**I Çalışmanın Tanımı**

**1.Çalışmanın Amacı ve Hedefleri**

Çalışmanın amacı, çözümlenmesi hedeflenen problemi en üst düzeyde tanımlar. Bu nedenle tek bir amaç verilmelidir. Hedef ise proje çalışması sonunda gerçekleştirileceklere yönelik sıralanışlardır. Amaçlardan daha kesin belirtilmesi gereken hedefler projenin tamamlanması ile değerlendirilecek niceliksel ve niteliksel ölçütlerdir.

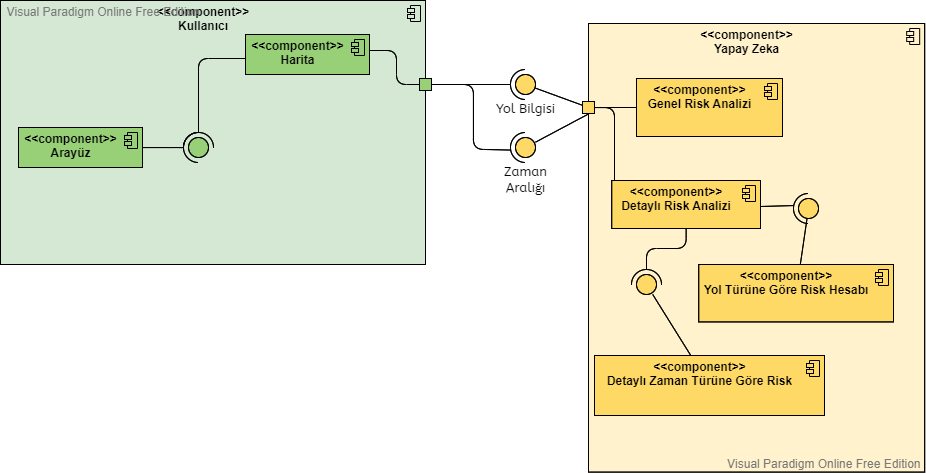
**2 Çalışmanın Kapsamı**

Geliştirilecek yazılım ürünü ile ilgili olarak yapılan çalışmanın ilk olmadığı kabul edilmektedir. Piyasadaki güncel yazılım ürünlerinin pek çoğu önceden geliştirilmiş, daha sonra yeniden yapılandırılmış ürünlerdir. Yeni bir ürün geliştirirken iş analistleri eski ürün yerine yenisinin niçin geliştirildiğini ya da önceden elle yapılan birtakım işlerin neden otomatikleştirilmesi ya da değiştirilmesi gerektiğini araştırırlar. Mevcut sistemde ne gibi eksilikler vardır ki ürünün yeniden geliştirilmesi istenmiştir? şeklindeki sorular mutlaka cevaplanmalıdır.

**2.1 Çalışmanın İçeriği**

Proje çalışmasına ait “*component diagram*” ın çizilmesi istenmektedir. Böylece çalışmanın üst düzeydeki açıklaması görsel olarak yapılmış olacaktır. Diğer bir ifadeyle, “*component diyagram”* ile projenin sınırları belirlenecektir. Böylece, geliştirilecek ürün tüm alt bileşenleri ile birlikte açıklanmış olacaktır.

<https://www.uml-diagrams.org/component-diagrams.html> “visual paradigm community edition”



**Şekil 1:** ………..sistemine ait “component” diyagramı

Çalışma içerisinde çizilen tüm diyagramlara metin içerisinde açıklanırken mutlaka atıf yapılması gerekir.

Bu bölüm de dahil olmak üzere sonraki bölümlerde sadece UML diyagramının çizilmesi yeterli olmayıp, görsel çözümlemelerin yazılı olarak ta açıklanması gerekir.

**3 Fonksiyonel Olmayan Gereksinimler**

Fonksiyonel olmayan gereksinimler geliştirilen sistemin başarısının göstergesi olacağı için çevik ekipler için hayati beceridir ve sistemin niteliklerini belirtir. Bunlar, sistemin ne yaptığının değil, *ne kadar iyi yaptığını*n göstergesidir. Dolayısıyla sistemin çeşitli girdilere yanıt olarak ne yapacağını tanımlayan fonksiyonel gereksinimlerden tamamen farklıdır. Fonksiyonel olmayan gereksinimlerin karşılanmaması, sistemin yapılan işi yani müşterilerin ihtiyaçlarını karşılamaması demektir. Bu da uygulama geliştirilirken kullanılan standartların da karşılanmayacağı ankamına gelir.

Örneğin uyumsuzluk (non-compliance) maliyet, geri çekme, gizlilik (privacy), güvenlik (security), güvenlik riski (safety risk), yasal yaptırımlar (legal exposure) gibi önemli sorunlara neden olabilir. O nedenle fonksiyonel olmayan gereksinimlerin doğru tanımlanması ve uygulanması kritik öneme sahiptir. Aşırı sayıda belirtim durumunda çözüm çok maliyetli veya uygulanamaz olabilir. Hedeflenen hiçbir gereksinim de yoksa sistem kullanım açısından yetersiz olabilir. Sistemin kapsamı ne olursa olsun, fonksiyonel olmayan gereksinimler belirtilmelidir.

Analiz ve tasarımı gerçekleştirilecek probleme ilişkin fonksiyonel olmayan gereksinimler ve açıklamaları gerekçeleri de ekleyerek hazırlamalıdır. Fonksiyonel olmayan gereksinimlerin belirlenmesi ve açıklamaları için <https://en.wikipedia.org/wiki/Non-functional_requirement> linkini incelenebilir.

Fonksiyonel olmayan gereksinim örnekleri:

**Güvenilirlik** : …………. sistemi, insan ve canlı/cansız etkilenebilecek tüm varlıkların maddi manevi duyarlılıklarının korunmasını hedeflediğinden çalışma ortamı ve kriterleri dahilinde %99,9 hatasız çalışmalıdır.

**Sürdürülebilirlik**: ………….sisteminin arıza vermesi durumunda bakım yapılabilmesi için, hiçbir kullanıcı tarafından kullanılmadığına emin olunmalı ve ……. seviyesi olabildiğince düşük tutulmalıdır.

**Kullanılabilirlik**: Kullanıcının veri girişi yaparken internet bağlantısında bir sorun olmadığı kabul edildiğinde, verilerin başarılı analizi sonucunda sistem tüm kullanıcılar için 7/24 kullanılabilir durumda olmalıdır.

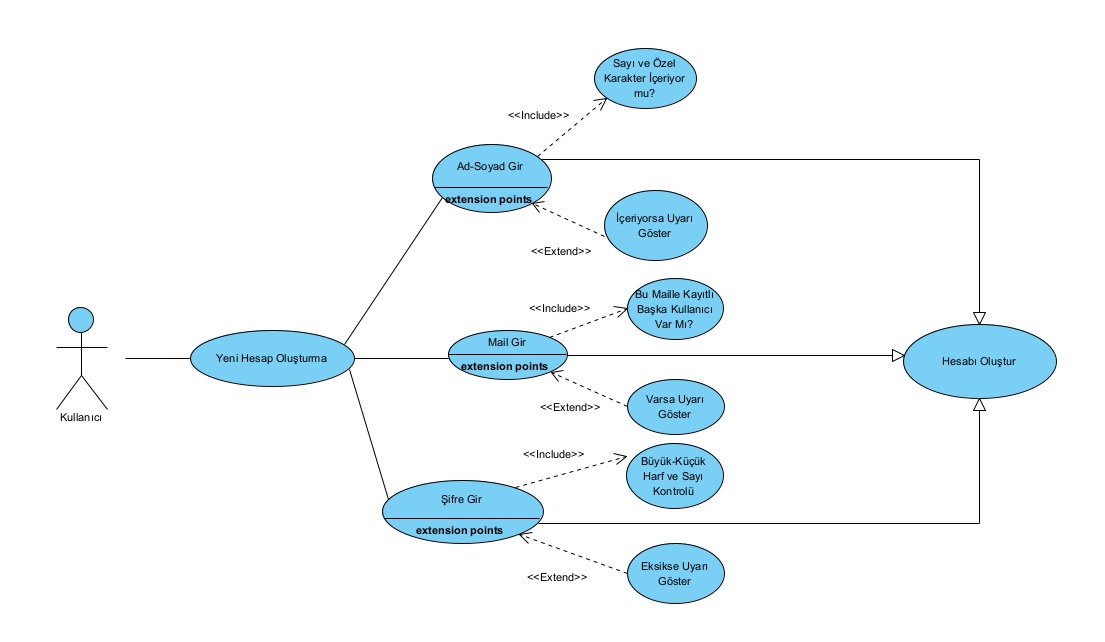
…………………..

**II Gereksinimlerin Belirlenmesi**

**4 *Use Cases* ve Tablo Açıklamaları**

*Uses cases*, geliştirilecek yazılım sistemini ve sistemin sınırlarını ayrıntılı olarak tanımlaması ile birlikte sistemin fonksiyonel gereksinimlerinin tanımlanmasını da sağlar. Use casediyagramları (<https://www.uml-diagrams.org/use-case-diagrams.html>) linki referans alınarak geliştirilebilir.

Bir *use case* diyagramı ve bir başka örnek için use case diyagramının tablo üzerindeki açıklaması örneği aşağıdadır.

****

**Şekil 2 : ………**…..’a ait use case diyagramı

*Tüm şekil ve tablolar metninizin bu bölümünde yapılan açıklamalar içerisinde mutlaka refere edilemelidir.*

**\*Tablo1 :** …………..use case diyagramı açıklaması (tablonun üstüne yazılır)

|  |  |
| --- | --- |
| Use case | **SK.1.0** |
| Aktörler | Admin,Owner, Housekeeper |
| Ön Koşullar | Owner ya da Admin bulunmalıdır. Property belirlenmelidir |
| Son Durum | Housekeeper verilen görevi tamamlayacak, gerekli durumlarda Assets stoğu güncellenecektir. |
| Başarılı Senaryo | 1. Owner property belirler, admin üyeliği oluşturur. 2. Owner ya da Admin sector ve space alanı belirler, Housekeeper üyelikleri oluşturur. 3. Owner ya da Admin Task bilgileri gönderir. 4. Housekeeper Task’ı alır,Status inprogress halini alır 5. HouseKeeper Task’ın durumuna göre Status tekrar düzenlenir. 6. Task tamamlanır,belirli durumlara göre Assets stoğu güncellenir. |
| Alternatif Senaryo | Housekeeper Task için issued durumu işaretlerse, gerekli Assets stok düzenlemeleri yapılır, Task’ın done olması sağlanır |

\*Tablo 1 ‘de olduğu gibi İngilizce sözcükler çalışma alanında özel terim olarak kullanılıyorsa Türkçe karşılığı ile yazılmayabilir.

Her *use case* diyagramının bir listesini yapmak, her birini ayrı ayrı modellemek ve tanımlamak çalışmasının analiz aşamasının izlenebilirliğini kolaylaştıracaktır.

Use case diyagramlarının sistem üzerinde iki amacı vardır: Her bir *use case* diyagramının problemin ilgili parçasına ait kavramsal çözümlemede *include* ve/veya  *extend* ilişkileri tanımlanmalıdır. Bu ilişkiler karmaşık bir problemin çözümüne doğal olarak bulunur. Her bir *use case* diyagramını listeleyen tablo, bu *use case* diyagramının sistemin bir parçası olarak hangi eylemleri içerdiğini ve de aynı zamanda sınırlamaları (kısıtları) ifade ederek sistemin neleri içermediğini gösterir.

Aktörlerin canlı ve/veya cansız bir varlık olması mümkündür. Çözülen problemin karmaşıklığına bağlı olarak, bazı *use case* diyagramları için birden fazla diyagram kullanılması gerekebilir.

Her bir olay (vaka), yani use case kullanımı ile *fonksiyonel gereksinimler belirlenmiş* olacaktır. Pek çok yazılım ürünü geliştirme aracında *fonksiyonel gereksinimler* tablo olarak tutulur ve her bir fonksiyonel gereksinim için tek (unique) bir sayı belirlenir. Bu, çalışmanın *izlenebilirliğini (tracebility)* kolaylaştıran önemli bir uygulamadır.

**III Tasarım Dokümantasyonu**

**5 Tasarım Hedeflerinin Tanımlanması**

Tasarım hedefleri, geliştirilecek sistemin genel tasarımında oldukça etkili olup birtakım ayırt edici özellikler içerebilir. Örneğin bilgisayar oyunlarında öncelik, çözümün *doğruluğundan* (accuracy property) ziyade sistemin *çalışma hızının* yeterince yüksek olmasıdır. Bir bilgisayar oyununda kullanılan motorla, oyunun daha hızlı çalışmasını sağlayacak alternatifler aranabilir; böylece geliştirilen sistemin *doğruluğu* tasarım hedefi olmayacaktır. Öte yandan farklı bir yazılım ürününde, örneğin hassas bilimsel hesaplamaların yapıldığı bir çalışmada öncelikli tasarım hedefi, hızın düşmesi olasılığına rağmen *sonuçların doğru* elde edilmesi olacaktır.

Tasarım hedefleri ile gereksinimler arasındaki *önemli bir fark vardır*: Gereksinimler, ürünün müşteriye kabul edilebilir olması için gerçekleştirilmesi gereken *her şeydir*. Oysa tasarım hedefleri, tasarımcıların mümkün olan *en iyisini yapmak için* gerçekleştirmeyi arzuladığı özelliklerdir. Tasarımcılar hedeflerini belirlerken belirli *kabul edilebilirlik* şartlarını sağlamak zorunda değildir. Oysa gereksinimler müşterinin istekleridir ve *yerine getirilmelidir*.

Bu bölümle ilgili olarak tanımlanacak herhangi bir kavramın hem gereksinimler aşamasında, hem de tasarım hedefinde verilmesi mümkündür. Bu nedenle bir tasarım hedefi, bir gereksinimin gerçekleştirilmesi ile sistemin mümkün olabildiği kadar hızlı çalışmasını sağlayabilir. Ayrıca tasarım hedefine göre sistemin belirli bir eşiğin altındaki bir hızı kabul etmeyeceği de ifade edilebilir.

Yazılım ürünü geliştirirken tasarım hedeflerine karar vermede bazı birbirinin *karşıtı* betimlemelerle karşılaşılabilir. Örneğin, bir uygulamanın *fonksiyonelliği* ile *kullanılabilirliği* örtüşmeyebilir. Bir sistem 100 fonksiyonlu olarak kullanılabilir mi? Bunun için ne büyüklükte büyük bir menü tasarımı daha uygundur? gibi soruların cevabı gerekir.

Bir başka karşıtlık durumu olarak ürünün maliyetinin düşüklüğü ile güçlülüğü (robustness) çelişebilir. Örneğin, düşük maliyetli bir sistem, kullanıcı hatalı veri girişi yaptığında hataların kontrol edilmediği bir çözüm içerebilir.

Geliştirilen sistemin etkinliği (efficiency) ile taşınabilirliği (portability) özelliği de zıt yönde gerçekleşebilir.

Ürünün modellemesi için tercih edilen hızlı geliştirme (rapid development) yöntemi, geliştirilen sistemin fonksiyonelliği ile çelişebilir. Örneğin bir projenin geliştirilmesi için 5 hafta planlandığı ve bunun 5 geliştirici ile yapılacağı kabul edilsin. Tasarım süresi ise 2 hafta olsun. Bir sorunla karşılaşılması durumunda, ürünün teslim tarihinin uzatılması mümkün değil ise, gerçekleştirilecek fonksiyonellik azaltılacaktır. Bu durumda modeldeki tüm kullanıcı hikâyeleri (use cases) hayata geçirilmeden proje tamamlanacak ya da ürün zamanından sonra teslim edilecektir.

Hedeflenen *ürün maliyeti*, *yeniden kullanılabilirliğe* (reusability) karşıt olan bir özelliktir. Geçmiş yıllarda tasarımın yeniden kullanılabilirliği oldukça güçtü, ekstra çaba gerektiriyordu. Günümüz uygulamalarında tasarım şablonlarıyla yeniden kullanılabilirlik modern bir yapıya oluşturularak yaygınlaştı.

Probleme **uygun olan** tasarım hedefleri aşağıdaki kavramlar içerisinden **seçilerek** problemin çözümü içerisindeki işlevleriyle açıklanacaktır.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Reliability | Modifiability | Maintainability | Understandability | Adaptability |
| Reusability | Efficiency | Portability | Traceability of requirements | Fault tolerance |
| Backward-compatibility | Cost-effectiveness | Robustness | High-performance | Good documentation |
| Well-defined interfaces | User-friendliness | Reuse of components | Rapid development | Minimum number of errors |
| Ease of learning | Readability | Ease of remembering | Ease of use | Increased productivity |
| Low-cost | Flexibility | ……… | ……….. | ……….. |

**6 Önerilen Yazılım Mimarisi**

Çalışmanın yazılım mimarisi problemin çözümüne uygun olacak şekilde araştırılmalı ve mimari önerisi açıklanmalıdır.

Görsel betimleme olarak, geliştireceğiniz yazılım ürününün bağlam bakış (context) açısına göre çözümü anlatılacaktır. Bu UML diyagramı “deployment” diyagramdır (https://www.uml-diagrams.org/deployment-diagrams-overview.htm). Burada geliştireceğiniz sistemin alt sistemlere parçalanmasını gösteren fonksiyonel bir model tasarlanmaktadır. UML çizimi açıklanmalıdır.

**7 Sınıf Diyagramları**

Bu bölümde mantıksal bakış açısına göre tasarıma devam edilmektedir. “use case” diyagramları ile açıklanan davranışsal betimlemelerin tümüne ait statik etkileşimleri gösteren sınıf diyagramları çizilecektir (<https://www.uml-diagrams.org/class-diagrams-overview.html>).

“use case” diyagramlarının açıklamaları ile örtüşerek tanımlanan sınıflar ve sınıflar arasındaki ilişkilerde **“aggregation”, composition”, “association”,”generalization/specilization**” ilişkileri kullanılmalıdır.

**8 Dinamik Model**

Sistemin dinamik modeli (interaction modeling) UML diyagramları ile “collaboration diagrams” ya da “sequence diagrams” (<https://www.uml-diagrams.org/communication-diagrams.html>) olarak verilebilir.

Çözüm bu diyagramların ikisini birden içermez. Bu diyagramlardan biri diğerinin yerine tercih edilebildiği için Sequence diyagram tercih edilebilir.

**UML diyagramlarının tasarımı ile ilgili diğer referanslar:**

[*https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/*](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/)

[*https://www.uml-diagrams.org/index-references.html*](https://www.uml-diagrams.org/index-references.html)

(UML indeksidir, tüm UML terimlerinin açıklamalarını içerir)