T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ÇEVİK SÜREÇ İLE MODEL YÖNELİMLİ YAZILIM GELİŞTİRME

GÜRKAN ALPASLAN

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI

DANIŞMAN
PROF. DR. OYA KALIPSIZ

İSTANBUL, 2015

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

ÇEVİK SÜREÇ İLE MODEL YÖNELİMLİ YAZILIM GELİŞTİRME

Gürkan ALPASLAN tarafından hazırlanan tez çalışması 26.06.2015 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Prof. Dr. Oya KALIPSIZ

Yıldız Teknik Üniversitesi

Jüri Üyeleri

Prof. Dr. Oya KALIPSIZ

Yıldız Teknik Üniversitesi _____

Yrd. Doç. Dr. Zeynep ORMAN

İstanbul Üniversitesi _____

Yrd. Doç. Dr. Mehmet AKTAŞ

Yıldız Teknik Üniversitesi _____

ÖNSÖZ

Bu tezin hazırlanmasında bilgi ve deneyimleri ile bana her türlü katkıyı sunan sayın hocam Prof. Dr. Oya Kalıpsız' a teşekkürlerimi bir borç bilirim.

Bu tezi, maddi ve manevi hiçbir fedakârlıktan kaçınmayıp desteklerini hiçbir zaman esirgemeyen sevgili aileme ithaf ediyorum.

Haziran, 2015

Gürkan ALPASLAN

İÇİNDEKİLER

Sayfa

KISALTMA LİSTESİ

CSS	Cascading Style Sheets
HTML	Hyper Text Markup Language
MDD	Model Tabanlı Yazılım Geliştirme
MTM	Model Tabanlı Mimari
OMG	Object Management Group
SLAP	System-level Agile Process
SRS	Sistem Gereksinim Raporu
UML	Unified Modeling Language

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

ÖZET

ÇEVİK SÜREÇ İLE MODEL YÖNELİMLİ YAZILIM GELİŞTİRME

Gürkan ALPASLAN

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Tez Danışmanı: Prof. Dr. Oya KALIPSIZ

Çevik geliştirme yöntemi, klasik geliştirme yönteminin dezavantajlarını gidermek üzerine ortaya çıkmış bir yöntemdir. Çevik geliştirme yöntemini geliştirerek model tabanlı yapı ile birleştirmenin, yazılım başarısını arttıracağı düşünülmektedir. Bu amaç ile literatürde son yıllarda bu iki yöntemi birleştiren birkaç yöntem önerilmiştir. Bu tez çalışmasında çevik süreç ile model tabanlı geliştirme yöntemleri incelenmiş; web uygulamalarında kullanılmak üzere literatüre yeni bir yaklaşım kazandırılmıştır. Bu yaklaşım, üniversitemizin yazılım mühendisliği dersinde öğrenci projeleri üzerinde, öğrenci grupları ile birlikte çalışılarak denenmiş; elde edilen sonuçlar değerlendirilip, uygulanabilirliği ve avantajları sonuçlandırılmıştır.

Anahtar Kelimeler: Çevik modelleme, Model tabanlı yazılım geliştirme, Çevik süreç ile model tabanlı yazılım geliştirme, Mockup tabanlı geliştirme

ABSTRACT

MODEL DRIVEN SOFTWARE DEVELOPMENT WITH AGILE PROCESSES

Gürkan ALPASLAN

Department of Computer Engineering

MSc. Thesis

Adviser: Prof. Dr. Oya KALIPSIZ

Agile development is a method that is produced in order to eliminate the disadvantages of the classical development method. Developing the agile method and combining it with the model-driven method is thought to increase the software achievement. For this purpose, some methodologies combining these two methods are suggested. In this thesis, the agile model-driven methodologies in literature are explored and a new approach for using the web applications is developed. This approach is implemented with the student projects in ''Software Engineering'' class in our university and the results are evaluated.

Keywords: Agile modeling, model-driven software development, agile model-driven software development, mockup-driven development

BÖLÜM 1

GİRİŞ

1.1 Literatür Özeti

Bir yazılım projesinin, başlangıç aşamasından çalıştırılabilir kod haline gelene kadar olan süre içinde etkin bir proje yönetimini sağlamak, projeye uygun ve başarılı sonuçlar üreten yöntemler kullanmak yazılım mühendisliğinin alanıdır. İlk dönemler yazılım projeleri klasik yöntem olarak da adlandırılan "Şelale Yöntemi" ile geliştirilmekteydi. Şelale yöntemi uzun analiz ve tasarım süreçlerini içeren, projenin başlangıç halinin değiştirilmeyeceği göz önüne alınarak uygulanan bir yöntemdir. Ancak, müşteri istekleri stabil olmamakta, yazılım gereksinimleri, proje geliştirilmesi sırasında değişebilmektedir. Bu sebeple, başarılı sonuçlanan yazılım projeleri azlığı dikkat çekmiştir. Yazılım geliştirilmesine esneklik katmak için tekrarlı yazılım geliştirme yöntemi kullanılmaya başlanmıştır. Tekrarlı yazılım geliştirme, projenin parçalar halinde geliştirilmesini hedeflemektedir. Her tekrarda, projenin bir miktarı çalıştırılabilir kod halinde ortaya çıkmaktadır. Çevik yazılım geliştirme yöntemi de bir tekrarlı yazılım geliştirme yöntemidir. Çevik süreç, çevik prensipler, pratikler ve metodlar bütünüdür.

Projenin izleyeceği yol kadar, projenin temel öğeleri de kaliteli ve esnek yazılım geliştirilmesinde etkilidir. Nesneye dayalı yazılım geliştirme, yazılım elemanlarını nesne olarak tanımlayarak kodlama yapılmasını öngörür. Bir başka yöntem de model tabanlı (model yönelimli) yazılım geliştirmedir. Model tabanlı yazılım geliştirmede temel öncelik modellerdir. "Herşey modeldir" anlayışı benimsenir ve modellemeler yapılmadan önce kod üretimine geçilmez [1]. Modeller üzerinde çalışmak, yazılımın daha üst bir soyutlama düzeyinde geliştirilmesini sağlar.

Yakın zamanda yapılan çalışmalarda, model tabanlı yöntem ile çevik süreç prensiplerini birleştirmenin, iki sürecinde avantajlı yönlerinin alınması hususunda faydalı olacağı belirtilmiştir. Bu amaçla, iki süreci birleştiren metodlar üretilmiştir. Literatürde bu iki süreci birleştiren ilk metod, model tabanlı *çevik süreç üst seviye yaşam döngüsüdür* [2], [3]. Daha sonrasında, küçük ölçekli projelerde kullanılmak üzere tasarlanan *Hybrid* model tabanlı süreç [4] tasarlanmıştır. Askeri ve yüksek güvenlik arz eden projeler için *Sage* geliştirme yöntemi [5] de iki süreci birleştiren bir yöntemdir. Literatürdeki bir başka çalışma, Motorola şirketi tarafından kendi projeleri için geliştirilen *Slam-MDD* yöntemidir [6].

1.2 Tezin Amacı

Çevik süreç ve model tabanlı süreç incelendiğinde, ikisi de çeşitli avantaj ve dezavantajlar içermektedir. Çevik süreç, dökümantasyonu ikinci plana atar, çalıştırılabilir yazılım üretmeyi ön planda tutar [7]. Model tabanlı yazılım geliştirme ise, bir dökümantasyon kaynağı olan modellemeleri ön planda tutar. Tezin amacı, bu iki yöntemin birleştirilmesinin incelenmesidir. Literatürde, bu iki yöntemi birleştiren yöntemler olmakla birlikte, web uygulamalarında kullanılmak üzere özelleştirilmiş bir çevik model tabanlı süreç bulunmamaktadır. Tezin bir diğer amacı, bu iki süreci birleştiren, web uygulamalarında uygulanmak üzere özelleştirilmiş, kendi yaklaşımımızı önermek; ve bu yaklaşımı gerçek projeler üzerinde uygulayarak değerlendirmektir.

1.3 Hipotez

Bir yazılım projesinden beklentiler özetle aşağıda belirtilen temel kriterler ile belirtilebilir:

- Yazılım geliştirme hızı,
- Yazılım geliştirme maliyetinin azlığı,
- Müşteri gereksinimlerinin tam ve doğru olarak karşılanması,
- Müşteri memnuniyetinin arttırılması,
- Projenin geliştirilme safhasında değişime açık olması,

- Yazılımın üretiminden sonra, bakımının ve güncellenmesini kolaylığı

Yapılan çalışmaları incelediğimizde, çevik süreç ile model tabanlı geliştirmenin, yukarıda bahsedilen kriterlerde klasik metod ya da yalnız çevik süreç yöntemlerinden daha efektif olabileceği öngörülmektedir.

Bu öngörüyü destekleyen görüş ise, iki sürecin birleştirilmesinin iki sürecin avantajlı yönlerini alırken, kısmen dezavantajlarını kapatarak birbirlerini tamamlamasıdır. Örneğin, çevik süreç ile geliştirme, yazılımcıya tekrarlı bir ortamda ve yazılımı küçük parçalara bölerek geliştirme ortamı sunmaktadır. Bu sayede yazılım karmaşıklığı ve süreç içerisindeki değişikliklere geliştiricinin yazılımı entegre edebilmesi kolaylaşmaktadır. Geleneksel süreçteki uzun analiz ve tasarım süreçlerini de ortadan kaldırarak, buradaki vakit kaybını azaltmaktadır. Ayrıca müşteriyi de yazılım geliştirme sürecinin bir elemanı olarak görerek, gereksinim analizinin ve süreç boyunca yapılan işlerin doğruluğunu daha iyi bir ortamda sağlamaktadır. Ancak çevik süreç, dokümantasyona önem vermemektedir.

Model tabanlı geliştirmenin ise en önemli avantajları, programlama dillerine göre daha üst bir soyutlama düzeyinde geliştirme imkanı sağlaması ve model tabanlı geliştirme araçlarının tüm avantajlarından faydalanabilmesidir. Az sayıda ifade ile uzun kod parçalarının üretilmesi, modellerin kolay ve müşteri tarafından da okunabilirliğinin ve anlaşılabilirliğinin fazlalığı model tabanlı geliştirmenin bir diğer avantajlarıdır. Ayrıca, üretilen modeller birer dokümantasyon ürünü olarak ta kullanılabileceğinden, çevik süreçteki dokümantasyon eksiğini gidermektedir.

BÖLÜM 2

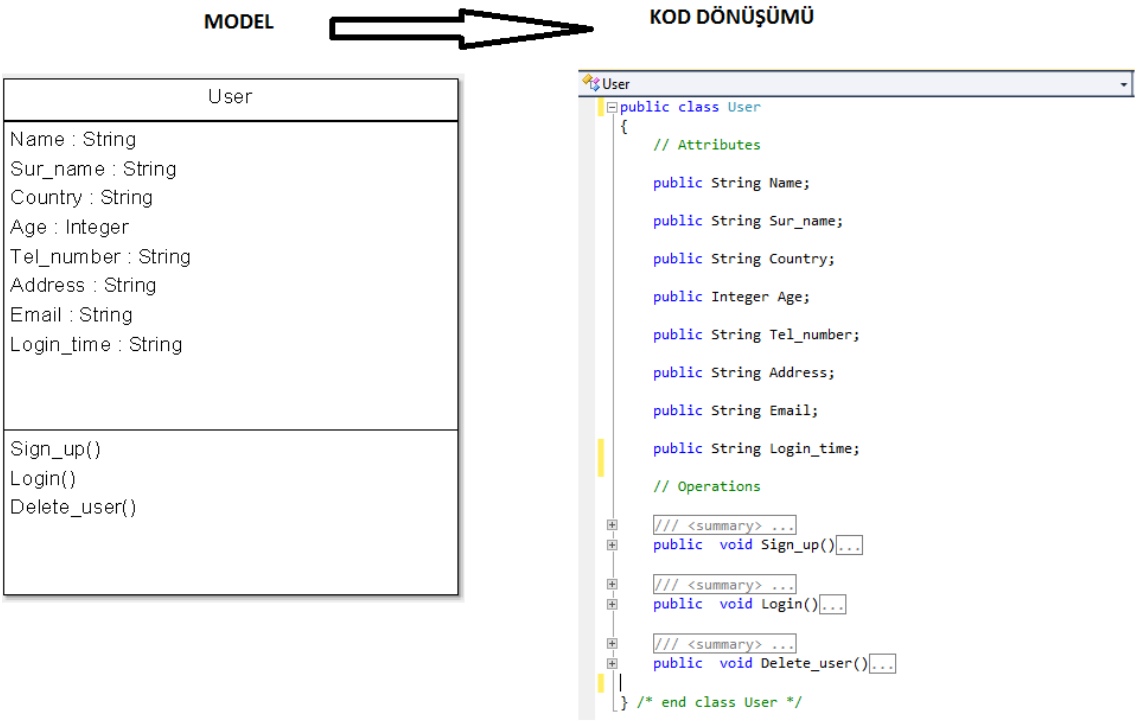
MODEL TABANLI YAZILIM GELİŞTİRME

Bu bölümde model tabanlı yazılım geliştirme metodu, model tabanlı geliştirmenin sağladığı avantajlar ve çevik süreçle uyumlu modelleme yapmamızı sağlayan çevik modelleme prensipleri anlatılmıştır.

2.1 Model Tabanlı Yazılım Geliştirme Temelleri

Yazılım geliştiriciler ve akademisyenlerin oluşturduğu ortak bir oluşum olan Object Management Group (OMG) tarafından, bir dizi standartlar bütünü olan Model Tabanlı Mimari (MTM) oluşturulmuştur. Model tabanlı yazılım geliştirme, MTM standartlarına uygun yazılım geliştirme metodudur. Model tabanlı yazılım geliştirmede, yazılım geliştirmenin ana gövdesini modeller oluşturmaktadır [8]. Model, bir sistemin soyutlanmış gösterimi olarak tanımlanır [9]. Yazılım geliştirme evresine baktığımızda, makine kodlarından, gelişmiş programlama dillerine doğru uzanan bir süreç vardır. Bu süreç boyunca, geliştirme mekanizmalarında soyutlama düzeyi arttırılarak, yazılım geliştirmenin karmaşıklığı azaltılmaktadır. Model tabanlı yazılım geliştirme, programlama dillerinin üzerinde yer almakta ve modeller ile çalışmak, yazılım geliştirmenin, daha üst bir soyutlama düzeyinde yapılmasını sağlamaktadır. Ayrıca modeller, platform bağımsız bir ortam sunmaktadır. Modeller, platform bağımsız ortamda geliştirilerek, sonrasında platform bağımlı ortama dönüştürülmesi hedeflenen yapılardır [10], [11]. Model tabanlı yazılım geliştirmede, geliştirici, öncelikli olarak model geliştirimini yapar, sonrasında üretilen modeller hedeflenen platforma uygun kaynak koda dönüştürülür.

Model tabanlı geliştirmede, modellemeler ile sistem varlıkları ve varlıklar arası ilişki gösterilebilir. Ayrıca bu modellemeler, sistemin kelime sözlüğünün üretilmesini de sağlayabilmektedir. Oluşturulan modeller, model tabanlı dönüşüm araçları sayesinde otomatik kod dönüşümü ile kaynak kodlara dönüştürülebilmektedir. Şekil 3.1'de ArgoUML aracı [12] kullanılarak oluşturulan modelden koda dönüşüm örneklendirilmiştir.



Şekil 2. 1 MDD aracı ile otomatik kod dönüşümü örneği

2.2 Model Tabanlı Yazılım Geliştirme Avantajları

Model tabanlı yazılım geliştirme avantajları [13], [14] genel anlamıyla şu başlıklar altında toplanabilir.

Daha hızlı yazılım geliştirme: Model tabanlı yazılım geliştirme, geleneksel programlama dillerinin üzerinde bir soyutlamada yer almaktadır ve modeller otomatik kod dönüşümü ile geleneksel programlama dillerine dönüştürülebilmektedirler. Yani model üzerinde tanımlanan her eleman, birkaç satır koda tekabül etmektedir. Bu

sayede, aynı sürede daha fazla fonksiyonu model üzerinde gösterilmesi sağlanmaktadır.

Maliyet yönünden kazanç: Bu avantajın iki temel sebebi vardır. Birincisi, ilk özellik olarak bahsettiğimiz daha hızlı yazılım geliştirme sayesinde, ürünün daha seri bir şekilde piyasaya ulaşmasını sağlamasıdır. İkinci sebep, MDD kullanımında daha az kaynak ve işgücü kullanımı sağlanmasıdır.

Artan kalite: MDD ile yazılım geliştiriminde, otomatik kod dönüşümünü destekleyen MDD araçları kullanılmaktadır. Bu araçlar, gelişen teknoloji ve bilgiye göre kendisini yenilemektedirler. Kullanacağımız en yeni araç, muhtemelen mevcut deneyimlerin birikimi olarak ortaya çıkacaktır ve iyi düzeyde kalite imkanı sunacaktır.

Düşük hata eğilimi: Bir yazılım geliştirimi aşamasında, test süreci oldukça maliyetli, deneyim isteyen ve zaman harcanan bir süreçtir. MDD ile kod yazım hataları azaltılabilmekte; sistem test ekibi, sistem kabul testlerine ve güvenlik açıklarına odaklanabilmektedir.

Güncel dokümantasyon: Model tabanlı yazılım geliştirme ile dokümantasyon için ayrı bir efor harcanmasına gerek yoktur. Çünkü modellerin kendisi, yazılım dokümantasyonu olarak kullanılabilmektedir.

Teknoloji yerine iş problemlerine odaklanmayı sağlaması: Model tabanlı geliştirmede, teknolojik problemler, MDD araçları tarafından sağlanmaktadır. Son teknolojiyi kullanmak için son versiyon bir MDD aracı edinmek yeterlidir. Bu sayede, takımın teknolojik problemler yerine, iş problemlerine odaklanması sağlanmaktadır.

Değişime uyum: Teknoloji hızlı bir şekilde ilerlemektedir ve bu ilerlemeyi takip etmek ve uyum göstermek model tabanlı araçlar sayesinde daha kolaydır. Ayrıca modeller üzerinde değişim yapmak, kod üzerinde değişim yapmaktan daha kolaydır.

Mimari ile uyum: Mimari prensiplere uygun yazılım tasarlamak ve geliştirmek, başarılı bir proje yönetimi için gereklidir. Model tabanlı geliştirme araçları, mimari tasarımı yaparken, oluşturulan kodun mimari prensiplere uygunluğunu da kontrol eder, ve genel prensiplere uygun olmayan tasarımın üretilmesine engel olur.

2.3 Çevik Modelleme

Çevik prensiplere uygun kriterlerde yapılan modellemeler çevik modelleme [15], [16], [17] olarak tanımlanır. Model tabanlı yazılım geliştirmede çevik süreç katkısı sağlamak için çevik modelleme kullanılır. Çevik modellemenin ana prensipleri aşağıda incelenmiştir.

Basitlik: Çevik modelleme yapılırken, mümkün olduğunca basit bir şekilde yapılması öngörülür. Modellemelerde aşırı detaylandırma, çevik sürece zarar vermektedir. Bu sebeple amacımızı yerine getirecek kadar modelleme detayına yer verilir.

Paydaşların katkısının arttırılması: Çevik modellemede, tüm paydaşlar bir bütün halinde çalışması gerekmektedir. Müşteri ile geliştiriciler arasında iletişim yoğun olması ve müşterinin isteklerinin tam olarak yerine getirilmesi hedeflenmektedir. Bu sebeple, paydaşlar ürünün ilerleyişi, kararların alınışı vs. konularda bilgi sahibi olması ve ortak kararlar alması önerilir.

Çoklu modelleme: Her bir model, sistemin farklı bir parçasını tanımlamaktadır. Bu sebeple tek bir model üzerinde tüm sistemi göstermek yerine, her bir alt sistem için alt modellemeler yapılarak, çoklu modellemeler ile karmaşığın azaltılması önerilir.

Hızlı geri bilgi akışı: Yapılan modellemelerin uygunluğu, gereksinimleri karşılayıp karşılamadığı gibi konularda anlık olarak hızlı bir şekilde geri dönüş alınması gerekmektir. Sistem geliştirme aşamasında, yapılan ilerlemeler için müşteri ve takım içerisinde etkileşimin kuvvetli olması ve onaylamanın hızlı bir şekilde alınması istenmektedir.

Öncelikli amaç olarak çalıştırılabilir yazılım: Çevik modellemede amaç, kapsamlı modellemeler yapmak değildir. Modellemeler, çalıştırılabilir yazılıma ulaşmak için kolaylaştırıcı ara etmenler olarak üretilir. Bu sebeple modellemeler, sonuç odaklı olmalı; çalıştırılabilir yazılıma kısa yoldan ulaştıracak şekilde tasarlanmalıdır.

Değişime uyum: Gereksinimler süreç içerisinde değişmektedir. Aynı şekilde proje paydaşları da süreç içerisinde değişebilmekte, yeni insanlar sürece katılabilmektedir. Bu konsepte yapılan çalışmalar değişime çabuk adapte olacak şekilde tasarlanmalıdır. Değişim durumunda, minumum efor ile değişiminin uygulanması hedeflenir.

Arttırımlı değişim: Önemli bir konsepte, projenin başlangıcında tüm detayları tasarlamak yerine, adım adım ilerleyerek değişimi gerçekleştirmektir.

Kaliteli tasarım: Çevik modellemelerde, basitlik ön planda olmasına rağmen, yapılan tasarımların belli bir kalite standartında olması gerekmektedir.

Geleceğe yatırım: Belli bir projeyi yürütürken ve tamamlandığında, sonraki çalışmaların da düşünülmesi gerekir. Bu sebeple bir sonraki çalışmanın daha az efor ile tamamlanabilmesi için kolaylaştırıcı etmenler, süreç içerisinde veya süreç bitiminde raporlanması önerilir.

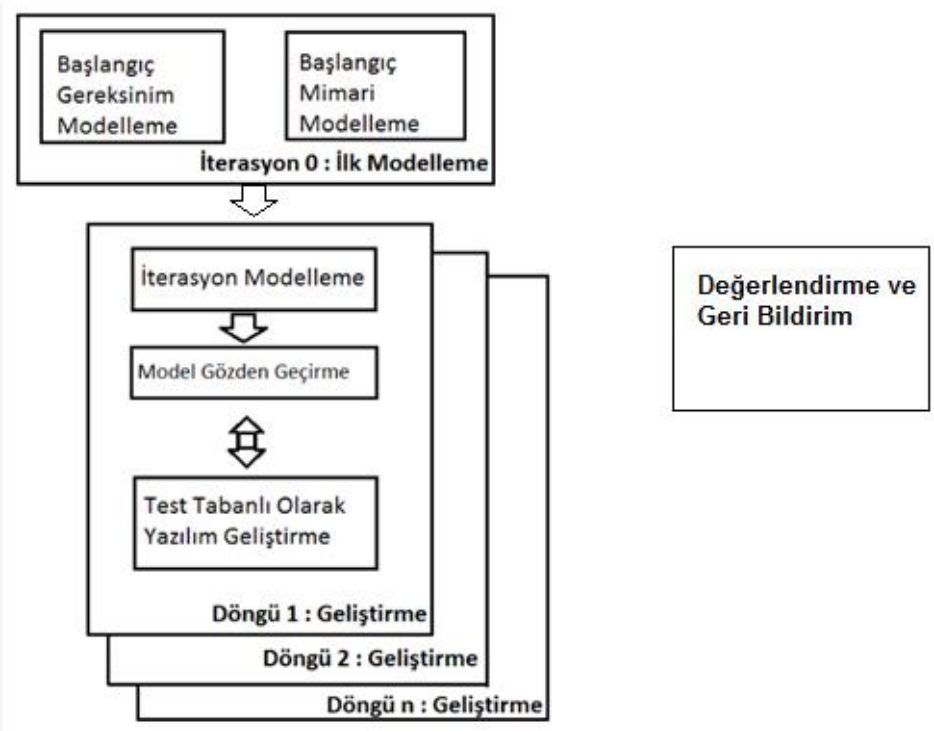
BÖLÜM 3

ÇEVİK SÜREÇ İLE MODEL TABANLI YAZILIM GELİŞTİRME

Bu bölümde literatürdeki model tabanlı geliştirme ile çevik süreci birleştiren yöntemler incelenmiş, yöntemlerin özelliklerinden ve süreç akışlarından bahsedilmiştir. *Üst seviye yaşam döngüsü* [3], *Sage* geliştirme yöntemi [5], *Hybrid* geliştirme yöntemi [4] ve *MDD-SLAP* yöntemi [6] olmak üzere dört yöntem bu bölümde incelenmiştir.

3.1 Üst Seviye Yaşam Döngüsü Tasarımı

Üst seviye yaşam döngüsü, iki yöntemi birleştiren literatürdeki ilk metoddur. Başlangıç ve geliştirme safhası olmak üzere iki temel süreçten oluşmaktadır. Başlangıç evresi, projenin kapsamının ve amacının belirlendiği; gereksinim analizi yapılıp, mimari yapının modellendiği aşamadır. Gereksinim modellemesi ve mimari modelleme olmak üzere iki alt aktivite içerir. Başlangıç aşaması, iterasyon 0 olarak da adlandırılmaktadır. Bu aşama, proje başlangıcında bir kere gerçeklenen aşamadır. Geliştirme aşaması, iterasyonların gerçekleştirildiği aşamadır. Proje planına göre başlangıç evresinde iterasyonlar belirlenir, bu iterasyonlar geliştirme aşamasında sırasıyla gerçeklenir. İterasyon modellemesi, model gözden geçirme ve test tabanlı yazılım gerçeklemesi olmak üzere üç alt aktivite içerir. Üst seviye yaşam döngüsünün süreç akışı Şekil 3.1'de gösterilmiştir [2], [3].



Şekil 3. 1 Üst Seviye Yaşam Döngüsü süreç akışı [3]

Başlangıç aşaması, projenin kapsamının, amacının belirlendiği; gereksinim analizi yapılıp, mimari yapının modellendiği aşamadır. Bu amaç için, üst seviye gereksinim modellemesi ve üst seviye mimari modelleme yapılır. Amaç, detaylı bir dokümantasyon çıkarmak değildir; bunun yerine, projenin stratejisini belirlemek temel alınır. Küçük ölçekli projeler için birkaç saat; büyük ölçekli projeler için ise, ortalama iki haftalık süre ayrılması önerilmektedir. Daha fazla süre ayırmak, aşırı modelleme tehlikesini getirebilmektedir; veya çok fazla detay ile model üzerinde çok fazla problem oluşmasına ve modelin esnekliğinde sorun yaratılmasına sebep olabilmektedir.

Başlangıç gereksinim modellemesi: Sistemin ne yapacağının belirlendiği aşamadır. Bu aşamada kullanım diyagramları, sistem kullanıcıların sistem ile etkileşimlerini göstermek için kullanılabilir. Kullanıcı arayüz tasarımı ve arayüz problemlerini belirlemek için kullanıcı arayüz modeli çizilebilmektedir.

Başlangıç mimari modelleme: Sistemin işleyişinin genel olarak tasarlandığı aşamadır. Mimari tasarımda UML diyagramları, varlıkları ve ilişkileri modellemek için yararlanılabilir. Başlangıç modellemelerinde genel amaç mümkün olduğunca basitlik ilkesini gerçekleştirebilmektir. Bu aşamalarda yapılan tasarımlar, geliştirme

safhalarında muhtemelen yenilenecektir. Bu aşamalar, geliştirmeye başlayabilmek için ön çalışma aşamalarıdır.

Geleneksel yazılım geliştiriciler için başlangıç modellemesi safhasını çevik sürece uygun olarak uygulamak zor olabilmektedir. Çünkü, klasik geliştirmeden gelen, proje başlangıcında geniş tasarımlar yapılması alışılmış bir durumdur. Ancak, çevik sürecin tam olarak uygulanabilmesi için bu süreçler genellikle geliştirme safhalarında tamamlanmaktadır.

Başlangıç aşamasında proje iterasyonlara ayrılırken, bu iterasyonların gerçeklenme sırası da önemlidir. İterasyon kuyruğu olarak isimlendirilen bir yapıda gerçeklenecek modüllerin önem ve öncelik sırasına göre düzenlenmesi önerilir.

Geliştirme aşaması, iterasyonların sırasıyla gerçeklendiği aşamadır. Başlangıç aşamasında oluşturulan iterasyon kuyruğundan ilk itarasyon seçilir ve bu iterasyon için iterasyon modelleme, model gözden geçirme ve test tabanlı yazılım geliştirme aşamaları uygulanır. İterasyonlar tamamlanana kadar, bu aşama tekrarlı olarak devam eder.

İterasyon modellemesi: Mevcut iterasyonda neler yapılacağının belirlendiği aşamadır. Her iterasyon için kendi içinde planlamalar yapılır. Eğer gerekiyor ise, o iterasyona özgü modellemeler tasarlanır. İterasyon içinde yapılacak işler, gereken efor ve önem derecelerine göre kuyruğa konulup, planlanabilir.

Model gözden geçirme: İterasyon modellemesi ve gerçekleştirme aşamaları arasında bir ara katmandır. Geliştirme aşamasında bir sorun çıktığında ekibin toplanarak, sorun için çözüm bulduğu veya model üzerinde değişiklik yapılması gerektiğinde ekibin modelin yeni haline şekil verdiği aşamadır. Tüm paydaşların etkin iletişim halinde olması önemlidir. Geliştirme aşamasında, herhangi bir anda, bu aşamaya geri dönülüp, paydaşlar toplanarak yeni kararlar alabilmektedirler.

Test tabanlı yazılım geliştirme: Üst seviye yaşam döngüsü yönteminde, yazılım geliştirilmesinin test tabanlı olarak yapılması önerilmektedir. Test tabanlı yazılım geliştirme , yazılım kodları üretimi öncesinde, test birimlerinin üretilmesini öngörür. Geliştirici, test üniteleri tamamlanmayan yazılımın geliştirilmesini reddeder. Bu yaklaşımda, geliştirilen kod parçaları, test ünitelerinden geçirildiğinde başarılı, başarısız

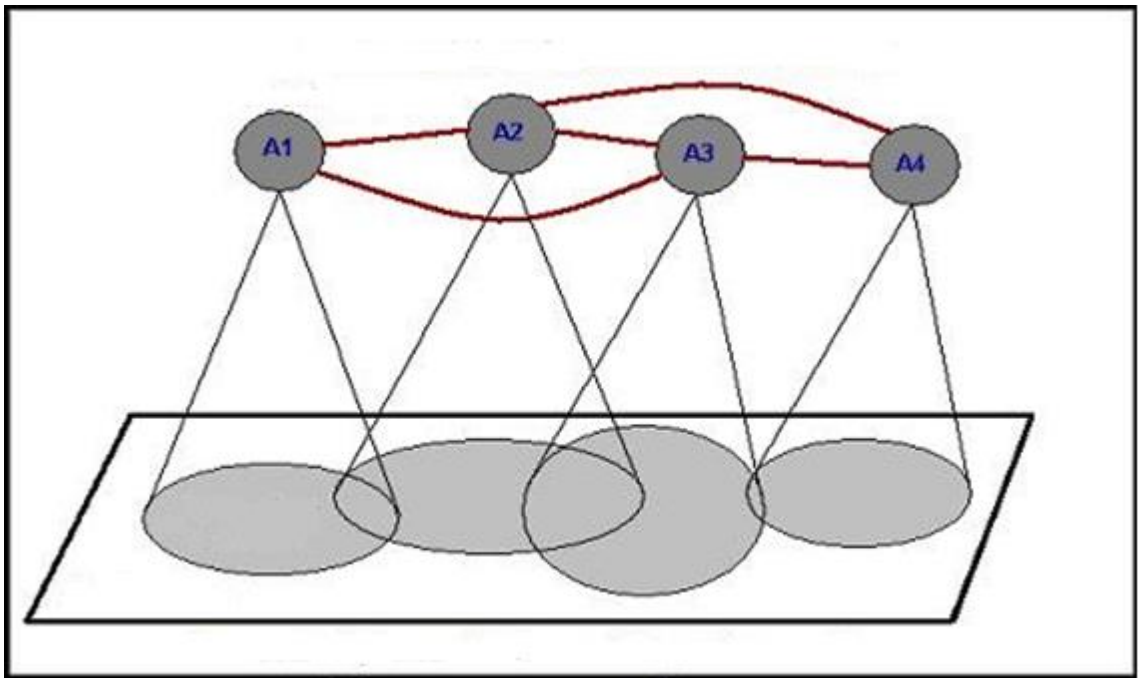
ve yeniden gözden geçir olarak üç sonuç üretir. Başarılı yapı yeşil renk ile ifade edilir, kod parçasının testi geçtiğini ifade eder. Başarısız yapı kırmızı renk ile ifade edilir, kod parçasının testi geçemediğini ifade eder. Yeniden gözden geçir sonucu ise mavi renk ile ifade edilir, kod parçasının testi geçtiğini ancak iyileştirmelere ihtiyaç olduğunu ifade eder.

Şekil 3. 2 Test Yönelimli Geliştirme Yaşam Döngüsü

Değerlendirme ve Geri Bildirim Aşaması: Değerlendirme aşaması, yapılan iterasyon hakkında geri dönüşlerin toplandığı aşamadır. Bu aşamada toplanılan bilgilerin, sonraki çalışmalara kaynak olması ve kolaylaştırması hedeflenir.

3.2 Sage Geliştirme Yöntemi

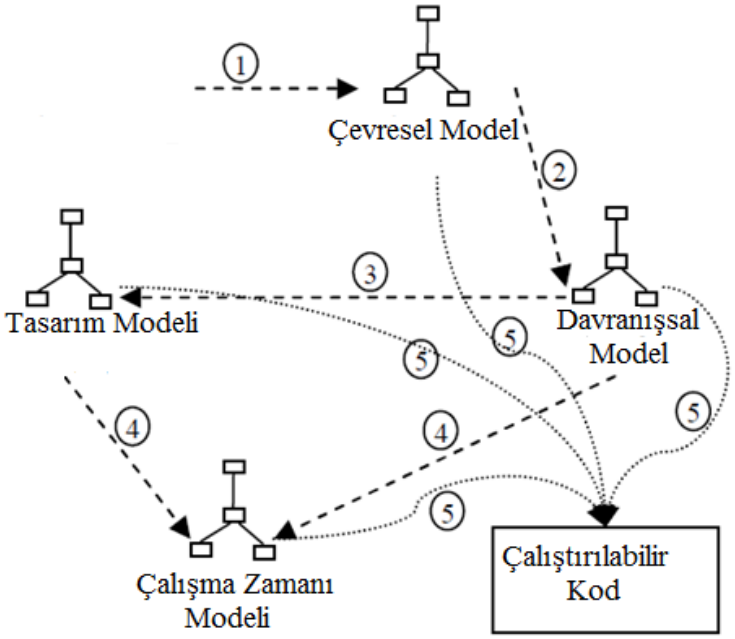
Sage geliştirme yöntemi [5], yüksek güvenlik öncelikli çok ajanlı sistemlerin geliştirilmesi amacıyla kullanılmış bir yöntemdir. Ajanlar, çevresinden aldığı veriler ile ortamı algılayabilen yapılardır. Ajanlar, etrafı algılayabilmek için üzerine konumlandırılmış sensörleri kullanırlar. Çok ajanlı sistemler de, birden çok sayıda ajanın birbirleri ile etkileşim halinde haraket edebildikleri sistemlerdir. Çok ajanlı sistemlere örnek olarak, otonom haraket kabiliyetine sahip futbol maçı yapan robot kümeleri gösterilebilir. Şekil 3.3'de çok ajanlı sistemlerin yapıları gösterilmiştir. Şekil 3.3'de tanımlanan A1, A2, A3 ve A4 ifadeleri sistem içerisindeki ajanları temsil etmektedirler.



Şekil 3. 3 Çok Ajanlı Sistemlerin yapısı

Ajanlar, etkileşim halinde oldukları diğer ajanlara sensör verilerini gönderebilir; aynı zamanda diğer ajanlardan da veri alabilmektedirler. Bu değerler, bir ajanın giriş ve çıkış değerleri olarak adlandırılır. Yüksek güvenlik öncelikli sistemler için çalışan bir sistem tasarlamak yeterli değildir. Müşteri geniş dokümantasyon ile sistem yapısının tutulmasını talep etmektedir. Ayrıca, çok ajanlı sistemler, arttırımsal programlama mantığı ile sistemin parça parça geliştirilmesine ihtiyaç duymaktadır. Bu iki isteği de karşılayabilmek adına bu yapılarda bir model tabanlı çevik süreç yöntemi olan *Sage* yöntemi kullanılmıştır. Modeller ile dokümantasyon, model tabanlı araçlar ile düşük hata eğilimi sağlanırken, arttırımsal çevik süreç kullanılmaktadır.

Sage geliştirme yöntemi dört adet birbirleriyle bağlantılı model sürecinin arttırımsal olarak birbirlerini tamamlaması mantığı ile çalışır. Bu modeller, çevresel model, davranışsal model, tasarım modeli ve çalışma zamanı modeli olarak isimlendirilir. *Sage* geliştirme yönteminin süreç akışı Şekil 3.4'de gösterilmiştir.



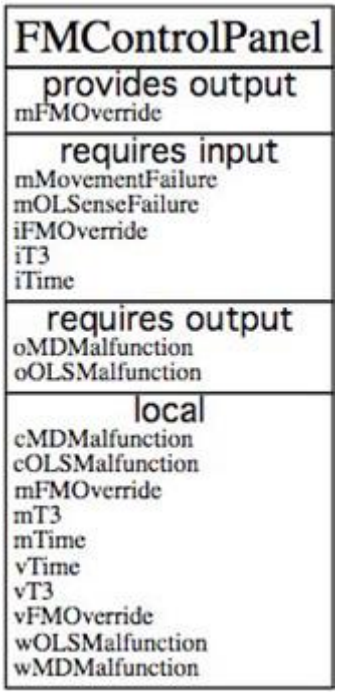
Şekil 3. 4 Sage Geliştirme Yöntemi süreç akışı [20]

Şekil 3.4'de süreç akışı, numaralar ile gösterilen sırada haraket etmektedir. Sıra ile modeller, bir önceki modelden yararlanılarak üretilmekte; tüm modellemeler yapıldıktan sonra ise bu modellerin tamamı döngünün çalıştırılabilir kod yapısını oluşturmaktadır.

Çevresel model, sistemin sınırlarının belirlendiği yapıdır. Sistemin neyi etkileyici ve nelerden etkileneceği, değişkenler olarak belirlenir. Çevresel etkenler tür ve karakteristiklerine göre tanımlanır. Bu değişkenler çevresel nitelikler olarak isimlendirilir.

Davranışsal model, sistemin mimari yapısıyla, fonksiyonelliği ve çalışma mantığı ile ilgilidir. Bu modelde, değişkenlerin fonksiyon yapıları ve kısıtları belirlenir. Hangi niteliklerin sistemi kontrol özelliğinde olduğu tanımlanır. Bu değişkenler kontrol değişkenleri olarak isimlendirilir.

Tasarım modeli, oluşturulan davranış modeline göre sınıfların oluşturulduğu yapıdır. Değişkenler, tasarım sınıflarına dönüştürülür. *Sage* modelin örnek sınıf yapısı Şekil 3.5'te gösterilmiştir.



Şekil 3. 5 Sage Geliştirme Yöntemi sınıf yapısı [5]

Sınıf yapıları, gereksinim duyulan giriş ve çıkış değişkenleri, sınıfın sağladığı giriş ve çıkış değerleri ve yerel değişkenlerden oluşmaktadır. Gereksinim duyulan değişkenlerden kasıt, sınıfın çalışması için gerek duyduğu değişkenlerdir. Sağlanan değişkenler ise, sınıfın ürettiği değişkenlerdir. Yerel değişkenler, sınıf içerisinde kullanılan değişkenlerdir.

Çalışma zamanı modeli, sınıf diyagramlarının çok ajanlı sistemlerde çoklu yapıya paylaştırıldığı yapıdır. Sistemin performans ve gereksinim karşılama kriterlerine göre sınıflar bölüştürülür. Her ajan için bellek kullanım miktarı da göz önünde bulundurulması gerekmektedir. Gereksinim duyulan ve sağlanan giriş değişkenleri ajanların sadece yazma kapasiteli portları (write-only port) kullanılarak oluşturulur. Gereksinim duyulan ve sağlanan çıkış değişkenleri de, ajanların sadece okuma kapasiteli portları (read-only port) kullanılarak oluşturulur.

3.3 Hybrid-MDD Geliştirme Yöntemi

Hybrid geliştirme yöntemi [4], üst seviye yaşam döngüsünün geliştirilmesi ile üretilmiştir. Üst seviye yaşam döngüsünden en önemli farkı, birbirleri ile paralel ve ortaklaşa çalışan takımların tanımlanmasıdır. *Hybrid* geliştirme yöntemi, üst seviye

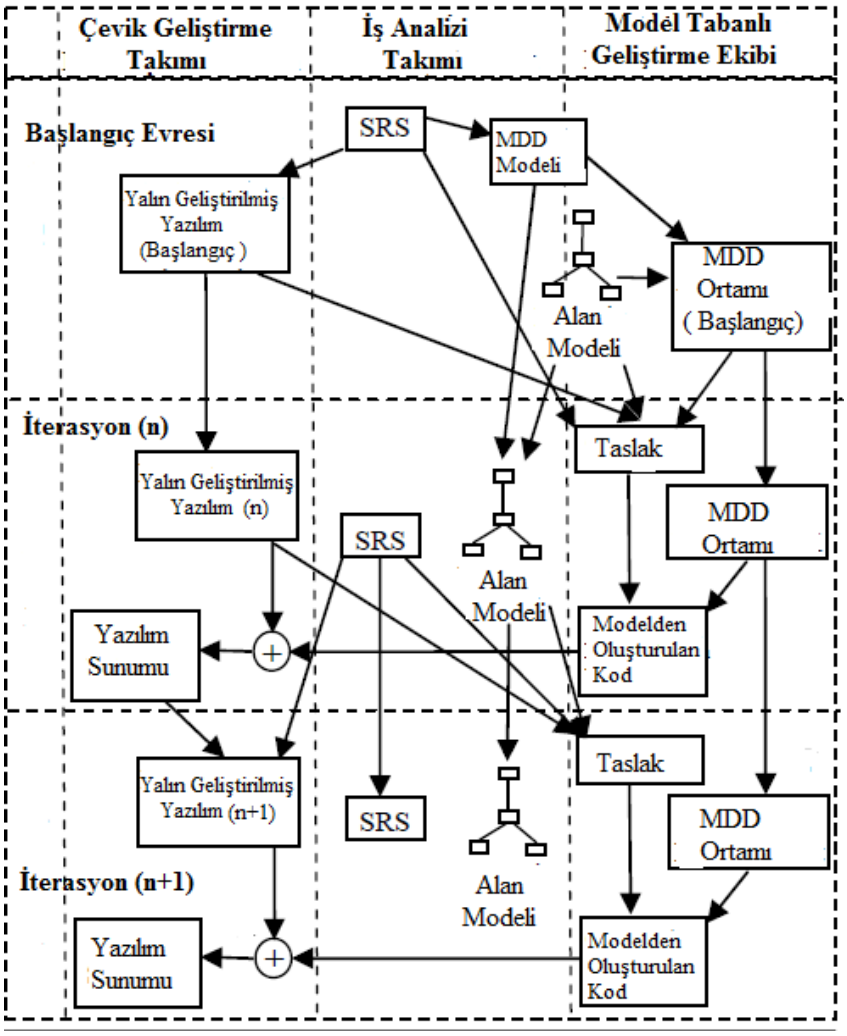
yaşam döngüsü gibi iki temel aşamadan oluşmaktadır. Bu aşamalar, başlangıç ve geliştirme aşamalarıdır. Geliştirme aşaması, iterasyonların gerçeklendiği aşamadır. Ancak *Hybrid* yöntemde, üç geliştirme takımı tanımlanmaktadır. Bu takımlar, çevik geliştirme takımı, model tabanlı geliştirme takımı ve iş analisti takımı olarak adlandırılır.

İş analisti takımı, müşteri ile iletişim halinde olunmasından ve gereksinimlerin üretilmesinden sorumludur.

Model tabanlı geliştirme takımı, sistem modellerinin oluşturulmasından, model tabanlı geliştirme altyapısının oluşturulmasından ve modelden koda dönüşümden sorumludur. İş analisti takımının paylaştığı gereksinim analizine ve sisteme uygun model tabanlı geliştirme araçların seçiminden sorumludur.

Çevik geliştirme takımı, otomatik geliştirilen kodların haricindeki el ile yazılan kod parçalarından sorumludur.

Hybrid geliştirme yönteminin süreç akışı Şekil 3. 6'da gösterilmiştir.



Şekil 3. 6 Hybrid Geliştirme Yöntemi yaşam döngüsü [20]

Şekil 3.6'da üst tarafta takımlar tanımlanmıştır. Takımların sınırları içinde kalan işlemler o takım tarafından gerçekleştirilecek işlemleri belirtmektedir. İki takım arasındaki ortak sınırda yer alan işlemler, iki takımın birlikte gerçekleştirdiği işlemlerdir. Yatay eksenle belirtilen sınırlar, yaşam döngüsü aşamalarını ayırmış ve her döngüde yapılan işlemler kendi yatay sınırında gösterilmiştir. Ayrıca şekil okuması yukarıdan aşağıya doğrudur ve daha yukarıda olan işlemlerin, daha önce gerçekleştirildiği belirtilmektedir. Eğer bir işlem, ona bağlı bir ok ile gösterilmiş ise, o işlemin uygulanabilmesi için, ona bağlanan işlemin tamamlanması gerektiği anlaşılmaktadır.

Süreç, iş analizi takımının müşteri ile toplantı yaparak gereksinimleri belirlemesi ile başlar. İş analizi takımı, gereksinim spefikasyon raporunu (SRS) çıkarır. Gereksinim raporunu, model tabanlı geliştirme ve çevik süreç takımları ile paylaşır. MDD takımı ile

birlikte gereksinimlere uygun ilk modellemeleri yapar. MDD takımı, kullanılacak MDD aracını belirler ve MDD altyapısını oluşturur. Çevik takımı ise, başlangıca uygun veritabanı tasarımı ve test süreçleri ile ilgilenir. Bu süreçlerin tamamlanması ile başlangıç döngüsü tamamlanmış olur.

İterasyon döngülerinde MDD takımı otomatik geliştirilecek kodları oluşturur. Bu kodlar çevik süreç takımı ile paylaşılır ve çevik süreç takımı otomatik geliştirilen kodları el ile yazılabilecek kodlar ile tamamlar. Test sürecinden geçirip, ilk iterasyon sonucunun ilk halini oluşturur. İş analizi takımı da süreç içerisinde gereksinim değişiklikleri olmuşsa, gereksinim analiz raporlarını günceller ve müşteri ile daimi iletişim halinde bulunur. Her iterasyon aynı süreçten ve kontrollerden geçirilerek süreç tamamlanır.

3.4 SLAP-MDD Geliştirme Yöntemi

Motorola şirketi tarafından, kendi projelerinde uygulanmış bir model tabanlı çevik süreçtir. SLAP (System-level agile process), Motorola'nın kullandığı bir çevik süreçtir. SLAP-MDD yapısını geliştirirken bu yapıyı iskelet olarak kullanmıştır [6]. SLAP yapısı, geliştirilecek yazılımı çoklu iterasyonlara böler ve her iterasyon üç adet döngü'den (sprint) oluşmaktadır. Bu döngüler uygulama gereksinimi ve mimari, geliştirme döngüsü ve test döngüsü olarak adlandırılır. Her döngü için ortalama beş hafta süre ayrılması öngörülmüştür ve her döngü benzer küme aktivitelerden oluşmaktadır. İlk hafta döngü başlangıcı, uygulama gereksinim analizi, üst seviye tasarım; ikinci hafta detaylı tasarım; üçüncü hafta gerçekleştirme; dördüncü hafta teknik gözden geçirme ve düzenleme; ve beşinci hafta entegrasyon hazırlığı ve döngü sonrası toplantı aktiviteleri gerçeklenir. SLAP yöntemi iterasyon ve döngüleri Çizelge 3.1'de gösterilmiştir.

Çizelge 3. 1 SLAP yöntemi iterasyon ve döngüleri [6]

	Döngü 1	Döngü 2	....	Döngü (n)
İterasyon 1	Gereksinim analizi ve Mimari Tasarım	Geliştirme	Test	
İterasyon 2		Gereksinim analizi ve Mimari Tasarım	Geliştirme	Test
.....			Gereksinim analizi ve Mimari Tasarım	Geliştirme
İterasyon n				Gereksinim analizi ve Mimari Tasarım

Çizelge 3.1'de belirtildiği üzere, ikinci iterasyonda üçüncü döngünün başlaması ile farklı iterasyonlardaki tüm döngüler paralel şekilde ilerlemektedir. Belirtilen iterasyon için, gereksinim ve mimari döngüsü, bir sonraki iterasyon için mimari ve gereksinim hazırlamaktadır.

SLAP-MDD yapısı, SLAP yapısı üzerinde MDD pratiklerinin uygulanması ile yapılmaktadır. Çevik süreç pratikleri, MDD pratikleri ile uyumlu olacak şekilde eşleştirilir. Süreç, modellemeler ve "herşey modeldir" prensibi üzerinden yürütülür. MDD pratikleri ile çevik pratiklerinin eşleştirilmesi Çizelge 3.2'de gösterilmiştir.

Çizelge 3. 2 MDD pratikleri ile çevik pratiklerinin eşleştirilmesi [6]

		Çevik pratikler							Çevik Yönetim Pratikleri			
		İkili Geliştirme	Test tabanlı geliştirme	Tekrarlı ve arttırımsal geliştirme	Dokümantasyon yerine çalışan	Otomatik bütünlük testi	Sürekli birleştirme	Hızlı geri bilgi akışı	Günlük toplantı	Günlük durum takibi	Kısa döngü süreleri	Döngü sonrası toplantı
MDD pratikleri	Kod olarak modelleme	*										
	İkili modelleme	*										
	Test-tabanlı modelleme		*									
	Tekrarlı ve arttırımsal modelleme			*								
	Canlı dökümantasyon olarak modeller				*							
	Otomatik toplu modda test					*						
	Sürekli modelleme						*					
	MDD araç zinciri							*				
Çevik MDD yönetim pratikleri	Günlük MDD plan güncellemesi								*			
	Günlük MDD görev durum güncellemesi									*		
	Mevcut döngü için MDD plan										*	
	Önceki döngüler için MDD geri bilgi akışı											*

Çizelge 3.2'de belirtilen prensipler SLAP-MDD nin uyması gereken temel nitelikler olarak tanımlanabilir. Bu prensiplere bağlı olarak SLAP te olduğu gibi SLAP-MDD'de de üç ana döngü içermektedir. Gereksinim ve mimari analiz döngüsü SLAP'te olduğu şekliyle aynı kalmıştır. Geliştirme döngüsünde geliştirme gereksinim analizi, üst seviye tasarım, detaylı tasarım, kod dönüşümü, UML ünite testi ve entegrasyon testi aktiviteleri içerir. Test döngüsü de, alt sistem testi ve bütün haliyle sistem testi aktivitelerini içermektedir.

Kod olarak modelleme: Bu prensipte kodlar, modeller ile ifade edilir. Model için UML kullanılırken, UML araçları ile UML'den C ve C++ kodlarına dönüşüm uygulanır.

İkili modelleme: Bu prensip çevik pratiklerden ikili geliştirme ile uyumludur. Bu aktivitede iki kişi, tercihen bir geliştirici ve bir müşteri temsilcisi, modellemeleri birlikte oluştururlar.

Test-tabanlı modelleme: Bu prensip, çevik pratiklerden test-tabanlı geliştirme ile uyumludur. Bu pratikte öncelikli olarak test kriterleri oluşturularak, sonrasında modeller bu kriterlere bakılarak üretilir. Bu yaklaşım, UML modelini oluşturduktan sonra kontrol üniteleri oluşturarak diyagramı test etmekten daha efektif olduğu öngörülmektedir.

Canlı dokümantasyon olarak modeller: Çevik manifestodaki canlı dokümanın çalışan yazılım olduğu ifadesinin UML'e uygulanmasıdır. Bu prensip, UML modellemeleri, dokümantasyon olarak kullanılmasını öngörür ve elle yazılan dokümantasyonun minimalize edilmesini önerir.

Otomatik toplu modda test: Bu prensipte, oluşturulan yeni modellerin veya var olan bir modele yapılan güncellemenin, modelin doğruluğunu bozmaması hedeflenmektedir. Bazı temel kuralların MDD araçlarında otomatik olarak modelleri kontrol etmesini öngörür.

Sürekli modelleme: Bu prensip, çevik pratiklerden sürekli entegrasyon ile uyumludur. Bu prensipte, sürekli olarak üretilen UML diyagramları birleştirilir. Otomatik kod dönüşümü, simülasyon ve test düzenli olarak gerçekleştirilir.

MDD araç zinciri: Model tabanlı geliştirmede, MDD araçları seçimi ve kullanımı, projenin başarısını ve hızını doğrudan etkilemektedir. MDD aracı secerken diğer MDD araçları ile uyumu da göz önüne alınması gerekmektedir. Bu yöntemde, Motorola geliştiricileri IBM Tau aracını UML modelleme için tercih etmiş; IBM tester aracını da birim ve entegrasyon testi için kullanmıştır.

Günlük MDD plan güncellemesi: Sprint methodunda uygulanan prensip, burada modeller için uygulanmaktadır. Günlük olarak, kısa Sprint toplantıları ile günün modelleme planı kararlaştırılmaktadır.

Günlük MDD görev durum güncellemesi: Görevler günlük olarak güncellenmekte ve yapılan işlerin durumlarının kaydedilmesi istenmektedir. Görevler, tamamlandı, henüz tamamlanmadı ya da çalışma halinde şeklinde kategorize edilebilmektedir.

Mevcut döngü için MDD plan: MDD planları makro ölçekte çıkarıldığı gibi, mevcut durum için de bir plan oluşturulması istenir. Mevcut geliştirilen modelin tartışması ve geri dönüşün alınması hedeflenmektedir.

Önceki döngüler için MDD geri bilgi akışı: Bir döngü bittikten sonra, o döngü içinde modelleme ve modelleme altyapısı hakkında neler öğrenildiği değerlendirilir. Bu değerlendirme sonraki döngülerin daha verimli geçmesi için kullanılır.

BÖLÜM 4

WEB UYGULAMALARINDA MODEL TABANLI GELİŞTİRME

Bu bölümde, web uygulamalarının yapısı, web uygulamalarında model tabanlı geliştirmenin nasıl uygulandığı incelenmiştir. Sonrasında, web uygulamalarında kullanılmak üzere özelleştirilmiş, kendi yaklaşımımız olan *Hybrid Mockup* tabanlı geliştirme yöntemi tanıtılmıştır.

4.1 Web Uygulamalarının Yapısı

Web uygulamalarında yazılım geliştirirken, web uygulama geliştirme teknolojilerinden faydalanılması gerekmektedir. Bu teknolojiler en basitinden HTML (Hyper Text Markup Language), CSS (Cascading Style Sheets) ve JavaScript'tir. HTML, web sayfalarının kullanıcı görünümünü sağlayan bir biçimlendirme dilidir. Bu biçimlendirmeyi, tanımlanan etiketler yardımı ile yapmaktadır. Her etiket farklı bir konsepti etkileyerek onun içeriğini belirler. CSS teknoloji ise, HTML verilerinin nasıl görüneceğini belirleyen yapıdır. Web sayfasının stili ve nasıl görüneceği CSS ile belirlenir. CSS bilgileri HTML etiketleri içerisinde tanımlanabileceği gibi, ayrı bir dosya yardımı ile de tanımlanarak işlemleri kısaltabilmektedir. JavaScript teknolojisi, yaptığı işlemler ile sayfada değişiklik yapabilmemizi sağlar. JavaScript, içerisinde fonksiyonlar barındırır ve bu fonksiyonlar ile sayfaya dinamizm katar.

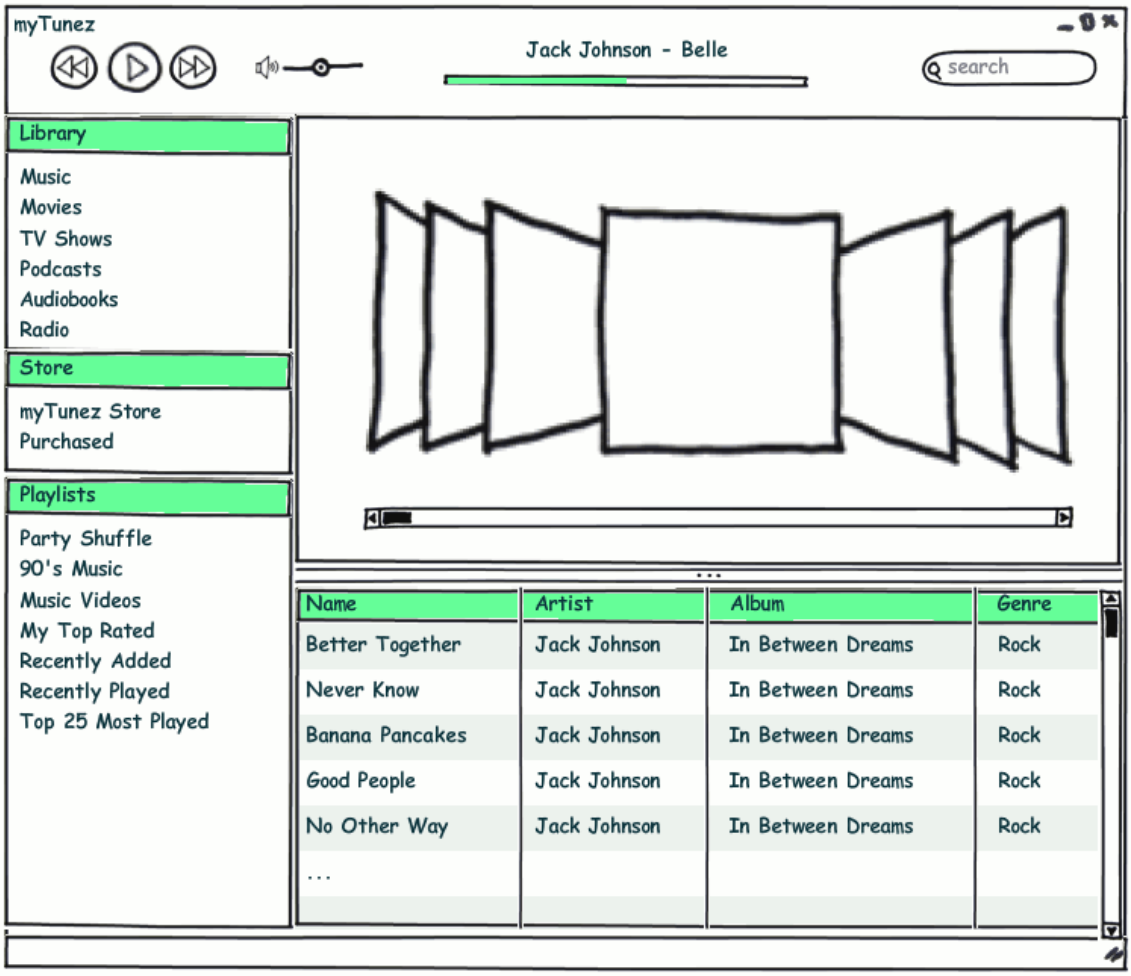
Web uygulamaları, sunucu ve kullanıcı arasında etkileşim mantığına dayanır. Kullanıcı talep eder, ilgili veriler sunucu tarafından gönderilir. Programlama, sunucu ve istemci taraflı programlama olarak ayrılmaktadır. İstemci, kullanıcı bilgisayarı ve web tarayıcısı olarak belirtilir. Sunucu tabanlı programlama, işlemlerin sunucuya gönderilip, sunucu

üzerinde yapılıp, geri gönderildiği sistemlerdir. İstemci tabanlı programlama ise, işlemlerin kullanıcı web tarayıcısında yapıldığı işlemlerdir.

4.2 Web Uygulamalarında Model Tabanlı Geliştirme

Web uygulamalarında model tabanlı yazılım geliştirme [21], [22], [23], [24] mockup'ların kullanımı ile gerçekleştirilmiştir. Mockup'lar, web sayfaları ve onun elemanlarının modelleridir. Web modelleri olan mockup'ların, kaynak kodlar yazılmadan üretilmesi yöntemi mockup tabanlı yazılım geliştirme olarak tanımlanır. Mockup tabanlı yazılım geliştirmede, öncelikli olarak web sayfası, mockup yardımı ile modellenir, modellere özellik ve fonksiyonlar eklenir; sonrasında ise mockup araçları ile web sayfalarına dönüştürülür. Mockup'lar ilk olarak tasarım amacı ile ortaya çıkmış, görsel tasarımın yapımında kullanılmış. Sonrasında bu yapılar dinamikleştirilerek model tabanlı geliştirmeyi sağlamışlardır [25], [26].

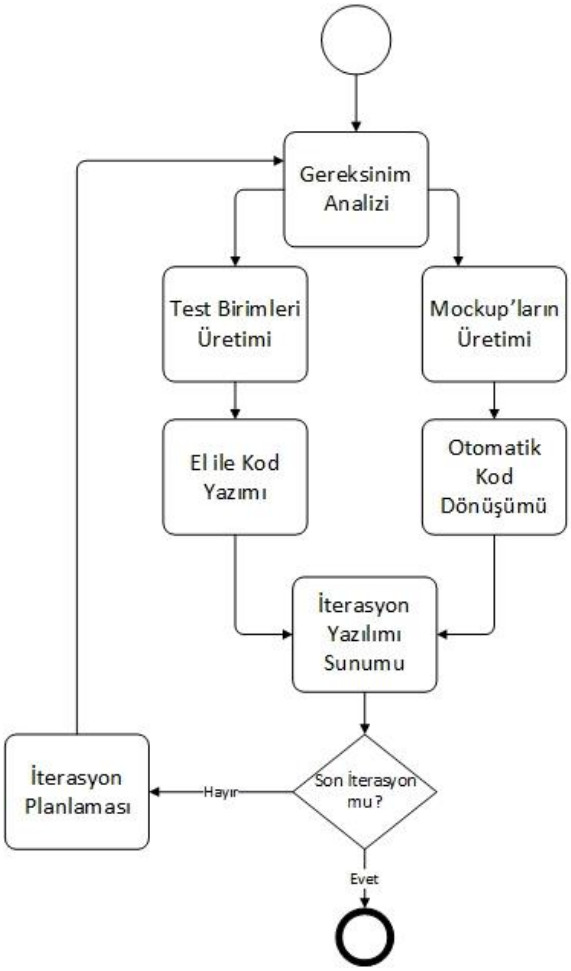
Mockup'lar ile oluşturulmuş bir web sayfası örneği Şekil 4.1'de gösterilmiştir. Bu web sayfasındaki her eleman boyut, konum gibi biçimsel özellikler alabildiği gibi; fonksiyonel özellikler de alabilmektedir.



Şekil 4. 1 Mockup kullanılarak oluşturulmuş örnek bir web tasarımı [27]

4.3 Hybrid Mockup Tabanlı Geliştirme

Web uygulamaları için model tabanlı çevik süreç gerçeklemesi amacıyla, tezin üçüncü bölümünde incelediğimiz *Hybrid* yöntemi yapısı, iskelet olarak kullanılmıştır. Bu yöntemi web uygulamalarında kullanabilmek için mockup tabanlı geliştirme yöntemi ile birleştirilmesi düşünülmüştür. Bu amaç ile, literatüre yeni bir katkı sağlayacağını düşündüğümüz, *Hybrid mockup* tabanlı yöntem [28] önerilmiştir. Bu yöntemi tasarlarken, dördüncü bölümde incelenen tüm yöntemlerin değerlendirilmesi yapılmış ve web uygulamalarının ihtiyaçları da göz önüne alınarak Şekil 4.2'de süreç akışı gösterilen yapı üretilmiştir.



Şekil 4. 2 Hybrid Mockup Tabanlı Geliştirme süreç akışı

Şekilde aktiviteler dikdörtgen ile belirtilmiştir. *Hybrid* yönteminde olduğu gibi web uygulaması için, birbirleri ile iletişim halinde üç takım öngörülmüştür. Süreç akışı yukarıdan aşağı şekilde devam etmektedir ve aynı düzlemde yer alan aktivitelerin paralel bir şekilde uygulanması önerilmiştir. Paralel çalışma ile, geliştirme hızında artış ve süreç içi sorunların hızlı çözümü hedeflenmektedir. Döngüsel olarak geliştirme sağlanmaktadır. Süreç işleyişi şu adımları izlemektedir.

- İş analizi takımı, müşteri ile iletişimden sorumludur. İş analizi takımı tarafından uygulama gereksinimleri raporlanır. Burada ikili geliştirme uygulanır ve müşteri temsilcisi ile birlikte gereksinimler çıkartılır.
- MDD takımı kullanılacak mockup aracını belirler ve mockup geliştirme altyapısını oluşturur. Gereksinimlere uygun mockup tasarımı yapılır. Mockup tasarımları aynı zamanda web sayfalarının arayüzünü de oluşturmakta ve mockupların sağladığı

görsel sunum sayesinde gereksinimlerin daha net bir şekilde oluşturulması sağlanmaktadır. Oluşturulan mockup tasarımı müşteriye sunulur ve gerekli düzenlemeler yapılır.

- Gereksinim raporu ve mockup tasarımına göre çevik süreç takımı veritabanı altyapısını oluşturur. Çevik süreç takımı test birimlerinden de sorumludur. Gereksinimlere uygun test birimlerini üretir.
- MDD takımı, oluşturulan mockup'lara fonksiyonellik kazandırır ve otomatik kod dönüşümü ile çalışan yazılıma dönüştürülür. Yazılım bu haliyle kullanıcı taraflı fonksiyonları gerçekleştirebilir durumdadır. Sistemin arayüz tasarımları tamamlanmış, HTML,CSS ve JavaScript kodları elde edilmiştir.
- Yazılımın bundan sonraki tamamlanması ve test ünitelerinden geçirilip iterasyon yazılımını haline gelmesi sorumluluğu çevik süreç takımındadır. El ile yazılan kodlar ile mockup araçlarının ürettiği kod parçaları tamamlanarak çalışan yazılım elde edilir.
- Yazılım testlerini, bu yöntemde web uygulamaları için kara kutu test (black box test) önerilmektedir. Çünkü mockup araçları ile geliştirme sürecinin büyük çoğunluğu otomatik olarak işlemektedir. Bu anlamda sistemin gereksinimlerinin karşılanması ve sistemin doğru çalışması kara kutu test ile kontrol edilir.
- İterasyon yazılımı, testi geçtikten sonra, müşteriye sunulur. Çevik sürecin bir gereği olarak müşteri ile sürekli iletişim zorunlu tutulmuştur.
- Gerçekleştirilen iterasyon için geri dönüşümler rapor edilir.
- Bir sonraki iterasyonu gerçekleştirmek ve plan çıkarmak için tüm takım toplanarak iterasyon planlaması yapar [28].

4.4 Yaklaşımın Diğer Metodlar ile Karşılaştırılması

Yaklaşımımızı üç farklı grup metod ile karşılaştırabiliriz. Yaklaşımın geleneksel sürece göre karşılaştırılması, sadece model tabanlı geliştirme ile karşılaştırılması ve sadece çevik süreç ile karşılaştırılması bu grupları oluşturmaktadır.

Geleneksel süreç, uzun analiz ve tasarım aktiviteleri içermektedir. Bu aktiviteler tamamlanana kadar kodlamaya geçilmemekte ve doğrusal bir yapıda tekrarsız olarak aktiveler birbirini izlemektedir. Bizim yaklaşımımızda çevik sürecin gereği olarak aktivitelerin tekrarlı olarak geliştirilmesi ve arttırımlı programlama esas alınır. Ayrıca test tabanlı yazılım geliştirme de önerilmektedir. Test süreçleri yazılım kodlamasından önce gerçekleştirilmektedir. Geleneksel süreçte geniş dokümantasyon imkanı sunmakla birlikte, dokümantasyonu modele bağlamamaktadır. Bizim yaklaşımımız yeterli dokümantasyon sağlamaktadır; ancak çevik modellemenin gereği olarak dokümantasyon olarak modelleme yapılmaktadır. Geleneksel süreç dokümantasyon temelli bir yazılım geliştirirken, yaklaşımımız model tabanlı geliştirme yapmaktadır; ancak modeli değil, çalışan yazılımı hedef almaktadır.

Sadece model tabanlı geliştirmeyi web uygulamaları için düşünürsek, Şelale modelini izleyen mockup tabanlı geliştirme ortaya çıkar. Yine geleneksel sürecin çevik sürece karşı olan süreç içerisindeki değişime uyum problemi, paydaşların sürece katılım eksiği gibi dezavantajları bünyesinde barındırmaktadır. Model tabanlı geliştirmenin sağladığı görsel analiz üzerinden kod üretme, arayüz tasarım kolaylığı ve arttırılmış anlaşabilirlik gibi avantajlar ise yaklaşımla ortaktır.

Sadece çevik süreç ile karşılaştırdığımızda, model tabanlı geliştirme prensibini kullanmayan tekrarlı ve arttırımsal bir süreç modeli düşünmeliyiz. Bu model ile çevik modelleme prensipleri arasında üçüncü bölümde tanıttığımız SLAP-MDD yaklaşımında ayrıntısıyla ortaya koyan sıkı bir bağ vardır. Bu anlamda aktivitiler birbirleri ile ilişkilidir; ancak çevik modellemede prensiplerin model üzerinde uygulanması düşünülmektedir. Örneğin çevik sürecin ikili programlama prensibi ikili modelleme olarak uygulanmaktadır. Çevik sürecin arttırımsal programlama prensibi ise, arttırımsal modelleme olarak uygulanmaktadır. Ayrıca otomatik kod dönüşümü ile düşük hata olasılığı ve zaman kazanımını, sadece çevik süreçte sağlayamayız.

BÖLÜM 5

UYGULAMA ÇALIŞMASI

Dördüncü bölümde tanımlanan web uygulamaları için *Hybrid* mockup tabanlı yaklaşım, literatürde daha önce denenmemiş bir yöntemdir. Yaklaşımın uygulanabilirliği, artı ve eksi yönlerinin değerlendirilmesi ve diğer yöntemler ile karşılaştırılabilmesi için, yeni yaklaşım, "Yazılım Mühendisliği" dersinde, öğrenci dönem projelerinde uygulanmıştır. Bu bölümde, yapılan uygulama çalışması anlatılmaktadır.

5.1 Uygulanma Yöntemi

Yazılım mühendisliği dersinde dönem projesi geliştirecek olan öğrenci gruplarından iki proje grubu seçtik. Bu proje gruplarından biri online sinema bilet sistemi geliştirirken, diğer grup online dil kursu kayıt sistemi geliştirmektedir. Sinema bilet sistemini geliştiren grup beş öğrenciden oluşmaktadır. Dil kursu kayıt sistemini geliştiren grup ise dört öğrenciden oluşmaktadır. Öğrenciler genel programlama ve yazılım mühendisliği bilgisine sahiptirler.

Öncelikli olarak proje ekipleri ile toplantı yaptık. Bu toplantıda yaklaşımımız ve yaklaşımı projelerine nasıl uygulayacakları anlatıldı. Ekiplerin kendi içlerinde çevik geliştirme sorumlusu, MDD sorumlusu ve iş analisti sorumlusu olacak şekilde üç gruba ayrılmaları istendi. Projelerinde MDD ortamı olarak Axure geliştirme aracının kullanılması kararlaştırıldı. Axure bir mockup tabanlı geliştirme aracıdır ve otomatik kod dönüşümü ile yapılan tasarımın HTML, CSS ve JavaScript kodlarına dönüştürülmesini sağlamaktadır.

Ekipler ile her hafta düzenli olarak toplantı düzenlendi. Toplantıda projelerin ilerleyişi ve geliştirme yapılırken yöntemi uygulayışları kontrol edilerek, yaklaşıma sadık kalmaları sağlandı. Proje ekiplerinden kodlama ve model tabanlı geliştirmeye başlamadan önce, gereksinim analizi çıkarmaları istendi. Ekiplerin, projeleri için ürettikleri gereksinim analizi raporları Ek-A'da gösterilmiştir.

Gereksinim aşamasından sonra model tabanlı ekibin tasarım yapması ve mockup'ları üretmesi istenmiştir. Mockup'ların üretimi ilk aşamada görsel olarak tasarım sağlamaktadır. Bu görsel arayüz, sistemin mimari yapısının ve gereksinim analizinin uygunluk yönünden değerlendirilmesinde faydalı olmaktadır. Görsel tasarım sonrasında, tasarım üzerinde web elementlerine eylemler atanarak fonksiyonellik kazandırılmıştır. Burada fonksiyonellik istemci taraflı programlama ile sınırlıdır. Mockup tabanlı geliştirme aşamasında, istemci taraflı programlama kodlarının mockup geliştirme araçları tarafından üretilmesi sağlanmaktadır. Proje gruplarının ürettikleri mockup tasarımları Ek-B'de gösterilmiştir.

Model tabanlı geliştirme ekibine paralel olarak çevik süreç takımlarından test birimlerini üretmeleri istenmiştir. Test yöntemi olarak kara kutu test tekniğinin model tabanlı web uygulamaları için uygun olduğuna karar verildi. Kara kutu test tekniği, uygulamanın kodlarının test edilmesi yerine, çalışan yazılımın doğru çalışıp çalışmadığının kontrol edilmesi prensibine dayanır. Otomatik kod dönüşümü ile el ile yazılan kodun azlığı ve kod üretiminin büyük kısmının araçlara bırakılmasından dolayı bu yöntem, hızlı test süreci geçirilmesi için seçilmiştir. Ayrıca test birimlerinin üretimi sonrasında el ile yazılacak kodlarında üretilmesi istenmiştir. Tüm aşamaların tamamlanması ile birlikte iterasyon ürünü oluşturulmuştur. Bu aşamalar tekrarlanarak son yazılım üretilmiştir. Proje grupları, üç iterasyon sonucunda son yazılıma ulaşmışlardır. Projelerin gerçeklenmesi üç aylık bir sürede tamamlanmıştır.

5.2 Proje Grupları ile Yapılan Anket Çalışması

Projenin son yazılımları tamamlanıp, öğrenciler projelerini sunduktan sonra, öğrenci gruplarından yaklaşımı ve süreci değerlendirmeleri istenmiştir. Bu amaç için hazırlanan sorular Çizelge 5. 1'de gösterilmiştir.

Çizelge 5. 1 Öğrenci gruplarına yöneltilen sorular

1) Yaklaşımı klasik yöntem ile karşılaştırdığınızda nasıl değerlendirirsiniz ?
2) Klasik yöntem ile karşılaştırdığınızda yeni yaklaşımın güçlü yönü nedir?
3) Yeni yaklaşım ile geliştirme yapmaktan genel anlamda memnun musunuz?
4) Bu yaklaşımın zayıf yönü olarak neyi ön plana çıkartabilirsiniz?
5) İleride web uygulaması yapmayı düşündüğünüzde bu yaklaşımı kullanır mısınız?
6) Ekip içi iletişim kuvvetli mi ?
7) Mockup ile web tasarımını kolaylık yönünden nasıl değerlendirirsiniz?
8) Kullandığınız geliştirme aracının (Axure) yardımını nasıl değerlendirirsiniz?
9) Yaklaşımı kullanmakta zorluk yaşadınız mı ?
10) Mockup'lar ile çalışmak analiz ve anlaşılabilirliği arttırdı mı, yoksa geliştirme karmaşıklığını arttırdı mı?
11) El ile yazılan kod ile otomatik oluşturulan kodları birleştirmekte zorluk yaşadınız mı ?

5.2.1 Sinema Bilet Sistemi Ekibinin Anket Soruları Sonucu Değerlendirmesi

Sinema bilet sistemi proje grubu ile yapılan anket çalışması sonucunda alınan değerlendirmeler aşağıda belirtilmiştir.

- Yaklaşımın tasarım kolaylığı sağladığını; ancak model tabanlı geliştirme aracını kullanmakta zorluk yaşadıklarını belirttiler.
- Yardımcı geliştirme aracının (Axure), veritabanı bağlantısı ile ilgili bir eklentisi bulunmamaktadır. "Repeater" isimli elemente sahiptir, ancak ilişkisel veritabanı

ile bağlantı kuramamaktadır. Bu sebeple veri tabanı ile bağlantı sağlamakta sıkıntı yaşadıklarının altını çizmişlerdir.

- Yaklaşım ile hızlı bir şekilde geliştirme sağladıklarını ve istemci taraflı programlamayı seri bir şekilde geliştirdiklerini belirttiler.
- Ayrıca yaklaşımın düzenli iş takibi sağladığını belirttiler.
- Uygulama kolaylığı yönünden tasarım kolaylığını ön plana çıkardılar.
- Görsel tasarım açısından, web uygulamasının yapısının tasarlanmasında ve sayfalar arası bağlantıda iyi düzeyde olduğunu belirttiler.
- Yaklaşımın zorluk düzeyini orta olarak değerlendirdiler. İlk defa kullanıyor olmasının etkisi olduğunu vurguladılar.
- Otomatik dönüşüm ile beklenenden uzun kod verdiğini belirttiler.
- Veritabanı ile bağlantıda zorluk yaşadıklarının altını çizdiler.
- Şuan günümüzde şirketlerin bir çoğu çevik süreç kullandıklarını ve proje gruplarının da ileride bu yaklaşımı kullanmayı tercih edeceklerini söylediler ve genel anlamıyla yaklaşımdan memnun olduklarını belirtmişlerdir.
- Kodlama yönünden karmaşıklığı ve anlaşılabilirliği arttırdığını; ancak tasarım yönünden karmaşıklığı azalttığını belirttiler.

5.2.2 Dil Kursu Otomasyonu Ekibi ile Yapılan Anket Çalışması

Dil kursu otomasyon grubu ile yapılan anket çalışması sonucunda değerlendirmeleri aşağıda belirtilmiştir.

- Tekrarlı geliştirme ile daha kolay geliştirme sağladıklarını ve parçalar halinde geliştirme yaparak, projelerini kolaylaştırdıklarını belirttiler.
- Düzenli olarak iş takibinin sağlandığını ve her hafta, tüm grubun yapması gereken işler olduğu için devamlılık sağladığını ön plana çıkardılar.
- Yardımcı geliştirme aracında (Axure), CSS'leri ayarlamakta zorluk yaşadıklarını belirttiler. Bu sebeple Axure programına ek olarak Pingendo aracını

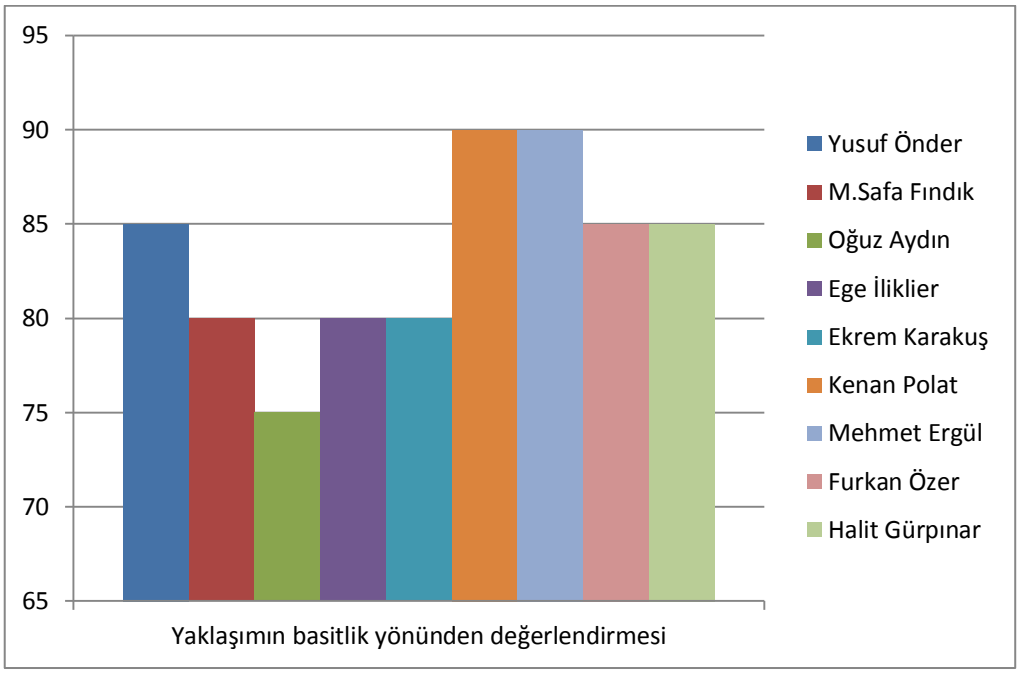
kullandıklarını; Axure ile ilk tasarımı sağlarken, Pingendo aracı ile yaşadıkları CSS probleminin üstesinden geldiklerini belirttiler.

- Sadece Axure aracını kullanarak kolaylık yönünden orta; iki aracı birlikte kullanarak kolay olarak değerlendirdiler.
- Web uygulamamıza aktiflik kazandırdığını, ilk defa kullanmalarına rağmen fazla zorluk yaşamadıklarını; Axure ve Pingendo aracını birlikte kullanarak karmaşıklık ve anlaşabilirliği azalttığını belirtmişlerdir.
- Axure aracının daha fazla tecrübe edilmesine ihtiyaç duyulduğunun altını çizmişlerdir.
- Genel anlamıyla yaklaşımdan memnun olduklarını belirttiler.

5.3 Sayısal Değerlendirme

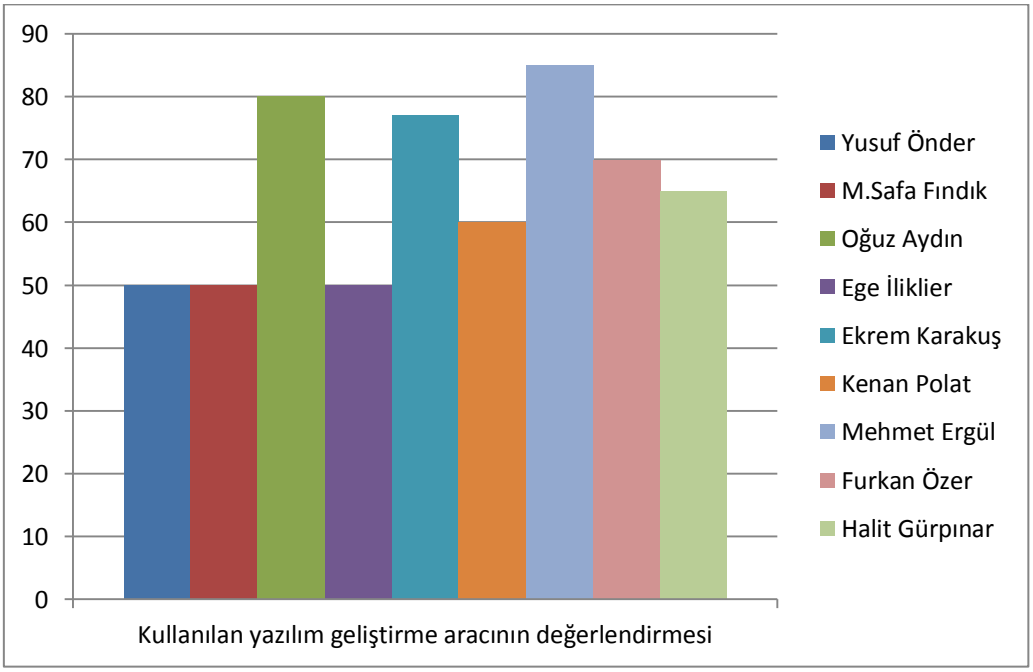
Yapılan çalışmanın sayısal değerlendirmesini yapmak amacı ile iki proje grubununda tüm elemanlarına sayısal notlandırma yapabilecekleri beş adet soru yöneltilmiştir. İki grubun toplam dokuz üyesi, sorulara birbirlerinden bağımsız şekilde 100 üzerinden notlandırma yapmışlardır. Yöneltilen sorular ve proje grup elemanlarının bu sorulara verdikleri değerlendirme puanları aşağıda gösterilmiştir.

1. Kullandığınız yaklaşımı basitlik yönünden değerlendiriniz?



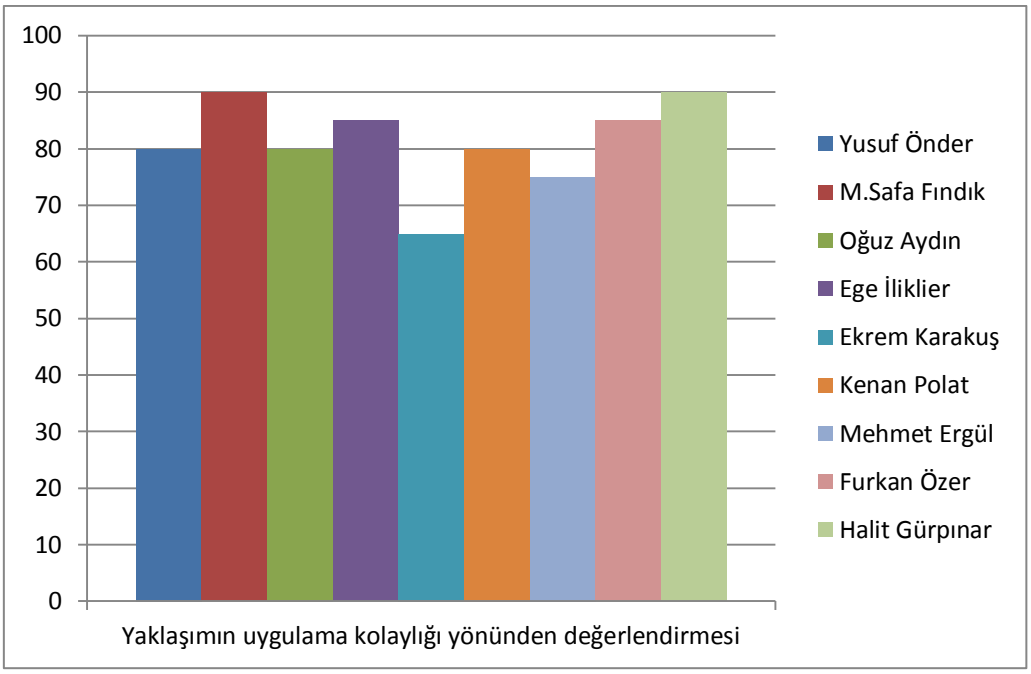
Şekil 5. 1 Yaklaşımın basitlik yönünden değerlendirmesi

2. Kullandığınız geliştirme aracını (Axure) değerlendiriniz?



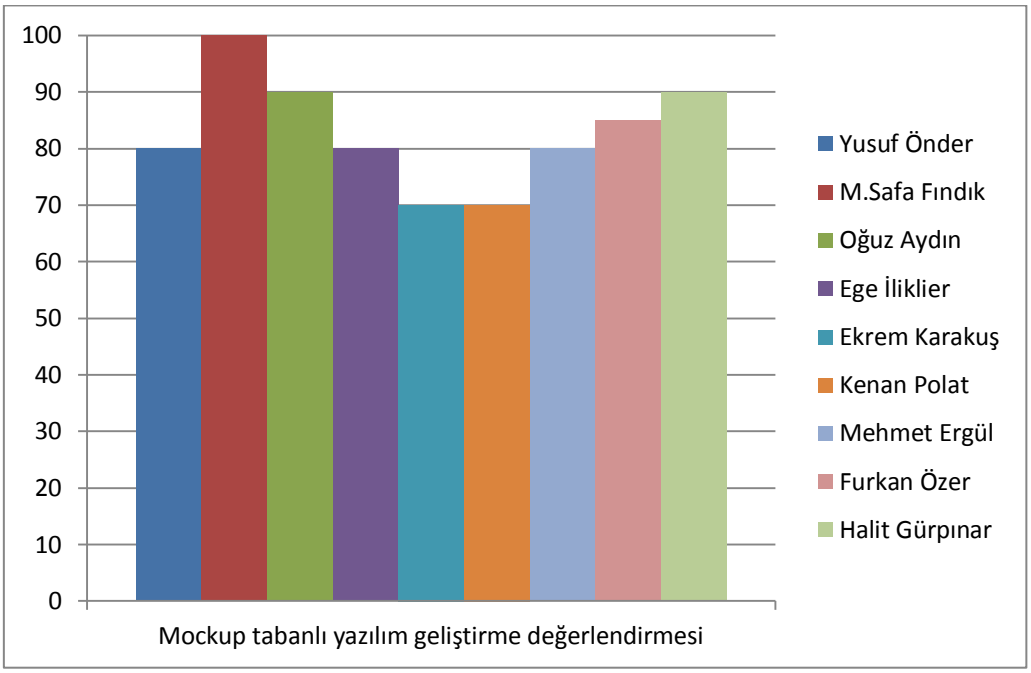
Şekil 5. 2 Kullanılan yazılım geliştirme aracının değerlendirmesi

3. Kullandığınız yaklaşımı uygulama kolaylığı yönünden değerlendiriniz?



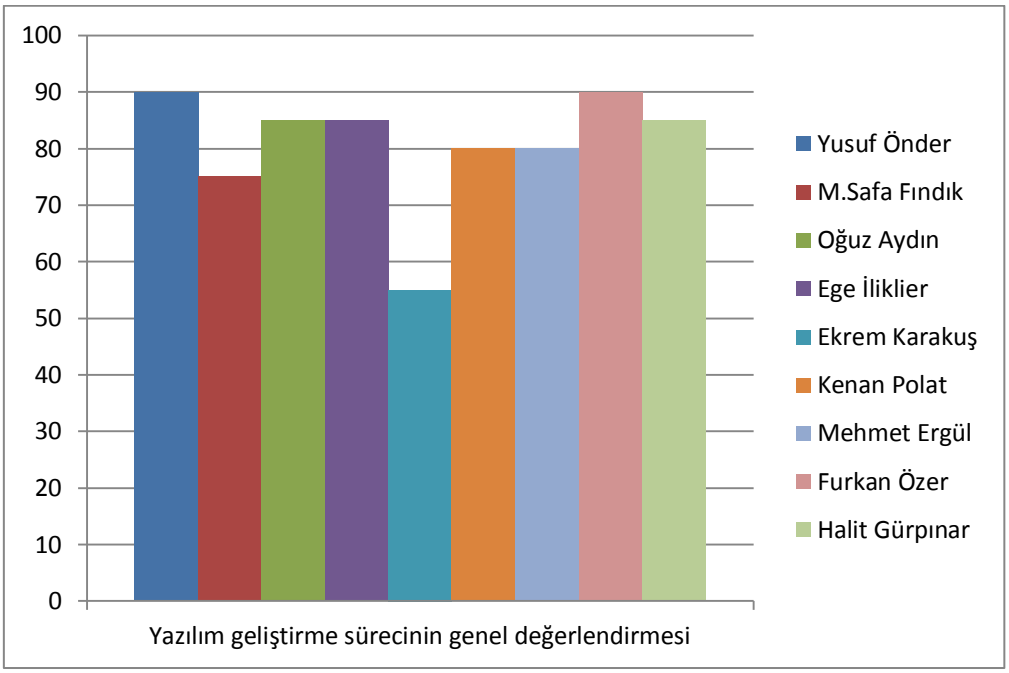
Şekil 5. 3 Yaklaşımın uygulama kolaylığı yönünden değerlendirmesi

4. Mockup ile web uygulaması geliştirmeyi değerlendiriniz?



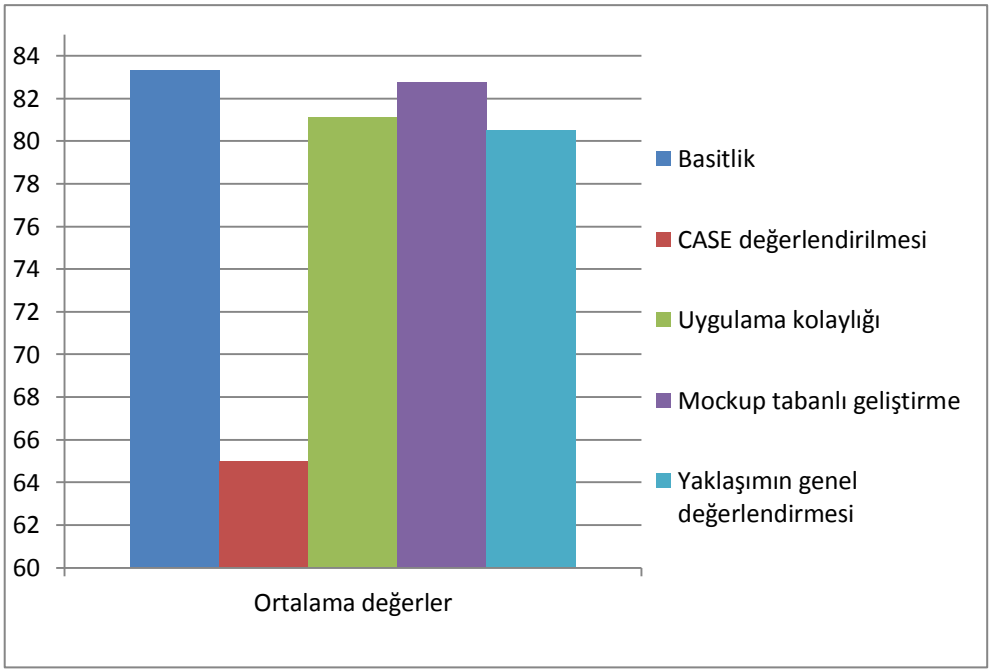
Şekil 5. 4 Mockup tabanlı geliştime değerlendirmesi

5. Tüm süreci genel anlamıyla değerlendiriniz?



Şekil 5. 5 Yazılım geliştirme sürecinin genel değerlendirmesi

Değerlendirme ortalamaları 80 ile 83 arasında çıkmıştır. Bu düzeni bir tek yardımcı geliştirme aracı değerlendirmesi daha düşük oran ile bozmaktadır. Yapılan değerlendirmede proje grup elemanlarının yaklaşımda genel olarak zorluk yaşadıkları kısım Şekil 5. 6'da görüldüğü üzere yardımcı geliştirme aracının kullanılması yönündedir.



Şekil 5. 6 Sayısal verilerin ortalama değerleri

İlk defa yeni bir geliştirme aracı kullanılması, öğrenci projeleri için kısıtlı zamanda yeni bir proje ortaya koymakta zorluk yaratmıştır. Bu sebeple, yeni bir yaklaşımda öncelikli olarak yardımcı geliştirme araçlarının iyi etüt edilip, tam olarak öğrenilmesi ve deneyimlenmesi ön plana çıkmaktadır. Uygulama kolaylığı ve yaklaşımın basitliği genel olarak olumlu olarak değerlendirilebilir. Web uygulaması geliştirenler için mockup tabanlı yazılım geliştirme yaklaşımı, proje grupları tarafından beğenilmiş ve uygulama da kolaylık sağlamıştır.

5.4 Uygulama Sonrası Değerlendirme

Yazılım mühendisliği dersi kapsamında birlikte çalıştığımız gruplar dışındaki gruplar klasik yöntem ile web uygulaması geliştirmesi yapmışlardır. Bu kapsamda diğer proje gruplarının yaptıkları çalışmaları incelediğimizde iki yönden bizim sağladığımız avantajları elde edememişlerdir. Bunlar mockup kullanımın sağladığı avantajlar ile çevik sürecin klasik sürece göre avantajlarıdır. Çevik sürecin avantajları tezin önceki bölümlerinde incelenmiştir. Mockup tabanlı yazılım geliştirme mockup'lar ile analiz ve tasarım yaparak bu aktivitelerin seri bir şekilde üretilmesini sağlamaktadır. Mockup geliştirme araçlarını kullanmak yaklaşımımızı kullanan proje gruplarına otomatik kod dönüşümü avantajı sağlamıştır. Kullandığımız *Axure* yazılım geliştirme aracı ile istemci

taraflı programlama bölümlerini mockup geliştirme araçları ile yapılmıştır. Bu sayede kodlama bölümünde de bu aktivitelerin daha hızlı bir şekilde geliştirilmesi sağlanmıştır. Ayrıca müşteri ile iletişim de mockup araçlarının çıktıları üzerinden yapılarak kullanıcı gereksinim onayı daha anlaşılabilir bir ortamda yapılmıştır. Yapılan tasarım ve kodlama üzerinde istenilen değişikliklerin yapılması daha esnek bir temele oturtulmuştur. Ayrıca mockup tabanlı araçların gelişimini takip ederek, web uygulamalarındaki teknolojik gelişmeyi araçlar yardımı ile daha kolay sağlanabilmiştir.

BÖLÜM 6

SONUÇ VE ÖNERİLER

Uygulama çalışmasında öğrencilerin yaşadığı en büyük sıkıntı, otomatik kod dönüşümü ile oluşturulan kodlar ile sunucu tabanlı kodların birleştirilmesidir. Mockup tabanlı araçlar, dönüşüm sonucunda beklenenden çok uzun kod parçası üretmiştir. Basit bir web sitesi tasarımı için binlerce satır HTML,CSS ve JavaScript kodu üretmiştir. Bu uzunluğun sebebi en basit bir gösteriminin bile ayrı ayrı etiketler içerisinde tanımlanarak dönüşüm yapılmasıdır. Aslında basit HTML ifadelerinden oluşan bu kod yığını, öğrenciler için karmaşıklık yaratmıştır. Bizim bu konudaki önerimiz, koda değil, modele odaklanmalarıdır. Kod üzerinde geliştirme yapmaya alışkın öğrenciler için modeller ile geliştirmeye adapte olup, eski uyguladıkları yöntemleri bırakmaları ve alışmaları kolay olmamıştır. Bu sebeple bu yöntemin düzenli takip edilme ihtiyacı vardır. Kod birleşimi için, etiketler ile HTML içerisine server tabanlı kod parçalarının yerleştirmeleri önerilmiştir. Bunun için örneğin "<% %>" etiketleri kullanılarak ilgili yerlerde ASP.NET programlama dili kullanmaları tavsiye edilmiştir.

Yaklaşım ile görsel olarak programlama yapılmış, basitlik yönünden beğenilmiştir. Sistem analizi görsel tasarım ile yapılarak gereksinimlere uygun olarak geliştirme yapılması sağlanmıştır. Otomatik kod dönüşümü ile kodlama aktivitesi kısa bir süreye indirgenmiş ve yazılım geliştirme hızı arttırılmıştır. Hatalı kod yazımı ve tespiti gibi sıkıntılar minimalize edilmiştir. Kod yazımınında insan faktörünün minimalize edilmesi, gelişen teknolojinin gereği olarak düşünülebilir. Bu düşünce ile çok sayıda yardımcı araç üretilmiş ve sürekli olarak geliştirilmektedir. Bu araçların kullanımı ile yazılım geliştirme ekiplerinin daha hızlı üretim yapmaları ve gelişen teknolojiyi takip etmeleri

kolaylaştırılmıştır. İnsan faktörünün azaltılması ile, üretim maliyetinin düşürülmesi sağlanmaktadır. Bunun yanısıra test süreçleri de kısalmaktadır. Bu sayede, yazılım geliştirme ekipleri, kaynak kodun üretilmesi, kod hatalarının tespiti gibi konular yerine sistem analizine ve mimariye daha çok vakit ayırabilmektedir.

Eğer sadece model tabanlı yazılım geliştirme, şelale yöntemi kullanılarak yapılsaydı, uzun analiz ve tasarım süreçleri modeller üzerinde yapılması gerekecekti. Bu durumda model tabanlı geliştirme araçlarının avantajlarından faydalanılabilecekti, ancak yazılım sürecinde esneklik sağlanamayacaktı. Müşteri isteklerinin süreç içerisinde değişmesi durumunda veya yapılan tasarımın beğenilmemesi durumunda geçen zaman boşa gitmiş olacaktı. Kısacası bu durumda model tabanlı geliştirme avantajı yakalanırken, klasik geliştirmenin sebep olduğu dezavantajlar yerinde kalacaktı.

Ancak yazılım geliştirme ekiplerini bu yöntemlere adapte etmek kolay olmamaktadır. Klasik yöntemden veya çevik yöntemden, çevik model tabanlı geliştirmeye geçiş yapılması için ön çalışma sürecine ihtiyaç vardır. Eğer tam verim alınması isteniyor ise, eski alışkanlıklarından kopup, yeni prensiplere sıkı sıkıya bağlı bir şekilde uygulama gerçeklemesi gerekmektedir. Ayrıca, araçların seçimi ve kullanımı içinde ciddi bir hazırlık dönemi gerekmektedir.

Sonuç olarak, yaklaşımın küçük ölçekli web uygulamalarında uygulanabilirliği kanıtlanmıştır. Ancak büyük ölçekli projelerde uygulanabilirliği araştırmaya açık bir alandır. Bu amaç ile, yazılım şirketleri ile birlikte bir çalışma verimli olabilir. Ayrıca koddan mockup'a dönüşüm gibi tersine mühendislik çalışması da incelenebilecek çalışmalardandır.

KAYNAKLAR

[1] Schmidt, D.C., (2006). “Model-Driven Engineering”, IEEE Computer , 39(2):25-31.

[2] Ambler, S.W., (2004). Agile Model Driven Development with UML 2.0, Third Edition, Cambridge University Press, New York.

[3] Ambler, S.W., (2007). “Agile Model Driven Development” , XOOTIC Magazine, 12(1): 13-21.

[4] Guta, G., Schreiner, W. ve Draheim, D., (2009), “A Lightweight MDSD Process Applied in Small Projects”, In Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications, 27-29 August 2009, Patras, Greece.

[5] Kirby, J., (2006). “Model Driven Agile Development of Reactive Multi Agent Systems”, In Proceedings of the 30th Annual International Computer Software and Applications Conference, 17-21 September 2006, Chicago, United States of America

[6] Zhang, Y. ve Patel, S., (2011). “Agile Model-Driven Development in Practise”, IEEE Software , 28(2): 84-91.

[7] Cockburn, A., (2001). Agile Software Development, First Edition, Addison-Wesley, Boston, United States of America.

[8] Selic, B., (2003). “The Pragmatics of Model-driven Development”, IEEE Software, 20(5): 19-25.

[9] Seidewitz, E., (2003). "What Models Mean", IEEE Software, 20(5):26-32.

[10] Nguyen, V.C. ve Qafmolla, X., (2010). “Agile Development of Platform Independent Model in Model driven Architecture”, 3th International Conference on Information and Computing, 4-6 June 2010, Wuxi, China, 344-347.

[11] Stahl, T. ve Volter, M., (2006). Model-Driven Software Development, First Edition, John Wiley & Sons, Chichester, United Kingdom.

[12] ArgoUML, http://argouml.tigris.org/, 26 Nisan 2015.

[13] Hailpern, B. ve Tarr, P., (2006). “Model-driven development: The Good, the Bad, and the Ugly”, IBM Systems Journal, 45(3): 451-461.

[14] 15 reasons why you should start using Model Driven Development, http://www.theenterprisearchitect.eu/blog/2009/11/25/15-reasons-why-you-should-start-using-model-driven-development/, 27 Nisan 2015.

[15] Ambler, S.W., (2003). "Agile Model Driven Development Is Good Enough, Point-Counterpoint article in IEEE Software, 20(5): 71-73.

[16] The practices of Agile Modeling, http://agilemodeling.com/practices.htm, 27 Nisan 2015.

[17] Dyba, T. ve Dingsory, T., (2009). "What Do We Know about Agile Software Development?", IEEE Software, 26(5) :6-9.

[18] Astels, D., (2003). Test Driven Development: A Practical Guide, First Edition, Prentice Hall, New Jersey, United States of America.

[19] Tilley, S., (2004). "Test-Driven Development and Software Maintenance", Proceedings of the 20th IEEE International Conference on Software Maintenance , 11-14 September 2004, Chicago, United States of America, 488-491.

[20] Matinnejad, R., (2011). "Agile Model Driven Development: An Intelligent Compromise" , 9th International Conference on Software Engineering Research, Management and Applications , 10-12 August 2011, Maryland, United States of America, 197-202.

[21] Rivero, J.M., Grigera J., Rossi, G., Luna, E.R., Montero, F. ve Gaedke, M., (2014). "Mockup-Driven Development: Providing agile support for Model-Driven Web Engineering", Information and Software Technology, 56(6): 670-687.

[22] Benson, E., (2013). "Mockup Driven Web Development", 22nd Intenational World Wide Web Conference, 13-17 Mayıs 2013, Rio de Janeiro, Brazil, 337-342.

[23] Basso, F., Pillat, M., Frantz, F.R. ve Frantz, R.Z., (2014). "Study on Combining Model-Driven Engineering and Scrum to Produce Web Information Systems", 16th International Conference Enterprise Information Systems, 27-30 April 2014, Lisbon, Portugal, 137-144.

[24] Rivero, J. M., Grigera, J., Rossi, G., Luna, E.R., ve Koch, N., (2011). "Improving agility in model-driven web engineering", Proceedings of the CAiSE Forum, 22-24 June 2011, London, United Kingdom, 163-170.

[25] Rivero, J.M., Rossi, G., Grigera J., Luna, E.R., Navarro, A., (2011). "From interface mockups to web application models", Proceedings of the 12th international conference on Web information system engineering, 13-14 October 2011, Sydney, Australia, 257-264.

[26] Lecarpentier, J., Crosnier, H.L., Brixtel, R. ve Bazin, C., (2014). "Document Model and Prototyping Methods for Web Engineering", International Journal of Information System Modeling and Design, 5(4): 91-117.

[27] Creating Your First Mockup | Balsamiq, http://support.balsamiq.com/customer/portal/articles/871902-creating-your-first-mockup, 03 Mart 2015.

[28] Alpaslan, G. ve Kalıpsız, O., (2015). “Hybrid Mockup-driven Development: An Agile Model-driven Development for Web Applications”, The First International Conference on Advances and Trends in Software Engineering, 19-24 Nisan 2015, Barselona, Spain, 56-59.

EK-A

UYGULAMA GEREKSİNİM ANALİZİ

Bu bölümde öğrenci grupları ile yapılan uygulama çalışmasının gereksinim analizleri gösterilmiştir.

A-1 Sinema Bilet Sistemi Projesi için Gereksinim Analizi

Birinci iterasyon için gereksinim analizi aşağıda gösterilmiştir.

- G.4.1.1 Admin giriş paneli olmalıdır.

Öncelik : 1

Açıklama : Sitenin yönetimi için admin paneli en öncelikli koşuldur.

- G.4.1.2 Admin filmlerin salonları ve seanslarını belirleyebilmelidir.

Öncelik : 1

Açıklama : Sitenin güncel kalması için gerekli bir koşuldur.

- G.4.1.3 Admin tadilat durumunda bir salonu kapatabilecek

Öncelik : 2

Açıklama : Beklenmedik durumlara karşı admine salonu tamamen kapatma yetkisi verilecek.

- G.4.1.4 Site müşteri kullanıma hazır hale gelecek

Öncelik : 1

Açıklama : Müşterilerin anasayfa, filmlerin seans ve salonlarını görme, vizyondaki ve

gelecek filmleri görme gibi bazı hizmetler verebilecek duruma gelecektir.

İkinci iterasyon için gereksinim analizi aşağıda gösterilmiştir.

- G.4.2.1 Site müşterilerin kullanımına geçilmiş olmalı

Öncelik : 1

Açıklama : Projede müşteri websitesini kullanmaya başlayıp görüş bildirmeye başlayarak etkin bir rol alacak.

- G.4.2.2 Müşteri vizyondaki ve gelecek filmleri görebilmeli

Öncelik : 1

Açıklama : Müşteri anasayfa yardımıyla ile vizyondaki ve gelecek filmlerin bulunduğu sayfalara gidebilmelidir.

- G.4.2.3 Müşteri filmlerin seans ve salonlarını görebilecek

Öncelik : 1

Açıklama : Müşteri vizyondaki filmlerin seans ve salonları hakkında bilgi sahibi olabilecek.

Üçüncü iterasyon için gereksinim analizi aşağıda gösterilmiştir.

- G.4.3.1 Müşteri bilet alma paneli oluşturulmalı

Öncelik : 1

Açıklama : Müşterinin online bilet alabilecek bir bilet alma sistemi yapılacaktır.

- G.4.3.2 Müşteri bilet alırken istediği filmin salon ve seans seçebilecek

Öncelik : 1

Açıklama : Müşterinin online bilet alırken gitmek istediği filmin kendine uygun salon ve seansını seçebilecek.

- G.4.3.3 Müşteri salondaki dolu koltukları görüp, boş koltuklardan birini seçebilecek

Öncelik : 1

Açıklama : Müşteri boş koltuklardan istediği yeri seçerek bilet satın alma işlemine devam edecektir.

- G.4.3.4 Müşteri online ödeme yapabilecek

Öncelik : 1

Açıklama : Müşteri biletin ücret tutarını online ödeyerek bilet alma işlemini tamamlamış olacak.

A-2 Dil Kursu Otomasyon Projesi için Gereksinim Analizi

Birinci iterasyon için gereksinim analizi aşağıda gösterilmiştir.

- G.1.1 Kullanıcılar Sisteme Mail Ve Şifreleriyle Giriş Yapabilmelidir

Öncelik : 1

Açıklama : Kullanıcılar kendilerine verilen mail adresi ve şifreyi kullanarak sisteme giriş yapabilmelidir.

- G.1.2 Kullanıcılar Üyelik Bilgilerini Görebilmelidir

Öncelik : 1

Açıklama : Kullanıcılar üyelik bilgilerini kendilerine ait sayfadan görebilmelidir

- G.1.3 Kullanıcılar Üyelik Bilgilerini Değiştirebilmelidir

Öncelik : 1

Açıklama : Kullanıcılar üyelik bilgilerini kendilerine ait sayfadan değiştirebilmelidir.

- G.1.4 Öğretmenler Not Bilgisi Girebilmelidir

Öncelik : 1

Açıklama : Öğretmenler verdikleri dersler için öğrencilerin not girişlerini yapabilmelidir.

- G.1.5 Öğretmenler Not Bilgisi Değiştirebilmelidir

Öncelik : 1

Açıklama : Öğretmenler verdikleri dersler için öğrencilerin not girişlerini değiştirebilmelidir.

- G.1.6 Öğretmenler Devamsızlık Bilgisi Girebilmelidir

Öncelik : 1

Açıklama : Öğretmenler verdikleri dersler için öğrencilerin devamsızlık girişlerini yapabilmelidir.

- G.1.7 Öğretmenler Devamsızlık Bilgisi Değiştirebilmelidir

Öncelik : 1

Açıklama : Öğretmenler verdikleri dersler için öğrencilerin devamsızlık girişlerini değiştirebilmelidir.

- G.1.8 Kayıt Görevlisi Yeni Öğrenci/Öğretmen Kayıtları Ekleyebilmelidir

Öncelik : 1

Açıklama : Kayıt görevlileri kurs şubesine yeni öğrenci/öğretmen kayıtları ekleyebilmelidir.

- G.1.9 Öğrenciler kayıt oldukları dersler için ödeme yapabilmelidir.

Öncelik : 1

Açıklama : Öğrenciler kayıt oldukları her ders için o dersin ücretini ödeyebilmelidir.

İkinci iterasyon için gereksinim analizi aşağıda gösterilmiştir.

- G.2.1 Kayıt ile ilgili kişi sisteme giriş yapmalıdır.
- G.2.2.Başarılı giriş yapıldığı takdirde yetkisine göre ilgili sayfaya yönlendirilmeli.
- G.2.3 Eğer giriş yapan öğrenci ya da hoca ise kendi ders kontrolünü yapabilir, yönetici ise kayıt işlemlerini gerçekler.

Üçüncü iterasyon için gereksinim analizi aşağıda gösterilmiştir.

- G.3.1 Giriş yapan öğrenci aldığı dersleri ve derslerin programlarını görebilir.
- G.3.2 Öğretmen ise verdiği derslerin programını görebilir veya hangi dersleri vereceğini kendi sayfasından öğrenebilir.

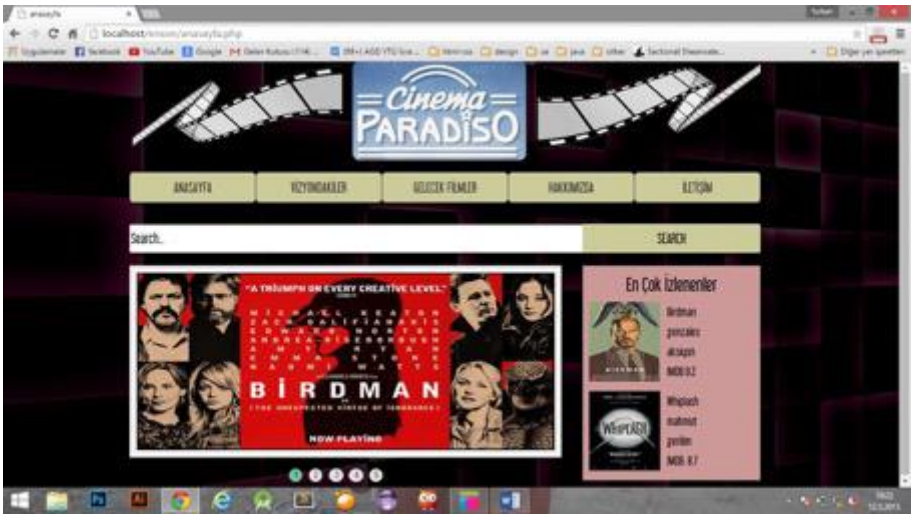
EK-B

MOCKUP TASARIMLARI

Bu bölümde, yapılan uygulama çalışmasında üretilen mockup tasarımları gösterilmiştir.

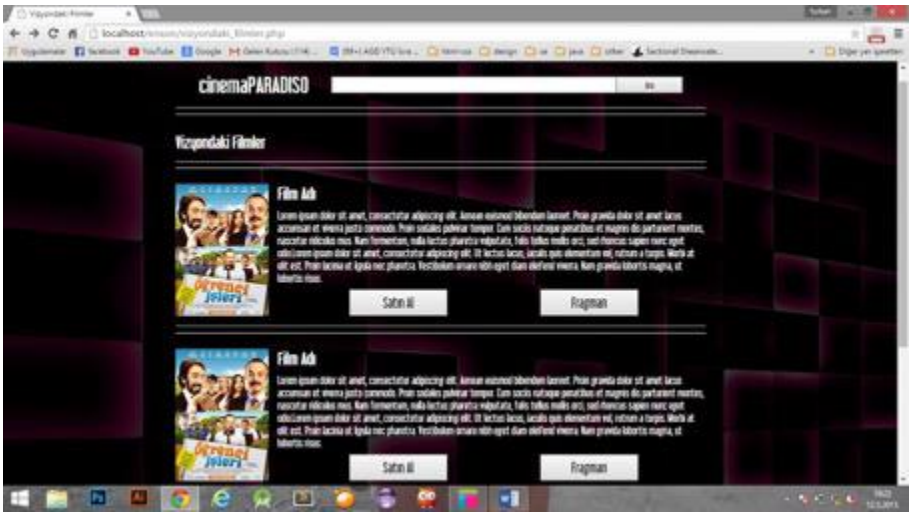
B-1 Sinema Bilet Sistemi Projesi için Mockup Tasarımları

Sinema bilet sistemi projesi için mockup tasarımları aşağıda gösterilmiştir. Şekil B.1'de ana sayfa mockup tasarımı gösterilmiştir.



Şekil B. 1 Sinema bilet sistemi projesi ana sayfası

Şekil B.2'de vizyondaki filmleri gösteren ve film seçilmesinin yapıldığı ilgili sayfa gösterilmiştir.

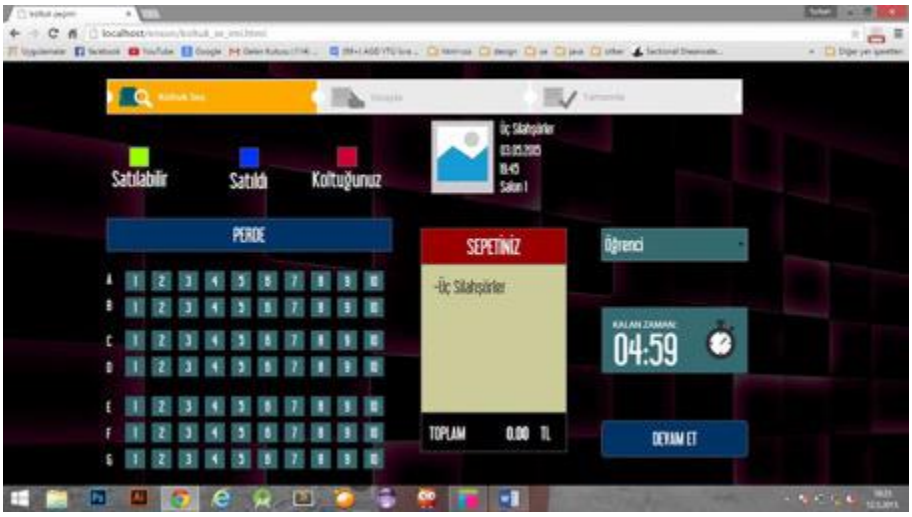


Şekil B. 2 Film seçme sayfası

Seans ve salon seçiminin yapıldığı sayfa Şekil B.3'de gösterilmiştir.

Şekil B. 3 Seans ve salon seçme sayfası

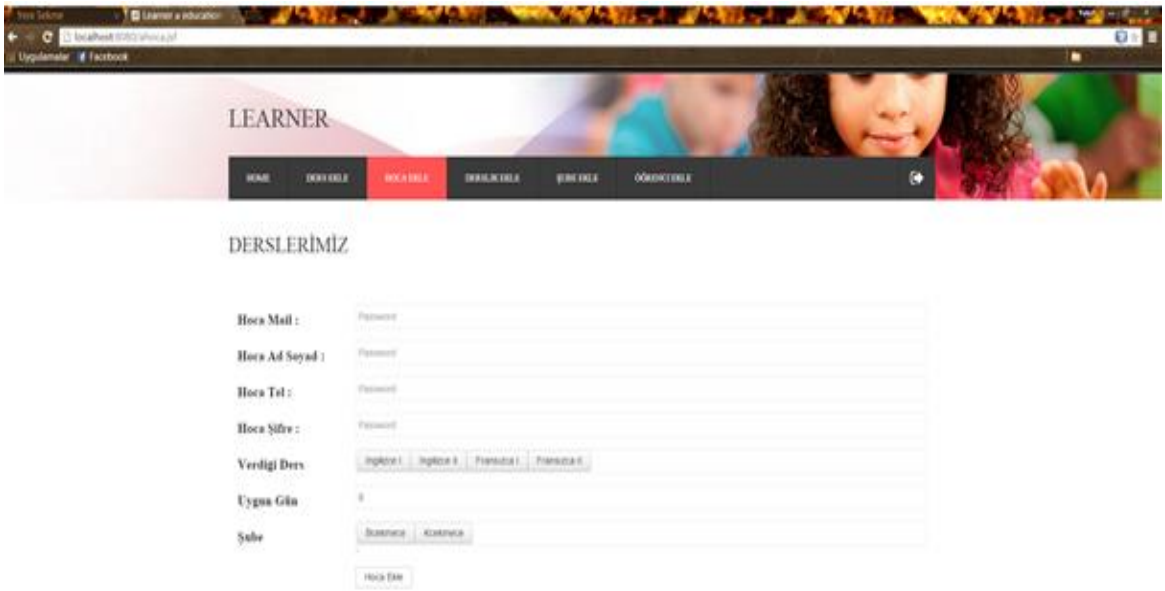
Koltuk seçiminin yapıldığı ilgili sayfa Şekil B.4'de gösterilmiştir.



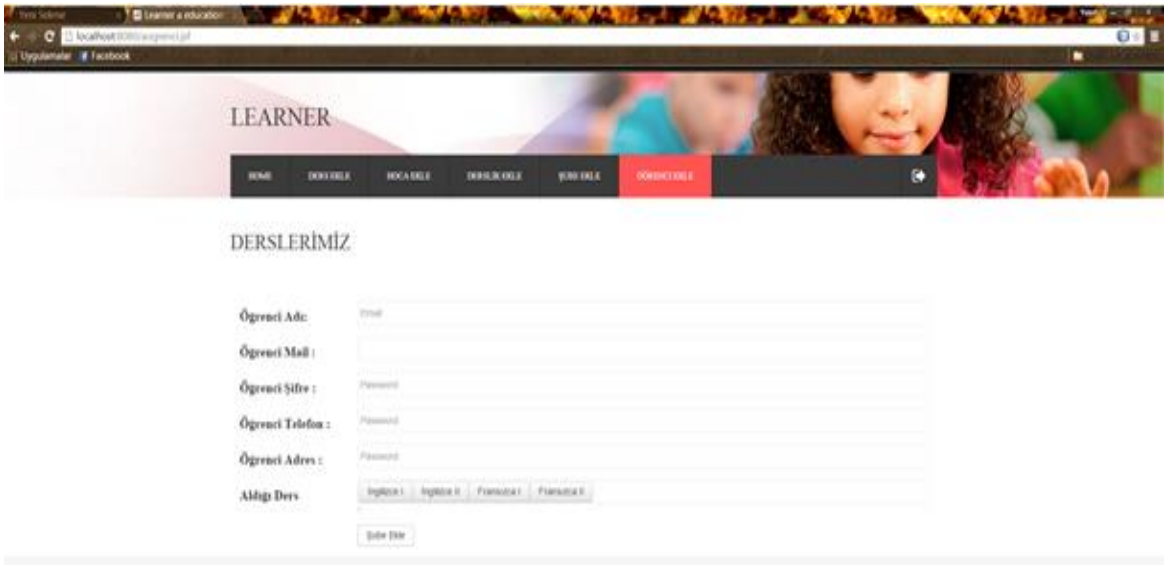
Şekil B. 4 Koltuk seçme sayfası

B-2 Dil Kursu Otomasyon Projesi için Mockup Tasarımları

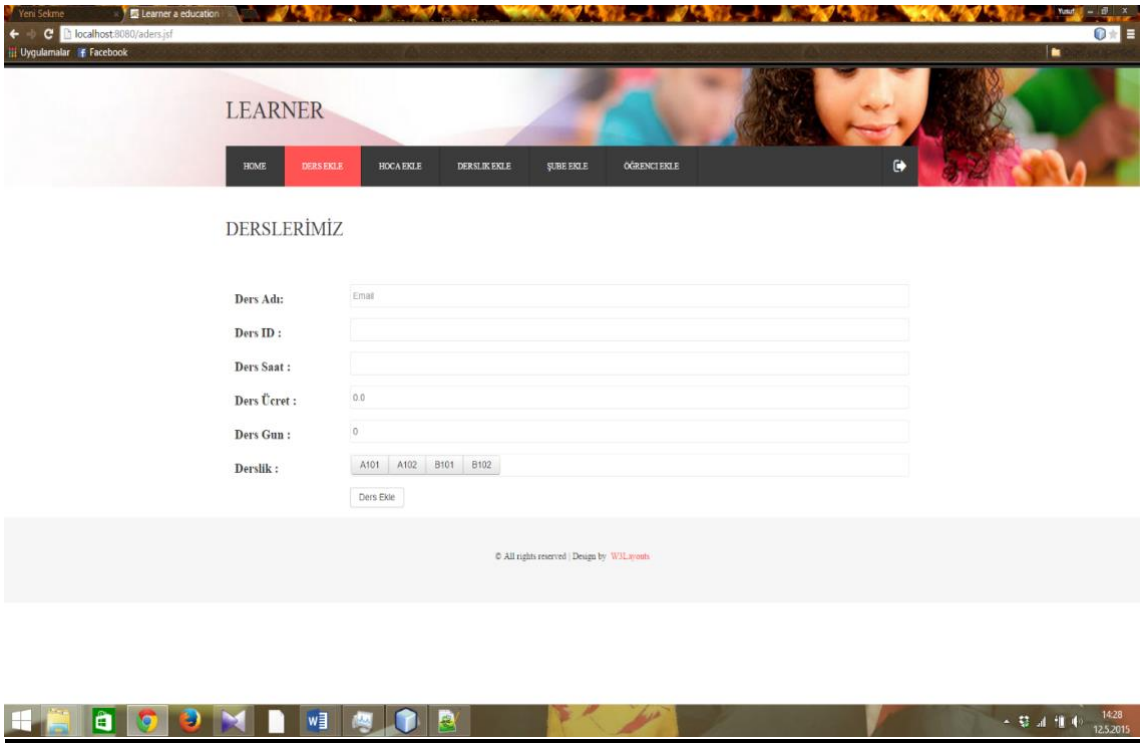
Dil kursu otomasyon projesi için mockup tasarımları aşağıda gösterilmiştir.



Şekil B. 5 Dil kursu otomasyon projesi öğretmen kaydı ekleme sayfası



Şekil B. 6 Dil kursu otomasyon projesi öğrenci kaydı ekleme sayfası



Şekil B. 7 Dil kursu otomasyon projesi ders kaydı ekleme sayfası

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Gürkan ALPASLAN

Doğum Tarihi ve Yeri : 21/09/1989 – Ankara

Yabancı Dili : İngilizce

E-posta : gurkan89@hotmail.de

ÖĞRENİM DURUMU

Derece	**Alan**	**Okul/Üniversite**	**Mezuniyet Yılı**
Lisans	Bilgisayar Mühendisliği	İstanbul Üniversitesi	2012
Lise	Fen Bilimleri	Ankara Atatürk Anadolu Lisesi	2007

YAYINLARI

Ulusal Bildiri

1. **Alpaslan, G.** ve Kalıpsız, O., (2014). "Çevik Süreç ile Model Tabanlı Yazılım Geliştirme", Elektrik - Elektronik, Bilgisayar ve Biyomedikal Mühendisliği Sempozyumu, 27-29 Kasım 2014, Bursa.

2. **Alpaslan, G.** ve Kalıpsız, O., (2014). "Model Tabanlı Çevik Süreç Yöntemlerinin İncelenmesi ", Ulusal Yazılım Mimarisi Konferansı, 3-4 Aralık 2014, Ankara.

Uluslararası Bildiri

1. **Alpaslan, G.** ve Kalıpsız, O.,(2015). “Hybrid Mockup-driven Development: An Agile Model-driven Development for Web Applications”, The First International Conference on Advances and Trends in Software Engineering, 19-24 Nisan 2015, Barselona.