Yazılım Yeniden Muayenesine Karar Vermek İçin Uygun Politikanın Seçimi

Doç. Dr. Gülser KÖKSAL[1] Yük. Müh. Serkan NALBANT[2]

[1,2] Endüstri Mühendisliği Bölümü, Orta Doğu Teknik Üniversitesi, Ankara

[1]e-posta: koksal@ie.metu.edu.tr [2]e-posta: e130576@metu.edu.tr

Özet

Yazılım muayeneleri yazılım projelerinde oluşan hataların ortadan kaldırılması için etkili bir tekniktir. İlk muayaneden sonra yapılabilen yeniden muayene sayesinde ise yazılım ürününde kalan hatalar daha da azaltılabilir. Bu çalışma, yazılım kod muayenelerine ilişkin yeniden muayeneye karar verme politikalarının performanslarını, Yazılım Yetenek Olgunluk Modeli Seviye 3'e göre yapılanmış bir yazılım organizasyonu kontekstinde değerlendirmektedir. Bu değerlendirme, farklı yeniden muayeneye karar verme politikalarının yazılım projelerinin maliyet, takvim, ve kalite hedeflerine etkilerinin simülasyon ve istatistiksel deney yöntemleri ile karşılaştırılmasına dayanmaktadır. Çalışma, yeniden muayene kararının majör hataların etkililik ölçüsünün nispen yüksek bir eşik değerine göre karşılaştırılması sonucu alınmasını tavsiye etmektedir.

Abstract

Software inspection is an effective defect removal technique for software projects. After the initial inspection, a reinspection may be performed for decreasing the number of remaining defects further. This study evaluates the performance of different software reinspection decision methods for code inspections conducted in the context of a Software Capability Maturity Model Level 3 organization. The evaluation is based on comparisons of different reinspection decision policies with respect to cost, schedule and quality objectives pursued by software projects, through simulation and statistical experimentation. The study recommends concluding the reinspection decision by comparing inspection effectiveness measure for major defects with respect to a moderately high threshold value.

1. Giriş

Günümüzde yazılım sistemlerinin gerçekleştirilmesi karmaşık projelerin başarıyla, yani zamanında, tahsis edilen bütçe içersinde kalınarak, ve yüksek kalite ile, tamamlanmasına bağlıdır. Bu tip yazılımların üretilmesi yazılım geliştirme yaşam döngüsünün gereksinim analizi, yazılım tasarımı, kodlama gibi çeşitli aşamalarında yaratılan hataların ortadan kaldırılması ile başarılabilir. Yazılım geliştiren organizasyonlar bu hataların yazılımdan çıkarılması sağlamak ve dolayısıyla kullanıcıya ulaşmasını engellemek için birçok teknik kullanırlar. Bu tekniklerden biri de eş gözden geçirmeleridir. Eş gözden geçirme kapsamında, yazılım ürünü (kod veya doküman) bünyesinde bulunan hataların ve üzerinde yapılabilecek iyileştirmelerin tespiti amacıyla üründen bağımsız (yazardan farklı) proje personeli tarafından incelenir.

Yazılım muayenesi özel bir eş gözden geçirme çeşididir. Bu eş gözden geçirme yönteminde, yazılım ürününün incelenmesi muayene prosedürleri ve teknikleri hakkında eğitilmiş kişiler tarafından belirli aşamaları olan iyi tanımlanmış prosedürlere göre gerçekleştirilir. Yazılım muayenesinin amacı, yazılım projesinin maliyet, takvim, ve kalite bakış açılarına göre ortaya çıkan performansını geliştirmektir. İlk muayene tamamlandıktan sonra, yazılım ürünündeki hataları bir seviye daha azaltmak ve dolayısıyla muayenenin getirdiği avantajlardan daha fazla faydalanmak amacıyla, yazılım ürününe ikinci bir muayene uygulanabilir. Fakat, yeniden muayene olarak da adlandırılan bu ikinci muayenenin yapılabilmesi yazılım muayenesi faaliyetlerine ayrılan proje kaynaklarının arttırılmasını gerektirir. Bu yüzden, proje yönetimi yazılım ürününün yeniden incelenmesi (muayenesi) için projenin değerli kaynaklarının planlanandan fazla tahsis edilmesi kararını akılcı bir şekilde almalıdır. Literatürde bu önemli kararın verilmesine yönelik nesnel (objektif) ve öznel(subjektif) metodlar önerilmektedir. Fakat, yeniden muayene kararının sonuçlandırılması için önerilen nesnel metodların uygun olanının seçilmesinde yazılım alanında çalışanları yönlendirecek bir bilgi literatürde mevcut değildir. Bu çalışmada literatürdeki bu boşluğun bir nebze de olsa azaltılması, yazılım kod muayenelerine ilişkin yeniden muayene kararının verilmesine yönelik farklı metodların performanslarının değerlendirilmesiyle amaçlanmaktadır.

Bu amaç çerçevesinde, Bölüm 2'de, literatürde konuya ilişkin bulunan bilgiler verilmekte, Bölüm 3'de ise, çalışma, hedefi, kapsamı ve adımları tanımlanarak, ve elde edilen temel sonuçlar sunularak açıklanmaktadır. Bölüm 4, çalışma sonucunda varılan noktayı çalışmaya ilişkin önemli hususları vurgulayarak özetlemektedir.

2. Çalışmaya İlişkin Bilgiler

Yazılım sektörünün yazılım muayeneleri ile tanışması Michael Fagan'ın 1970'lerin başında IBM'de gerçekleştirdiği muayene metodolojisi geliştirme çabaları sonucunda olmuştur [1]. Yazılım muayenesinin ana hedefi yazılım ürünündeki (kod veya doküman) hataların ürüne girmesinden hemen sonra tespit edip ortadan kaldırmaktır. Bunun sonucunda yazılım muayenesi, hataların yazılım yaşam döngüsünün sonraki fazlarına geçmesi sonucu harcanacak düzeltme maliyeti ve zamanından tasarruf edilmesini, ayrıca, yazılımın güvenilirliğinin ve bakım yapılabilirliğinin iyileşmesi sonucu yazılım kalitesinin artmasını sağlar. Yazılım muayenelerinin yazılıma ait kod için kullanımı, gereksinim spesifikasyonları, tasarım tanımları vb. ürünler için yapılan uygulamalarla karşılaştırıldığında daha yaygındır. Bu muayeneler sırasında karşılaşılan hatalar, kodun ilgili standartlara, gereksinime, veya tasarıma göre uygun olmamasından kaynaklanmaktadır. Koda ilişkin hatalar, genellikle, ciddiyet seviyesine göre majör ve minör olarak sınıflandırılmaktadır. Ciddiyet, bu kapsamda, kodda yer alan bir hatanın neden olduğu olumsuz etkilerin önemini ifade etmektedir. Ciddiyet seviyesi ne olursa olsun, tespit edilen hataların gerekli düzeltmelerin yapılması sonucu yazılımdan kaldırılması gerekir. Bunun gerçekleştirilmesi ise yazılım geliştirme aşamaları ilerledikçe artan düzeltme maliyetlerinin harcanmasını beraberinde getirir.

Yazılım muayenesine tabi bir ürünün içerdiği hata miktarının tahmin edilmesinde Hata Miktarı Tahmin Teknikleri (HMTT) (Defect Content Estimation Techniques - DCET) ekseriyetle kullanılmaktadır. Bu bilgi muayene ekibi tarafından yazılım muayenesi sonrası üründe kalan hata miktarının öngörülmesi için (ilk muayene öncesi veya ürünün yaratılması sonrası) toplam tahmini hata sayısından muayene sırasında bulunan hata sayısını çıkararak kullanılabilir. Kalan hata sayısının bilinmesi veya en azından tahmin edilmesi ikinci bir muayene, yani yeniden muayene, kararının sağlıklı ve veriye dayalı bir

şekilde verilebilmesi için önem arzetmektedir. Literatürde yer alan Hata Miktarı Tahmin Teknikleri nesnel (objektif) ve öznel (subjektif) olmak üzere ikiye ayrılmaktadır [2], [3], [4], [5], [6], [7] . Öznel teknikler tahminin kolay yapılmasını sağlamasına rağmen insan hükmüne dayandığı için yanıltıcı olabilmektedir. Bu güçlük önemli ölçüde verinin özenle toplanmasını gerektirdiği için maliyeti daha fazla olan nesnel tekniklerle aşılabilir. Nesnel teknikler ise kendi içerisinde eğri yerleştirme (curve fitting) modelleri ve yakalama-yeniden yakalama (capture-recapture) modelleri olarak sınıflandırılabilir. Eğri yerleştirme modelleri hata verilerinin grafiksel olarak işaretlenmesi, işaretlenen verilerin üzerine bir eğrinin yerleştirilmesi ve oluşturulan eğrinin formülasyonunun kullanımı sonucu üründeki toplam hata sayısı tahmininin üretilmesini sağlar. Yakalama-yeniden yakalama modelleri yazılım mühendisliğine biyoloji alanından uyarlanmıştır. Bu modeller biyolojide bir hayvan popülasyonundaki canlı sayısının istatistiksel olarak tahmin edilmesi amacıyla geliştirilmiştir. Yakalama-yeniden yakalama modellerinin yazılım muayenesi için uygulanması hataların hayvanlar olarak ve yakalama olaylarının ise muayeneciler olarak düşünülmesiyle mümkün olmaktadır. Bu benzerliğin kurulması ve kullanılmasıyla muayene edilen bir yazılım ürünündeki hataların tahmin edilmesi aynı belirli bir alandaki hayvan popülasyonunun boyutunu tahmin eder gibi gerçekleştirilebilmektedir. Bu modellerin yazılım muayeneleri için kullanımının arkasında şu mantık bulunmaktadır; farklı muayeneciler tarafından bulunan hataların çoğu birbiriyle aynı ise kalan hatalar azdır, aksine buldukları hatalar ayrık ise üründe daha tespit edilemeyen birçok hata vardır. Bu benzetim ve çıkarım düzeninin takip edilmesi sonucu, yazılım ürünündeki hataların tahmini için literatürde dört farklı kapalı popülasyon yakalama-yeniden yakalama modeli kullanılmıştır. Bu modeller varsayımları ve tahmin edicileri ile Tablo 1'de listelenmiştir.

Tablo 1. Yazılım Muayenesi için Kullanılan Yakalama-Yeniden Yakalama Modelleri ve Tahmin Edicileri

Model	Varsayımlar	Tahmin Edici
M0	Hataların bulunma olasılığı eşittir. Muayenecilerin hata tespit yetenekleri eşittir.	Maksimum Olabilirlik Tahmin Edicisi [8]
Mt	Hataların bulunma olasılığı eşittir. Muayenecilerin hata tespit yetenekleri farklıdır.	Maksimum Olabilirlik Tahmin Edicisi [8] Chao'nun Zaman Tahmin Edicisi [9]
Mh	Hataların bulunma olasılığı farklıdır. Muayenecilerin hata tespit yetenekleri eşittir.	Jack-knife Tahmin Edicisi [10] Chao'nun Heterojenlik Tahmin Edicisi [11]
Mth	Hataların bulunma olasılığı farklıdır. Muayenecilerin hata tespit yetenekleri farklıdır.	Chao'nun Heterojenlik-Zaman Tahmin Edicisi [12]

Yazılım muayenesi tamamlandıktan, yani bulunan hatalar düzeltildikten ve doğrulandıktan sonra, muayeneye giren ürünün yazılım yaşam döngüsündeki bir sonraki aşamaya geçirilmesi veya ilk muayenede gözden kaçan hataların bulunması amacıyla yeniden muayene edilmesi kararı proje yöneticisi tarafından verilmelidir. Bu ikinci seçeneğin tercih edilmesi durumu yeniden muayene olarak adlandırılır. Daha açık bil şekilde ifade etmek gerekirse, yeniden muayene, yazılım ürününü ihtiva

ettiği hata sayısını daha iyi bir seviyeye indirmek amacıyla yeni baştan muayene etmektir. Bunun sonucunda, yazılım geliştirme projesi yazılım muaynelerinin sunduğu faydalardan daha fazla istifade etmiş olur. Fakat, muayenenin tekrar edilmesi aynı zamanda, muayene faaliyetleri için ek kaynak ve zaman tahsis edilmesi anlamına gelmektedir. Literatürde yeniden muayene kararının alınmasına yönelik gelişigüzel metodlardan geçmiş veriye dayalı metodlara ve nesnel metodlara kadar birçok yöntem önerilmiştir. Bunlardan en sağlıklı bilgiyi sağlayan nesnel karar verme metodları aşağıdakilerdir:

1. İlk muayenenin etkinliğine (bulunan hata sayısının toplam hata sayısına oranı) göre karar verme [13]
2. İlk muayene sonucundaki hata yoğunluğuna (birim ürün boyutu başına düşen hata sayısı) göre karar verme [14]
3. Yeniden muayene sonucunda ulaşılacak hata yoğunluğuna göre karar verme [15]
4. Yeniden muayene sonucunda ulaşılacak etkinliğe göre karar verme [16]
5. Yeniden muayene sonucunda elde edilecek net faydaya göre karar verme [15]

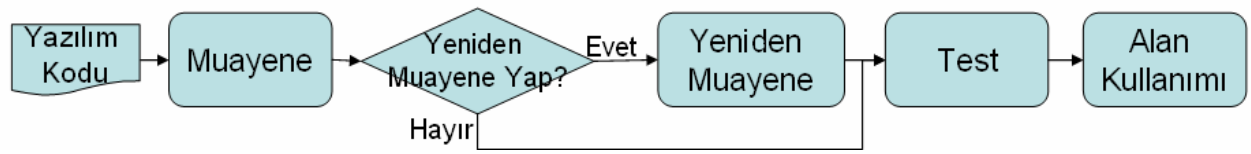
Yukarıdaki metodlardan bazıları için yeniden muayene yapıldığı takdirde bulunacak yeni hataların miktarının öngörülmesi gerekmektedir. Bunun için Hata Tespit Yeteneği Tahmin Teknikleri (HTYTT) (Defect Detection Capability Estimation Techniques-DDCET) kullanılmakta, ve dolayısıyla muayene ekibi yeniden muayene kararının yeniden muayene yapılması durumunda elde edilecek tahmini getiriye dayalı olarak sonuçlandırabilmektedir. HTYTT teknikleri yukarıda açıklanan HMTT tekniklerine göre daha yeni olduğu için, bu tekniklere ilişkin daha az çalışma bulunmaktadır. Literatürde karşılaşılan HTYTT teknikleri şunlardır; (i) İyimser Doğrusal Model (Optimistic Linear Model-OLM), (ii) Geliştirilmiş Doğrusal Model (Improved Linear Model-ILM), ve (iii) Güvenilirlik Artışı Modeli (Reliability Growth Model-RGM). Bu çalışmadakine benzer şekilde literatürde başka çalışmalarda da simülasyon yöntemi uygulanmıştır [13], [17].

3. Çalışmanın Tanımı, İcrası ve Sonuçları

Bölüm 2'de de belirtildiği gibi, literatür, yeniden muayene kararının alınmasını yapılan ilk muayene sırasında elde edilen verilerden yararlanarak yönlendiren nesnel metodları yazılım mühendisliğinin kullanımına sunmaktadır. Yeniden muayene kararı yazılım geliştirme projelerinde karşılaşılan birçok teknik ve yönetimsel kararların iyi bir örneğidir. Yazılım geliştirme projeleri yürüttükleri tüm faaliyetleri ve verdikleri tüm kararları projelerin üç temel hedefini, yani, maliyet, takvim, ve kalite hedeflerini dikkate alarak yapılandırmaktadır. Bu yüzden aynı diğer kararlar gibi yeniden muayene kararı da bu kararın yazılım projesi hedeflerine etkilerinin ortaya konması sonucu verilmelidir. Ancak, literatürde, yeniden muayene kararının sonuçlandırılmasına yönelik metodlar önerilmiş olmasına rağmen, farklı metodların yazılım geliştirme projesinin maliyetine, takvimine ve kalitesine yaptıkları etkilerin açığa çıkarılması sonucu değerlendirilmesi ve karşılaştırılmasına ilişkin bir çalışma bulunmamaktadır. Dolayısıyla, bu çalışmada literatürdeki bu eksiklik ele alınmaktadır.

Çalışma, Yazılım Yetenek Olgunluk Modeli'ne (Software Capability Maturity Model-SW CMM) [18] 3. seviyede olan bir organizasyonda kod muayenesinin tamamlanmasına mütakiben verilmesi gereken yeniden muayene kararını dikkate almaktadır. Çalışma kapsamında, ilk muayenenin hemen sonrasında alınan kararın sonuçlarının elde edilmesi için karar öncesi ve sonrasında yazılım koduna ilişkin yürütülen faaliyetleri simgeleyen bir Monte-Carlo simülasyon modeli oluşturulmuş ve çalıştırılmıştır.

Bu model, kodun ilk muayeneye girmesinden hedef oratamda (alanda) kullanılmasına kadar olan süreci kapsamaktadır (Bakınız Şekil 1). Simülasyon modeli, temelde, yazılım kodundaki hatların yakalanabileceği noktaları temsil etmektedir. Bu yüzden, yazılım hataları modelin ana unsuru olarak tayin edilmiştir.



Şekil 1. Muayene-Yeniden Muayene Sürecinin Akışı

Çalışmanın amacı kod muayenesine ilişkin farklı yeniden muayene kararını verme metodlarının değerlendirilmesidir. Bu amaç doğrultusunda, çalışma, herbir karar verme metodu için ortaya çıkan maliyeti, kullanıcılar tarafından karşılaşılan hata seviyesini ve projenin planlanan bitiş tarihine göre takvim gecikmesini ortaya koymaya çalışmaktadır. Bunlardan, maliyetin ve hata seviyesinin hesaplanabilmesi için farklkı hata tespit faaliyetleri (muayene, yeniden muayene, test, alan kullanımı) sonucunda yakalanan hata sayılarının bilinmesi gereklidir. Takvim gecikmesi ise projenin başlagıcında öngörülmediği için planlanmamış yeniden muayene faaliyetinin yapılmasından kaynaklanan zaman sapmasını ifade etmektedir. Tüm bunlar göz önüne alındığında simülasyon modelinin hataların maruz kaldıkları olaylara göre yapılandırılması mantıklı gözükmektedir. Bunu başarabilmek için, (i) belirli bir hatanın tespit edildiği faaliyet, (ii) ilk muayenenin başlangıcında kodun ihtiva ettiği hata sayısı ve içeriği, (iii) tespit edilen bir hatanın dzüeltme maaliyeti, ve (iv) yeniden muayenenin maliyeti, Şekil 1 kapsamında ve literatürdeki ilgili verilerden yararlanılarak modele dahil edilmiştir.
Çalışma süresince çalışmanın yukarıda belirtilen amacı ve kapsamı doğrultusunda oluşturulan simülasyon modeli kullanılarak aşağıda kısa açıklamaları verilen adımlar gerçekleştirilmiştir.
1. Simülasyon modelinin tanımladığı yazılım yaşam döngüsü sonucunda oluşan maliyetin, takvim gecikmesinin, ve hata muhteviyatının uygulalan farklı yeniden muayene karar metodları için karşılaştırılabilmesi ve değerlendirilebilmesi için aşağıdaki ölçüler belirlenmiştir:

- **Toplam Maliyet:** Hata düzeltme ve (yapılması durumunda) yeniden muayene maliyeti (harcanan iş gücü, yani adam-saat cinsinden).
- **Takvim Gecikmesi:** Yeniden muayenin yapıldığı simülasyon replikasyonlarının yüzdesi. Bu replikasyonlar ek bir muayene yapılmasından dolayı proje tamamlanma zamanının olumsuz olarak etkilendiği durumları temsil göstermektedir.
- **Majör Alan Hataları:** Kullanıcı tarafından karşılaşılan majör hataların sayısı. Bu ölçü kullanıcı tarafından algılanan yazılım kalitesinin bir göstergesi olarak kabul edilmektedir.

2. Bu adımda çalışmanın sonuçlarını etkilemesi muhtemel faktörler ortaya konmuştur. Bunu takiben, simülasyon modelindeki değişken koşulları göstermeleri amacıyla belirlenen faktörlerin farklı seviyeleri seçilmiştir. Çalışmada dikkate alınan faktörler şunlardır; (i) bir hatanın majör olma olasılığı, (ii) bir hatanın zor bulunan türden olma olasılığı, (iii) zor bulunan ve kolay bulunan türden hataların muayene sırasında tespit edilme olasılıkları, (iv) muayeneci sayısı, (v) yeniden muayenedeki muayenecilerin sayısı, (vi) muayenecinin hata yakalama yeteneği, ve (vii) bir hatanın test sırasında tespit edilme olasılığı.

3. Adım 2'de belirlenen faktörleri ve faktör seviyeleri kullanılarak, simülasyon modelinin konu edindiği süreç kapsamında karşılaşılan farklı koşulların sıralanması amacıyla bir Taguchi deney tasarımı (orthogonal array, fractional factorial design) oluşturulmuştur.

4. Çalışmanın ana öznesi yeniden muayene kararı verme metodları olduğu için, çalışma kapsmında karşılaştırması yapılacak kod muayenelerine yönelik yeniden muayeneye karar verme politikaları ortaya konmalıdır. Bu adımda, literatürdeki bu karara ilişkin metodlardan yola çıkılarak 47 tane yeniden muayeneye karar verme politikası belirlenmiştir.

5. Adım 4'te tayin edilen politikaların kodda muayene sonrası kalan hata sayısını tahmin etme amacıyla HMTT tekniklerini kullandığı için, bu adımda çalışmanın hedefleri doğrultusunda kullanılabilecek genel geçer bir HMTT seçilmiştir. Tüm HMTT tekniklerini kapsayan gerekli analizlerin yapılması sonucunda Yakalam-Yeniden Yakalama Jack-knife 2. Kademe Tahmin Edicisi ihtiyaç duyulan tahmin edici olarak belirlenmiştir.

6. Çalışmada dikkate alınan bazı karar metodlarının muhtemel bir yeniden muayene sırasında bulunabilecek hata sayısı bilgisini gerektirmesinden dolayı, uygun bir HTYTT tekniği seçilmelidir. Gerçekleştirilen detaylı karşılaştırmaların sonucunda, çalışmada kullanılacak HTYTT olarak Geliştirilmiş Doğrusal Model tespit edilmiştir.

7. Seçilen tüm politikalar için toplam maliyet, takvim gecikmesi ve majör alan hataları ölçüleri simülasyon modeli deney tasarımındaki tüm durumlar için çalıştırılarak hesaplanmıştır.

8. Çalışmada içerilen faktörlerin yazılım projelerin üç ana hedefini temsil etmek üzere belirlenen ölçüler üzerindeki etkileri varyans analizi (Analysis of Variance-ANOVA) yöntemiyle incelenmiştir. Bu inceleme sonucu elde edilen çıkarımlar Tablo 2'de özetlenmiştir.

Tablo 2. Faktör Seviyelerinin Maliyet, Takvim ve Hata Ölçülerine Etkisi

	Toplam Maliyet	Takvim Gecikmesi	Majör Alan Hataları
Test Etkinliğinin Artması	+	0	+
Muayeneci Yeteneğinin Artması	+	+	+
Muayeneci Sayısının Artması	+	+	+
Kolay ve Zor Hataların Bulunma Olasılıklarının Yakın Olması	+	-	+
Zor Bulunan Hataların Azalması	+	0	+
Majör Hataların Azalması	+	+	+
Yeniden Muayenedeki Muayenecilerin Sayısının Artması	0	-	+

(+ : ölçünün azalması sonucu olumlu etkiler, - : ölçünün artması sonucu olumsuz etkiler, 0 : etkisi yok)

9. Son adım olarak, simülasyon replikasyonlarıyla elde edilen maliyet, takvim, ve kalite bakış açılarına ilişkin ölçülerin karşılaştırılması vasıtasıyla yeniden muayeneye karar verme politikalarının değerlendirilmesi gerçekleştirilmiştir. Bu değerlendirme sırasında, yazılım geliştirme projeleri arasındaki hedef öncelikleri açısından çeşitliliği dikkate almak amacıyla maliyet, takvim, ve kalite boyutları için farklı ağırlık kombinasyonları belirlenmiştir. Bu sayede elde edilen yazılım projesi profilerinin herbiri için hangi politikanın daha uygun sonuçlar verdiği ortaya çıkarılmıştır. Bu adımın tamamlanmasıyla elde edilen temel sonuçlar aşağıda sunulmaktadır:

- 8 yeniden muayene politikası en az bir profilde en iyi politika olmuştur. Bunların ortak özelliği muayene etkinliğine ve ürün hata yoğunluğuna ilişkin eşik değerlerinin ideale yakın olmasıdır.
- Yeniden muayenenin etkinliğine dayalı karar veren politikalar genel olarak yazılım projelerinin hedeflerinin yakalanmasına yönelik beklentilerin karşılanması açısından daha başarılıdır.
- Çalışma sonucunda üstünlüğü görülen politikaların çoğu majör hatalara ilişkin verilere dayalı olarak yeniden muayene kararını sonuçlandırmaktadır. Bunun temelinde majör hataların zamanında ortadan kaldırılmamasından dolayı neden oldukları önemli maliyet ve kalite sorunları yatmaktadır.
- Muhtemel yeniden muayene sonarasında bulunan majör hataların yüzdesini yüksek sayılabilecek bir eşik seviyesine göre (75%) karşılaştıran 39 numaralı politika yeniden muayene kararının uygun şekilde alınması açısından ön plana çıkmaktadır.
- 'Hiçbir Zaman Yeniden Muayane Yapma' ve 'Her Zaman Yeniden Muayane Yap' sabit kararlarının uygulanması maliyet, takvim, ve kalite açılarından en uygun sonuçları vermemektedir.

4. Sonuç

Yazılım geliştirme organizasyonları uygun görüldüğü takdirde gerçekleştirecekleri yeniden muayeneler ile yazılım muayenesinin faydalarından daha fazla istifade edebilirler. Ancak, bu ikinci muayene ilave kaynak ayrılmasını gerektirdiği için her muayeneyi tekrar etmek mümkün değildir. Dolayısıyla, yeniden muayene, belirlenen kriterler sağlandığı zaman yapılmalıdır. 'Yeniden Muayene Yap' veya 'Yeniden Muayene Yapma' seçenekleri arasında tercih yapmayı konu edinen karar 'Yeniden Muayene Kararı' olarak adlandırılır. Literatür yeniden muayene kararının sonuçlandırılmasına yönelik çeşitli nesnel metodlar sunmaktadır. Fakat, yazılım alanında çalışanlar farklı yeniden muayaneye karar verme metodlarından uygun olanını seçme konusunda kendilerini yönlendiren bir kaynak bulamamaktadırlar. Bunun farkına varılması üzerine gerçekleştirilen bu çalışmada, yazılım geliştiren projelere yeniden muayene politikası belirleme konusunda rehberlik etmek amacıyla, farklı politikaların proje hedeflerine etkileri irdelenerek elde edilen değerli sonuçlar sunulmuştur. Bunun yanında, bu hedeflerle ilgili yazılım muayenesine ilişkin faktörler, hedeflerin yakalanmasına olan etkilerini anlamak amacıyla analiz edilmiştir. İleride bu çalışmanın geliştirilmesi amacıyla şu işlemler yapılabilir; (i) çalışmanın gerçek bir yazılım mühendisliği ortamından elde edilen verilerle tekrarlanması, (ii) çalışmanın Yazılım Yetenek Olgunluk Modeli'nin daha düşük ve daha yüksek seviyeleri için adapte edilmesi, (iii) yeniden muayeneye karar verme politikaları karşılaştırılırken gelişmiş çok amaçlı karar verme yöntemlerinin kullanılması, (iv) test sürecini detaylı olarak ele alan daha kapsamlı bir simülasyon modelinin kurulması.

5. Referanslar

[1]. Fagan, M.E. (1976). Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, 15(3), 182-211.

[2]. Gilb, T., Graham, D., 1993. *Software Inspection.* Addison-Wesley Longman Limited.

[3]. Strauss, S.H., Ebenau, R.G., 1993. *Software Inspection Process.* Mcgraw Hill Systems Design and Implementation Series.

[4]. Radice, R.A., 2002. *High Quality Low Cost Software Inspections.* Paradoxicon Publishing.

[5]. Miller, J., 1999. "Estimating the Number of Remaining Defects after Inspection". *Software Testing, Verification, and Reliability*, 9(3), 167-189.

[6]. Briand, L., El Amam, K., Freimut, B., Laitenberger, O., 2000. "A Comprehensive Evaluation of Capture-Recapture Models for Estimating Software Defect Content". *IEEE Transactions on Software Engineering*, 26(6), 518-540.

[7]. Petersson, H., Thelin, T., Runeson, P., Wohlin, C., 2004. "Capture–Recapture in Software Inspections after 10 Years Research—Theory, Evaluation and Application". *Journal of Systems and Software*, 72(2), 249-264.

[8]. Otis, D.L., Burnham, K.P., White, G.C., & Anderson, D.R. (1978). Statistical Inference from Capture Data on Closed Animal Populations. *Wildlife Monographs*, 62, 1–135.

[9]. Chao, A. (1989). Estimating Population Size for Sparse Data in Capture-Recapture Experiments. *Biometrics*, 45(2), 427-438.

[10]. Burnham, K.P., & Overton, W.S. (1978). Estimation of the Size of a Closed Population when Capture Probabilities Vary among Animals. *Biometrica*, 65(3), 625-633.

[11]. Chao, A. (1987). Estimating Population Size for Capture-Recapture Data with Unequal Catchability. *Biometrics*, 43(4), 783-791.

[12]. Chao, A., Lee, S.M., & Jeng, S.L. (1992). Estimating Population Size for Capture-Recapture Data When Capture Probabilities Vary by Time and Individual Animal. *Biometrics*, 48(1), 201-216.

[13]. El Amam, & K., Laitenberger, O. (2001). Evaluating Capture-Recapture Models with Two Inspectors. *IEEE Transactions on Software Engineering*, 27(9), 851-864.

[14]. El Amam, K., Laitenberger, & O., Harbich, T. (2000). The Application of Subjective Estimates of Effectiveness to Controlling Software Inspections. *Journal of Systems and Software*, 54(2), 119-136.

[15]. Biffl, S., & Halling, M. (2001). In: M. Williams, Investigating Reinspection Decision Accuracy Regarding Product-Quality and Cost-Benefit Estimates. Proceedings of Twenty-Fifth Annual International Computer Software and Applications Conference (sayfa 87-96). Los Alamitos: IEEE Computer Society.

[16]. Biffl, S., Gutjahr, & W.J. (2002). Using a Reliability Growth Model to Control Software Inspection. *Empirical Software Engineering: An International Journal*, 7(3), 257-284.

[17]. Thelin, T., Runeson, P., 1999. Capture–recapture estimations for perspective-based reading—a simulated experiment. In: Proceedings of the International Conference on Product Focused Software Process Improvement, 182–200.

[18]. Paulk, M.C., Curtis, B., Chrissis, M.B., & Weber, C.V. (1993). *Capability Maturity Model for Software, Version 1.1* (1-82). Pittsburgh, PA: Software Engineering Inst., Carnegie Mellon University.