmıştır. Bu veriler sınıflandırılırken proje geliştirme süreleri baz alınmıştır.



Şekil 3.6 Açılış sayfası

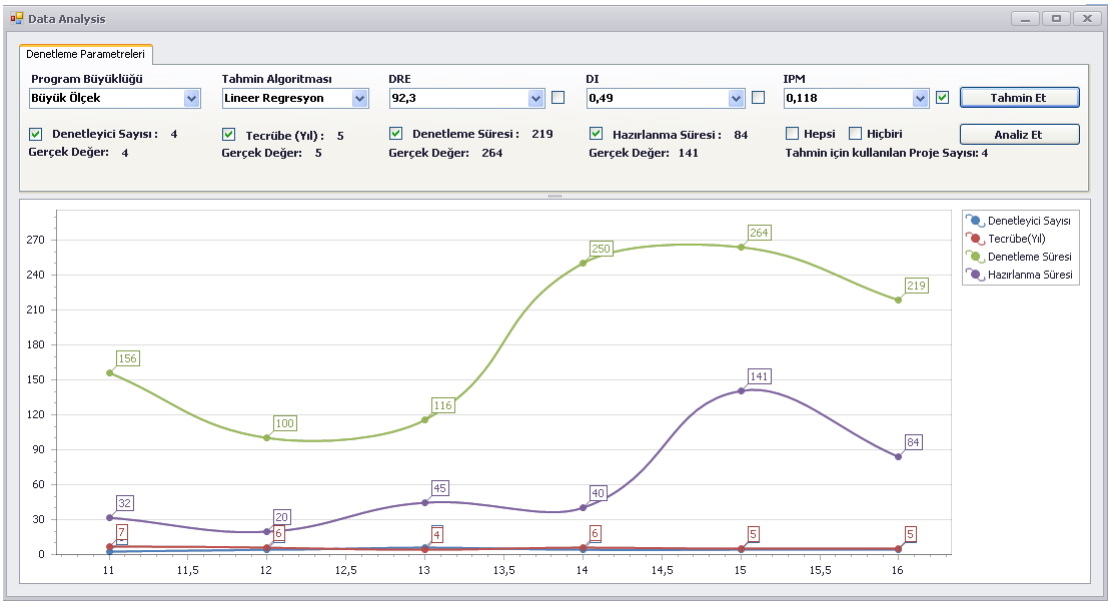
İlk olarak proje büyüklüğü açılan kutusundan tahmin işleminin gerçekleşeceği büyüklüğe uygun değer seçilmelidir. Seçilen proje büyüklüğüne göre IPM, DRE ve DI değerleri ilgili açılan kutularda listelenecektir. Daha sonra tahmin algoritması bölümünden tahmin işleminde kullanılmak istenilen algoritma seçebilir. Kullanıcıya üç adet seçenek sunulmaktadır. Bunlar Lineer regresyon, Eğri Uydurma ve Gauss-Newton algoritmalarıdır. Tahmin işleminde kullanılmak istenen değerler belirtmek

zorundadır. Örneğin büyük ölçekli projelerde lineer regresyon ve IPM değerleri kullanılarak işlem yapılmak isteniyorsa seçimler Şekil 3.7 deki gibi yapılmalı tahmin et butonu tıklanmalıdır. Tahmin edilen değerler Denetleyici Sayısı: Tecrübe (Yıl): Denetleme Süresi: ve Hazırlanma Süresi: alanlarında görülecektir.



Şekil 3.7 Grafiksiz tahmin edilen değerler ekranı

Kullanıcı tahmin işleminde kullanılan projelere ait değerlerini ve tahmin edilen değerleri grafik şeklide görmek istiyorsa ilgili kutucukları şekil 3.8 deki gibi seçmelidir. Yeni değerlerin eski değerler ile ortak bir tabloda görünmesi kullanıcıya yapılan tahminin tutarlılığını analiz etme imkânı sunmaktadır.



Şekil 3.8 Grafikli tahmin edilen değerler ekranı

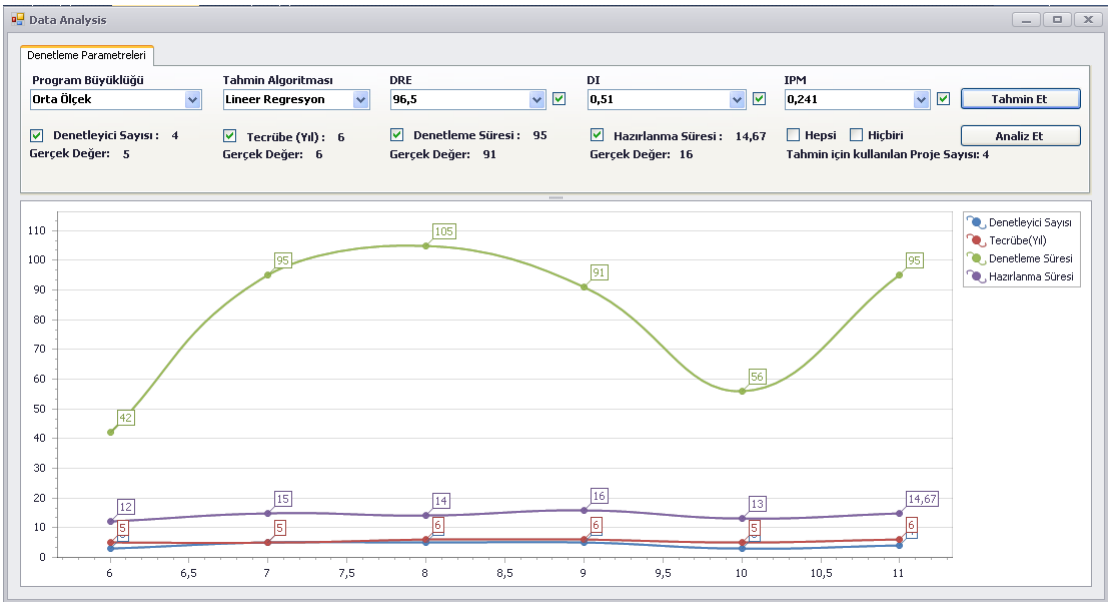
Kullanıcı eğer ekranda belirlediği metrikler doğrultusunda hangi algoritmanın daha iyi tahmin yaptığını görmek istiyorsa Analiz Et butonuna tıklamalıdır. Şekil 3.9 da analiz et butonuna tıklandıktan sonra açılan ekran gösterilmiştir. Burada amaç uygun algoritmayı ve metrikleri belirlemek için aradaki farklılıkları görmektir.

Algoritma	Tecrübe(Yıl)	Denetleyici Sayısı	Hazırlanma Süresi	Denetleme Süresi
Lineer Regresyon	5	4	84	219
Curve Fitting	5	4	134	234
Gauss-Newton	5	4	133,24	233,81

Şekil 3.9 Algoritmalar ve tahmin edilen değerler

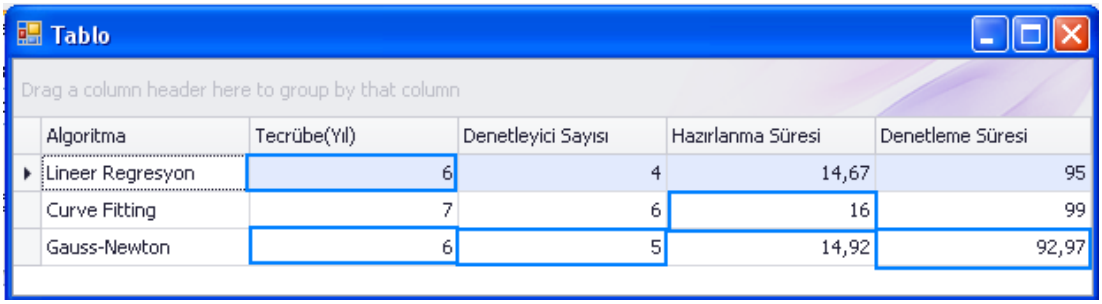
3.4.4.1 Tahmin Sonuçlarının Değerlendirilmesi

Örnek olarak orta ölçekli bir proje tercih edilmiştir. Şekil 3.10 de görüldüğü üzere orta ölçekli bir projede Denetleyici sayısı, Tecrübe (Yıl), Denetleme Süresi ve Hazırlanma Süresi değerlerinin tahmin edilmesi için Lineer Regresyon tahmin algoritması tercih edilmiş ve tahmin işleminde kullanılacak metrikler DRE, DI ve IPM olarak belirlenmiştir. Tahmin et butonuna tıklanmasıyla beraber değişkenlere yönelik tahmini değerler hesaplanmış ekranda grafik olarak gösterilmiştir.



Şekil 3.10 Orta ölçekli bir proje için değişken tahmini

Aynı metrikler ile farklı algoritmalar kullanılarak yapılan hesaplamaları görmek için Analiz Et butonuna tıklanmalıdır. Şekil 3.11 deki ekranda gerçek değere en yakın değerler kutular içine alınmıştır. Orta ölçekli projelerde üç metriğinde hesaplamaya katıldığı durumlarda Gauss-Newton algoritmasının daha iyi çalıştığı görülmüştür.



Algoritma	Tecrübe(Yıl)	Denetleyici Sayısı	Hazırlanma Süresi	Denetleme Süresi
Lineer Regresyon	6	4	14,67	95
Curve Fitting	7	6	16	99
Gauss-Newton	6	5	14,92	92,97

Şekil 3.11 Orta ölçek, DRE, DI ve IPM değerleri kullanılarak analiz yapılması.

Burada amaç hangi algoritmanın daha iyi tahmin ettiğinden ziyade hangi metrikler kullanılarak daha iyi sonuç elde edileceğinin belirlenmesidir. Algoritmada değişiklik yapmadan sadece metriklerin değiştirilmesinin sonuç üzerindeki etkisi Şekil 3.12 da gösterilştir. DRE değeri devre dışı bırakılmış hesaplama IPM ve DI değerleri kullanılarak yapılmıştır. DRE metriğinin devre dışı bırakılması sonucunda değerlerde

ciddi değişiklikler meydana gelmemiştir.

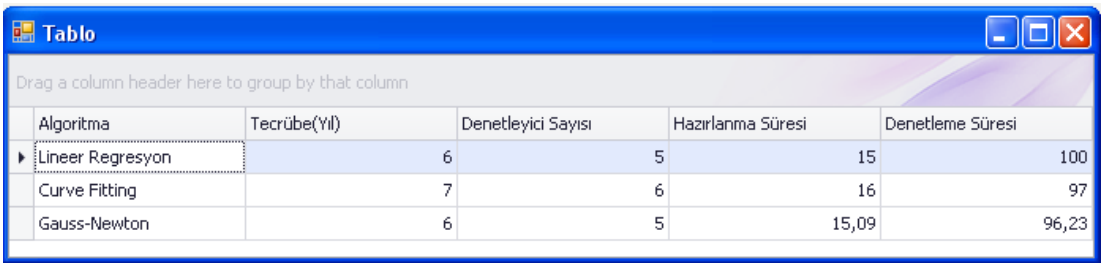

Tablo

Drag a column header here to group by that column

Algoritma	Tecrübe(Yıl)	Denetleyici Sayısı	Hazırlanma Süresi	Denetleme Süresi
Lineer Regresyon	6	4	14,5	91
Curve Fitting	7	6	16	101
Gauss-Newton	6	5	14,78	91,94

Şekil 3.12 Orta ölçek, DI ve IPM değerleri kullanılarak analiz yapılması.

Aynı durumda DI değeri devre dışı bırakıldığında söz konusu olmuştur. Değerlerde ciddi değişiklikler gözlemlenmemiştir. IPM değeri devre dışı bırakılıp DI ve DRE kullanılarak hesaplama yapıldığında hazırlanma süresi değerlerinin gerçek değere yaklaştığı, denetleme süresi değerinin ise gerçek değerden uzaklaştığı gözlemlenmiştir. Şekil 3.13 da bu değerler görülmektedir.


Tablo

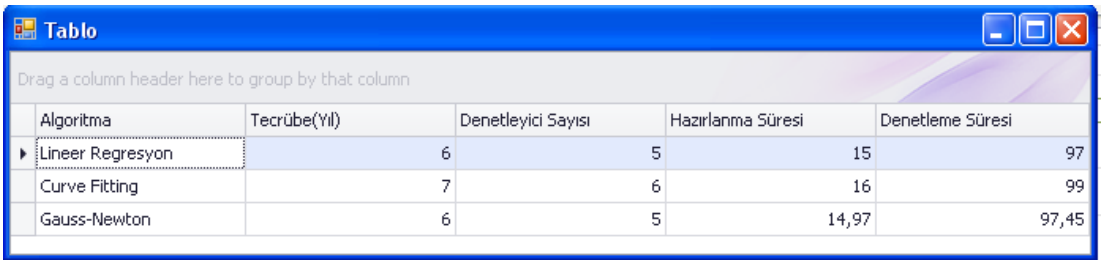
Drag a column header here to group by that column

Algoritma	Tecrübe(Yıl)	Denetleyici Sayısı	Hazırlanma Süresi	Denetleme Süresi
Lineer Regresyon	6	5	15	100
Curve Fitting	7	6	16	97
Gauss-Newton	6	5	15,09	96,23

Şekil 3.13 Orta ölçek, DI ve IPM değerleri kullanılarak analiz yapılması.

Hangi metriğin hangi değerler üzerinde etkiye sahip olduğunu görebilmek için hesaplama algoritması sabit tutulmuş. DRE, DI ve IPM değerleri hesaplamaya tek tek gönderilmiştir.

Şekil 3.14 de sadece DI kullanılarak yapılan hesaplama sonuçlarına yer verilmiştir. Hazırlanma süresi gerçek değere yaklaşmış denetleme süresi değeri gerçek değerden uzaklaşmıştır.



Algoritma	Tecrübe(Yıl)	Denetleyici Sayısı	Hazırlanma Süresi	Denetleme Süresi
Lineer Regresyon	6	5	15	97
Curve Fitting	7	6	16	99
Gauss-Newton	6	5	14,97	97,45

Şekil 3.14 Orta ölçek, DI değerleri kullanılarak analiz yapılması.

Sadece IPM metriği kullanılarak hesaplama yapıldığında denetleme süresi değeri gerçek değerden uzaklaşmış bu değerin altında bir değere düşmüştür. Aynı hesaplama sadece DRE kullanılarak yapıldığında gerçek değerlere en yakın hazırlanma süresi değerlerine ulaşılmış. Denetleme süresi gerçek değerlere diğer metriklerle yapılan hesaplamalara göre daha çok yaklaşmıştır. Orta ölçekli projelerde tecrübe ve denetleyici sayısı değerleri Eğri Uydurma algoritmasının kullanıldığı durumlar haricinde tüm hesaplamalarda doğru sonuçları vermiştir. Hazırlanma süresi için en uygun değer IPM, DI ve DRE metriklerinin bir arada kullanıldığı durumlarda elde edilmiştir. Denetleme süresi için en yakın tahmin sadece DRE kullanılarak yapılan hesaplamalarda elde edilmiştir.

Denetleme sürecinin verimliliğini ölçmeye yönelik yapılan pek çok çalışmada denetleme süresi, hazırlanma süresi, denetleyici sayısı, denetleyici tecrübe seviyesi ve büyüklük değerleri parametre olarak kullanılmıştır. Değerlerin kullanılma sebebi denetleme süreci verimliliği üzerinde etki sahibi olmalarıdır. Bu tez çalışmasında denetleme süresi, hazırlanma süresi, denetleyici sayısı, denetleyici tecrübe seviyesi ve büyüklük değerlerinin verimlilik hesaplama parametresi olarak kullanılmasının yanında sürecin başlangıcında değerlerin doğru belirlenmesiyle verimliliğin kontrol altında tutulabileceği gösterilmeye çalışılmıştır. Değerlerin algoritmalar ve metrikler kullanılarak tahmin edilmesi sağlanmıştır.

4. Sonuç

Sonuç olarak kaliteli bir yazılım ürünü için hata yönetim sistemlerinin kullanılması kaçınılmazdır. Kod denetleme kaynak kodda yer alan hataların belirlenmesi hususunda çok etkili bir süreçtir. Fakat yanlış yönetilen bir kod denetleme süreci başarısız olmaya mahkûmdur. Bu nedenle süreci etkileyen faktörler belirlenmeli ve sürecin başlangıcında bu değişkenler için doğru değerler belirlenmelidir.

Bu tez çalışmasında değişken değerlerinin doğru belirlenmesi sorununu çözmeye yönelik bir yapı oluşturulmuştur. Lineer regresyon, Eğri uydurma ve Gauss-Newton algoritmaları tahmin işlemini gerçekleştirmek için kullanılmıştır. Değişken değerlerinin tahmini işleminde denetleme performans metriği, hata giderme etkinliği ve denetleme etkinliği metrikleri baz alınmıştır. Denetleme derinliği ve hata giderme etkinliği metrikleri denetleme sürecinin verimliliğini ölçmek için kullanılmaktadır. Denetleme performans metriği ise denetleme sürecini gerekleştiren denetleyicilerin verimliliğinin belirlenmesine yardımcı olmaktadır.

Tez çalışmasının ilk bölümünde tezin amacı ve genelinden bahsedilmiştir. İkinci bölümünde kod denetleme yapısından bahsedilmiştir. Kod denetleme tanımı yapılmış ve kod denetleme elementlerinden bahsedilmiştir. Kod denetlemenin başlaması ve bitmesi için gerekli giriş-çıkış kriterleri ve ilk denetleme yöntemi olan Fagan denetleme yöntemi anlatılmıştır. Daha sonra geçmişten günümüze kadar geliştirilen kod denetleme süreçlerinden bahsedilmiştir. Denetleme sürecinde yer alacak ekibin rolleri ve bu rollerin görev ve sorumlulukları anlatılmıştır. Denetleme işlemi sırasında kullanılan okuma tekniklerinden bahsedilmiştir. Kod denetleme yönteminin faydaları anlatılmıştır. Tez çalışmasının üçüncü bölümünde Kod denetleme sürecini etkileyen değişkenler ve denetleme metrikleri anlatılmıştır. Metrik kavramından bahsedilmiş, kod denetleme metriklerine yer verilmiştir. Daha sonra denetleme sürecini etkileyen değişkenleri bulunmasına yönelik bir çalışma ve bu değişkenler için optimal değerler öneren bir çalışmaya yer verilmiştir. Kod denetleme işlemi için optimal değerlerin bulunmasına yönelik yapılan çalışmada kullanılacak metriklere ve bu metriklerin tercih edilme sebeplerine yer verilmiştir. Hata giderme etkinliği ürünün içerdiği tüm hatalar kullanılarak hesaplandığı için tercih edilmiştir. Diğer hata denetimi baz alınarak hesaplanılan metriklerde müşteri tarafında belirlenen hata

sayısı işleme dahil edilmemektedir. Oysaki müşteri ürün eline geçmeden önce ne kadar hata giderildiğiyle değil ürün eline geçtikten sonra karşılaştığı hata miktarıyla ilgilenmektedir. Müşteri memnuniyeti kazanılmak isteniyorsa hata giderme etkinliği metriği tercih edilmelidir. Denetleme derinliği sürece daha çabuk müdahale edilmesini sağlayan önemli bir metriktir. Test aşamasında ve denetleme aşamasında tespit edilen hatalar baz alınarak hesaplanır. Verimli bir kod denetleme sürecinin test ve kod denetleme esnasında bulunan hataların yarısından fazlasını bulmuş olması beklenmektedir. Bu durum göz önünde bulundurularak denetleme sürecinin verimliliği kolayca takip edilebilir. Kod denetleme sürecinde denetlemeyi gerçekleştiren ekibin performansı büyük önem taşımaktadır. Bu sebeple denetleme performans metriği tercih edilmiştir. Bu metriğin tercih edilme sebebi hesaplama yaparken denetleyici sayısı, hazırlanma süresi, denetleme süresi ve denetleyicilerin tecrübe seviyesi gibi önemli değişkenleri içeriyor olmasıdır.

Bu tez çalışmasında yeni gerçekleştirilecek kod denetleme işleminde değişken değerleri tahmini yapılabilmesi için aynı proje büyüklüğüne sahip projeler arasından bir tanesi seçilmiş ve diğer projelere ait DRE, DI ve IPM değerlerinden yola çıkılarak hesaplama yapılmıştır.

Daha çok veriye ulaşılması halinde mevcut DRE, DI ve IPM değerlerini kullanmak yerine hedeflenen DRE, DI ve IPM değerleri belirlenerek bu veriler doğrultusunda en uygun değişken değerlerinin hesaplanması planlanmaktadır. Tez çalışmasının gerçekleşmesi için veri araması yapılırken Türkiye'de kod denetleme sürecini gerçekleştiren firma sayısının yok denecek kadar az olduğu gerçeğiyle karşılaşılmıştır. Yazılım ürününün kalitesini arttıran, bakım süresini ve maliyetini düşüren böylesine etkili bir yöntem ne yazık ki göz ardı edilmektedir.

Kaynaklar

[5] Köksal, G., Nalbant S., Yeniden Muayenesine Karar Vermek İçin Uygun Politikanın Seçimi, Araştırma Çalışması, Endüstri Mühendisliği Bölümü, Orta Doğu Teknik Üniversitesi, Ankara

[6] Nair, TR Gopalakrishnan, V. Suma, and P. Kumar Tiwari. "Significance of depth of inspection and inspection performance metrics for consistent defect management in software industry." IET Software 6.6 (2012): 524-535.

[7] Yu, Liguo, Robert P. Batzinger, and Srini Ramaswamy. "A comparison of the efficiencies of code inspections in software development and maintenance."Proceeding of 2006 International Conference on Software Engineering Research and Practice, Las Vegas, Nevada, June. 2006.

[8] Kendrick, T.: 'Defining and Implementing Metrics for Project Risk Reduction', 2005

[9] Tervonen, Ilkka, and Juha Iisakka. "Monitoring software inspections with prescriptive metrics." Software QA 4.3 (1997): 16-23.

[10] Bilik, Ersan, et al. "Yazılım Geliştirme Sürecinde Yazılım Metriklerindeki Değişimi Değerlendirmek: Dev| Efor Projesi Metrik Toplama Modülü."

[11] Suma, V., and T. R. Nair. "Defect management strategies in software development." arXiv preprint arXiv:1209.5573 (2012)

[12] Fagan, Michael E. "Advances in software inspections." Software Engineering, IEEE Transactions on 7 (1986): 744-751.

[13] Ackerman, A. Frank, Lynne S. Buchwald, and Frank H. Lewski. "Software inspections: An effective verification process." Software, IEEE 6.3 (1989): 31-36.

[14] Fagan, Michael E. "Design and code inspections to reduce errors in program development." IBM Systems Journal 15.3 (1976): 182-211.

[15] Porter, Adam, Harvey Siy, and Lawrence Votta. "A review of software inspections." Advances in Computers 42 (1996): 39-76.

[16] A.F. Ackerman, P.J. Fowler, and R.G. Ebenau, “Software Inspections and the Industrial Production of Software,” in Software Validation, H.L. Hausen, ed., Elsevier, Amsterdam, 1984, pp. 134

[17] Kandt, Ronald Kirk. "A software defect detection methodology." Jet Propulsion Laboratory (2003).

[18] NASA–STD 8739.9 SOFTWARE FORMAL INSPECTIONS STANDARD

[19] NASA-GB-A302,SOFTWARE FORMAL INSPECTIONS GUIDEBOOK

[20] IEEE Standard for Software Reviews

[21]Software Engineering1 Course Page, erişim adresi: http://www.cs.toronto.edu/~sme/CSC444F/inspections, erişim tarihi: 2 Ocak 2013

[22] Laitenberger, Oliver. "A survey of software inspection technologies." Handbook on Software Engineering and Knowledge Engineering 2 (2002): 517-555.

[23] Laitenberger, Oliver. "Studying the effects of code inspection and structural testing on software quality." Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on. IEEE, 1998.

[24]Aurum, Aybuke, Håkan Petersson, and Claes Wohlin. "State-of-the-art: software inspections after 25 years." Software Testing, Verification and Reliability 12.3 (2002): 133-154.

[25] Aurum, Aybiike, Claes Wohlin, and Hakan Petersson. "Increasing the understanding of effectiveness in software inspections using published data sets." Journal of Research and Practice in Information Technology 37.3 (2005): 253-266.

[26] Ebenau, R. G. and Strauss, S. H. (1994): Software Inspection Process. McGraw Hill (System Design and Implementation Series), ISBN 0-07-062166-7

[27] McCarthy, P.; Porter, R.; Riedl, J. (1995): 'An Experiment to Assess Cost-Benefit of Inspection Meetings and Their Alternatives'. Technical Report, Computer Science, Dept., University of Maryland.

[28] Porter, A. A. and Votta, L. G. (1994): 'An Experiment to Assess Different Defect Detection Methods for Software Requirements Inspections'. Proceedings on 16th International Conference on Software Engineering, ICSE-16, 103-112.

[29] Eick, S. G.; Loader, C. R.; Long, M. D.; Votta, L. G.; Vander Wiel, S. (1992): 'Estimating Software Fault Content Before Coding'. Proceedings of the 14th International Conference on Software Engineering, 49-65

[30] Mashayekhi, V., Drake, J. M., Tsai, W.T., and Riedl, J., 1993. Distributed, Collaborative Software Inspection. IEEE Software, 10:66-75.

[31] Petersson, Håkan, and Thomas Thelin. "A Survey of Capture-Recapture in Software Inspections." First Swedish Conference on Software Engineering Research and Practise: Proceedings. 2001.

[32]Wheeler, D. A.; Brykczynski, B.; Meeson, R. N. (1996): 'Peer Review Process Similar to Inspection'. Software Inspection: An Industry Best Practice. IEEE Computer Society Press, USA. ISBN 0-8186-7340-0

[33]Gilb, T. and Graham, D. (1993): Software Inspection. Addison Wesley Publishing Company. ISBN 0-201-63181-4

[34] Parnas, D. L., 1987. Active Design Reviews: Principles and Practice. Journal of Systems and Software, 7:259-265.

[35]Oladele, Rufus O. "Reading Techniques for Software Inspection: Review and Analysis."

[36] Kang, P. Software inspection Process:Integration into a Software Development Cycle September 19, 2009

[37] O'Neill, Don. "Issues in software inspection." Software, IEEE 14.1 (1997): 18-19.arnumber=00566420)

[38]Kubendran, G., Software Inspection and Defect Management Erişim adresi: http://www.slideshare.net/ajaykemparaj/software-inspection-and-defect-management, Erişim tarihi:Aralık 2012

[39] Alexandrov, I., et al. "Impact of Software Review and Inspection." Proceedings of the CHEP2000 conference. 2000.

[40]Kyung, Taewon, and Sangkuk Kim. "A Study on the Development of Rules for Effective Code Inspection: Case Study of Company "A" Information System."International Journal of Computers 1.4 (2007): 223-227.

[41] Li, Q., Shu, F., Boehm, B., Wang, Q.: 'Improving the ROI of software quality assurance activities: an empirical study'. (LNCS, 6195), 2010, pp. 357 – 368

[42] Barney, S., Wohlin, C., Aurum, A.: 'Balancing software product investments'. Proc. Third Int. Symposium on Empirical Software Engineering and Measurement, Orlando, USA, October 2009, pp. 257 – 268

[43] L. Briand, K. El Emam, O. Laitenberger, T. Fussbroich, "Using Simulation to Build Inspection Efficiency Benchmarks for Development Projects", Proc. 20th International Conference on Software Engineering, Kyoto, Japan, pp.340-349, 1998.

[44] Barnard, Jack, and Art Price. "Managing code inspection information."Software, IEEE 11.2 (1994): 59-69.

[45] Nair, T. R. "Estimation of characteristics of a software team for implementing effective inspection process through inspection performance metric." arXiv preprint arXiv:1107.3201 (2011).

[46] Lazic, Ljubomir, and Nikos Mastorakis. "Cost effective software test metrics."WSEAS Transactions on Computers 7.6 (2008): 599-619.

[47] Goodman, Paul. 2004. Software metrics: best practices for successful IT management, USA: Rothstein Associates Inc.

[48] Nair, T. R. "Estimation of characteristics of a software team for implementing effective inspection process through inspection performance metric." arXiv preprint arXiv:1107.3201 (2011).

[49] Tervonen, Ilkka, and Juha Iisakka. "Monitoring software inspections with prescriptive metrics." Software QA 4.3 (1997): 16-23.

[50] Grady R.B., and van Slack T., Key Lessons In Achieving Widespread Inspection Use, IEEE Software, vol 11, no 4, 1994, pp. 46-57

[51] Yu, Liguo, Robert P. Batzinger, and Srini Ramaswamy. "A comparison of the efficiencies of code inspections in software development and maintenance."Proceeding of 2006 International Conference on Software Engineering Research and Practice, Las Vegas, Nevada, June. 2006

[52] Christenson, Dennis A., Steel T. Huang, and Alfred J. Lamperez. "Statistical quality control applied to code inspections." Selected Areas in Communications, IEEE Journal on 8.2 (1990): 196-200.

[53]Milicic, D., Wohlin, C.: 'Distribution patterns of effort estimations'. Proc. IEEE Conf. Euromicro, Rennes, France, 2004, pp. 422 – 429

[54] Kanabar, V.: 'A parametric cost model to estimate oracle-based application development effort'. Proc. Project Management in Practice Conf., Boston, MA, 15 – 17 May 2006

[55] Gopalakrishnan Nair, T. R., and V. Suma. "Defect Management Using Depth of Inspection and the Inspection Performance Metric." (2012).

[56] Suma, V., and T. R. Nair. "Defect management strategies in software development." arXiv preprint arXiv:1209.5573 (2012).

[57] Kan, Stephen H. Metrics and models in software quality engineering. Pearson Education India, 2003.)

[58] Jones, Caper. "Measuring defect potentials and defect removal efficiency."CrossTalk The Journal of Defense Software Engineering 21.6 (2008): 11-13.

[59] Software Defect Removal Efficiency 00488361 Jones, Capers. "Software defect-removal efficiency." Computer 29.4 (1996): 94-95

[60] Regresyon Analizi, Erişim Adresi: http://tr.wikipedia.org/wiki/Regresyon_analizi Erişim Tarihi: Mart 2013

[61] Regresyon Analizi, Erişim Adresi : www.baskent.edu.tr/.../veri_analizi_regresyon, Erişim Tarihi: Mart 2013

[62] Regresyon Analizi, Erişim Adresi : www.baskent.edu.tr/.../dersler/.../dersnotlari, Erişim Tarihi: Mart 2013

[63]Eğri Uydurma:, Erişim Adresi : http://web.itu.edu.tr/~yukselen/HM504/02-%20E%F0ri%20uydurma%20ve%20interpolasyon.pdf, Erişim Tarihi: Mart 2013

[64]Curve Fitting, Erişim adresi: http://en.wikipedia.org/wiki/Curve_fitting, Erişim tarihi: Mart 2013

[65] Eğri Uydurma, Erişim Adresi: http://www.cadcamokulu.net/wp-content/uploads/downloads/2012/11/egri_uydurma.pdf, Erişim Tarihi: Mart 2013

[66]Gauss-Newton Algoritması, Erişim Adresi: http://en.wikipedia.org/wiki/Gauss–Newton_algorithm Erişim Tarihi: Mart 2013

[67] McConnell, Steve. Software project survival guide. Microsoft press, 2009.

[68]Caldiera, Victor R. Basili1 Gianluigi, and H. Dieter Rombach. "The goal question metric approach." Encyclopedia of software engineering 2 (1994): 528-532.

Özgeçmiş

Kişisel Bilgiler

Soyadı, adı : AYIK, Hilal

Uyruğu : T.C.

Doğum tarihi ve yeri : 07.09.1987 Ankara

Medeni hali : Bekar

Telefon : 0 (312) 339 98 94

e-mail : hayik@etu.edu.tr

Eğitim

Derece	Eğitim Birimi	Mezuniyet tarihi
Lisans	Bilkent Üniversitesi/ Bilgisayar Teknolojisi ve Bilişim Sistemleri	2010

İş Deneyimi

Yıl	**Yer**	**Görev**
2012 Ocak – Aralık	TCHealth Bilgi Teknolojileri A.Ş.	Yazılım Mühendisi

Yabancı Dil

İngilizce