AIntelia

Termal Görüntüleme Sistemleri İçin Yazılım Ürün Hattı Tasarımı

Zeynep Telli[1], Hacer Karacan[2*]
[1]*Yazılım Tasarım Müdürlüğü, MGEO, ASELSAN A.Ş., ztekdas@aselsan.com.tr*
[2*] *Bilgisayar Mühendisliği Bölümü, Mühendislik Fakültesi, Gazi Üniversitesi, hkaracan@gazi.edu.tr*

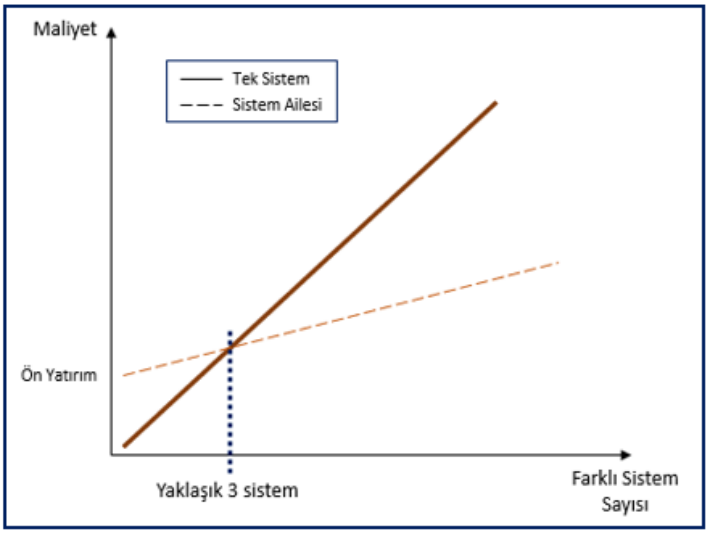
Yazılım geliştirme süreçlerinin hızlı, düşük maliyetli ve kaliteli bir şekilde yeniden kullanılabilecek şekilde tasarlamanması günümüzde aranan yaklaşımlardan biridir. Yazılım geliştiren birçok firma farklı özelliklere sahip fakat temelde birbirine çok benzeyen yazılımlar üretmektedir. Yazılım mühendisliği her ne kadar orijinal geliştirmeye daha çok odaklansa da kısa zaman içerisinde düşük maliyetli ve kaliteli bir yazılım ürünü ortaya çıkarmak istemektedir. Bu da yazılımın yeniden kullanımını benimseyen bir tasarım yaklaşımıyla mümkündür. Yazılım ürün hattı bu yaklaşımlardan biridir. Yazılım ürün hatları, belirli bir ürün ailesinin ortaklıklarını ve değişkenliklerini göz önünde bulundurarak temel varlıkları oluşturmayı ve bu varlıkların yeniden kullanımını sağlayarak, yazılım geliştirme sürelerinin ve maliyetlerinin azaltılmasında önemli avantajlar sağlayan bir yaklaşımdır. Bu nedenle YÜH yaklaşımı, klasik yaklaşımda da kullanılan yeniden kullanım yönteminin çok daha sistematik bir şekilde kullanılması temel almaktadır. Yazılım ürün hattı mühendisliğinde Alan Mühendisliği ve Uygulama Mühendisliği olmak üzere birbirine paralel iki ayrı süreç bulunmaktadır. Bu bildiride, yazılım ürün hattı yaklaşımı çerçevesinde gerçekleştirilen alan mühendisliği çalışmaları anlatılacaktır. Bu kapsam da yazılım ürün ailesi kullanan 20 farklı yazılım incelenerek, öncesinde belirlenen ölçütler çerçevesinde 4 pilot yazılım belirlenmiştir. Bu yazılımlar üzerinde çalışarak, problem tanımı çıkarılmıştır. Problem tanımına göre referans mimarisi oluşturulmuştur. Yazılımlar arasındaki ortaklıklar ve farklılıklar belirlenerek bir mimari tasarım ortaya çıkarılmıştır.

Keywords: *Yazılım Ürün Hattı, Özellik Modeli, Alan Mühendisliği, Yazılım ürün hattı*

1. Giriş

Yazılım, modern ve rekabetçi ürünler arasında giderek popülerliği artan bir varlık haline gelmiştir. Basit veya karmaşık, büyük veya küçük olmasına bakılmaksızın, yazılımsız neredeyse hiç modern ürün bulunmamaktadır. Bu nedenle yazılım geliştirme, şirketler açısından büyük bir öneme sahiptir [1]. Yazılım müşterilerinin artan bir hızla yeni ürünler ve hizmetler talep etmesi, yazılım endüstrisini süreçlerinin verimliliğini ve ürünlerinin kalitesini arttırmayı sağlayan yeni yaklaşımlar geliştirmeye zorlamıştır [2]. Yazılım mühendisliği orijinal geliştirmeye daha çok odaklansa da kısa zaman içerisinde düşük maliyetli ve kaliteli bir yazılım ürünü ortaya çıkarmak, yazılımın yeniden kullanımını benimseyen bir tasarım yaklaşımıyla mümkündür. Yazılım ürün hattı bu yaklaşımlardan biridir.

Yazılım ürün hattı (YÜH), belirli bir pazar kesiminin ya da görevin ihtiyaçlarını karşılamak için, ortak yetenek kümelerini kullanan sistemlerden oluşan yapıya verilen isimdir. Yazılım ürün hattı tasarlanırken tüm ürün hattı için değişken ve ortak olan gereksinimler bir araya getirilerek yeniden kullanımın artırılması amaçlanmaktadır. Genellikle ekonomik hususlara dayanan sebepler, şirketleri yazılım ürün hattı mühendisliği (YÜHM) yaklaşımına yönlendirmiştir [4]. YÜHM yeniden kullanımı desteklemesi nedeni ile, maliyetleri düşürmekte, piyasaya sürülme süresini kısaltmakta ve ortaya çıkan ürünün güvenirliliğini artırmaktadır. Fakat yeniden kullanımın sağlanabilmesi için bir ön yatırım gerekmektedir. Bu yeniden kullanılabilir varlıklar oluşturmak, organizasyonu dönüştürmek vb. için gereklidir [3]. Yaklaşık 3 üründen sonra YÜHM yaklaşımı ile maliyetlerin azaldığı Şekil 1'de görülmektedir.



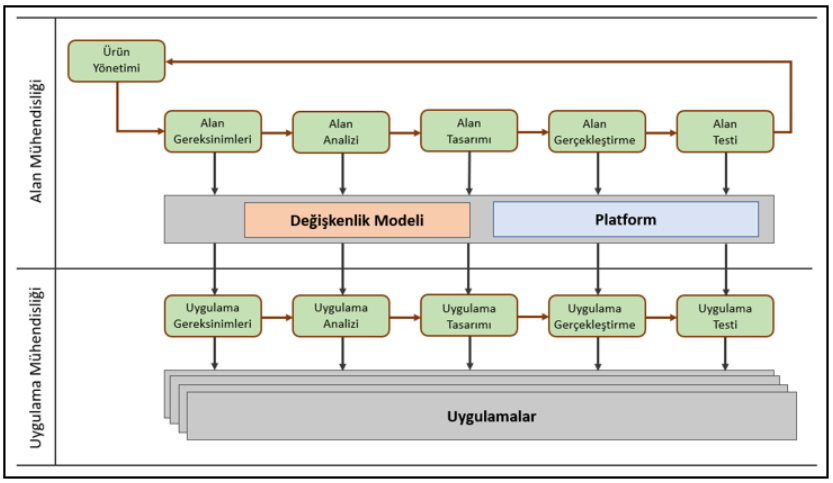
Şekil 1. *Yazılım Ürün Hattı Yaklaşımın Maliyete Etkisi [3]*

Yazılım ürün hattı yaklaşımı, belirli bir alanda, düşük maliyetli ve kaliteli yazılımları hızlı bir şekilde geliştirmeye yönelik ortaya atılmıştır [5]. Bu yaklaşım, belirli bir ürün ailesinin ortaklıkları ve değişkenlikleri göz önünde bulundurarak temel varlıkları oluşturmayı ve bu varlıkları kullanarak yazılımların geliştirilmesini söylenmektedir [6]. Bu amaçla YÜH yaklaşımı, klasik yaklaşımda da kullanılan yeniden kullanım yönteminin çok daha sistematik bir şekilde kullanılması ve sadece yazılım kodlarında değil, her türlü yazılım varlığında (gereksinim, tasarım, kod, test tanımı) uygulanmasını temel alır [7].

2. Yazılım Ürün Hattı Mühendisliği veya Yazılım Ürün Hattı Tasarımı

Yazılım ürün hattı mühendisliği, ürünler arasındaki değişkenliği ve benzerliği kullanarak bir yazılım ürünleri ailesinin sistematik olarak geliştirmesini sağlamaktadır. Kitlesel özelleştirme, yazılım geliştirmenin hem geliştiriciler hem de kullanıcılar için daha ekonomik hale gelmesi için düzenli yeniden kullanım ile elde edilir. Farklı ürünlerin temel varlıkları bir platform olarak sunulmaktadır. Ürünler arasındaki ortaklıklar ve farklılıklar, değişkenlik modellerinde yakalanmaktadır [8].

Ürün hattı mühendisliğinin amaçladığı hedefler arasında, geliştirme maliyetini azaltmak, kaliteyi artırmak, piyasaya sürme süresini azaltmak ve karmaşıklık ile başa çıkmak vardır. Ürün hattı mühendisliği ayrıca özel ihtiyaç ve isteklerine uyarlanmış ürünleri alabilmelerine imkan vermesi sebebiyle müşterilere de fayda sağlamaktadır. Yazılım ürün hattı mühendisliği de yazılım projelerinde bu yaklaşımı kullanarak yeniden kullanımı sağlamakta ve hem üretici hem de müşteriler için aynı faydaları sunmaktadır. Yakın zamana kadar yazılım, özellikle gömülü yazılım, nispeten küçük ölçekli ve her ürün varyantının kendi yazılım çeşitliliğine sahip olduğu bir şekilde geliştirilmiştir. Ancak, günümüzde bu durum değişikliğe uğramıştır. Çevremizdeki sistemler yazılımın yoğun olduğu sistemlere dönüşmeye başlamıştır. Çünkü yazılım donanımdan daha esnek yapıdadır. Bu da ürünlere donanımdan bağımsız yeni işlevsellikler sunmaktadır. Ayrıca, donanımla kıyaslandığında yazılımı kopyalamak, taşımak ve değiştirmek daha kolay ve ucuzdur. Gömülü sistem ürünleri dışında, yazılım genellikle değişken değildir. Bir müşteri, ihtiyaç duyabileceği tüm olası özellikleri içeren bir yazılım sistemi satın alabilir veya tek bir müşteri siparişi ile yazılım tek bir amaç için üretilebilir [3]. Fakat gömülü yazılımlar düşünüldüğünde değişiklik çok fazladır ve gömülü sistem ürünlerinde karmaşıklığın neredeyse tamamını yazılım oluşturmaktadır. Geleneksel yazılımlar ile bu karmaşıklığı gidermek mümkün olmamaktadır. Bu da yazılımcıları yazılım ürün hattı mühendisliğine yönlendirmiştir.



***Şekil 2**. Yazılım Ürün Hattı Mühendisliği Çerçevesi [1]*

YÜHM alan mühendisliği ve uygulama mühendisliği olmak üzere iki süreç üzerinden açıklanmıştır (Bkz. Şekil 2). Alan mühendisliği süreci tekrar kullanılabilir bir platformun geliştirilmesinden, dolayısıyla ürün hattının ortaklık ve değişkenlik tanımının yapılmasından sorumludur [9]. Uygulama mühendisliği sürecinde ürün hattı uygulamaları, alan mühendisliğinde tanımlanan ürün hattı değişkenliğinden faydalanılarak, büyük oranda varlıkların yeniden kullanımı ile oluşturulmaktadır.

A. Alan Mühendisliği:

Alan mühendisliği, yeniden kullanılabilir platformun kurulmasından ve böylece ürün hattının ortaklığını ve değişkenliğini tanımlamaktan sorumludur. Platform, her türlü yazılım araçlarından oluşmaktadır (gereksinimler, tasarım, gerçekleştirme, testler, vb.). Bu eserler arasındaki izlenebilirlik bağları, sistematik ve tutarlı yeniden kullanımı kolaylaştırmaktadır. Alan mühendisliği, mevcut değişkenliğin, uygulamaları üretmek için uygun olmasını sağlamaktan sorumludur. Bu, belirli bir uygulamanın türetilmesi için yaygın mekanizmaları içermektedir. Platform, birçok yeniden kullanılabilir üründe doğru esneklik miktarı ile tanımlanmaktadır [1][3]. Alan mühendisliği sürecinin temel hedefleri şunlardır [1]:

- Yazılım ürün gruplarının ortaklık ve değişkenliklerinin tanımlanması
- Yazılım ürün grubunun kapsamının tanımlanması
- İstenilen çeşitliliği sağlayan yeniden kullanılabilir yapıların tanımlanması

Alan mühendisliği süreci, uygulamaların ortaklığını ve kitlesel kişiselleştirmeyi destekleme değişkenliğini içeren bir platform üretmektedir. Bu süreç;

- ürün yönetimi,
- alan gereksinim mühendisliği,
- alan tasarımı,
- alan gerçekleştirme ve
- alan testi

olmak üzere beş temel alt süreçten oluşur. Alan mühendisliği hedefleri, alan mühendisliği alt süreci ile gerçekleştirilir. Her birinin şunları yapması beklenmektedir [8]:

- Bir önceki işlem tarafından belirlenen değişkenliğin detaylandırılması
- Önceki alt işlem için gerekli değişkenliğin gerçekleşmesi hakkında geri bildirim sağlanması

B. Uygulama Mühendisliği

Uygulama mühendisliği, ürün hattının ürünlerini oluşturma ve temel varlıkları üretim planına göre yeniden kullanma sürecidir. Bu süreçte, ürün tasarımcıları, istenen ürünlerin doğru bir şekilde elde edilmesini sağlamak için alan mühendisliği sırasında yaratılan değişkenliği ve alan varlığını kullanır [1][3].

Uygulama mühendisliği aşamasında, alan mühendisliği sürecinde oluşturulan temel varlıkların kullanılması ile ürün maliyetinin düşürülmesi hedeflenmektedir. Aynı zamanda uygulama mühendisliği sürecinde gerçekleştirilen adımların otomatize edilmesi ile de maliyetin düşürülmesi için çalışmalar yapılmaktadır [10][11]. Uygulama mühendisliğinin temel amacı, mümkün olduğu kadar çok sayıda alan eserini yeniden kullanarak bir yazılım ürün hattı uygulaması elde etmektir. Bu, alan mühendisliğinde oluşturulan ürün hattının ortaklığı ve değişkenliğinden yararlanılarak elde edilmektedir [3]. Uygulama mühendisliği sürecinin temel hedefleri şunlardır [1]:

- Bir ürün hattı uygulaması tanımlarken ve geliştirirken alan varlıklarının mümkün olduğunca yeniden kullanılması sağlanmalıdır.
- Bir ürün hattı uygulamasının geliştirilmesi sırasında yazılım ürün grubunun ortaklığı ve değişkenliği ortaya çıkarılmalıdır.
- Uygulama eserlerini, yani uygulama gereksinimlerini, mimariyi, bileşenleri ve testleri belgeleyerek, bunları alan eserleri ile ilişkilendirilmelidir.
- Uygulama gereksinimleri mimari, bileşen ve test durumlarına göre değişkenliğe açık olmalıdır.
- Uygulama, alan gereksinimleri ile mimarlık, bileşenler ve testler arasındaki farklılıkların etkilerini tahmin edebilmelidir.

Bu çerçevede dört uygulama mühendisliği alt süreci tanıtılmaktadır. Bunlar [1]:

- Uygulama gereksinimleri mühendisliği,
- Uygulama tasarımı,
- Uygulama gerçekleştirme ve
- Uygulama testidir.

3. Değişkenlik Yönetimi

Değişkenlik Yönetimi (Variability Managing), Yazılım Ürün Hattı Mühendisliği yaklaşımının temelini oluşturmaktadır. DY, benzer özellik gösteren ürünler arasında değişken ve ortak özellikleri belirlemeyi hedeflemektedir. Ortaklık ve Değişkenlik analizi kullanılarak, ürün ailelerinin oluşturulması için bir yol sağlanmaktadır. Bu sistematik yaklaşımda ortaklık, bir dizi nesne arasında aynı değere sahip olan nitelikleri ifade etmektedir. Değişkenlik ise bunlar arasında farklı değerler alan nitelikleri tanımlamaktadır [12].
Yazılım ürün hatlarının ortak ve değişkenliklerini tanımlamak için birden çok model bulunmaktadır. Bunlar arasından 3 tanesi yaygın olarak kullanılmaktadır: Karar Modeli, Değişkenlik Modeli ve Özellik Modeli.

A. Karar Modeli

Benzer yazılım sistemleri için değişken ve ortak noktaların belirlenmesi, ürün oluşturmak için etkili tasarım kararları ile şekillendirilmesi, karar odaklı değişkenlik yönetimi olarak adlandırılmaktadır. Bu yaklaşımda, bir ürün hattından özelleştirilmiş bir ürün oluşturmak için, her varyasyon noktasında karar alınarak ilerlenmektedir [13]. Karar modeli, alan mühendisliğinin ihtiyaç analizi ve mimari tasarım aşamalarında üretilir ve bunlara karşılık gelen değerler ile kararlar, ürün yapım sırasında uygulama mühendisleri tarafından çözümlenir. Karar modeli, bir ürün hattı için değişkenlikleri kapsamlı bir şekilde göstermektedir [14].

B. Değişkenlik Modeli

Değişkenlik Modeli, yazılım geliştirme yaşam döngüsünün farklı yapılarında üretilen değişkenlik noktaları arasında kesitsel bir ilişki sağlamaktadır. Değişkenlik modeli, biri varyant diğeri varyant noktaları olmak üzere iki temek yapıya sahiptir. Değişkenlik modeli yaklaşıma göre, bir varyasyon noktası harici ve dahili olabilmektedir. Dahili varyasyon

noktaları sadece geliştiriciler tarafından görülebilirken, harici varyasyon noktaları hem geliştiriciler hem de müşteriler tarafından görülebilmektedir. Varyasyon noktaları ile varyantlar arasında bir bağımlılık ilişkisi bulunmaktadır. Bu durum her varyasyon noktasının en az bir varyant ile ilişki içinde olması gerektiği anlamına gelmektedir. Bu değişkenlik bağımlılığı isteğe bağlı veya zorunlu olabilmektedir. İsteğe bağlı değişkenlik bağımlılığı, bir ürün hattının özel bir ürününde bir varyantın olabileceği fakat var olması gerekmediği anlamına gelmektedir. Zorunlu değişkenlik bağımlılığı durumunda özelleştirilmiş bir ürün için varyasyon noktası mevcutsa, bir uygulama için varyantın olması gerekmektedir [15].

C. Özellik Modeli

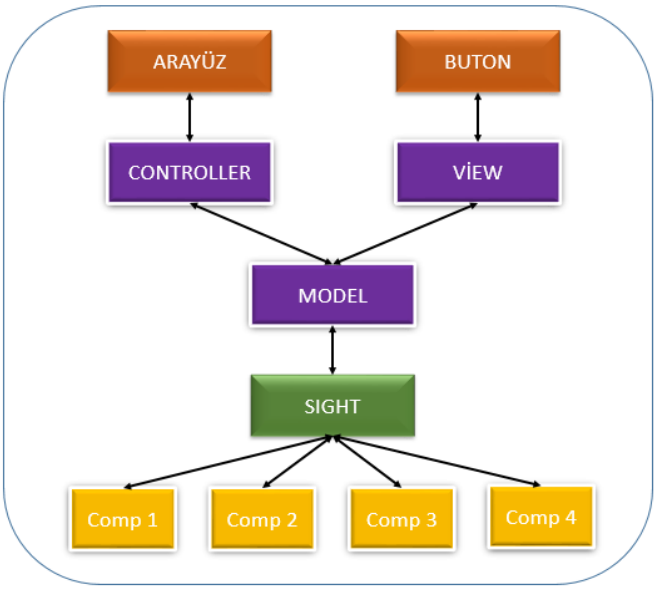
Özellik Modelleri, bir YÜH'daki tüm ürünlerin kompakt bir temsili olarak yaygın olarak kullanılmaktadır. Bir özellik modeli, düğümlerin özellikleri temsil ettiği ve bağlantıların aralarındaki ilişkileri gösterdiği ağaç benzeri bir yapı olarak temsil edilmektedir. Bu ilişkiler, özelliklerin ürünler oluşturmak için nasıl birleştirilebileceğini belirlemektedir. Özellik Modelleme yaklaşımı ilk olarak Kang ve arkadaşları tarafında Özelik Odaklı Alan Analizi (FODA) yönteminin bir parçası olarak tanıtılmıştır [16]. Çalışmaya göre, büyük ve karmaşık yazılım yoğun sistemlerin herkes tarafından anlaşılmasını sağlamak için yeteneklerin ve özelliklerin netleştirilmesi gerekmektedir. FODA, öncelikle alan analizi sırasında ürün ailesinin yeteneklerini ve özelliklerini keşfederek, yazılım bileşenlerinin etkili bir şekilde yeniden kullanılmasının gerçekleşebileceğini savunmaktadır.

4. Yazılım Ürün Hattı Tasarımı

A. MVC Tasarım Örüntüsü

MVC, yazılımın yapılan iş ile kullanıcı arayüzü bölümlerini birbirinden ayrıştırmaya yarayan bir tasarım örüntüsüdür. 'Model', 'View' ve 'Controller' olmak üzere 3 temel parçadan oluşmaktadır. Uygulama bu parçalara ayrılırken girişlerin kontrolü, işlemlerin gerçekleşmesi ve çıkışın oluşturulması aşamaları göz önüne alınmaktadır. 'Model' yapısı uygulama da verilerin işlenmesi ve işlemlerin gerçekleştirilmesinden sorumludur. 'View' kısmı, yapılan işlerin çıktılarının kullanıcıya iletilmesi, gösterimin sağlanmasından sorumludur. 'Controller' ise dış dünyadan gelen girişleri denetlemek ve gerekli işlemleri başlatmaktan sorumludur.

Mevcut projelerde yapı, bir model, bir view ve bir controller olacak şekilde tasarlanmıştır. Tasarlanan mimari de controller dış dünyadan gelen arayüz kontrol komutlarını işlemekten sorumludur. Gelen mesajların yapısal kontrolleri bu katmanda gerçekleştirilir ve model'e iletilmek üzere yeni bir mesaj oluşturulur. Oluşturulan yeni mesaj model yapısına iletildikten sonra model gerekli işlemleri gerçekleştirir ve dönüş işlemlerini oluşturur.



***Şekil 3**. Mevcut MVC Yapısı*

a. Controller Yapısının Çalışması

Controller yapısı gelen ve giden mesajların işlemlerinin yapıldığı iki temel fonksiyon içermektedir. Bu fonksiyonlar farklı önceliğe sahip taskların içerisinden kullanılmaktadır. Çıkış işlemlerinin gerçekleştirildiği task giriş işlemlerinden daha yüksek önceliğe sahiptir. Diğer yandan çıkış işlemleri aktifleşmedikçe bu task uyutulmaktadır. Giriş işlemlerinin yönetildiği task ise kullanılacak haberleşme arayüzünün yapısına bağlı olarak sürekli uyuyabilmekte ya da sürekli kontrol işlemini gerçekleştirmektedir. Sürekli uyutulan durumlarda dışarıdan gelen kesmeler sayesinde uyanıp işlemini tamamlayıp modele komutu yönlendirmektedir.

b. View Yapısının Çalışması

View yapısı buton işlemleri dışında pasif olarak çalışan bir yapıya sahiptir. Termal görüntüleme sistemlerinin bir kısmı sadece masaüstü uygulamaları ile çalışırken bir kısmı dışarıdan buton yardımları ile de kontrol edilebilmektedir. View yapısı buton kontrolü olan projeler dışında kullanılmamaktadır. Menü işlemlerinin gerçekleşeceği durumlarda view buton girişlerini yakalayarak bunlardan yeni girişler oluşturmaktadır. Bu durum dışında yalnızca çıkış işlemlerinin gerçekleşeceği durumlarda aktifleşmektedir.

c. Model Yapısının Çalışması

Model yapısı giriş işlemlerinden çıkışların oluşturulmasının sağlandığı yapıdır. Ek olarak periyodik işlemlerin gerçekleştirildiği ve bu işlemlere bağlı olarak yeni çıkışların oluşturulduğu birimdir. Model yapısı, controller ve view üzerinden gelen girişleri içeriğinde bulunan farklı sensörlerin donanım yazılımlarına aktararak cihaz davranışlarını kontrol etmektedir. Model, giriş işlemlerini controller ve view ile aynı task içerisinde gerçekleştirmektedir. Oluşan sonuç daha yüksek önceliğe sahip olan çıkış taskı aktifleştirerek döngünün tamamlanmasını sağlamaktadır.

d. Model Donanım Yazılımları

Farklı sensörlerden oluşan donanımlar içermektedir. Bu donanım yazılımları nesne tabanlı programlama kullanılarak oluşturulan arayüz sınıfları aracılığı ile biçimsel olarak aynı yapıya sahip fakat içerikte farklılaşan işlemler gerçekleştirmektedir. Bu durum tüm donanım yazılımları için geçerlidir. Her donanım yazılımı bir arayüz sınıfına sahiptir.

e. Sorunlar

İncelenen yazılımlar için projeye özel kod tasarımları yapılmaktaydı. Sonrasında proje sayısının artması ile birlikte şu an kullanılmakta olan MVC yapısı geliştirilmiştir. Projeler genelde benzer özellik ve donanımlara sahip olduğu için oluşturulmuş olan bu yapı ile az bir efor harcanarak yeni projeler için yazılım geliştirilmiştir. Fakat gelişen teknoloji ve müşterilerin isterlerinin çeşitlenmesi ile birlikte kullanılan MVC yapısı yeterli olmamaya başlamıştır. Farklı platformlarda, farklı işletim sistemi isterlerine ve donanım çeşitliliğine sahip projeler gelmeye başlamıştır. 10 proje için kullanımı kolaylaştıran bu yapı proje sayısının 100'e yükselmesi ile işlevselliğini kaybetmiştir. Hem yeni gelen bir proje için eski kodların kullanımının imkansızlaşması hem de yeni isterlerin mevcut projeye uygulanması zorlaşması bizleri yenilik yapmaya doğru itmiştir.

Şekil 3'te mevcut MVC yapısı gösterilmektedir. Projeler de donanım ihtiyaçlarının artması, mevcut projeler içerisinde müşteri ihtiyaçlarına göre farklı markalara ait donanım isterlerinin gelmesi kurulmuş olan yapı da kullanım zorluğu oluşturmaya başlamıştır. Yapının modülleri, arasında bağımlılık çok fazla olduğu için küçük bir değişiklik durumunda bile birçok yer etkilenmektedir. Ayrıca farklı marka isterlerinden kaynaklı arayüz sınıflarında gereksiz metotların yığılmasına neden olmaktadır.

Bu kapsamda bir yazılım ürün hattı çalışması yürütülerek yeni bir yapının kurulmasına karar verilmiştir. İlk olarak yazılım ürün hattı süreçlerinden alan mühendisliği çalışmalarına başlanmıştır. Mevcut tasarım incelenmiştir. Bu incelemeler doğrultusunda referans mimariyi oluşturabilmek için değişkenlik yönetimi modellerinden özellik ve karar modellerinin uygulanması kararlaştırılmıştır.

B. Alan Mühendisliği Çalışmaları

a. Özellik Modelinin Çıkarılması

İlk olarak yazılım ürün ailesi kullanan 20 farklı yazılım incelenmiştir. İncelenen bu yazılımlardan farklı platform ve sistemlere sahip olduğu düşünülen 4 pilot yazılım belirlenmiştir. Belirlenen bu 4 yazılım ile detaylı bir çalışma yapılmıştır.

Yazılım ürün hattı oluşturma çalışmaları kapsamında ilk olarak projelerdeki 'Model', View ve Controller katmanlarındaki base sınıflardaki metot sayıları incelenmiştir. Şekil 3'te görüldüğü gibi yapı birbiri ile bağlantılı olduğu için her yeni metot tüm yapıyı etkilemektedir.

Tablo 1. *1.Proje Toplam Metot Sayıları*

PROJE 1				
	Model Sınıfı	Controller Sınıfı	View Sınıfı	Sight Sınıfı
Public Metotlar	2	3	2	139
Private Metotlar (Genel)	4	6		
Private Metotlar (Component)	72	70	29	28

Tablo 2. *2.Proje Toplam Metot Sayıları*

PROJE 2			
	Model Sınıfı	Controller Sınıfı	View Sınıfı
Public Metotlar	2	3	2
Private Metotlar (Genel)	3	2	
Private Metotlar (Component)	152	309	29

Tablo 3. *3.Proje Toplam Metot Sayıları*

PROJE 3			
	Model Sınıfı	Controller Sınıfı	View Sınıfı
Public Metotlar	4	3	2
Private Metotlar (Genel)	3	13	
Private Metotlar (Component)	147	101	29

Tablo 4. *4.Proje Toplam Metot Sayıları*

PROJE 4				
	Model Sınıfı	Controller Sınıfı	View Sınıfı	Sight Sınıfı
Public Metotlar	3	3	2	219
Private Metotlar (Genel)	4	7		
Private Metotlar (Component)	123	106	29	13

Projeler incelendiğinde, View yapısının tüm projelerde aynı şekilde kullanıldığı gözlemlenmiştir. Bu nedenle ilk aşama da yapılacak çalışmalarda Model ve Controller sınıflarına odaklanmak gerektiği anlaşılmıştır. Tablo 1'de 1.projeye ait veriler görülmektedir. Bu verilere göre Model yapısı içerisinde bulunan model sınıfında 78 metot ve sight sınıfında 167 metot, Controller yapısı içerisindeki controller sınıfında 79 metot olduğu görülmektedir. Tablo 2'de 2.projeye ait veriler görülmektedir. Bu verilere göre Model yapısı içerisinde bulunan model sınıfında 157 metot Controller yapısı içerisindeki controller sınıfında 314 metot olduğu görülmektedir. Bu proje de sight sınıfı bulunmamaktadır. Tablo 3'de 3.projeye ait veriler görülmektedir. Bu verilere göre Model yapısı içerisinde bulunan model sınıfında 154 metot Controller yapısı içerisindeki controller sınıfında 117 metot olduğu görülmektedir. 3. proje de sight sınıfını kullanmamıştır. Son olarak Tablo 4'de 4.projeye ait veriler görülmektedir. Bu verilere göre Model yapısı içerisinde bulunan model sınıfında 130 metot ve sight sınıfında 232 metot, Controller yapısı içerisindeki controller sınıfında 116 metot olduğu görülmektedir. Projelerde yapılmış olan incelemeler sonucunda public metotların sayısının az ve genelde birbirleri ile ortak oldukları gözlenmiştir. Aynı zamanda Private metotlar incelendiğinde 2 farklı gruba ayrılmıştır. Genel metotlar mesajları bölme veya gelen mesajı anlamlandırıp ilgili metota yönlendirme gibi işlemleri yaptıkları ve tüm projelerde ortak oldukları gözlemlenmiştir. Component özel metotlar ise bölünmüş ve ilgili komponente gerekli işlemi yapması için mesaj oluşturmaya yarayan metotlardan oluşmaktadır. Bu metotların sayısının mevcut projelerde bile yeterince büyük olduğu görülmektedir. Model sınıfını kullanmak isteyen bir kişinin projeye özel metotlarda eklemesi ile birlikte bu sayı çoğalmakta ve izlenebilirlik azalmaktadır. Ayrıca projelere göre kod satır sayıları da Tablo 5'te görülmektedir.

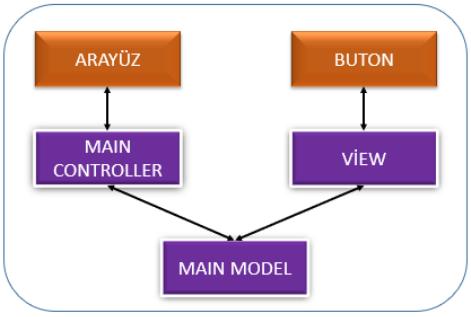
Tablo 5. *Projelerde Toplam Satır Sayıları*

	Proje 1	Proje 2	Proje 3	Proje 4
Model	5038	7813	5558	7404
Controller	2326	5790	4246	3768
Sight	3875			5689

Satır sayılarına baktığımız zamanda kullanılan veya kullanılmayan tüm kodların yeni projelere eklenecek olması, proje özelinde istenecek yeni metotlar ile sayıların daha da çok artacağı düşünüldüğünde bunların ayrılarak yeni bir yapının oluşturulması gerektiğine karar verilmiştir.

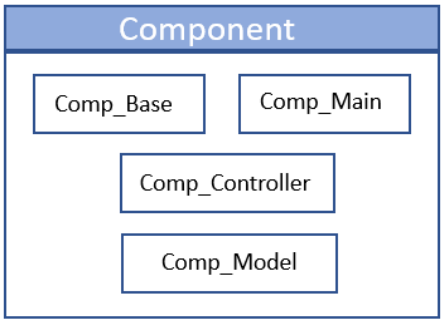
b. Referans Mimarisi

Yapılmış olan analiz sonrasında, ilk olarak model ve controller sınıflarının tüm projeler için ortak kullanılacak şekilde yeniden tasarlanması gerektiği açıkça görülmektedir. Bunun da birer adet olan sınıf sayısının çoğaltılması ve araya belirli katmanlar eklenerek sağlanabileceği düşünülmüştür. View sınıfı şu an tüm projelerde ortak olarak kullanıldığı için herhangi bir değişiklik yapılmasına gerek görülmemiştir.



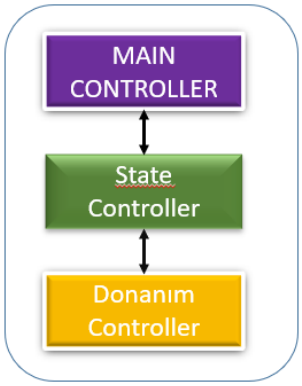
***Şekil 4**. Yeni MVC Tasarımı*

Şekil 4'te görüldüğü gibi MVC yapısı aynen korunmak istenmiştir. Dış arayüzden gelen mesajların ilk olarak ana controller sınıfına gelmesi planlanmaktadır. Ana model sınıfı ise aynı şekilde kullanılan yardımcı donanımların yapacağı işler için kodlar üretmeye devam edecektir. Fazla sayıda metota sahip olan controller ve model sınıflarının daha anlaşılabilir olması için her component yapısı içerisine model ve controller sınıfı eklenerek yeni bir yapının oluşturulması düşünülmüştür. Şekil 5'te görüldüğü gibi her donanıma özel bir base, bir ana,bir controller ve bir model sınıfı oluşturulması planlanmıştır. Eski yapı da model ve controller sınıflarında yapılan işlemlerin, yeni yapı da artık donanıma özel sınıflarda yapılması planlanmaktadır.



***Şekil 5**. Yeni Tasarım Donanım Yazılım Yapısı*

Bu yapıya ek olarak bir de state katmanı eklenmesi planlanmaktadır. Şekil 6'da controller haberleşmesi için oluşturulması düşünülen yapı görülmektedir. Bu state katmanı Ana Controller sınıfının donanım controller sınıfları ile haberleşmesi için bir ara katman olarak düşünülmektedir. Aynı şekilde ana model ile donanım özel model sınıflarının da haberleşmesinde de state katmanı kullanılacaktır.



***Şekil 6**. Yeni Tasarım Controller Haberleşme Yapısı*

5. Gelecekte Yapılacak Çalışmalar

Yazılım ürün hattı 2 süreçten oluşmaktadır. Bu makale kapsamında alan mühendisliği çalışmalarından bahsedilmiştir. Yaklaşık 1 yıllık bir çalışmanın sonunda analizler çıkarılmış ve tasarım oluşturulmuştur. Bundan sonra çalışma da ise

uygulama mühendisliği sürecinin yapılması planlanmaktadır. Böylece yazılım ürün hattı hayata geçilmiş ve bundan sonra gelecek olan projeler için yazılım ürün hattı oluşturulmuş olacaktır.

6. Sonuçlar

Mevcut MVC yapısı, kullanılan yazılımlarda gerekli isterleri karşılayabilmektedir. Fakat mevcut yazılımlarda yeni isterlerin gelmesi veya yeni bir yazılım talebinin olması durumunda yetersiz kalabilmektedir. Eski ve yeni mimari karşılaştırıldığında gelen yeni isterlere uyum sağlama konusunda, yeni mmimarinin daha esnek ve kolay çözümler üretebildiği görülmektedir. Yeni mimarinin daha katmanlı bir yapıdan oluşması, gelecek yeni isterlerin yazılıma uygulanmasını kolaylaştırması beklenmektedir.

Mevcut mimari bütüncül yaklaşımla geliştirildiği için takip edilebilirlik kolay gibi düşünülse de artan metot ve kod satır sayıları düşünüldüğünde bu özelliğini yitirmiş durumdadır. Tüm metotların tek bir sınıfta toplanmış olması ve işlemlerin katmanlar olmadan gerçekleştirilmesi yazılım isterlerinin eklenmesi açısından problem oluşturmaktadır. Yeni mimarinin hem model ve controller yapılarının donanım özelinde bölünmesi hem de state katmanının kullanılacak olması sınıflardaki metot sayısını azaltması ve izlenebilirliği artırması beklenmektedir. Kod satır sayılarının da yaklaşık %20 oranında azalması beklenmektedir.

Tasarlanmış olan yeni mimari ile yeni gelen isterlerin daha hızlı yazılıma uygulanması, yeni gelen yazılım isterlerinde ortak sınıflarının kullanımının artması, daha hızlı yazılım ürünlerinin çıkarılması ve ürünlerin piyasaya sürülme sürelerinin azalması beklenmektedir. Uygulama mühendisliği sürecinin hayata geçilmesi ile iyileşme verileri de ortaya çıkacaktır.

Kaynaklar

[1] Klaus Pohl, Günter Böckle, Frank van der Linden. "Software Product Line Engineering: Foundations, Principle, and Techniques" Springer-Verlag (2005)
[2] Sepúlveda Samuel, Ania Cravero, and Cristina Cachero."Requirements modeling languages for software product lines: A systematic literature review." Information and Software Technology 69 (2016): 16-36.
[3] Frank van der Linden, Klaus Schmid, Eelco Rommes. "Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering" Springer (2007)
[4] Catal, C., "Yazılım Kusur Kestirimi Probleminde Yapay Bağışılık Sistemlerinin Uygulanması", Doktora tezi, Yıldız Teknik Üniversitesi, İstanbul, 2008
[5] E. Kahraman, T. Ipek, B. Iyidir, C. Bazlamaci ve S. Bilgen, "Bileşen Tabanlı Yazılım Ürün Hattı Geliştirmeye Yönelik Alan Mühendisliği Çalışmaları," Ulusal Yazılım Mühendisliği Sempozyumu, İstanbul, Türkiye, 2009.
[6] B. C. Kasikci ve S. Bilgen, "Etkin Yeniden Kullanım: Yazılım Ürün Hatlarında Değişkenliğin Modellenmesi," Ulusal Yazılım Mühendisliği Sempozyumu, İstanbul, Türkiye, 2009.
[7] Krueger, C.W.: The emerging practice of software product line development. Military Embedded Systems (2nd semester), pp. 34–36 (2006)
[8] Felix Schwägerl, Version Control and Product Lines in Model-Driven Software Engineering, Master's Thesis, Nürnberg, 2018.
[9] P. Clements, L. Northrop. Software Product Lines: Practices and Patterns. Boston, MA:Addison-Wesley, 2002.
[10] Krueger, W. C., "New Methods in Software Product Line Development", 10th International Software Product Line Conference, 21-24 August 2006, Baltimore, Maryland, USA
[11] Groher, I., Papajewski, H., Voelter, M., "Integrating Model-Driven Development and Software Product Line Engineering", Eclipse Summit Europe 2007, Ludwigsburg, Germany
[12] Coplien, J., Hoffman, D., & Weiss, D., "Commonality and Variability" 1998
[13] Dhungana, D., Rabiser, R., & Grünbacher, P. "Decision-Oriented Modeling of Product Line Architectures", 2007
[14] Schmid K., John I., "A customizable Approach to Full Lifecycle Variability", 2004
[15] Gülşah ERDİL, "A DYNAMIC SOFTWARE PRODUCT LINE FOR REMOTE MONITORING OF COMPUTER SYSTEMS", 2019
[16] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., & Peterson, A. S., "Feature-oriented Domain Analysis (FODA) Feasibility Study", 1990