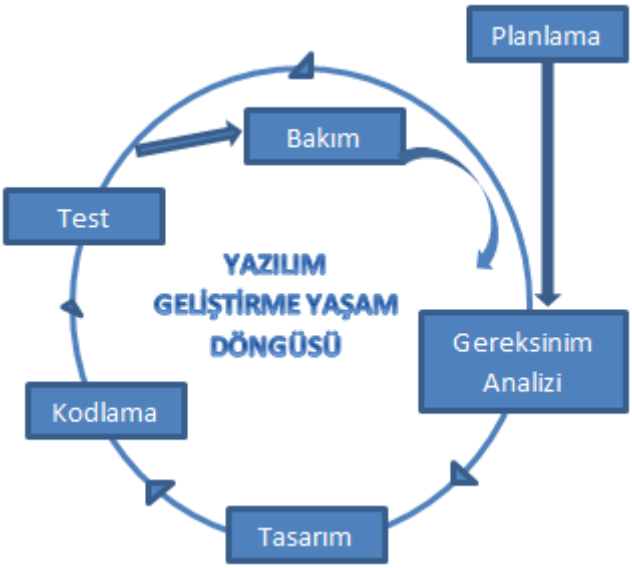
2

Yazılım Mühendisliğinde Kalite

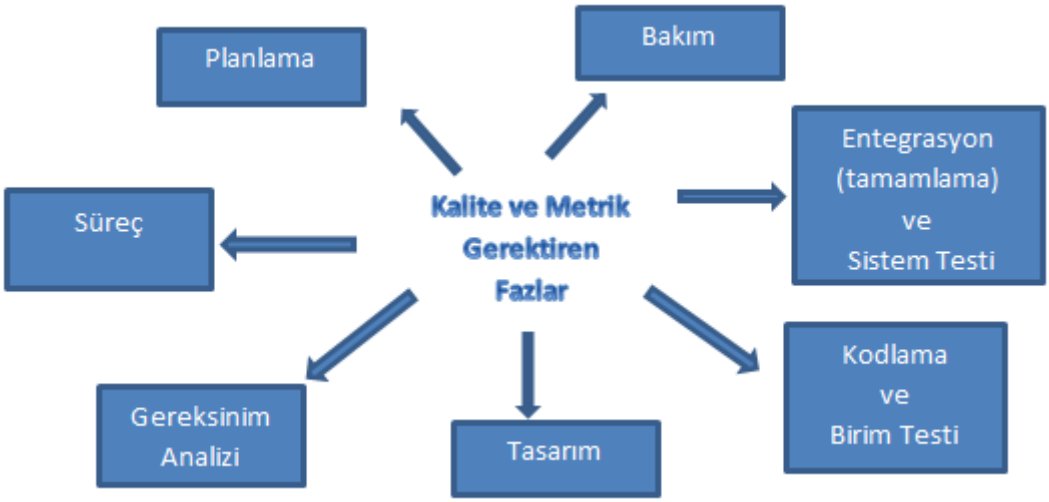
- "Yazılım Kalitesi" ne anlama gelir?
- Uygulamalardaki kusurlar nelerdir?
- Yazılım geliştirmede "doğrulama" ve "geçerleme-onaylama" arasındaki fark nedir?
- Yazılım kalitesini nasıl ölçeriz?



Şekil 2.1 Bu bölümün içeriği ve amaçlarının öğrenilmesi

Yüksek kalitede bir yazılım inşa etmek bu kitabın en önemli temalarından biridir ve başarılı yazılım sistemlerinin üretilmesinde kritik bir faktördür. Ancak, yüksek seviyede bir kaliteye ulaşmak kolay bir iş değildir. Kalite başlangıcından ve ürün geliştirmenin her fazı süresince projelere entegre edilmelidir. Kitapta kalite kavramının tartışılmasının sebebi budur ve kitabın ilk bölümünde tanıtılmıştır.

Yazılım projesinin tüm süreci boyunca, işlemlerin etkinliğini anlamak için veri – ya da ölçümler- toplanır ve yazılım inşa edilir. Toplanan bu verilere "metrik" adı verilir.



Şekil 2.2 Ölçümlerin uygulamasını gerektiren yazılım geliştirme aşamaları

Bu metrikler bir projenin çok çeşitli yönlerini belirtir, yazılım kalite seviyesi, proje zaman sürecine olan bağlılığı ve diğer ölçülebilir durumları, örneğin mühendislerin (verimlilik) üretkenliklerini. Bu bölüm yazılım metrikleri ile ilgili bir genel bakış sunar ve bunların yazılım kalitesi ile olan ilişkilerini açıklar.

Bu kitabın ana bölümlerinden her biri kalitenin nasıl hesaplanacağı ve metriklerin nasıl toplanacağı ve uygulanacağı ile ilgili bölümler içerir. Şekil 2.2 kalite uygulaması ve metrik toplanmasını gerektiren farklı aşamaları resmetmektedir.

Şimdiye kadar "yazılım kalitesi" kavramını serbestçe kullandık. Şimdi somut tanımını verelim.

2.1 YAZILIM KALİTESİNİN ANLAMI

Herkes "kalite" ister, fakat yazılım kalitesi tam olarak ne demektir? Bu konuda değişen görüşler mevcuttur. Bazı kalite ölçüm terimleri güvenilirlik (reliability), taşınabilirlik (portability) vb. gibi kategorilere ayrılmıştır. Ancak eğer bunlardan biri –örneğin taşınabilirlik- yazılım ürününün herhangi bir formu için gerek duyulan bir şey değilse amaca uygun değildir. Bazıları sahip olacağı daha kaliteli bir uygulama ile dünyayı daha iyi hale çevireceğini hisseder. Diğerleri yazılım kalitesini "ne kadar iyi bir geliştirme süreci-işlemi" kavramı ile ifade eder ya da çıkan ürünün ne kadar kapsamlı test edildiği ile ifade eder. Eninde sonunda, bir şirket için kendisi açısından ve müşterileri açısından "kalite" kavramından ne anladıklarını belirlemek en iyisidir. Bilhassa, "yazılım kalitesi" kavramı proje paydaşları arasında tutarlı bir anlamı olması önemlidir. Kitap boyunca kullanılacak bir tanım aşağıdadır.

Biraz alt yapı oluşturmak için, öncelikle "kaliteli" eski moda bir kurşun kalem düşünelim (genellikle sarı). Herkes bir kalemin şu sıralanan gereksinim duyulan özelliklerini karşılaması gerektiğinde hemfikir olacaktır; sağlam bir ağacı olmalıdır, sert bir kurşuna sahip olmalıdır, kalem düz kağıda ne bıraktıysa onu silebilecek bir silgisi olmalıdır, ihtiyaç duyulduğu kadar uzunlukta olmalıdır vs.. Eski stil kaliteli bir kalem olması

için. Ancak geleneksel bazı özellikler umarız; alışılagelenden daha zor kırılması, ortalama kalem ömrünün iki katından uzun gitmesi, çok etkin bir silgiye sahip olması vs.. Özet olarak, kaliteli geleneksel bir şey, onun gereksinimlerinin ötesinde özelliklere sahip olmalıdır. Yazılım Mühendisliğinde, aslında bu kalitenin tipik bir tanımı değildir.

Yazılım uygulamalarındaki fark, yazılım uygulamalarının kaçınılmaz hatalar içermesidir. Her hata, ihtiyaçlardan birer sapmadır, ihtiyaç analizleri dikkatlice yazılmış olsa bile. Bundan dolayı yazılım kalitesi hakkında düşünürken kurşun kalemin kalitesi hakkında düşündüğümüz gibi bir yol izleyemeyiz. Bunun yerine, yazılım kalitesi için iyi bir tanım aşağıdaki gibidir;

> Gereksinimlerine daha fazla yaklaşan yazılım ürününün, kalitesi daha yüksektir.

Şekil 2.3 Kalitenin bu yönünü resmetmektedir.

Şekil 2.3 Yazılım kalitesinin anlamı: hangi gerçek gereksinimlerin karşılandığının bir derecesi

Henüz gereksinimlerin nasıl toplanıp hazırlandığını tartışmadık. Şimdilik, gereksinimlerin, gereksinim analizi sırasında en iyi bilinen bilgilerden oluştuğunu düşünelim. Bu durumda bile, yazılım geliştirilip müşteriye sevk edildikten sonra bazı gereksinimlerin yanlış ya da eksik olduğunun farkına varılabilmektedir. Bu gibi durumlarda, yazılım, gereksinimlerin dokümente edildiği şekliyle inşa edilir ve bu durumda yazılım "açıkça belirtilmiş" gereksinimleri karşılar diyebiliriz. Fakat bu müşteriyi tam olarak tatmin etmeyebilir. Bunun farklı yolları vardır.

Örneğin, eğer gereksinim tanımlaması sürecinde müşterilere danışılmadıysa, sistem onların girişleri olmadan inşa edilecektir. Sonunda yazılımı kullandıklarında istedikleri gibi çalışmıyor bulabilirler ve haliyle tatmin olmazlar. Müşteriler girişler-input için danışılsa bile, yazılımın kullanımında memnun kalmayabilirler. Örneğin, diyelim ki bir yazılım sistemi, gereksinim dökümanında belirtildiği gibi bir ekran düzenine ve menülere sahip bir kullanıcı arayüzüne sahip olsun. Eğer müşteriye yazılım tam olarak kurulmadan arayüzü kullanma imkanı verilmemişse, çalışmak

için kağıt üzerinde gördükleri şeyler elverişsiz gelebilir. Bu ve bunun gibi durumlar için, ürünün yüksek kalite sergilemediğini söyleyebiliriz; çünkü müşteriyi memnun etmemektedir. Bu yüzden biraz daha tamamlanmış bir kalite tanımı şöylece verilebilir:

> Belirtilen gereksinimlerine daha fazla yaklaşan ve bu gereksinimleri müşterinin istek ve ihtiyaçlarını karşılayan yazılım daha kaliteli bir yazılımdır.

Gereksinimlerin toplanması yöntemleri 6. Referanstaki kitabın 4. Bölümde Gereksinim Analizi kısmında verilmiştir.

Müşteri memnuniyetinin yararı dışında, yüksek kaliteli yazılım üretmek başka avantajlar da sağlar. Yapılan çalışmalar göstermiştir ki, en az hataya sahip yazılım ürünü ile en kısa çalışma programına sahip yazılım ürünü arasında bir ilişki vardır. Yine düşük kaliteli yazılım ile zaman taşmalarının yaşandığı yazılımlar arasında ve düşük kaliteli yazılım ile proje ücretlerinin yükseldiği yazılımlar arasında da ilişki vardır. Daha fazla hatanın olduğu yazılımlarda, mühendisler daha fazla zaman harcarlar böylelikle diğer önemli proje işlevleri aksamış olur. Ayrıca, daha fazla hata mevcutsa, bu hataların düzeltilmesinin gerektirdiği değişikliklerden dolayı daha fazla hatalar ile sonuçlanacaktır. Bu ve bunun gibi nedenlerden dolayı "kalite" yazılımda en önemli özelliktir ve yazılımın başarısına katkıda bulunan bir anahtardır.

2.2 YAZILIM HATALARI

Yazılım hatası yazılım uygulamasının bir aşamasıdır ve şöyle tanımlanır: o fazın gerektirdiği çalışma şeklinden sapma durumudur. Pratikte, birçok yazılım sistemi hatalara sahiptir ve bu yüzden kendi gereksinimlerini karşılamazlar. Yazılım kazaları 6. Referanstaki kitabın 1. bölümünde tanımlanmıştır ve hataların maliyetlerinin onlarca milyon dolar olduğu tasvir edilmiştir. Hatalar geliştirme sürecinin herhangi bir aşamasında meydana gelebilir. Yazılım mühendisliğinin ana hedeflerinden birisi, ürün geliştirme süreci boyunca yapay olguları denetleyerek ve sistemi test ederek, mümkün olabildiğince çok sayıda ve olabildiğince erken zamanda hataları bulup temizlemektir. Örneğin, yazılım kodlama aşamasında ise, kod bir önceki tasarım aşamasından sapabilir veya kod yanlış bir çıkış verebilir. Her iki durumda da sonuç birer hatadır. Bu özel örnekte test öncesinde olabildiğince fazla sayıda hatayı belirlemek amacıyla kod denetimi yapılır. Yazılım sisteme tam anlamıyla entegre edildiğinde, belirlenen gereksinimlere uygunluğu için test işlemi yapılır. Hatalar kalite eksikliğinin en açık belirtileridir. Daha sonra geliştirme döngüsünde keşfedilir ve gidermek için daha fazla harcama yapılır. Bu yüzden yazılım geliştirme işlemlerinde iki kalite amacı şöyledir;

1. Yazılım müşteriye teslim edilmeden olabildiğince hatalar giderilmelidir.
2. Bu hatalar geliştirme süresince mümkün olan en erken zamanda giderilmelidir.

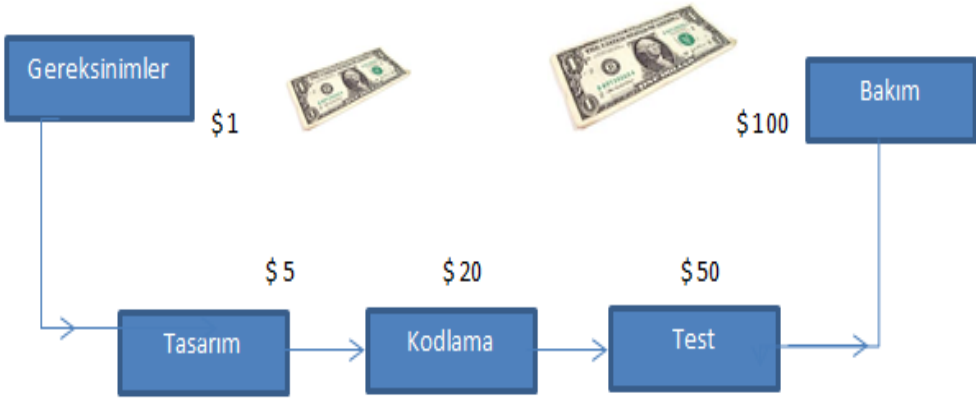
Birinci nokta için, elbette geliştirme sürecinde keşfedilen ve düzeltilen hatalar müşteri tarafından karşılaşılmayacaktır. Ne kadar çok hata temizlenirse daha yüksek yazılım kalitesi muhtemeldir ve daha fazla müşteri memnuniyeti demektir.

İkinci nokta için, yazılım geliştirilirken bulunan hatalar, erken bulunduğunda onarılmaları için maliyet daha azdır.

Boehm ve diğerlerine göre [2], yazılım müşteriye teslim edildikten sonraki hata maliyeti, geliştirme döngüsünün erken zamanlarında ortaya çıkacak aynı hatanın maliyetinden 100 kat daha büyük olabilmektedir. Çeşitli çalışmalar, çeşitli maliyet artış faktörlerini göstermişlerdir fakat hepsinin ortak özelliği, belirleme ve temizleme maliyetlerinin dramatik yükselişidir. Şekil 2.4 her geliştirme aşamasındaki nisbi hata maliyet kestirimini göstermektedir. Örneğin, eğer bir hata gereksinim analizi kısmında ortaya çıkmışsa tespiti ve onarımı aynı aşamada yapılırsa maliyeti 1$ olarak varsayılmaktadır. Aynı hata kodlama aşamasında ortaya çıktıysa maliyeti 20$ olarak varsayılmaktadır. Eğer yazılımı artık müşterinin kullandığı bakım aşamasında ortaya çıkarsa maliyet 100$ olabilmektedir.

Birbirini takip eden aşamalarda hatanın giderilmesindeki maliyetin bu yükselişinin sebebi bir örnekle tasvir edilebilir [1]. Çok sayıda dosya yazma ve okuma işlemi içeren bir uygulamaya sahip olduğunuzu varsayın. Geliştirme işleminizin bir parçası olarak, tasarım aşamasının sonunda bir tasarım testi gerçekleştirdiniz ve iki hata aldınız; birincisi dosya okuma ve yazma işlemlerinde kullandığınız algoritmada, diğeri dosya kayıtlarının tutulduğu hafıza yerleşiminde. Varsayalım ki bu hataların düzeltilmesi için yapılan denemeler iki gün sürecek, yapılacaklar aşağıda verildiği gibidir;

1. Hatayı düzeltmek için algoritmayı değiştirin
2. Tasarımın doğruluğunu kontrol amaçlı yeni bir denetim gerçekleştirin



Şekil 2.4 Bakım aşaması için önceki aşamalara kıyasla hata denetleme ve düzeltme maliyetleri

Varsayalım ki hatalar kodlama aşamasına kadar tespit edilemedi. Hataların tespit edilmesi şimdi daha zordur ve yüzlerce satır kod mevcuttur. Eğer hatalar bir kod denetimi sırasında tespit edilirse, hataların

düzeltilmesi işlemi için bir haftalık bir süreç gerekecektir. Yapılacaklar aşağıdaki gibidir;

1. Hatayı düzeltmek için algoritmayı değiştirin
2. Tasarımın doğruluğunu kontrol amaçlı yeni bir denetim gerçekleştirin
3. Kodun etkilenen kısmını yeniden yazın
4. Kodun doğruluğunu kontrol amaçlı yeni bir denetim gerçekleştirin ve yeni tasarıma uyumluluğunu kontrol edin.

Şimdi varsayalım ki hatalar test aşamasına kadar tespit edilemedi. Hataların tespit edilmesi daha zordur ve çok daha fazla onarım zamanı gerekmektedir. Ek olarak kod denetimi sırasında bulunduysa bir hafta süre gerektirir ve aşağıdaki adımlar izlenmelidir;

5. Hataları yenilemek için test ediciden yardım isteyin
6. Yazılım içindeki kesin nedeni keşfedin
7. Hata ayıklamak için ayrılmış bir zaman belirleyin
8. Düzeltmelerin doğru çalışıp çalışmadığını yeniden test edin.

Bu işlemler, hataların izole edilmesi ve onarılmalarının zorluğuna bağlı olarak bir aya bir hafta zaman süre ekletebilir. Bu örnek hataların kendi enjeksiyonlarına yakın olarak tespit edilememesi ve onarılamamasının maliyetini ve mümkün olan en kısa zamanda tespit edilip düzeltilmesi için niçin efor gösterilmesi gerektiğini açıklıyor.

2.3 DOĞRULAMA VE GEÇERLİLİK SINAMASI

Yazılım doğrulama ve geçerlilik sınaması yazılım yaşam döngüsü boyunca uygulanan bir işlemdir ve aşağıdakileri garantiler;

1. Geliştirme işleminde her adım doğru bir şekilde gerçekleştirildi. (doğrulama-verification)
2. Üretilen her yapay olgu önceki aşamada belirtilen gereksinimlerine uyuyor.(geçerlilik sınaması-validation)

IEEE sözlüğünde [3] bunların tanımları şu şekildedir:

Doğrulama: Bir sistemin ya da bir bileşenin başlangıc aşamasında dayatılan şartları, ürünün verilen geliştirme aşaması emniyetini saptamak için değerlendirilmesidir. Örneğin;

Yazılım tasarımı önce belirlenen gereksinimleri uygulamak açısından yeterli mi?

Kod tamamıyla ve doğru bir şekilde tasarımı kodlayabiliyor mu?

Geçerleme: Bir sistemin ya da bileşenin geliştirme aşamasının sonunda ya da süresince belirlenen şartları yerine getirip getirmediğini saptamak için değerlendirilmesidir. Diğer bir deyişle;

Kodlanan sistem belirlenen kullanıcı gereksinimlerini karşılıyor mu?

Doğrulama, günlük aktivite planlarınızı içeren çizelgenizdeki güncel kalanınızı korumak ile karşılaştırılabilir. Kontrolleri yazıp mevduat yapmak

isterseniz, günlüğünüzde kayıtlı miktarı girip kalanınızı güncellersiniz. Her giriş yapıldığında hesabınızın doğru olduğunu, yeni kalan bakiyenizin doğru olduğunu kontrol edin. **Bu yazılım geliştirme süreçlerinin her aşamasının doğru olarak gerçekleştirildiğini garanti etmekle benzer birşeydir.**

Geçerleme, günlüğünüzü bankanın açıklaması geldiğinde dengelemek ile karşılaştırılabilir. Kayıtlarınızdaki aylık bütün hareketleri dahil edilmeyen veya yanlış girilen var mı diye kontrol etmek için karşılaştırırsınız. Ek olarak, kalan son bakiyenizin ifade edilenle karşılaştırıp geçerlersiniz. Bakiyenizi eksiksiz olarak bugünkü şekliyle korusanız bile, hatalar yapılabilir. Geçerleme bu hataları yakalar ve bakiyenizin doğruluğunu garantiler. **Bu yazılım sisteminin test edilmesiyle benzerdir.**

Doğrulama ve geçerleme sırasındaki iki temel işlem **denetleme** ve **yazılım testidir**. Denetleme ve aynı zamanda gözden geçirme, işlemleri doğrulama aktivitelerdir ve proje olgularının sınanmasını içerir (gereksinimler, tasarım, kodlama vs.) hataları tespit etmek için. Amaç olabilecek tüm hataları, olabilecek en erken zamanda yakalamaktır. Denetlemeler, bölüm 5 ' te daha geniş şekilde yer almaktadır. Test, geçerlemenin başlıca işlemidir.

Yazılım testi, öncelikli olarak bir geçerleme işlemidir, yazılım yaşam döngüsünün sonunda yapılır ve yazılımın kullanıcı gereksinimlerini karşıladığını garanti eder. Test işlemi, sistemi bir giriş verisi kümesi ile sistem davranışının geçerlenmesi ve sistemin çıkışında beklenen değerlerin elde edilen değerlerle karşılaştırılmasıdır. Beklenen değerlerden herhangi bir sapma, hata olarak değerlendirilecektir. Test işlemi Bölüm 7'de detaylıca anlatılmıştır.

Kaliteli bir yazılımın doğrulama-denetlemenin öncelikli şekli- ve geçerleme-yazılım testinin öncelikli şekli- kombinasyonunun bir sonucu olduğuna dikkat edilmelidir. Test tek başına yeterli değildir. Yazılım mühendislerinin hatayı nasıl giderdiği mesele değildir, kalite seviyesi düşükse, geçerlemeye girilir, görünmeyen hataların ölçülmesiyle, büyük olasılıkla bu geçerleme umulandan daha uzun olacaktır ve kalite seviyesi istenenden daha düşük olacaktır. Geçerlemeden önce mümkün olabildiğince hata temizlenmiş olmalıdır.

Doğrulama ve geçerleme kalite uygulamalarıdır, yazılımın temel aşamaları ile ilgili aktivitelerdir, kitabın her bölümünün kalite kısmında yer almaktadır. Şekil 2.5 doğrulma ve geçerleme konseptlerini özetlemektedir.

Doğrulama veya geçerleme arasındaki ayrımları daha da güçlendirmek için bir örnek üzerinde bu iki kavrama bakalım. Farzedin ki, müşteri "ax+b=c" lineer problemi çözmek için bir uygulama istemiş olsun. Bunu müşterinin gereksinimine şöyle dönüştürebiliriz;

1. Kullanıcı giriş rakamlarını a, b, c yi 10 ar rakama kadar girebilecek. Ondalıklı kısım için virgülden sonra 4 rakam olabilecek.
2. Uygulama ax+b=c formülüne 1/1000 içinde bir kesin çözüm üretecek.

Sonra bu gereksinimleri karşılayacak bir program kodlarız.

- **Doğrulama:** Her olgunun kendi spesifikasyonlarına uygun olarak inşa edilmesini sağlamak.
  - "Ürünü doğru inşa ediyor muyuz?"
  - Çoğunlukla denetlemeler ve gözden geçirmeler.
- **Geçerleme:** Tamamlanan her olgunun kendi spesifikasyonlarına uygunluğunun kontrol edilmesidir.
  - "Doğru ürünü mü inşa ediyoruz?"
  - Çoğunlukla test.

Şekil 2.5 Doğrulama ve Geçerleme

Uygulamayı başlangıcından sonuna kadar doğru bir şekilde yapmayı garanti etmek bizim yükümlülüğümüzdedir. Bu doğrulama işlemdir. Kolaylık olması açısından geliştirme aşamalarını aşağıdaki gibi numaralandırabiliriz;

1. Müşteri ihtiyaçlarını-gereksinimlerini al.
2. Gereksinimleri detaylıca yaz.
3. Gereksinimleri ifade şeklimiz için müşterinin onayını al.
4. Uygulamayı kodla (basitlik için, tasarım aşaması atlanmıştır)
5. Uygulamayı test et.

Doğrulama aşağıdaki soruları kontrol edecektir:

1. Gereksinimler gerçekten müşterinin isteklerini ve ihtiyacını ifade ediyor mu? Bazı doğrulama ifadeleri şöyledir: ax+b=c doğru form mu? Hassas gereksinimler nelerdir? a için 0 girilirse ne olur?

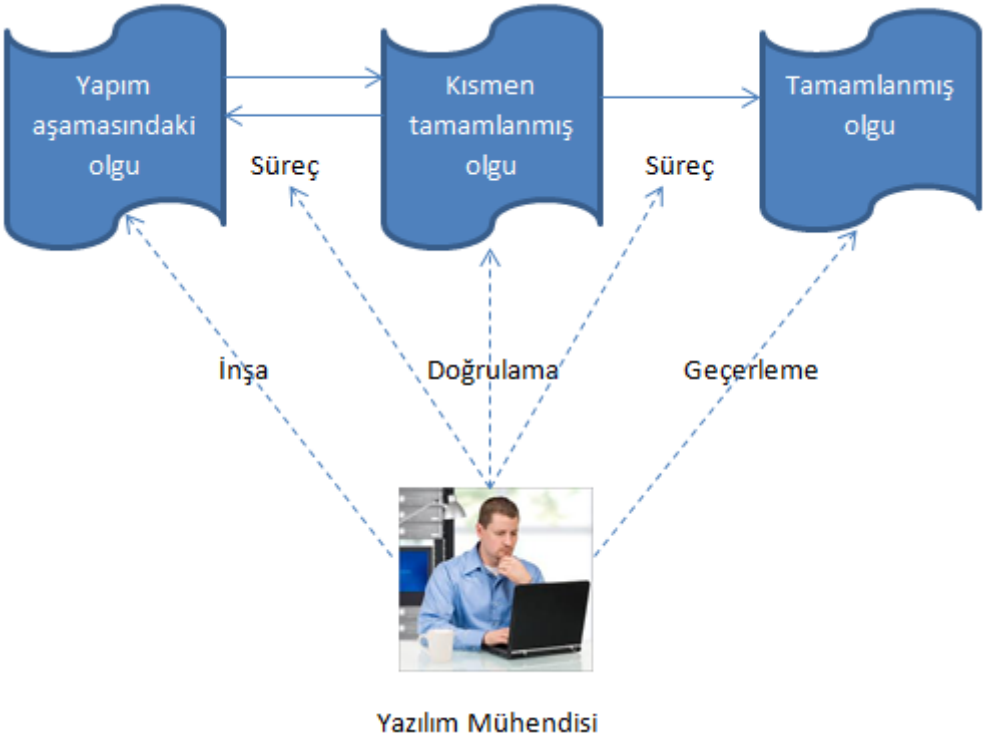
2. Müşterinin düşüncelerini uygun bir şekilde aldık mı? Bazı doğrulama ifadeleri şöyledir: Müşteriye yeterince zaman verdik mi? Müşteriye bütün ihtiyaç ifadelerini açıkladık mı?

3. Kod, yazılı bütün gereksinimleri her açıdan yerine getiriyor mu? Bu kodun kendi kendini denetlemesini gerektirir. Hangi gereksinimler kabul edilir ve karşılaştırma kodu incelenir. Bu matematiksel bir kanıt içerebilir. (açıkçası doğrulama kodu çalıştırmak için çağırmaz)

4. Önerilen testler uygulamayı kapsama açısından yeterli mi? Gerçekçi boyutta bir uygulama için, bu aşama test planları, test işlemleri ve test prosedürleri içerecektir. Bu kendi kendine testten ayrılır, aşağıda bir geçerleme adımı tartışılmıştır.

Müşterinin alınan yazılı görüşlerine göre bu ürünün geçerlemesi tamamlanmış gereksinimler ve tamamlanmış kodda bir küme test işleminin çalıştırılmasıdır. Örneğin, a=1, b=1 ve c=1 girildiğinde, program x=0 değerini üretmesi bir geçerlemedir.

Doğrulama ve geçerleme üzerine olan tartışmamızı özetlemek için, belirli bir anda bir yazılım mühendisi, ya bir olgu üretir, ya bunu doğrular veya tamamlanmış olgusunu geçerler. Şekil 2.6 da bu durum gösterilmiştir.



Şekil 2.6 Bir olgunun Doğrulama ve geçerleme durumu

2.4 KALİTE PLANLAMASI

Kalite özellikleri yazılım yaşam döngüsü boyunca uygulandığından dolayı kalite yaklaşımını ve amaçlarını projenin başlarında yazılı olarak ortaya koymak önemlidir. Diğer bir değişle, hedefler yazılım kalitesinin izlenmesinde kullanılmaktadır. Çevik proje kapsam olarak, kalite düzeyinin vurgulanmasında, müşteri ile sürekli bir iletişim halinde olmak, kodlama aşaması öncesinde ve devamında kümülatif testlerde bulunmanın kaliteyi arttıracağını vurgular. Bu amacı karşılamak için IEEE birçok standart yayınlamıştır. Bunlar ve bunlara benzer standartlar, çevik olmayan büyük projeler veya çevik metotları kısmen kullanan projeler olma eğilimindedirler.

Yazılım kalite güvencesi planı (SQAP) (IEEE Std 730-2002) yazılım yaşam döngüsü boyunca, yazılım kalitesine dair bütün yaklaşımları açıklayan bir dökümandır. SQAP kısım 5 te verilmiştir. Ana başlıkları aşağıda verildiği gibidir; plan şunları içermektedir:

- Kalite takımının organizasyonel yapısı
- Geliştirme sürecinin yönetimini belirleyen bir dökümantasyon: doğrulama, geçerleme, yazılımın kullanımı ve bakımı (yazılım gereksinim özellikleri, doğrulama ve geçerleme planı, yazılım tasarım özellikleri vs…)
- Denetlemelerin tanımlanması ve zaman tabloları ve bunların nasıl oluşturulduğu
- Toplama, izleme ve analiz etme için metriklerin toplanması
- Yazılım test planları için referenslar
- Raporlama nasıl yönetiliyor ve yürütülüyor

Doğrulama ve geçerleme planı, yazılım kalite güvencesi planında olduğu gibidir. IEEE, yazılım doğrulama ve geçerleme işlemlerinin özelliklerini belirtmek için, **Yazılım Doğrulama ve Geçerleme Planı** (SVVP) (IEEE Std 1012-2004) yayınlamıştır. SVVP' nin içerik ve uygulaması kısım 9' da anlatılmıştır.

2.5 METRİKLER

Metrikler yazılımın ya da işlemlerin verilen özniteliklerin hangilerine sahip olduklarını gösteren nümerik ölçülerdir. Metriklere örnek olarak; "her bin satır koddaki hata sayısı", "ortalama yazılım modülü boyutu", verilebilir. Metrikler yazılım yaşam süreci boyunca toplanır ve analiz edilirler ve bize aşağıda verilen konularda yardımcı olurlar;

- Yazılım kalite seviyesinin belirlenmesi
- Proje zamanlama kestirimi(tahmin)
- Zaman ilerlemesinin izlenmesi
- Yazılım boyutu ve karmaşıklığının hesaplanması
- Proje maliyetinin belirlenmesi
- Süreçlerin iyileştirilmesi

Yazılım kalite seviyesi ve proje maliyeti gibi amaçlar yeterince açıklanmadığı zaman, farklı yorumlara konu olabilir. Bu farklılıklar da karışıklığa ve çatışmaya sebep olabilirler. Örneğin, "uygulama çok nadiren çökmeli" şeklinde yapılan bir tanımlama yeterince belirli değildir. Burada "çok nadiren" kelimesini nasıl ölçeceksiniz? Bir günde bir mi?, hafta da bir mi? Ya da yılda bir mi?... Daha kesin bir amaç, "Uygulama yılda 3 defadan fazla çökmemelidir" şeklinde verilebilir. Bu şekilde belirtildikçe, amaç farklı yorumlara sebep olmaz ve daha ölçülebilir olur.

Bir projeye görünürlük sağlamak için ölçülebilir, nesnel ölçütler olmadan projenin ilerlemesini ölçmek zordur; zamanında olup olmadığını, kalite hedeflerini karşılayıp karşılamadığını, müşteriye sevke hazır olup olmadığını… Ek olarak, "ölçemediğini geliştiremezsin". Devamı olarak, bir organizasyonun süreçlerinin veya bir yazılımın geliştirilmesi, metriklerin toplanmasına ve nicel geliştirme amaçlarının belirlenmesine bağlıdır.

Başarılı bir değerlendirme için, metrik toplama projenin başlangıcında başlamalı ve tüm yazılım yaşam döngüsü süresince, bakıma kadar devam etmelidir. Metriklerin toplanması ve analizi, önemli miktarda yönetimsel ve mühendislik çabası gerektirmektedir ancak ödemeler de buna değer durumdadır.

Yazılım metrikleri kabaca iki kategoriye ayrılabilir: **ürün metrikleri** ve **süreç metrikleri**. Ürün metrikleri yazılım ürününün veya sistemin karakteristiklerini ölçer: boyutunu, karmaşıklığını, başarımını ve kalite seviyesini içerir. Örneğin, boyut koddaki satır sayısı ile ifade edilebilir, başarım saniyedeki işlem sayısı ile ve kalite seviyesi hata yoğunluğu olarak bilinen her bin satırdaki hata sayısı ile ifade edilebilir.

Süreç metrikleri yazılım geliştirme ve bakım süresince yazılım süreç, metot, araç özelliklerini ölçer. Süreç metrikleri, proje eforu, zaman çizelgesine bağlılığı, hata onarım oranını ve verimliliği içerir. Örneğin, proje eforu aylık geliştirmedeki kişi sayısı ve hata onarım oranı ise hafta

başına onarılan hata sayısı ile ifade edilebilir. Birçok farklı metrik toplanabilir, bunlar kitabın ilerleyen bölümlerinde detaylandırılmıştır. Bölüm 9 metriklerin toplanmasını daha fazla detaylı vermektedir.

2.5.1 KALİTE METRİKLERİ

Kalite, hangi yazılımın gereksinimlerini karşıladığının bir ölçüsü olduğundan dolayı, uyum derecesini ölçmemiz gerekmektedir. Yazılım kalite metriklerinin bir kümesini oluşturmakta yarar vardır. Bunlar proje için toplanan metrikler olup, tüm metriklerin bir alt kümesi olacaktır. Bu metrikler özel olarak yaşam döngüsü boyunca kullanılan yazılım kalite karakteristiklerine odaklanmışlardır.

Kan [4] tarafından tanımlanan kalite metrikleri aşağıdakileri içerir:

- Hata yoğunluğu
- Çökmesi için geçen ortalama süre
- Müşteri problemleri
- Müşteri memnuniyeti

Hata yoğunluğu, yazılımın boyutuyla ilişkili olarak hataların sayısını ifade eder. Boyut tipik olarak binlerce satır kod (KLOC) şeklinde ifade edilir, dolayısıyla hata yoğunluğu hatasayısı/KLOC olarak ifade edilebilecektir. Hatanın kullanıcı gereksinimlerinden herhangi bir sapma olduğunu hatırlayınız. Bu yüzden hata yoğunluğu kalite ölçümü için kullanılan en önemli metriklerden biridir. Genel olarak yüksek hata yoğunluğu, düşük kalite demektir. Şirketler tipik olarak hataları türüne göre ve şiddetine göre karakterize ederler ki göreceli olarak önemlerine göre ayırt edilebilsin. Aşağıdaki örnek yüksek şiddette bir hatanın nasıl tanımlanabileceğini ve kalite amacını tanımlamaktadır:

Bir sınıf bir hata, yazılım gereksinim şartnamesinin üçüncü bölümüne göre memnuniyeti karşılamak için başarısızdır. Uygulama 5 sınıftan oluşabilmekte ve ilk ayda her 1000 kullanıcı için 1 hata raporundan başka işlem yapamamaktadır.

Dikkat edin ki bu tanım ve amaç çok spesifik ve ölçülebilirdir. Tekraren, amaçlar açık bir şekilde ifade edilmediği zaman, farklı anlamalara yol açabilirler. Örneğin yukarıdaki tanım için alternatif bir tanım verelim: "çok az sayıda sınıf olabilir, hatalar ilk bir kaç ayda rapor edilebilir". Burada "çok az" ı ölçebilecek bir yol yoktur.

Yazılım kalitesinin genel göstergelerinden biri sistem güvenilirliğidir. Güvenilirlik sistemin uygunluğu ya da belirlenen bir peryotta sistem çökmesinin sıkılığıdır. Tipik olarak bu durum sistem çökmesi için ortalama zaman(MTTF) olarak kullanılır ve sistem çökmeleri arasındaki geçen zaman miktarıdır. MTTF emniyetle ilgili sıklıkla kullanılan bir metriktir(havayolu trafik kontrol sistemi, havacılık elektroniği ve silahlar). Örneğin U.S hükumeti hava yolu trafik kontrol sistemini yıllık bazda üç saniyeden daha fazla erişilmez olmamasını görev kabul eder[4]. Bir başka örnek hücresel ağlarda kullanılan iletişim anahtarlarında görülebilir. Büyük network sağlayıcılar, bu cihazların yılda %99.999 oranında erişilebilir olmasını görev sayar. Bu yılda 5 dakika 15 saniye kesinti anlamına gelmektedir. Okuyucu MTTF için neden %100 değil diye sorabilir. Cevap genel olarak bu oranın elde edilebilir olmadığıdır.

Müşteri problemleri, müşterinin ürünü kullanırken aldığı tüm problemlerin sayısıdır. Problemlerden bazıları yazılımdaki geçerli hatalardan kaynaklanabilir. Diğer bazıları hata kaynaklı olmayabilir, kafa karıştırıcı kullanıcı ara yüzü, zayıf hazırlanmış dökümantasyon veya hataları çoğaltmak(yinelenen)(çoktan rapor edilmiş ve çözülmüş ama raporlama biriminin bilmediği hata) gibi sebeplerden dolayı olabilir. Yinelenen hatalar bir hatanın ne kadar yaygın olduğunun en iyi göstergesi olabilir. Bunlar farklı kullanıcılar tarafından karşılaşılan aynı hatalardır. Tüm bu durumlarda, problem bir hatadan kaynaklı olsun veya olmasın, müşteriler kullanışlılık problemi olarak algılanan sorunlarla karşılaşıyorlar ve sonuç olarak yazılımdan memnun olmuyorlar. Bu da müşteri problemlerini en önemli kalite göstergesi yapmaktadır. Bu metrik tipik olarak problem/kullanıcı/aylar ile ifade edilir. Bu ve diğer metrikler yazılım ürünü teslim edildikten sonra bölüm 7 de tartışıldığı gibi test ve bakım aşamasında toplanabilir.

Müşteri memnuniyeti metrikleri yaygın olarak müşteri memnuniyeti anketleri sayesinde toplanırlar. Cisco System gibi şirketler [5] müşterilerinden memnuniyet ölçüsü olarak 5 üzerinden bir geri besleme alırlar (5=çok memnun, 1=çok memnuniyetsiz). IBM CUPRIMDSO kategorilerini kullanır (Yetenek/fonksiyonellik, kolay kullanım, başarım, güvenilirlik, bakım kolaylığı, yüklenebilirlik, dökümantasyon, servis, ayrıntılı tasarım). Hewlett-Packard FURPS(Fonksiyonellik, Kolay kullanım, güvenilirlik, başarım ve servis) metriğini kullanır [4]. Bu anketlerin sonucu ilgili şirketlerin kendi önceliklerini geliştirmeleri için yönlendirici olarak kullanılır.

2.6 ÖZET

Kaliteli yazılım bu kitabın ana temasıdır. Kalite, "müşterisinin istediği ve ihtiyaç duyduğu gereksinimleri karşılamaya daha yakın olan" şeklinde tanımlanabilir. Kaliteli yazılım üretiminin birçok avantajı vardır; yükselen müşteri memnuniyeti, azaltılmış geliştirme çizelgeleri, azalmış proje maliyeti gibi. Kalite çalışmaları yazılım yaşam döngüsünün başında uygulamaya konur ve nihayet yazılım teslimi ile bakım aşamasında sonlanır. Yazılımda hatalar gereksinimlerden sapmalardır. Ürün geliştirme aşamasında olabildiğince çok hata temizlenmelidir. Böylece bu hatalar müşteriye taşınmamış olacaktır. Erken süreçte hataların bulunması önemlidir, çünkü hem bu hataların temizlenmesi kolaydır, hem de onarım maliyeti düşüktür. Çalışmalar ürün tesliminden sonra farkedilen hataların maliyetleri 100 kat daha artırdığını göstermektedir.

Doğrulama ve geçerleme (V&V), yazılım yaşam süreci boyunca birkaç hata içermesi mümkün olabilen olguların doğrulanması ve yazılımın gereksinimleri ile uyduğunun geçerlenmesi demektir. Doğrulama, "Ürünü doğru inşa ediyormuyuz?" sorusuna cevaptır, geçerleme ise " doğru ürünü inşa ediyormuyuz?" sorusuna cevaptır.

Metrikler yazılım yaşam döngüsü sürecinde toplanan nümerik ölçümlerdir ve yazılımların hangi belirlenen ölçütlere uygunluğunu nicemlemek için bir derece sunar. Metrikler, kalite seviyesinin belirlenmesinde, zaman çizelgelerinin tahmininde, çizelge ilerleme durumlarının izlenmesinde, yazılım boyut ve karmaşıklığının

hesaplanmasında, proje maliyetlerinin hesaplanmasında ve süreç geliştirme gibi işlemlerde yardımcı olurlar. Metrikler, hata yoğunluğu, ortalama çökme zamanı, müşteri problemleri ve müşteri memnuniyeti ile ilişkilidir.

2.7 ALIŞTIRMALAR

1. Bu bölümdekilere ek olarak, kaliteli yazılım geliştirmenin avantajlarına en az iki örnek veriniz

2. Anlatıldığına göre tespit edilmeden yazılım yaşam döngüsünde ilerleyen hatalar sonra tespit edildiğinde keşfedilmeleri ve onarılmaları zordur. Bunun neden doğru olduğunu kendi kelimelerinizle iki örnek vererek açıklayınız.

3.(a) Doğrulama ve geçerleme arasındaki farkı kendi kelimeleriniz ile açıklayınız.

(b) Kendi spesifik projenizin yürütülmesi için plan yapmadan önce doğrulama ve geçerleme planı yapmanın avantaj ve dezavantajları nelerdir.

4. (a) Metrikler nelerdir?

(b) Metriklerin neden önemli olduğuna dair anladığınız bir sebep veriniz.

5. (a) Kendi kelimelerinizle ürün ve süreç metrikleri arasındaki farkları açıklayınız.

(b) Aşağıdaki hedeflerden her biri için, size amaca ulaşmanız için yardımcı olacak bir metrik belirtin. Bunların Ürün metriği yada süreç metriği olduğunu belirtin ve nasıl uygulanacağını açıklayın.

- Proje takvimi gecikmelerini önlemek.
- Bir projeden diğerine kalite ölçümlerini sürekli yapmak.
- Kalite sorunları yaşanan yazılım bölümlerini tanımlamak
- Zamanlama doğruluğunu geliştirmek için bir temel kurmak

2.8 BİBLİYOGRAFYA

1. Whitten, Neal, "Managing Software Development Projects", John Wiley & Sons, p. 195, 1995

2. Boehm, Barry, "Software Engineering Economics", Prentice-Hall, 1981

3. "IEEE Standard Glossary of Software Engineering Terminology", IEEE Std 610.12-1990, December 1990, pp. 80-81

4. Kan, Stephen N., 2003, "Metrics and Models in Software Quality Engineering", Second Edition, Addison-Wesley, Pearson Education Inc., pp.86-100.

5. Cisco Systems, Customer Satisfaction Metrics. http://www.cisco.com/web/about/ac50/ac208/about_cisco_approach_to_quality_customer_sat_survey.html [accessed Nov 4, 2009]

6. Software Engineering: A Modern Approaches 2/e Eric J. Braude, Michael E. Bernstein, June 2010, ©2011