GEREKSİNİM MÜHENDLİĞİNE GİRİŞ

Motivasyon

Yazılım geliştirmeyi endüstrileştirme yolunun başlarında, Royce (1975) aşağıdaki konulara dikkat çekti.

Birini bile eksik yaptığınız zaman yeterli gereksinim analizi yapılmadığından kaynaklanan dört tip sorun vardır. Baştan aşağı tasarım imkansızdır, test imkansızdır, kullanıcı dışlanmaktadır, yönetim kontrolde değildir. Bu sorunlar tartışmayı kolaylaştırmak için çeşitli başlıklar altında toplanmış olsa da aslında hepsi tek bir içeriğin varyasyonudur ve bunlardan biri kötü yönetimdir. Yazılım tedarikinin iyi proje yönetimi, bir takım açık ve düzenleyici gereksinimler olmadan mümkün değildir.

Baştan aşağı tasarım büyük ölçüde yerinin nesne tabanlı tasarımlar almış olsa da bu gerçek bugün hala geçerlidir.

Çok sayıda araştırma şunu doğruladı ki sistematik çabayı gereksinim mühendisliğine adamak yazılımın veya yazılım ağırlıklı ürünün sonraki çalışma ömründe ihtiyaç duyulan yeniden çalışma miktarını büyük ölçüde azaltabilir ve sistemin çeşitli niteliklerini maliyet açısından iyileştirebilir. Çoğu zaman sistem mühendisleri, gereksinim mühendisliğini nasıl uygun bir şekilde yapacaklarını anlamadıkları için yeterli gereksinim mühendisliği faaliyetinden vazgeçerler veya kod yazmak için acelesi vardır (bir yazılım ürünü olması durumunda). Açıkçası, bu olasılıklar istenmeyen bir durumlardır ve bu kitabın amacı, mühendislerin gereksinim mühendisliğinin uygun ilkelerini ve uygulamalarını anlamalarına yardımcı olmaktır.

Gereksinim Mühendisliği nedir?

Tanımlayıcının bakış açısına bağlı olarak gereksinim mühendisliği disiplinini tanımlamanın birçok yolu vardır. Örneğin bir köprü kompleks bir sistemdir ancak nispeten az sayıda kullanılabilecek tasarım modeline sahiptir (asmalı, makaslı, kablolu).

Köprülerin ayrıca yük gereksinimleri, kullanılan malzemeler ve kullanılan yapım teknikleri açısından belirli kurallara ve yerinde yönetmelikleri vardır. Yani bir müşteriyle köprünün

gereksinimleri hakkında konuşurken işlevselliğinin çoğu kısa ve öz bir şekilde ele alınabilir.

Köprü, Chadds Ford Pennsylvania Creek Road'daki Brandywine Nehri üzerindeki mevcut açıklığın yerini alacak ve çelik konstrüksiyondan bir konsol köprüsü olacaktır. Her yönde iki şeritli trafiği destekleyecek ve her yönde saatte minimum 100 araç kapasitesini sağlayacaktır.

Elbette ki bu "tanımlamada" birçok bilgi eksik (ağırlık kısıtlamaları gibi) ama bu köprünün ne yapacağını büyük ölçüde açıklar ve bu gereksinimleri yerine getirmek için mevcut tasarım seçenekleri nispeten basittir.

Oldukça uzmanlaşmış domain diline sahip biyomekanik veya nanoteknoloji sistemleri gibi diğer sistem türleri, görünürde garip gereksinimlere ve kısıtlamalara sahiptir. Yine de diğer karmaşık sistemler, somutlaştırılması gereken o kadar çok davranışa sahiptir ki söz konusu sistemlerin spesifikasyonu gerçekten zorlayıcı bir hale getirir.

Öncelikle Pamela Zave bir yazılım mühendisi olduğundan dolayı, gereksinim mühendisliği için az çok evrensel bir tanım için uygun bu disipline ulaşıyoruz.

Gereksinim mühendisliği, yazılım sistemlerinin gerçek dünya hedefleri, işlevleri ve kısıtlamaları ile ilgili yazılım mühendisliği dalıdır. Ayrıca bu faktörlerin yazılım davranışının belirli özellikleriyle ve bunların zaman içinde yazılım aileleri arasındaki evrimiyle de ilgilenir. (Zave 1997)

Ancak biz gereksinim mühendisliği kavramını yalnızca yazılım, yalnızca donanın veya donanım ve yazılım olsun herhangi bir sistemi içerek bir şeklide genelleştirmek istiyoruz ve böylece Zave'in tanımını aşağıdaki gibi yeniden yazdık.

Gereksinim mühendisliği, gerçek dünya hedefleri, işlevleri ve sistemler üzerindeki kısıtlamalarla ilgilenen mühendislik dalıdır. Ayrıca bu faktörlerin sistem davranışının belirli özellikleriyle ve bunların zaman içinde ilgili sistem aileleri arasındaki evriminin ilişkisiyle de ilgilenir.

Bu metin boyunca Gereksinim Mühendisliği'nden bahsettiğimizde bu değiştirilmiş tanıma atıfta bulunuyoruz ve ilerledikçe bu tanımın tüm sonuçlarını ve içerdiği faaliyetleri çok detaylı bir şekilde keşfedeceğiz.

Muhtemelen Yeterli Gereksinim Mühendisliği Yapmıyorsun

Araştırmalar gereksinim mühendisliğinin endüstride iyi yapılmadığını gösteriyor. Örneğin, 3 binden fazla mühendisten oluşan küresel bir ankette %37'si şirketlerindeki gereksinim mühendisliği uygulamalarının tatmin edici olmadığını söyledi (Kassab ve ark. 2014). Avrupa'da yedi gömülü sistem mühendisliği şirketinin başka bir çalışmasında benzer sonuçlara ulaştılar. Sikora ve arkadaşları (2012) özellikle mevcut gereksinim mühendisliği yöntemlerinin karmaşık gömülü sistemler için gereksinimlerin ele alınması için yetersiz olduğu sonucuna varmışlardır.

Yetersiz gereksinim mühendisliğinin olası bir nedeni Wnuk ve arkadaşları tarafından önerilmiştir. Wnuk, şirketlerin gereksinim mühendisliği uygulamalarını etkinlik kapsamı ve gereksinim yapıtlarının yapısı açısından ölçeklendirmekte zorlandıklarını gözlemledi. Onların anketinde Sikora ve arkadaşları endüstriyel gereksinimler mühendisliğindeki bariz yetersizlik için başka bir olası neden buldu. Uygulayıcıların "genel sistem mimarisine uygun şekilde entegre edilmiş ancak belirtilen işlev veya bileşenin kendisiyle ilgili olarak çözüm içermeyen gereksinim belirtimlerini" istediklerini buldular. Ankete katılanlar, mevcut yönetim desteğinin bu hedefi desteklemede yetersiz olduğunu belirtti. Öyle görünüyor ki, gereksinim mühendisliğinin uygulayıcıların ihtiyaçlarını karşılanmasında kat edilmesi gereken uzun bir yol var.

Gereksinimler nelerdir?

Gereksinim mühendisliğindeki zorluğun bir kısmı, bir "gereksinimin" gerçekte ne olduğunu anlamakla ilgilidir. Gereksinimler, üst düzey soyut ifadeler ve kağıt üstü taslaklardan resmi spesifikasyonlara kadar değişebilir. Bu değişen temsil biçimleri, paydaşların farklı seviyelerde ihtiyaçları olduğu ve dolayısıyla farklı soyutlama temsillerine bağlı olduğu için ortaya çıkar. Paydaşlar ayrıca bu temsilleri yapmak ve okumak için çeşitli yeteneklere sahiptir ve bu da gereksinimlerde çeşitli kalitelere yol açar. Paydaşların doğasını, ihtiyaçları ve yeteneklerini bir sonraki bölümde tartışacağız.

Gereksinimler ve Hedefler

Gereksinim mühendisi için temel bir zorluk müşterilerinin gereksinimleri ve hedefleri sıklıkla karıştırdığını kabul etmektir.

Hedefler, bir işletmenin organizasyonun veya sistemin üst düzey hedefleridir ancak bir gereklilik, önerilen bir sistem tarafından bir hedefe nasıl ulaşılacağını belirtir. Dolayısıyla, Ulaştırma Bakanlığının hedefi "dünyanın en güvenli köprüsünü inşa etmek" olabilir. Gerçekten burada amaçlanan nedir? Amaçlanan, güvenli bir köprüyü sağlayacak köprü malzemeleri, müteahhit ve mühendislerin nitelikleri ve yapım teknikleri ile ilgili performans gerekliliklerini belirlemektir. Bir hedefi bir gereklilik olarak ele almak soruna davetiye çıkarmaktır çünkü hedefe ulaşıldığını kanıtlamak zor olacaktır. Ayrıca, paydaşlar fikirlerini değiştirdikçe ve hedefleri davranışsal gereksinimlere dönüştürdükçe ve işledikçe hedefler de gelişir.

Gereksinim Seviyesi Sınıflandırması

Gereksinim türlerindeki çeşitlilikle başa çıkmak için Sommerville (2005), bunları üç soyutlama düzeyinde düzenlemeyi önerir.

- Kullanıcı gereksinimleri
- Sistem gereksinimleri
- Tasarım özellikleri

Kullanıcı gereksinimleri, resmi olmayan diyagramlarla birlikte doğal dilde yazılmış soyut ifadelerdir. Sistemin sağlaması beklenen hizmetleri (kullanıcı işlevselliği) ve herhangi bir kısıtlamayı belirtirler. Toplanan kullanıcı gereksinimleri genellikle bir "operasyon konsepti" (Conops) belgesi olarak görünür. Birçok durumda kullanıcı hikayeleri (user-story), kullanıcı gereksinimleri rolünü oynayabilir.

Sistem gereksinimleri, hizmetlerin ve kısıtlamaların ayrıntılı açıklamalarıdır. Sistem gereksinimlerine bazen fonksiyonel spesifikasyon veya teknik ek olarak atıfta bulunulur. Bu gereksinimler, kullanıcı gereksinimlerinin analizinden türetilir ve yapılandırılmış ve kesin olmalıdır. Gereksinimler, bir sistem gereksinimleri belirtimi (SRS) belgesinde toplanır.

Kullanıcı durum senaryoları, birçok durumda sistem gereksinimi rolünü oynayabilir

Son olarak, tasarım özellikleri, geliştiriciler tarafından uygulanması için temel olarak kullanılan analiz ve tasarım belgelerinden ortaya çıkar. Sistem tasarım spesifikasyonu, esas olarak, sistem gereksinimleri spesifikasyonunun analizinden doğrudan türetilir.

Bu spesifikasyon seviyelerindeki farklılıkları göstermek için havayolu bagaj taşıma sisteminden aşağıdakileri göz önünde bulundurun:

Bir kullanıcı gereksinimi,

- Sistem dakikada 20 bagaj işleme tabi tutabilecektir.

Bazı ilgili sistem gereksinimleri,

- İşlenen her bagaj bir bagaj olayını tetikleyecektir.
- Sistem dakikada 20 bagaj olayını kaldırabilecektir.

Son olarak ilgili sistem özellikleri,

1.2 Sistem, operasyonel modda dakikada 20 bagaj olayını işleyebilecektir.

1.2.1 Bir dakikalık aralıklarla 20'den fazla bagaj olayı meydana gelirse, sistem…

1.2.2 [daha fazla istisna işleme]…

Bir evcil hayvan mağazası POS sistemi için aşağıdakileri göz önünde bulundurun:

Bir kullanıcı gereksinimi,

- Sistem, indirimler, vergiler, geri ödemeler ve indirimler dahil olmak üzere satış toplamlarını doğru bir şekilde hesaplayacak, doğru bir makbuz yazdırıp, envanter sayımlarını buna göre güncelleyecektir.

Bazı ilgili sistem gereksinimleri,

- Her satışa bir satış ID’si atanacaktır.
- Her satışta bir veya daha fazla satış öğesi olabilir

- Her satışta bir veya daha fazla indirim olabilir.
- Her satışın yazdırılmış yalnızca bir makbuzu olabilir

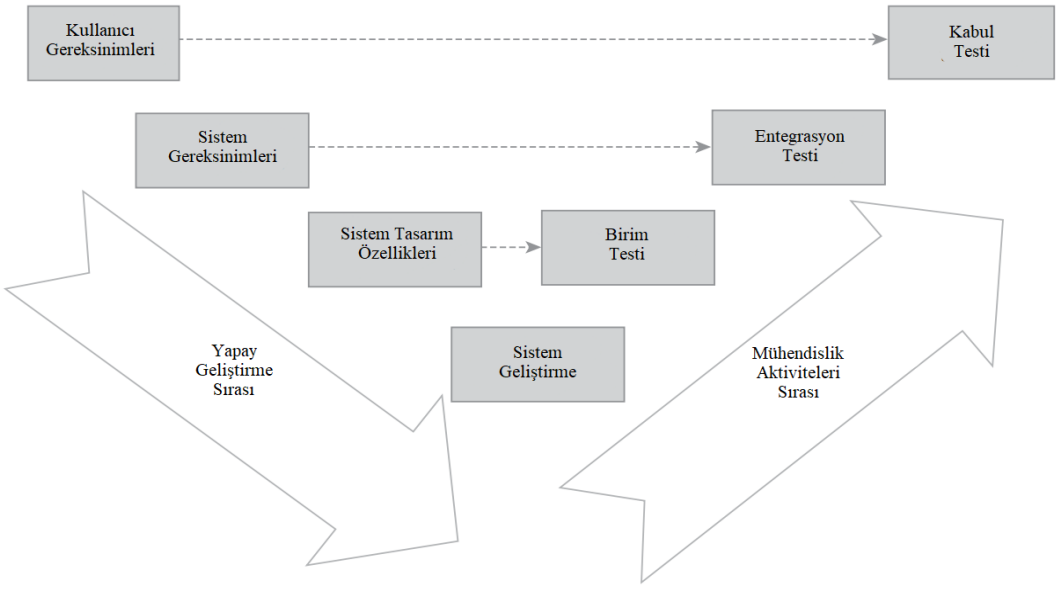
Son olarak ilgili sistem özellikleri,

1.2 Sistem, her satış işlemine benzersiz bir satış kimlik numarası atayacaktır.

1.2.1 Her satış kimliğinin kendisiyle ilişkilendirilmiş sıfır veya daha fazla satış öğesi olabilir, ancak her satış öğesi tam olarak bir satış kimliğine atanmalıdır.

Eklerdeki sistem özellikleri ayrıca, keşfetmeniz için seviyeye göre düzenlenmiş çok sayıda spesifikasyon içerir. Farklı spesifikasyon seviyeleri, proje yaşam döngüsü boyunca aşamalı testlere rehberlik eder (Resim 1.1).

Böylece ilk keşfedilen kullanıcı gereksinimleri, nihai kabul testi için temel olarak kullanılır. Genellikle kullanıcı gereksinimlerinden sonra geliştirilen sistem gereksinimleri, kabul testinden önce gelen entegrasyon testi için temel olarak kullanılır. Ve son olarak, sistem gereksinimlerinden türetilen tasarım özellikleri, her bir kod birimi uygulandıkça birim testi için kullanılır.



Resim 1.1 **Gereksinim spesifikasyon seviyeleri ve test arasındaki ilişki**

Gereksinim Özellikleri Tipleri

Gereksinim özellikleri için başka bir sınıflandırma, aşağıdaki olasılıklar listesinden gereksinim türüne odaklanır.

- Fonksiyonel gereksinimler (FRs)
- Fonksiyonel olmayan gereksinimler (NFRs)
- Domain gereksinimleri

Şimdi bunlara daha yakından bakalım.

Fonksiyonel Gereksinimler

Fonksiyonel gereksinimler (FRs), sistemin sağlaması gereken hizmetleri tanımlar ve sistemin girdilerine nasıl tepki vereceği. Ek olarak fonksiyonel gereksinimler, sistemin yapmaması gereken belirli davranışları açıkça belirtmesi gerekir. Fonksiyonel gereksinimler yüksek düzeyde ve genel olabilir (bu durumda bunlar daha önce açıklanan kullanıcı gereksinimleridir) veya ayrıntılı olabilir, girdileri, çıktıları, istisnaları vb. ifade edebilir (bu durumda önceden açıklanan sistem gereksinimleridir).

Doğal dilden, görsel modellerden ve daha titiz biçimsel yöntemlerden, fonksiyonel gereksinimler için birçok temsil biçimi vardır. 4. bölümde gereksinimlerin temsilini tartışmak için çok daha fazla zaman harcayacağız.

Bazı işlevsel gereksinimleri göstermek için, bagaj taşıma sistemi için aşağıdaki örneklemeyi göz önünde bulundurun.

2.1 Sistem dakikada 20 bagaja kadar taşımalıdır.

…

2.4 Sistem boşta iken konveyör bant hareket etmeyecektir.

…

2.8 Ana güç kesilirse, sistem 5 saniye içinde düzenli bir şekilde kapanacaktır.

…

2.41 Konveyör bant motoru arızalanırsa, sistem giriş besleme mekanizmasını 3 saniye içinde kapatacaktır.

Evcil hayvan mağazası POS sistemi için aşağıdakiler bazı işlevsel gereksinimler olabilir:

4.1 Operatör “total” düğmesine bastığında mevcut satış indirim duruma girer.

4.1.1 Bir ürün indirim duruma girdiğinde, her satılmayan kalemin toplamı, kalem sayısı ile kalemin liste fiyatının çarpımı olarak hesaplanır.

4.1.2 Bir satış indirim duruma girdiğinde, her bir satış kalemi için bir toplam hesaplanır.

Fonksiyonel Olmayan Gereksinimler

Yazılım sistemleri hem işlevsel davranışlarıyla (sistemin yaptığı) hem de işlevsel olmayan davranışlarıyla (sistemin güvenilirlik, yeniden kullanılabilirlik, sürdürme yeteneği gibi bazı gözlemlenebilir niteliklere göre nasıl davrandığı) ile karakterize edilir.

Fonksiyonel olarak eşdeğer ürünlerin aynı müşteri için rekabet ettiği yazılım pazarında, rakip ürünleri ayırt etmede işlevsel olmayan gereksinimler (NFR'ler) daha önemli hale gelir. Bununla birlikte, uygulamada, NFR'ler, FR'lere (Weber) göre hala çok az ilgi görmektedir. Bu sorun, temel olarak, NFR'lerin, geliştirme sürecinin erken bir aşamasında tedavi etme seçimini yaparken zorluk yaratan benzersiz doğasından kaynaklanmaktadır. NFR'ler özneldir, görecelidir ve gereksinimler alanından çözüm alanına eşleştirildiklerinde çoklu modüller arasında dağılma eğilimindedirler. Ayrıca, NFR'ler, bir NFR'ye ulaşma girişimlerinin diğer NFR'lerin başarılmasına yardımcı olabileceği veya engelleyebileceği anlamında sıklıkla etkileşime girebilir. Örneğin, yazılım güvenliğini artırma girişimi, performans pahasına olabilir (örneğin, gecikmeyi artırarak performansı düşürme). Bu tür etkileşimler, NFR'ler arasında izlemesi veya tahmin etmesi kolay olmayan kapsamlı bir karşılıklı bağımlılık ve değiş tokuş ağı yaratır (Chung ve diğerleri 2000).

NFR'lerin zorlu doğasına rağmen, raporlar sürekli olarak, bunların ihmal edilmesinin feci proje başarısızlıklarına veya en azından önemli gecikmelere ve sonuç olarak nihai maliyette önemli artışlara yol açabileceğini göstermektedir. Aşağıdaki liste birkaç örnek sağlar.

- 1992'de Londra Ambulans Servisi (LAS), halktan ve acil servislerden gelen çağrılara yanıt olarak ambulansları dağıtan sistemi otomatikleştirmeyi amaçlayan yeni bir bilgisayar destekli sevk sistemini tanıttı. Bu yeni sistem son derece verimsizdi ve ambulans müdahale süreleri önemli ölçüde arttı. Tanıtımından kısa bir süre sonra tamamen başarısız oldu ve LAS önceki manuel sisteme geri döndü. Sistemin başarısızlığı, esas olarak, sistemin tasarımında "insan ve organizasyonel faktörleri" dikkate almamaktan kaynaklanıyordu (Finkelstein ve Dowell 1996).

- Bir NASA Mars Climate Orbiter uzay aracı, bir yazılım "birlikte çalışabilirlik" sorunu nedeniyle başarısız oldu. Araç, yolculuğu sırasında rotasından saptı ve planlanandan çok daha düşük bir yörüngeye girdi ve atmosferik sürtünme tarafından yok edildi. Gemiyi yok eden metrik/İngiliz birimleri karışımı, Dünya'daki bir yazılım hatasından kaynaklandı. Dönme hızını kontrol etmesi amaçlanan uzay aracı üzerindeki iticiler, iticilerin etkisini 4.45 kat hafife alan bir bilgisayar tarafından kontrol ediliyordu. Bu, İngiliz sisteminde standart kuvvet birimi olan pound kuvveti ile metrik sistemdeki standart birim olan Newton arasındaki orandır. Dünya'daki yazılım pound gücünde çalışıyordu, uzay aracı ise Newton'da rakamlar bekliyordu (Breitman ve arkadaşları 1999)

- New Jersey Motorlu Taşıtlar Departmanı'nın lisanslama sistemi, geliştirme süresinden tasarruf sağlamak amacıyla dördüncü nesil bir programlama dilinde yazılmıştır. Ancak uygulandığında sistem o kadar yavaştı ki bir noktada milyonlarca New Jersey aracı işlenmemiş lisans yenilemeleriyle sokaklarda dolaştı. Proje, "uygun fiyat" ve "zamanlılık" hedeflerini karşılamayı amaçladı, ancak "performans ölçeklenebilirliği" sorunları nedeniyle başarısız oldu. (Babcock 1985)

- Ulusal Tıp Kütüphanesi MEDLARS II sistemi başlangıçta çok çeşitli gelecekteki yayın sistemlerini desteklemek için birçok soyutlama katmanıyla geliştirilmiştir. Sistemin ilk odak noktası, "taşınabilirlik" ve "gelişebilirlik" niteliklerini geliştirmeye

yönelikti. Sistem, "performans" sorunları nedeniyle iki pahalı donanım yükseltmesinden sonra hurdaya çıkarıldı. (Boehm ve In 1996)

NFR'lerin bu bariz önemine ve uygunluğuna rağmen, uygulama tamamlandıktan sonra neredeyse her zaman doğrulanmaya bırakılırlar, bu da NFR'lerin gereksinim mühendisliğinden uygulamaya doğrudan ve açık bir şekilde haritalanmadığı anlamına gelir (Matoussi ve Laleau 2008). Bu durum, esas olarak, yazılımı mümkün olduğunca hızlı dağıtmaya yönelik muazzam baskıdan kaynaklanmaktadır. Bu baskı, yazılım geliştirmeyi, sürecin çok geç saatlerine kadar tespit edilmeyen eski gereksinim hataları sorununun potansiyel olarak alevlenmesiyle karşı karşıya bırakır. Neto ve arkadaşları. (2000), NFR'lerin ihmal edilmesinden dolayı yazılım geliştirmenin iyi bilinen bazı problemlerini sıralamaktadır: (i) maliyet ve zamanlama aşımları, (ii) yazılım sistemlerinin durdurulması ve (iii) yazılım sistemleri kullanıcılarının memnuniyetsizliği. Tüm bunlar için, NFR'lerin yazılım/sistem yaşam döngüsünün tüm seviyelerini etkilemesi ve mümkün olan en kısa sürede tanımlanması ve bunların ortaya çıkarılmasının doğru ve eksiksiz olması gerektiğini teyit etmek önemlidir.

NFR'lerle uğraşmanın ilk adımı, NFR teriminin nasıl tanımlanacağı konusunda bir anlaşmaya varmaktır. Literatürde bu tür birçok tanım varken, NFR'yi “sistemin çalışacağı ortamın dayattığı gereksinimler” olarak tanımlıyoruz. Bu durumda, "çevre", açıkça "işlevsel" olarak tanımlanmayan tüm gereksinimleri kapsayan bir şemsiye terimdir.

Birçok gereksinim kategorisi çevreyi oluşturur. Beş ortak NFR kategorisini tanımlıyoruz: kalite, tasarım, ekonomik, işletim ve politik/kültürel.

Kalite gereksinimleri, NFR dünyasındaki en önemli kategoridir. Kalite, bir işletmenin belirtilen ve ima edilen ihtiyaçları karşılama kabiliyetine dayanan özelliklerin toplamıdır (ISO12601). Sistem kalitesi, nihai ürünün temel ve ayırt edici bir özelliğidir. Tipik kalite gereksinimleri arasında güvenlik, gizlilik, güvenilirlik, kullanılabilirlik ve sürdürülebilirlik gereksinimleri bulunur. Genel olarak, bir İngilizce "ility" ile biten herhangi bir yazılım kalitesi veya niteliği, işlevsel olmayan bir gerekliliktir. Bu sözde ilişkiler, yasalar ve yönetmelikler, standartlar, çevresel kısıtlamalar ve başka yerler dahil olmak üzere birçok kaynaktan türemiştir.

Yazılım/sistem ürün kalitesi, dahili nitelikleri (tipik olarak ara ürünlerin statik ölçümleri), harici nitelikleri ölçerek veya kullanımda kalite niteliklerini (kullanıcının ürünün kalitesine ilişkin görüşünü temsil eden) ölçerek değerlendirilebilir. Belirli bir ortamda ve belirli bir kullanım bağlamında kullanılır). Şekil 1.2, yazılım/sistem yaşam döngüsünün farklı aşamalarında ürün kalitesinin üç görünümünü sunar.

Pek çok yaklaşım, yazılım kalitesini, daha sonra alt karakteristiklere ayrıştırılan yapısal bir dizi özellik içinde sınıflandırır. Örneğin, Kassab (2009) 87 nitelik için bir kalite taksonomisi sunmuştur.

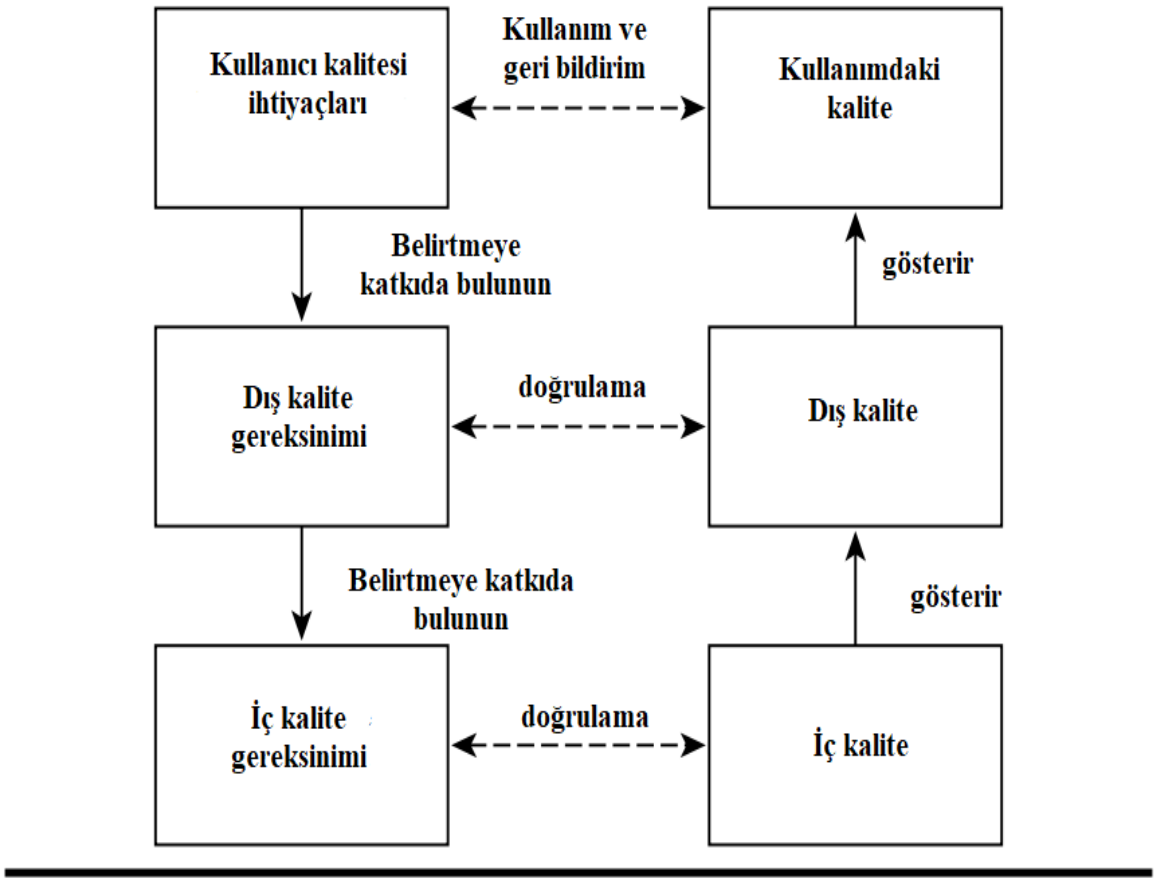


Figure 1.2 Quality in the software/systems lifecycle (ISO12601).

Tasarım/Uygulama Kısıtlamaları: Kısıtlamalar genellikle müzakereye tabi değildir ve bir

kez üzerinde anlaşmaya varıldıktan sonra tasarım değiş tokuşları sırasında sınırsızdır. Kısıtlamalar, sistemin tasarımına veya bir sistemin geliştirildiği sürece ilişkin kısıtlamalar olarak tanımlanır. Sistemin dış davranışını etkilemez, ancak teknik, ticari veya sözleşmeden doğan yükümlülükleri yerine getirmek için yerine getirilmesi gerekir (Leffingwell ve Widrig 2003). Bir kısıtlamanın önemli bir özelliği, kısıtlamaya uyulmaması durumunda bir tür ceza veya kaybın geçerli olmasıdır. Tasarım/uygulama kısıtlamalarına bir örnek, belirli mimari kalıpların veya belirli programlama dillerinin kullanımına ilişkin kısıtlamaları içerir.

Ekonomik Kısıtlamalar: Bunlar, acil ve/veya uzun vadeli geliştirme maliyetini içeren kısıtlamalardır.

Çalışma Kısıtlamaları: Bunlar, fiziksel kısıtlamaları, personel kullanılabilirliğini, beceri düzeyindeki hususları, bakım için sistem erişilebilirliğini vb. içeren kısıtlamalardır.

Politik/Kültürel Kısıtlamalar: Bunlar, politika ve yasal konuları içeren kısıtlamalardır (örneğin, ürün için hangi kanun ve standartların geçerli olduğu).

Varlıkları her zaman proje bağlamındaki diğer kavramlara/gerekliliklere bağlı olduğundan, NFR'lerin bağımsız gereksinimler olmadığını bilmek önemlidir. Örneğin, NFR'ler işlevsel gereksinimlerle ilişkilendirilebilir: "Yalnızca yetkili kullanıcılar sistemin X işlevine erişebilmelidir" veya proje için gereken kaynaklarla ilişkilendirilebilir: "Yazılım bakımcılarının en az iki yılı olmalıdır. Oracle veritabanında deneyim.”

Varlıkları her zaman proje bağlamındaki diğer kavramlara/gerekliliklere bağlı olduğundan, NFR'lerin bağımsız gereksinimler olmadığını bilmek önemlidir. Örneğin, NFR'ler işlevsel gereksinimlerle ilişkilendirilebilir: "Yalnızca yetkili kullanıcılar sistemin X işlevine erişebilmelidir" veya proje için gereken kaynaklarla ilişkilendirilebilir: "Yazılım bakımcılarının en az iki yılı olmalıdır. Oracle veritabanında deneyim.” NFR'nin ne olduğu ile yerine getirilmesi için neler gerektirebileceği arasında ayrım yapmak da önemlidir. Örneğin, e-posta mesajlarını okurken güvenli etkileşim gereksinimi NFR olarak sınıflandırılır. Ancak bu gereksinimin karşılanması, kimlik doğrulama, yetkilendirme veya mesaj şifreleme gibi bazı teknik tasarım/mimari kararların alınmasını gerektirebilir. Üstelik hedefleri NFR'lerle karıştırmak kolaydır.

Bir hedefin bir paydaşın genel bir niyeti olduğunu unutmayın, örneğin:

Sistem deneyimli operatörler tarafından kullanımı kolay olmalıdır

Doğrulanabilir bir NFR, bazı nesnel ölçüleri kullanan bir ifadedir:

Deneyimli operatörler, %0,5'ten fazla olmayan bir hata oranıyla iki saatlik uygulamalı eğitmen liderliğindeki eğitimden sonra aşağıdaki 18 sistem özelliğini kullanabilecektir.

Burada, "deneyimli" bir kullanıcının hangi özelliklerde ustalaşması gerektiği (kısa olması için burada listelenmemiş olsalar da), gerekli olan eğitim türünün tanımı (örneğin, eğitmen liderliğindeki, örneğin, kendi kendine çalışma) ve hiçbir insanın mükemmel olmadığını anlamak için yeterli boşluk.

Domain Gereksinimleri

Domain gereksinimleri, uygulama etki alanından türetilir. Bu tür gereksinimler, yeni işlevsel gereksinimlerden veya mevcut işlevsel gereksinimler üzerindeki kısıtlamalardan oluşabilir veya belirli hesaplamaların nasıl yapılması gerektiğini belirtebilirler.

Örneğin bagaj taşıma sisteminde çeşitli alan gerçeklikleri gereksinimler yaratır. Endüstri standartları vardır (yeni sistemin diğer havayollarının sistemlerine kıyasla daha düşük performans göstermesini istemeyiz). Mevcut mevcut donanım (örneğin, konveyör sistemleri) tarafından dayatılan kısıtlamalar vardır. Ve bagaj görevlileri sendikası ile yapılan toplu iş sözleşmeleri tarafından zorunlu kılınan performans üzerinde kısıtlamalar olabilir.

Evcil hayvan mağazası POS sistemi için alan gereksinimleri, geleneksel mağaza uygulamaları tarafından empoze edilir. Örneğin:

- Nakit, kredi kartları ve kuponların işlenmesi
- Ekran arayüzü ve makbuz formatı
- Evcil hayvan mağazası endüstrisindeki sözleşmeler
- Pound ile ürün satışı

Domain Sözlüğünü Anlayışı

Farklı alanlarda terimlerin kullanımında ince ve derin farklılıklar olabileceğinden, gereksinim mühendisi uygulama alanı kelime dağarcığını tam olarak anladığından emin olmalıdır (veya bu kelime dağarcığına hâkim birinin hazır olması gerekir). Aşağıdaki gerçek olay bu noktayı göstermektedir. Yazardan bir keresinde çok büyük, uluslararası bir paket dağıtım şirketine danışmanlık hizmeti vermesi istendi. Paket teslimat şirketinin mühendisleriyle birkaç saat iletişim kurduktan sonra, yazarın “kamyon” terimini yanlış kullandığı ortaya çıktı. Yazar, "kamyonun", paketleri doğrudan müşterilerin evlerine teslim edecek tanıdık araca atıfta bulunduğuna inanırken, şirket bu araçlara atıfta bulunmak için "paket araba" terimini kullandı. "Kamyon" terimi, bir dağıtım merkezinden diğerine büyük miktarda paket taşıyan herhangi bir uzun mesafeli araç (genellikle 18 tekerlekli bir kamyon) anlamına geliyordu. Dolayısıyla, bir "kamyonda" ve bir "paket vagonda" taşınan paketlerin hacmi arasında büyük bir fark vardı. O zaman, “1.000 kamyondan gelen paketlerin” işlenmesini içeren bir şartın yazıldığını hayal edin (gerçekten “1.000 paket araba” anlamına geldiğinde). Açıkça, alan terminolojisi anlayışındaki bu fark, önemli ve potansiyel olarak maliyetli olurdu.

Gereksinim Mühendisliği Aktiviteleri

Gereksinim mühendisi bir dizi faaliyetten sorumludur. Bunlar şunları içerir:

- Gereksinimlerin ortaya çıkarılması
- Gereksinim analizi ve uzlaşması
- Gereksinim gösterimi ve modelleme
- Gereksinim doğrulama ve doğrulama
- Gereksinim yönetimi

Bu faaliyetlerin her birini aşağıdaki bölümlerde kısaca ve sonraki bölümlerde ise daha fazla inceleyeceğiz.

Gereksinimi Oraya Çıkarma ve Keşif

Gereksinimlerin ortaya çıkarılması/keşfi, müşterinin neye ihtiyaç duyduğunu ve istediğini ortaya çıkarmayı içerir. Ancak ortaya çıkarma, bir ağaçtan alçakta asılı meyveleri hasat etmek gibi değildir. Bazı gereksinimler açık olsa da (örneğin, POS sisteminin satış vergisini hesaplaması gerekecek), birçok gereksinimin iyi tanımlanmış yaklaşımlarla müşteriden çıkarılması gerekecektir. Gereksinim mühendisliğinin bu yönü, paydaşların kim olduğunun ortaya çıkarılmasını da içerir; örneğin, gizli paydaşlar var mı? Çıkarma ayrıca genellikle gözden kaçan NFR'lerin belirlenmesini de içerir.

Gereksinim Analizi ve Anlaşması

Gereksinim analizi ve gereksinim anlaşması, gereksinimlerle ilgili bir dizi sorunu "ham" biçiminde, yani müşterilerden toplandıktan sonra ele almaya yönelik teknikleri içerir. Ham gereksinimlerle ilgili sorunlar şunları içerir:

- Her zaman anlam ifade etmezler.
- Genellikle birbirleriyle çelişirler
- Tutarsız olabilirler.
- Eksik olabilirler.
- Belirsiz veya sadece yanlış olabilirler.
- Etkileşime girebilir ve birbirlerine bağımlı olabilirler.

Daha sonra tartışacağımız ortaya çıkarma tekniklerinin çoğu, bu sorunları önlemeye veya hafifletmeye yöneliktir. Formel yöntemler de bu konuda faydalıdır. Gereksinim analizi ve anlaşma Bölüm 4'te tartışılmaktadır.

Gereksinim Yönetimi

Gereksinim mühendisliğinin en çok gözden kaçan yönlerinden biri olan gereksinim yönetimi, zaman içinde değişen gereksinimlerin gerçeklerini yönetmeyi içerir. Aynı zamanda, gereksinimlerin uygun şekilde bir araya getirilmesi ve bunlara tabi kılınması ve gereksinimlerdeki değişikliklerin bilinmesi gereken kişilere iletilmesi yoluyla izlenebilirliği geliştirmeyi de içerir. Yöneticilerin ayrıca, kapsam kayması meydana geldiğinde akıllıca geri adım atma becerilerini de öğrenmeleri gerekir. Değişiklikleri izlemek ve izlenebilirliği sürdürmek için araçlar kullanmak, gereksinim yönetiminin yükünü önemli ölçüde azaltabilir. Bölüm 8'de gereksinim mühendisliğine yardımcı olacak yazılım araçlarını ve Bölüm 9'da genel olarak gereksinim yönetimini tartışacağız.

Bilgi Organları

Yazılım sistemlerinin gereksinim mühendisliği için üç önemli yapılandırılmış sınıflandırma veya bilgi yapısı mevcuttur: Yazılım Mühendisliği Bilgi Grubu Sürüm 3.0 (SWEBOK 2014), Lisansüstü Yazılım Mühendisliği Referans Müfredatı (GSwE 2009) ve Yazılım İlkeleri ve Uygulamaları (P&P) Mühendislik Sınav Şartnamesi (Kilcay-Ergin ve Laplante 2013). Bu bilgi yapıları, yazılım mühendisliği disiplinine odaklanır, ancak yazılım, elektrik, mekanik veya hibrit olsun, her türlü sistemin mühendisliğine uygulanabilirler.

SWEBOK, tutarlı bir yazılım mühendisliği görüşünü desteklemek ve müfredat geliştirme ile sertifika ve lisanslama sınavları için bir temel sağlamak amacıyla kurulmuştur. SWEBOK, yazılım mühendisliği yüksek lisans programından mezun olan tüm kişilerin sahip olması gereken temel bilgi ve becerileri tanımlar.

GSwE raporu, yazılım mühendisliği ve sistem mühendisliği arasındaki güçlü bağı vurgular ve sistem mühendisliği bilgi alanlarının yazılım mühendisliği müfredatına entegre edilmesi gerektiğini önerir (Kilcay-Ergin ve Laplante 2013)

Yazılım Mühendisliği İ&P sınavı, Amerika Birleşik Devletleri eyaletleri ve yargı bölgeleri tarafından, halkın sağlığını, güvenliğini ve refahını etkileyen yazılım sistemleri üzerinde çalışan kişiler için bir lisans bileşeni olarak kullanılır. (Laplante ve diğerleri 2013)

Taksonomilerin amacı farklı olduğundan, gerekli bilginin kapsamı da bazı yönlerden farklılık gösterir, örneğin, SWEBOK'ta GSwE'nin başlatılması ve kapsam tanımının net bir kapsamı yoktur. Bu bilgi grupları için bilgi kapsamının bir karşılaştırması Tablo 1.1'de gösterilmektedir.

Bu metin, SWEBOK'teki çoğu konunun makul bir kapsamını sağlamayı amaçlamaktadır; bununla birlikte, diğer iki bilgi gövdesini önemli ölçüde kapsar.

Sistem mühendisliği ile ilgili yeni ortaya çıkan referans müfredat ve bilgi yapıları da vardır, örneğin, Sistem Mühendisliği Bilgi Grubu Kılavuzu (SEBoK 2012), Sistem Mühendisliği Lisansüstü Referans Müfredatı (GRCSE 2012) ve Bilgisayar Makineleri Derneği (ACM)/Institute for Electrical and Electronics Engineers (IEEE) Computer Society'nin Yazılım Mühendisliği Lisans Derecesi Programları için Müfredat Yönergeleri (ACM/IEE 2014). Bunlar ayrıca önemli ölçüde gereksinim mühendisliği süreçleri ve faaliyetlerine odaklanır.

Table 1.1 Comparison of GSwE, SWEBOK, and Software Engineering P&P Knowledge Areas

GSwE 2009	*SWEBOK Version 3.0*	*Software Engineering P&P*
Fundamentals of RE • Relationship between systems engineering and software engineering • Definition of requirements • System design constraints • System design and requirements allocation • Product and process requirements • Functional and nonfunctional requirements • Emergent properties • Quantifiable requirements	**Software Requirements Fundamentals** • Definition of a software requirement • Product and process requirements • Functional and nonfunctional requirements • Emergent properties • Quantifiable requirements • System requirements and software requirements	**Software Requirements Fundamentals** • Concept of operations • Types of requirements • Product and process requirements • Functional and nonfunctional requirements • Quantifiable requirements • System requirements • Software requirements • Derived requirements • Constraints, service level
RE Process • Process models • Process actors • Process support and management • Process quality and improvement	**Requirements Process** • Process models • Process actors • Process support and management • Process quality and Improvement	
Initiation and Scope Definition • Determination and negotiation of requirements • Feasibility analysis • Process for requirements review/revision	No equivalent	

(Continued)

Table 1.1 (Continued) Comparison of GSwE, SWEBOK, and Software Engineering P&P Knowledge Areas

GSwE 2009	*SWEBOK Version 3.0*	*Software Engineering P&P*
Requirements Elicitation • Requirements sources • Elicitation techniques	**Requirements Elicitation** • Requirements sources • Elicitation techniques	**Requirements Elicitation** • Requirements sources • Elicitation techniques • Requirements representation
Requirements Analysis • Requirements classification • Conceptual modeling • Heuristic methods • Formal methods • Requirements negotiation	**Requirements Analysis** • Requirements Classification • Conceptual modeling • Architectural design and requirements allocation • Requirements negotiation • Formal analysis	
Requirements Specification • Requirements specification techniques	**Requirements Specification** • The System definition document • System requirements specification • Software requirements specification	**Requirements Specification** • System definition document • System/subsystems specification • Software requirements specification • Interface requirements specification
Requirements Validation • Requirements reviews • Prototyping • Model validation • Acceptance tests	**Requirements Validation** • Requirements reviews • Prototyping • Model validation • Acceptance tests	**Requirements Verification and Validation** • Requirements reviews • Prototyping • Model validation • Simulation

(Continued)

Table 1.1 (Continued) Comparison of GSwE, SWEBOK, and Software Engineering P&P Knowledge Areas

GSwE 2009	*SWEBOK Version 3.0*	*Software Engineering P&P*
Practical Considerations • Iterative nature of requirements process • Change management • Requirements attributes • Requirements tracing • Measuring requirements	**Practical Considerations** • Iterative nature of the requirements process • Change management • Requirements attributes • Requirements tracing • Measuring Requirements	**Requirements Management** • Iterative nature of the requirements process • Change management • Requirement attributes • Requirements traceability • Measuring requirements • Software requirements tools

Source: Adapted from Kilcay-Ergin, N., and Laplante, P.A., *IEEE Trans. Educ.*, 56: 199–207, 2013.

GEREKSİNİM MÜHENDİSLİĞİ

Bir gereksinim mühendisinin sahip olması gereken özellikler nelerdir? Christian and Chang, gereksinim mühendisinin organize olmasını, (yazılım) mühendisliğin yaşam döngüsü boyunca deneyime sahip olmasını, ne zaman genel ne zaman spesifik olması gerektiğini bilen ve gerektiği zaman müşterinin karşısında durabilecek olgunluğa sahip olmasını önermiştir (Christensen ve Chang 1996.) Christensen and Chang ayrıca gereksinim mühendisinin iyi bir yönetici (süreci yönetmek), iyi bir dinleyici, adil, iyi bir müzakereci ve multidisiplinci(örneğin, geleneksel sert bir geçmişe sahip olmak iletişim ve yönetim becerileri ile güçlendirilmiş bilimler ve mühendislik). olması gerektiğini öne sürüyor. Nihayet, gereksinim mühendisi problemin etki alanını anlamalıdır.

Gorla ve Lam(2004) mühendis düşünebilen, algılayan ve Myers-Briggs yöntemiyle yargılayan demektir demiştir. Bu gözlemi şu anlamda yorumlayabiliriz, gereksinim mühendislerinin yapılandırılmış ve mantıklı(düşünme), edinilen bilgilere odaklanması ve onu yorumlamaya(algılamaya) çalışmaması ve yarım bırakmak yerine sonuç araması(yargılamak) gerektiğini ileri sürmüştür.

Nihayet, Elbert 2010 yılında gereksinim mühendislerinin aşağıdaki alanlarda yetkinlik sahibi olması gerektiğini önerir:

- **Gereksinim mühendisliği**
- **Sistem mühendisliği**
- **Yönetim**
- **İletişim**
- **Biliş(vukuf,idrak,algı)**
- **Sosyal etkileşim**

Ayrıca Ebert akademik programların bu özellikleri geliştirmek için yetersiz olduğunu ve bunların yıllarca uygulanarak ve çalışarak kazanılması gerektiğini belirtiyor. Akademik kuslarla birlikte yeni gereksinim mühendislerini doğru yönde başlatabiliriz.

GEREKSİNİM MÜHENDİSİNİN ROLLERİ

Gereksinim mühendisliğinin doğasını anlamanın başka bir yolu da gereksinim mühendisinin rollerine bakmaktır. Aşağıdaki rol modellerini belirledik:

- **Yazılım sistemleri mühendisi olarak gereksinim mühendisi**
- **Konu uzmanı olarak gereksinim mühendisi (SME)**
- **Mimar olarak gereksinim mühendisi**
- **İş süreci uzmanı olarak gereksinim mühendisi**
- **Bilgisiz gereksinim mühendisi**

Gereksinim mühendisi için yukarıdan gelen hibrit roller de vardır.

YAZILIM VEYA SİSTEM MÜHENDİSİ OLARAK GEREKSİNİM MÜHENDİSİ

Gereksinim mühendislerinin birçoğunun yazılım, elektrik ya da sistem mühendisi olması olasıdır. Durum böyle olduğunda, gereksinim mühendisliği bu geliştirme modellerinden (örneğin, yazılım tasarımı) olumlu yönde etkilenir. Bu durumda tehlike; gereksinim mühendisinin, gereksinim özelliklerini geliştirmesi gerektiğinde bir tasarım oluşturmaya başlayabilmesidir.

KONU UZMANI OLARAK GEREKSİNİM MÜHENDİSİ

Birçok durumda müşteri gereksinim mühendisinin ya problem alanını anlamaya yardım etmede ya da kendi istek ve dileklerini anlamada uzmanlığı ve KOBİ olan gereksinim mühendisi arıyorlar. Bir gereksinim mühendisi bazen bir KOBİ değildir-gereksinim mühendisliği uzmanlarıdır. Gereksinim mühendislerinin KOBİ olmadığı durumlarda, KOBİ lerle güçlerinizi birleştirmeyi deneyin.

MİMAR OLARAK GEREKSİNİM MÜHENDİSİ

Yapı inşaatı, yazılım inşaatı için genellikle bir metafor olarak kullanılır. Yazarın deneyimine göre, mimarlar ve peyzaj mimarları gereksinim mühendislerine benzer roller oynarlar (ve bu benzerlik genellikle yazılım mühendislerinin lisanslı olmasını gerektirir). Daniel Berry bu konu hakkında kapsamlı bir yazı yazmıştır(Berry 1999-2003). Berry, benzer faaliyetlerin kapsam kaymasını azalttığını ve müşterileri gereksinim mühendisliği sürecine daha iyi dahil ettiğini kaydetti. Ek olarak Zachman(1987), modeli sunulmak üzere olana göre büyük ölçüde farklı olmasına rağmen, bilgi sistemleri için mimari bir metafor ortaya koydu.

Mimari(spesifikasyon şeklinde) ve yazılım/sistem özellikleri arasındaki benzerlikler Tablo1.2'de özetlenmiştir. Bir ev inşa etme ya da yenileme sürecinden geçtiyseniz bu benzerlikleri takdir edeceksiniz.

İŞ SÜRECİ UZMANI OLARAK GEREKSİNİM MÜHENDİSİ

Gereksinim mühendisliğinin faaliyetleri bir tür problem çözmeyi içerir -- müşterinin bir sorunu vardır ve bu mutlaka çözülmelidir. Çoğu zaman, eldeki sorunu çözmek, sistem davranışının ifadesini basitleştirmek amacıyla iş süreçlerindeki değişiklikler için gereksinim mühendisini tavsiye eder. İş süreci iyileştimesini yürütmek gereksinim mühendisinin görevi olmasa da, bu faydalı yanı sıklıkla kullanırlar.

Table 1.2 Sistem Mühendisi için Mimari Model

Yapı İnşaası	**Yazılım/Sistem Yapısı**
Mimar müşterilerle buluşur ve görüşmeler yapar. Notlar ve resimler alır.	Gereksinim mühendisi müşterilerle buluşur, görüşmeleri ve çıkarım tekniklerini kullanır.
Mimar kaba eskizler yapar(müşterilere gösterir, feri bildirim alır).	Gereksinim mühendisi, müşterilere gösterilecek gereksinim modellerini yapar (örneğin, prototipler, taslak SRS).
Mimar daha fazla eskiz(ör. cepheler) ve belki de daha karmaşık modeller(ör. karton, 3D sanal modeller, geçiş animasyonları yapar).	Gereksinim mühendisi gereksinimleri iyileştirir, resmi ve yarı resmi öğeler (örneğin, UML) ekler. Daha fazla prototipleme kullanılır.
Mimar, ek ayrıntılarla modeller hazırlar.(kat planları)	Gereksinim mühendisi, SRS'yi eksiksiz bir şekilde geliştirmek için yukarıda belirtilen bilgileri kullanır.
Gelecekteki modeller (örneğin, inşaat çizimleri) müteahhitlerin kullanımı içindir.	Gelecek modeller (örneğin, yazılım tasarımı belgeler) geliştiricilerin kullanımı içindir.

ERDEM OLARAK CEHALET

Berry (1995), gereksinim mühendisliği sürecinin içinde problem alanında hem deneyimsizlerin hem de uzmanların olması gerektiğini önerdi. Berry'nin gerekçesi aşağıdaki gibidir. "Cahil" insanlar "aptalca" sorular soruyorlar ve uzmanlar bu soruları cevaplıyorlar. Grubun en bilgisiz gereksinim mühendisine ihtiyaç duymak problem alanında kötü bir şey değildir çünkü problem alanında onu zor sorular sormaya ve geleneksel inançlara meydan okumaya zorlar. Tabii ki, bilgisiz gereksinim mühendisi, KOBİ'nin rolüne tamamen karşı çıkıyor.

Berry ayrıca gereksinim mühendisliğinde biçimsel yöntemlerin kullanılmasının bir tür cehalet olduğuna dikkat çekti çünkü bir matematikçi, modelleme yapmaya başlamadan önce genellikle bir uygulama alanı hakkında bilgisizdir.

MÜŞTERİNİN ROLÜ

Gereksinim mühendisi müşterinin hangi rolü oynamasını beklemeli? Müşterilerin rolleri çeşitlidir ve şunları içerir:

- Gereksinim mühendisinin neye ihtiyaç duyduklarını ve ne istediklerini anlamalarına yardımcı olmak (çıkarım ve doğrulama).
- Gereksinim mühendisinin ne istemediklerini anlamasına yardımcı olmak (çıkarım ve doğrulama).
- Gerektiğinde ve mümkün olduğunda alan bilgisi sağlamak.
- Gereksinim mühendisini kendilerinin veya başkalarının hata yaptığını fark ettiklerinde hızlı ve net bir şekilde uyarmak.
- Değişiklikleri belirlediklerinde gereksinim mühendisini hızlı bir şekilde uyarmak
- Kapsamın büyük ölçüde kaymasına neden olan "aha anları(içgörü, karar verilen nokta)" yaşama dürtülerini kontrol etme.
- Tüm anlaşmalara bağlı kalma.

Özellikle müşteri, gereksinim mühendisinin de yardımıyla aşağıdaki dört soruyu yanıtlamaktan sorumludur.

1. İstediğim sistem uygulanabilir mi?
2. Eğer öyleyse, ne kadara mal olacak?
3. Sistemi inşa etmek ne kadar sürer?
4. Sistemi inşa ve teslim etmek için plan nedir?

Gereksinim mühendisi, müşterilerin bu konularla ilgili beklentilerini yönetmelidir. Müşterilerin ve paydaşların doğasını ve rolünü bir sonraki bölümde keşfedeceğiz.

GELENEKSEL MÜHENDİSLİĞİ İLE İLGİLİ SORUNLAR

Geleneksel gereksinim mühendisliği yaklaşımları, birçoğunun halihazırda ele aldığımız(ve ele alınacak olan diğerlerini) ve bunların çoğu kolayca çözülmeyen bir dizi sorundan muzdariptir. Bu sorunlar şunlardır:

- Doğal dil sorunları (örneğin, belirsizlik, belirsizlik)
- Etki alanı anlayışı
- Karmaşıklıkla başa çıkmak (özellikle zamansal davranış)
- Zarflama sistemi davranışındaki zorluklar
- Eksiklik (eksik işlevsellik)
- Aşırı eksiksizlik (altın kaplama)
- Aşırı genişleme (tehlikeli "tümü")
- Tutarsızlık
- Yanlışlık

- ve daha fazlası

Bu sorunların çözümünü kitap boyunca ve özellikle Bölüm5'te inceleyeceğiz.

Doğal dil sorunları, doğal(insan) dillerin belirsizliklerinden ve bağlam duyarlılığından kaynaklanır. Bu dil sorunlarının sadece gereksinim mühendisleri için değil, herkes için var olduğunu biliyoruz, avukatlar ve yasa koyucular, doğal dilde yazılmış herhangi bir yasa ve sözleşmede bulunan boşlukları bularak, istismar ederek veya kapatarak geçimlerini sağlarlar.

Etki alanını anlama konusunu zaten ele aldık ve diğerleri gibi gereksinim mühendisinin kurulacak sistemin çalışacağı alanda uzman olabileceğini gözlemledik. Sistem karmaşıklığı bütün sistem mühendislerinin karşısına çıkan yaygın bir sorundur ve bu birazdan tartışılacak. Sistem davranışlarını tam olarak belirleme sorunları ve kayıp davranış sorunları oldukça zorlayıcıdır ancak en azından bir sistemdeki en belirgin işlevselliğin kullanılmasına yardımcı olabilecek bazı teknikler vardır.

KARMAŞIKLIK

Gereksinim mühendisliği faaliyetleriyle başa çıkmanın en büyük zorluklarından biri ,özellikle ortaya çıkarmak ve temsil, çoğu sistem için kompleksdir. Karmaşıklık için bir tanım bulmaya çalışmadan, herhangi bir türden önemli olan davranışı yakalamanın zorluklarının ve karmaşıklığının karmaşık kavramını gösterdiğini çok zor iddia edin. Bu tür zorluklar tekrarlanabilir basit insan çalışmalarında bile bulunur.

Örneğin, birinin sabah uyandığındaki ilk 5 dakikanı tanımlamanı istediğini düşün. Tamamen yapabilir misiniz(deneyin)? Hayır yapamazdın. Neden? Davranışlarının olabileceği çok fazla farklı yol ve çok fazla belirsizlik var. Atomik bir saatle bile her gün aynı saatte uyandığınızı iddia edemezsiniz (çünkü atom saatleri bile kusurludur). Ama tabii ki, haftanın günlerine, işe ara verip vermediğine, ve tatil günü olup olmamasına bağlı olarak farklı zamanlarda uyanıyorsu. Açıklamanda bunu hesaba katmalısın. Fakat ya hasta uyanırsan? Bu olayların sırası nasıl değişir? Kalkarken komodininizdeki su bardağını yanlışlıkla devirirseniz ne olur? Bu, davranışınızın özelliklerini değiştirir mi? Ya da tuvalete giderken köpeğinize takılırsanız? Bu örnekle devam edebilir ve bu alıştırmayı çim biçme veya yiyecek alışverişi gibi basit görevlerle tekrarlayabiliriz. Aslında, sorunu gülünçlük noktasına kadar sınırlayana kadar,

herhangi bir önemli olan insan davranışını tam olarak yakalamanın zorlayıcı ve hatta imkansız olduğunu göreceksiniz.

Şimdi karmaşık bir bilgiyi veya gömülü işletim sistemini düşünün. Böyle bir sistemi muhtemelen insanlarla etkileşime girmek zorunda kalacak. Doğrudan insan etkileşimine bağlı olmayan sistemlerde bile, gereksinimleri ortaya çıkarmayı ve tanımlamayı bu kadar zorlaştıran (ve diğer tüm gereksinim faaliyetlerini de zorlaştıran), zamansal davranışın karmaşıklığı ve ayrıca beklenmeyen olayların sorunlarıdır.

Rittel ve Webber (1973), "kötü" olarak adlandırdıkları bir dizi karmaşık problem tanımladılar. Kötü problemlerin 10 özelliği vardır:

- Kötü bir sorunun kesin bir formülasyonu yoktur.
- Kötü problemlerin durma kuralı yoktur.
- Kötü sorunların çözümleri doğru ya da yanlış değil, iyi ya da kötüdür.
- Kötü bir sorunun çözümünün acil ve nihai bir testi yoktur.
- Kötü bir soruna her çözüm, "tek seferlik bir işlemdir"; deneme yanılma yoluyla öğrenme imkanı olmadığı için her denemenin önemi büyüktür.
- Kötü problemlerin sayısız (veya ayrıntılı olarak tarif edilebilir) bir dizi olası çözümü yoktur ve plana dahil edilebilecek iyi tanımlanmış bir izin verilebilir işlemler dizisi de yoktur.
- Her kötü sorun özünde benzersizdir.
- Her kötü sorun başka bir sorunun belirtisi olarak kabul edilebilir.
- Kötü bir sorunu temsil eden bir tutarsızlığın varlığı çeşitli şekillerde açıklanabilir. Açıklama seçimi, sorunun çözümünün doğasını belirler.
- Planlayıcının (tasarımcının) yanılmaya hakkı yoktur.

Rittel ve Webber, ekonomik, politik ve toplumsal nitelikteki kötü sorunların (örneğin, açlık, uyuşturucu kullanımı ve Orta Doğu'daki çatışmalar) olduğu anlamına geliyordu; bu nedenle, gereksinim mühendisliği için uygun bir çözüm stratejisi sunmazlar. Bununla birlikte, gereksinim mühendisliğini kötü bir problem bağlamında görmek faydalıdır çünkü görevin neden

bu kadar zor olduğunu açıklamaya yardımcı olur - çünkü çoğu durumda gerçek sistemler kötü bir problemin birçok özelliğini bünyesinde barındırır.

ALTIN KAPLAMA VE SAÇMA GEREKSİNİMLER

Sistem tasarımı üzerine yazılan olağanüstü kitabında Brooks (2010), "gülünç gereksinimlere", yani teklif edildikleri zamanda teknolojinin durumu göz önüne alındığında sağlanması zor olan gereksinimlere karşı uyarıda bulunur. Başka bir tür gülünç gereksinim, gerçekleştirilmesi mümkün olsa da kullanılması pek mümkün olmayan bir gereksinimdir.

Brooks, Atlantik üzerinde uçabilmesi gereken bir “kendi kendine hareket eden helikopter” olan Comanche örneğini veriyor. Bu özelliğin sık kullanılması amaçlanmamıştı, ancak tasarımı önemli ölçüde karmaşıklaştırdı.

DEMODE GEREKSİNİMLER

Eski bir gereksinim, önemli ölçüde değişmiş(sistem geliştirme sırasında) veya söz konusu yeni sistem için artık geerli olmayan gereksinimdir. İkinci, üçüncü ve benzeri üretim sistemleri ve ilgili ürünlerden veya ürün hatlarında türetilen sistemler için (bkz. Alıştırma 1.12), gereksinim özelliklerinin bölümlerini yeniden kullanmak yaygın bir uygulamadır. Gereksinim belirtimleri yeniden kullanıldığında, genellikle eski gereksinimlerin yayılması söz konusudur. Wnuk ve diğerleri (2013) 219 katılımcıdan eski gereksinimlerle ilgili anket verilerini bildirdi. Eski gereksinimlerin, yazılım yoğun ürünler ve büyük projeler için önemli bir sorun olduğunu bulmuşlardır. Ayrıca, ankete katılan şirketlerin çoğunun, eski yazılım gereksinimleriyle başa çıkmak için süreçlere veya otomatik araçlara sahip olmadığını da buldular - ankete katılanların yalnızca %10'u, eski gereksinimlerin belirlenmesini desteklemek için araçlar kullandığını bildirdi.

Şüpheli gereksinimler, eski gereksinimlerle ilgilidir. Şüpheli gereksinimler, kökeni bilinmeyen ve/veya bilinmeyen bir amaca hizmet eden gereksinimlerdir. Bu nedenle, şüpheli bir gereklilik mevcut sistemde geçerli olmayabilir ve bu nedenle eskimiş olabilir.

Eski ve şüpheli gereksinimler, gereksinim mühendisliği yaşam döngüsünün tüm aşamalarında agresif bir şekilde tanımlanmalı ve ayıklanmalıdır. Gereksinimler için kaynaklarına

ve ilgili gereksinimlere yönelik izlenebilirlik bağlantılarının sürdürülmesi, eski ve şüpheli gereksinimleri önlemenin önemli bir yoludur. Tarih damgalama gereksinimleri (oluşturuldukları andan itibaren ve herhangi bir değişiklik için) eski gereksinimlerin önlenmesine yardımcı olabilir. Eski ve şüpheli gereksinimleri yönetmenin daha emek yoğun bir başka yolu da düzenli gereksinim incelemeleri/incelemeleri yapmaktır (5. Bölümde tartışılmıştır). Gereksinim dalgalanması, gereksinimler üzerinde anlaşmaya varıldıktan sonra gereksinimlerde meydana gelen değişiklikleri ifade eder. Gereksinim dalgalanması, birim zaman başına değişen gereksinimlerin oranı veya belirli bir zamanda toplam gereksinim sayısı başına değişen gereksinimlerin oranı olarak ölçülebilir. Gereksinimler bir sistemin yaşam döngüsü boyunca değişecektir, ancak bu değişikliklerin dikkatli bir şekilde yönetilmesi gerekir. Çeşitli aşamalardaki projelerde kayıp oranı için geçmiş istatistiklerin tutulması ve izlenmesi, hızla değişen gereksinimler gibi sorunların belirlenmesine yardımcı olabilir. Yüksek bir kayıp oranı, modası geçmiş gereksinimler için önde gelen bir gösterge olabilir.

DÖRT ÖNEMLİ NOKTA

Geleneksel gereksinim mühendisliği ile ilgili sorunların çoğu "dört karanlık köşeden" kaynaklanmaktadır (Zave ve Jackson 1997). Burada göze çarpan noktaları italik yorumlarla kelimesi kelimesine tekrarlıyoruz.

1. Gereksinim mühendisliğinde kullanılan tüm terminoloji, bir makinenin üretileceği ortamın gerçekliğine dayanmalıdır.
2. İnşa edilecek makineyi (her ne kadar soyut olarak) tanımlamak gerekli veya arzu edilmez.

   Aksine, çevre, makine olmadan veya makineye rağmen olacağı ve makine sayesinde olacağını umduğumuz gibi iki şekilde tanımlanır.

   *Spesifikasyonlar, sistem tarafından **ne** elde edileceğidir, **nasıl** değil.*
3. Resmi tanımların eylemlere odaklandığını varsayarsak, hangi eylemlerin çevre tarafından, hangi eylemlerin makine tarafından kontrol edildiğini ve çevrenin hangi eylemlerinin makine ile paylaşıldığını belirlemek esastır.

Tüm eylem türleri gereksinim mühendisliği ile ilgilidir ve resmi olarak tanımlanmaları veya sınırlandırılmaları gerekebilir. Resmi tanımlar devletlere odaklanırsa, aynı temel ilkeler biraz farklı bir biçimde uygulanır.

Resmi temsil yöntemi, sistemin temel organizasyonunu takip etmelidir. Örneğin, duruma dayalı bir sistem, en iyi duruma dayalı bir resmileştirme ile temsil edilir.

4. Gereksinim mühendisliğinde alan bilgisinin birincil rolü, gereksinimlerin uygulanabilir şartnamelere uygun hale getirilmesini desteklemektir.

   Uygun alan bilgisi ile birlikte doğru spesifikasyonlar, gereksinimlerin karşılandığını ima eder.

   Alan bilgisinin rolünün tanınmaması, doldurulmamış gereksinimlere ve yasaklanmış davranışlara yol açabilir.

Bu karanlık dönüştürücüleri yönetmek, gereksinim mühendisleri ve proje yöneticileri için önemli bir görevdir.

Bu sistemin girdi setinin bulunduğu bir ortamda çalıştığını hayal edin, I, insan operatörlerinden, sensörlerden, depolama cihazlarından ve bunun gibi diğer sistemlerden türemiştir. Çıktılar, O, görüntüleme cihazları, aktüatörler, depolama cihazları,ve bunun gibi diğer sistemler. Sisteme koyacağımız tek kısıtlama, girişler ve çıkışlar sistem içinde dijital olarak temsil edilir(analog cihazlardan veya sistemlerden geliyorlarsa, uygun bir dönüştürme gereklidir). Sistem davranışını giriş/çıkış çifti olarak tanımlıyoruz. Girdiler ve çıktılar ayrık olduğundan, bu sistem sonsuz bir sisteme sahip olarak düşünülebilir ancak sayılabilir sayıda davranış, $B \subseteq I \times O$.

B davranış uzayının Şekil 1.3'teki Venn diyagramı ile temsil edildiğini hayal edin.

Diyagramdaki en soldaki daire, müşterinin anladığı şekliyle istenen davranış setini temsil eder. Bu dairenin dışındaki alan, istenmeyen ve istenen davranıştır, hangi nedenle olursa olsun, müşteri keşfetmemiştir.

Gereksinim mühendisi işine devam eder ve müşterinin istediği davranışı temsil etmesi amaçlanan bir belirtim üretir (Şekil 1.3'ün en sağındaki daire). Bu belirtim, istenen davranışların bir kısmını (hepsini değil) yakalar, ayrıca bazı istenmeyen davranışları da yakalar. Belirtilen davranışta yakalanmayan istenen davranış, eksik işlevselliktir. Yakalanan istenmeyen davranış yasak işlevselliktir.

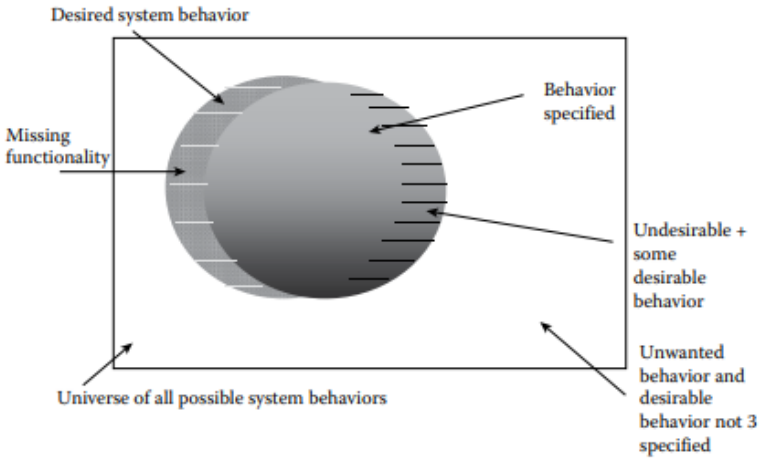


Figure 1.3 Universe of all possible behaviors, desired behavior, and specified behavior.

Özetle gereksinim mühendisinin amacı, sol ve sağ dairelerin mümkün olduğunca üst üste gelmesini sağlamak ve başlangıçta spesifikasyon tarafından kapsanmayan eksik istenen davranışları keşfetmektir(en soldaki daire). Sistemin istenen davranışlarından dışlanabilecek sonsuz sayıda davranış olduğunu kabul ederek, yine de bunların en kötüsüne odaklanmamız gerekiyor. Bu nedenle, keşfedeceğimiz ve ileteceğimiz bazı gereksinimler şu şekilde olacaktır:

"Sistem…"

Bu istenmeyen işlevselliğe bazen tehlike, ya da, "davranmaz" ya da "yasak davranış." denir. Tehlike terimi genellikle önemli veya yıkıcı başarısızlığa yol açabilecek yasaklanmış davranışı ifade eder, oysa "yasak davranış" daha az uğursuz bir çağrışıma sahiptir.

Bilinen "olmamalı" bir sorun değil, peki ya bilinmeyen yasak davranışlar? Bu bilinmeyen yasak davranışlar, bilinmeyen riskler veya belirsizlikler, bilinen yasak davranışların bilinen risklerle ilişkili olması gibi(Voas and Laplante 2010). Aşağıdaki trajik skeç, bu iki kavram arasındaki farkın bir örneğidir.

13 Ocak 1982'de karlı bir havada, bir Air Florida Boeing 737 jeti Washington DC'nin Ulusal Havaalanından ayrıldı ve havaalanından bir milden daha yakın bir mesafede 14. Cadde

Köprüsü'ne çarptı. Uçakta bulunan 79 kişiden sadece 5'i hayatta kaldı. Uçak ayrıca köprüdeki yedi araca çarparak dört sürücünün ölümüne ve dördünün de yaralanmasına neden oldu.

Uçaktaki talihsiz yolcular uçağın düşmesinin bir risk olduğunu bilmeliydiler. Bu risk ne kadar küçük olursa olsun. Bazı durumlarda, yolcular özel kaza sigortası bile yaptırmış olabilir. Peki ölen ya da yaralanan köprüdeki araçlarda bulunan sürücülerin beklentileri nelerdi? Elbette bir uçağın köprüye çarpması düşüncesi akıllarından hiç geçmedi. Bu nedenle, sürücüler için bu kaza, bilinmeyen ama belirgin bir belirsizliği temsil ediyordu.

Yasak davranışa örnek olarak bagaj taşıma sistemi için aşağıdakileri göz önünde bulundurun:

> "Konveyör sıkışması" sinyali yüksek duruma ayarlandığında, besleme mekanizması konveyör sistemine ilave maddelerin girmesine izin vermeyecektir.

Bu istenmeyen veya "yapmamalı" davranışların nasıl tanımlanacağını ve karakterize edileceğini Bölüm 3'te tartışacağız.

Spesifikasyonlarda "Hepsi" Tehlikesi

Gereksinim belirtimi cümleleri genellikle bazı evrensel nicelemeleri içerir.

(örneğin, "tüm kullanıcılar ..."e erişebilecektir). Ancak "tüm" spesifikasyonların kullanımı

tehlikeli çünkü genellikle doğru değiller (Berry and Kamsties 2000). Örneğin, bagaj taşıma sistemi için bir gereklilik şunları belirtiyorsa: *Tüm bagajlara benzersiz bir tanımlayıcı atanacaktır.*

Bu gerekliliğe meydan okumakta fayda var. Yani benzersiz bir tanımlayıcı vermek istemediğimiz bir tür "bagaj" var mı? Bir havaalanı check-in acentesine şüpheli bir paket verilirse, bir barkod etiketi yazdırıp paketin üzerine koymaları gerekir mi? Veya güvenlik personeli tarafından incelenmesi gibi daha ileri işlemler için bir kenara mı koymalılar? Cevabın "evet" veya "hayır" olması önemli değil, bu tür durumları dikkate almak ve bunları gereksinim analizi oluşana kadar ertelemek yerine, gereksinimleri taslak halinde yazarken ele almak istiyoruz. Bu nedenle, "hepsi" kelimesini içeren gereklilikler sorgulanmalı ve mümkün olduğunda gevşetilmelidir.

"Tümü" belirtimleriyle ilgili olarak "asla", "her zaman", "hiçbiri" ve "her biri" (çünkü resmi olarak evrensel nicelemeye eşitlenebilirler), bu kelimelerden ve matematiksel eşdeğerlerinden de kaçınılmalıdır.

> Amerika Birleşik Devletleri'nde, bir devlet kurumuna bir sistem sağlayan bir yüklenicinin ihtiyaç duyduğu mühendislik süreci oldukça titizdir. Savunma Bakanlığı veya Enerji Bakanlığı gibi devlet kurumları, genellikle önceden iyi tanımlanmış bir dizi gereksinime dayalı proje teklifleri isteyecektir. Örneğin, yeni bir nükleer enerji üretim tesisi için gereksinimlerin çoğu, sırasıyla büyük ölçüde standartlara, düzenlemelere ve yasalara dayanan önceki sistemlere dayanacaktır. Bu gereksinimler, sözleşmeyi yapan kurum (ve diğer paydaşlar) tarafından iş sözleşmesi verilmeden önce belirlenir.
>
> Müteahhitler, endüstri günleri aracılığıyla sözleşme teklifi oluşturmaya katılabilir. Bir endüstri günü boyunca, devlet kurumu, sistemin mevcut gereksinimlerini tartışmak için potansiyel yüklenicileri bir araya getirecektir. Bu süreç yinelemelidir ve önceden belirlenmiş gereksinimler olgunlaştıkça sistem daha iyi tanımlanmış hale gelir.
>
> Sözleşme yapıldıktan sonra gereksinim mühendisliği süreci devam eder. Kazanan yüklenici, sistem gereksinimlerini iyileştirmek için kitapta açıklanan tekniklerin birçoğunu kullanacaktır. Bu süreçte yüklenici müşteri iletişiminin iyi yapılandırılmış olması esastır. Bu nedenle, gereksinim mühendisliğine yönelik daha resmi olmayan bazı yaklaşımlar yasaklanacaktır (ve genellikle sözleşmelerde hangi metodolojilerin kullanılıp kullanılamayacağı belirtilir). Örneğin, birkaç JAD oturumu düzenlemek (Bölüm 3'te ele alınmıştır) genellikle sözleşmeye dayalı bir gerekliliktir.
>
> Açıklanan süreç, birçok ülkede hükümet sözleşmelerine benzer.

Egzersizler

1.1 Uygun gereksinim mühendisliği faaliyetlerine yönelik başlıca itirazlar ve caydırıcılardan bazıları nelerdir?

1.2 “Küçük” sistemler için gereksinim mühendisliği nasıl farklıdır?

1.3 Müşterilerin gereksinimleri değiştirmesine neden olabilecek bazı faktörler nelerdir?

1.4 Konu uzmanı olmayan bir gereksinim mühendisi gereksinimleri tanımlamaya yardımcı olması için bir konu uzmanı görevlendirdiğinde hangi sorunlar ortaya çıkabilir?

1.5 Bazı temsili kullanıcı gereksinimlerini, sistem gereksinimlerini ve yazılım özelliklerini listeleyin

- 1.5.1 Evcil hayvan mağazası POS sistemi
- 1.5.2 Bagaj taşıma sistemi

1.6 Beş tipik işlevsel gereksinimi listeleyin

1.6.1 Evcil hayvan mağazası POS sistem

1.6.2 Bagaj taşıma sistemi

1.7 Beş yasak işlevsel gereksinimi listeleyin

1.7.1 Evcil hayvan mağazası POS sistemi

1.7.2 Bagaj taşıma sistemi

1.8 İşlevsel olmayan gereksinimleri tanımlayın

1.8.1 Ek A'daki akıllı ev sistemi özellikleri

1.8.2 Ek B'deki ıslak kuyu pompalama sisteminin özellikleri

1.9 Beş olası işlevsel olmayan gereksinimi listeleyin

1.9.1 Evcil hayvan mağazası POS sistemi

1.9.2 Bagaj taşıma sistemi

1.10 Akıllı ev sistemleri için geçerli düzenlemeleri veya standartları (NFR) keşfetmek için bazı Web araştırması yapın.

1.11 Akıllı ev sistemi için, Alıştırma 1.9'da keşfettiğiniz düzenlemelere ve sahip olabileceğiniz diğer bilgilere dayanarak bazı tehlikelerin (bu sistemin yapmayacağı) bir listesini yapın.

1.12 Ürün grubu, belirli özellikleri paylaşan ilgili ürünler kümesidir. Ürün grupları planlanabilir veya ortaya çıkabilir. Ürün hatları için gereksinim mühendisliği zorluklarından bazıları nelerdir? Ritter and Webber'in kötü probleminin "tek seferlik çözüm" yönü nasıl uygulanır? Bu soruyu cevaplamak için biraz araştırma yapmak isteyebilirsiniz.

Referanslar

ACM/IEEE. (2014). Computer Society Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. https://www.acm.org/education/se2014.pdf (accessed January 2017). Babcock, C. (1985). New Jersey motorists in Software Jam. Computerworld, 30: 1–6. Berry, D. (1995). The importance of ignorance in requirements engineering. Journal of Systems and Software, 28(1): 179–184. Berry, D. M. (1999). Software and house requirements engineering: Lessons learned in combating requirements creep. Requirements Engineering Journal, 3(3&4): 242–244. Berry, D. M. (2003). More requirements engineering adventures with building contractors. Requirements Engineering Journal, 8(2):

142–146. Berry, D. M., & Kamsties, E. (2000). The dangerous ‘All’ in specifications. In Proceedings of the Tenth International Workshop on Software Specification and Design (IWSSD’00), San Diego, CA, 5–7 November. Brooks, F. P., Jr. (2010). The design of design: Essays from a computer scientist. Pearson Education. Addison-Wesley Professional, Boston. Boehm, B., & In, H. (1996). Identifying Quality-Requirement Conflicts. IEEE Software, IEEE Computer Society Press, Los Alamitos, CA. pp. 25–35. Breitman, K. K., Leite J. C. S. P., & Finkelstein, A. (1999). The world’s stage: A survey on requirements engineering using a real-life case study. Journal of the Brazilian Computer Society, 1(6): 13–37. Christensen, M., and Chang, C. (1996). Blueprint for the ideal requirements engineer. Software, March, p. 12. Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2000). Nonfunctional Requirements in Software Engineering. Kluwer Academic Publishing. Boston, MA. Ebert, C. (2010). Requirements engineering: Management. In P. Laplante (Ed.), Encyclopedia of Software Engineering, pp. 932–948. Taylor & Francis. Boca Raton, FL. Finkelstein, A., & Dowell, J. (1996). A comedy of errors: The London Ambulance Service case study. In Proceedings of the 8th International Workshop Software Specifications and Design, pp. 2–5. Guide to the Systems Engineering Body of Knowledge (SEBoK). (2012). http://www. sebokwiki.org/ (accessed January 2017). Graduate Reference Curriculum for Systems Engineering (GRCSE). (2012). http://www. bkcase.org/grcse/ (accessed January 2017). Graduate Software Engineering Reference Curriculum (GSwE). (2009). https://www.acm. org/binaries/content/assets/education/gsew2009.pf (accessed January 2017). Gorla, N., & Lam, Y.W. (2004). Who should work with whom?: Building effective software project teams. Communications of the ACM, 47(6): 79–82. Kassab, M. (2009). Non-Functional Requirements: Modeling and Assessment. VDM Verlag. Saarbrücken, Germany. Kassab, M., Neill, C., & Laplante, P. (2014). State of practice in requirements engineering: Contemporary data. Innovations in Systems and Software Engineering, 10(4): 235–241. Kilcay-Ergin, N., & Laplante, P. A. (2013). An online graduate requirements engineering course. IEEE Transactions on Education, 56(2): 199–207. Laplante, P. A., Kalinowski, B., & Thornton, M. (2013). A principles and practices exam specification to support software engineering licensure in the United States of America. Software Quality Professional, 15(1): 4–15.

Leffingwell, D., & Widrig, D. (2003). Managing Software Requirements: A Unified Approach. The Addison-Wesley Object Technology Series. Boston, MA. Matoussi, A., & Laleau, R. (2008). A Survey of Non-Functional Requirements in Software Development Process. Departement d’Informatique Universite, Paris, p. 12. Neto, D., Leite, J., & Cysneiros, L. (2000). Non-functional requirements for objectoriented modeling. In Third Workshop on Requirements Engineering, Rio de Janeiro, Brazil, pp. 109–125. Rittel, H., & Webber, M. (1973). Dilemmas in a general theory of planning. Policy Sciences, 4:

155–169. Royce, W. (1975). Practical Strategies for Developing Large Software Systems. AddisonWesley, Boston, MA, p. 59. Sikora, E., Tenbergen, B., & Pohl, K. (2012). Industry needs and research directions in requirements engineering for embedded systems. Requirements Engineering, 17: 57–78. Sommerville, I. (2005). Software Engineering. 7th edn. Addison-Wesley, Boston, MA. Software Engineering Body of Knowledge Version 3.0 (SWEBOK). (2014). http://www.
computer.org/portal/web/swebok/html/contentsch2#ch2 (accessed January 2017). Voas, J., & Laplante, P. (2010). Effectively defining shall not requirements. IT Professional, 12(3): 46–53. Wnuk, K., Gorschek, T., & Zahda, S. (2013). Obsolete software requirements. Information and Software Technology, 55(6): 921–940. Wnuk, K., Regnell, B., & Berenbach, B. (2011). Scaling up requirements engineering—Exploring the challenges of increasing size and complexity in market-driven software development. In D. Berry & X. French (Eds.), Requirements Engineering: Foundation for Software Quality. REFSQ 2011, LNCS, Vol. 6606, pp. 54–59. Zachman, J. A. (1987). A framework for information systems architecture. IBM Systems Journal, 26(3): 276–292. Zave, P. (1997). Classification of research efforts in requirements engineering. ACM Computing Surveys, 29(4): 315–321. Zave, P., and Jackson, M. (1997). Four dark corners of requirements engineering. ACM Transactions on Software Engineering Methodology, 6(1): 1–30.