- Sürüm tanımlama işlemi, tüm bileşenlerin sürüm tanımlayıcılarıyla birlikte bir envanterinin çıkarılması, problemlerin son durumları, sürümler arası farklılıklar, notlar ve yeniden üretim için gerekli yöntem ve komutların belirtilmesini içermelidir.

13.3. Geliştirmeye Yönelik Öneriler

Kitapta birçok yazılım geliştirme yönteminden, temel kurallardan bahsettik. Şimdi de pratik uygulamalarda kullanılabilecek geliştirmeye yönelik önerilere değinelim.

13.3.1. Tasarım

Yazılım geliştirme sürecinin başında yer alan çözümleme ve tasarım, gerçekleştirimin başarılı olabilmesi için zorunlu aşamalardır. İsterlerin tanımlanmasından sonra, kodlamaya geçmeden önce mutlaka çok sağlıklı bir tasarım aşamasından geçilmelidir. Oldukça yaygın ve eski bir deyişle, "iki kez tasarlayıp bir kez kodlamak" en başarılı sonucu verir. Tasarımın önemini vurgulamak üzere önerilerimiz şunlar olacaktır:

- Tasarım sürecinin tüm etkinlikleri Yazılım Geliştirme Planı'nda belirtilen yöntemlere göre yapılmalıdır.
- Ortaya çıkan ürünün özelliklerinin doğrulanması için testler yapılması tasarım standartlarının bir parçası olarak değerlendirilmelidir.
- İzlenebilirlik, tasarım, gerçekleştirim ve test sırasında devam ettirilmelidir.
- Her türlü tasarım düzenleşim yönetim sistemine girmeden önce yapısal bir incelemeden geçirilmelidir. Bu inceleme, işlevsellik, başarım, tekrar kullanım, güvenlik, emniyet, güvenilirlik, sağlamlık, beraber çalışabilirlik gibi yazılım nitelik ölçütlerini içermelidir.
- Artımlı geliştirme modeli kullanıldığı takdirde tasarım da artımlı olarak yapılabilir. Ancak düzenleşim yönetim sisteminin bu yapıya uygun destek vermesi gereklidir.
- Artımlı geliştirme planlamasında sistem ve yazılım mimarilerinin ilk sürümde tamamlanmasına dikkat edilmeli, sonra işlevsel artırıma devam edilmelidir.
- Var olan yazılımın tekrar kullanımı planlandığı takdirde sistem ve yazılım mimarisi tasarımı uygun şekilde yapılmalıdır.
- Sistem ve yazılım mimarileri üzerinde Yazılım Geliştirme Planı'nda belirtildiği şekilde doğrulama işlemi uygulanmalıdır. Bu işlem sonunda, mimarinin genel başarımı, tekrar kullanımı desteklemesi, güvenlik, emniyet, güvenilirlik, sağlamlık, beraber çalışabilirlik gibi yazılım nitelik ölçütlerine sahip olduğu denetlenmiş olur.

Tasarımın nasıl yapılacağına ilişkin en uygun kararlar uygulamalara ve deneyimlere göre alınır. Bunun yanında tasarımcıya kendi deneyimini geliştirmeye yardımcı olabilecek bazı önerileri de sıralamak istiyoruz. Aşağıdaki öneriler uygulamaya dönük tasarımın gerçekleştirilmesinde bir kontrol listesi olarak kullanılabilir:

- Yapmak istediğiniz şeyin ne olduğunu önceden tam olarak belirleyin. Amacı belirlemek için tasarımın ilerlemesini beklemeyin.
- Amaçlarınızın açık ve somut olmasına dikkat edin.
- Sosyolojik problemlere teknolojik açıdan yaklaşmayın.
- Var olan sistemlerden esinlenerek başlangıç noktası oluşturacak bir model kullanın.
- Tasarımı esneklik, geliştirme, taşıma ve tekrar kullanım özelliklerini karşılayabilecek şekilde yapın.
- Tekrar kullanılabilir yazılım parçalarını (modül, sınıf, kütüphane gibi) belgeleriyle birlikte kullanıma sunulacak şekilde hazırlayın.
- Ayrıntılı düşünme çabalarınızı tüm yazılımın değil de onu oluşturan yazılım birimlerinin tasarımı üzerine yoğunlaştırın.
- Bir arayüzün başka arayüzlere olan bağlılığını en aza indirin.
- Arayüzleri, gereksinimi karşılamaya yetecek en az miktarda bilgi sağlayacak şekilde tasarlayın.
- Tasarımı ve sonra da gerçekleştirimi tekrar tekrar gözden geçirip gerekli gördüğünüz düzenlemeleri yapın.
- Tasarımı ve gerçekleştirimi test edip sonuçlarını değerlendirebilecek yazılım geliştirme yardımcı araçları kullanın.
- Her şeyi olabildiğince basit halde ve küçük boyutta tutun, fakat bunda da aşırıya kaçmayın. Yapmak istediğiniz şeyin gerçekten kullanılabilirliğini düşünün.
- Etkinliği göz ardı etmeyin. Özellikle gerçek zaman uygulamaları ve gömülü sistemler için etkinliği artırıcı her türlü tekniği deneyin.
- Tasarımcıların, kodlayıcıların ve kullanıcıların da birer insan olduklarını ve her zaman hata yapabileceklerini unutmayın.
- Kullanıcıların her zaman bir mühendis kadar dikkatli ve bilgili olamayacağını düşünerek insan-makine arayüzlerinin uygunluğuna ve sistemin sağlamlığına gerekli önemi verin.
- Çevrenizde tasarımın, kütüphanelerin ve soyut veri tiplerinin tekrar kullanımı fikrini yayın ve bu fikri somut örneklerle destekleyin.

13.3.2. Tekrar Kullanım (13.3.2.)

Yazılımın tekrar kullanımı zaman ve maliyeti düşürür. Ancak, daha önce geliştirilmiş yazılımı tekrar kullanabilmek pahasına başka şeylerden ödün verilmesi çeşitli riskler doğurur ve genel maliyeti artırabilir. Bu konuda dikkat edilmesi gerekenleri şöyle sıralayabiliriz:

- Tekrar kullanım esasları Yazılım Geliştirme Planı'nda yer almalıdır.
- Tekrar kullanılacak ürünler, önce proje isterlerine göre kendi başlarına test edilmeli, daha sonra da tümleştirildikleri öğelerle birlikte bir test planı kapsamında testlerden geçirilmelidir.

- Tekrar kullanıma esas olacak öğeler, hazır ticari ürünler ve özel üretilmiş birimler gibi geliştirme sürecinde üretilmeyecek yazılımlar birer risk unsuru olarak ele alınmalıdır.
- Geliştirme süreci dışında temin edilecek öğelerin tekrar kullanımı ancak bu tür öğelerin özel bir denetim sonucu kabul edilmesinden sonra yapılmalıdır. Bu denetim, kullanım yöntemleri, uyumluluk, kullanıcıların bilgi düzeyi, belgelendirme ve bakım kolaylıkları gibi özelliklerin denetimini içermelidir.
- Bu tür öğelerin kullanımına karar verebilmek için tüm yaşam çevrimi boyunca gerekecek bakım maliyeti, üst sürümlere yükseltme, garanti, lisans ve diğer öğelerle olan uyumluluk hususları dikkate alınmalıdır.
- Bu tür öğelerin kullanımı için verilecek karar mimari ve tasarım tanımlamaları ile müşteri isteklerine uygun olmalı, isterleri tam olarak karşıladığından emin olunmalıdır.
- Sistemin yaşam çevrimi boyunca hazır olarak kullanılacak kritik öğelerin sorunsuzca kullanılabilmesi için gerekli çalıştırma lisansları, bakım ve teknik destek olanakları ile mümkünse kaynak kod ve geliştirme lisanslarının temin edilmesi, bunların şimdiki ve gelecekteki maliyetleri konusunda kestirimlerde bulunularak geliştirici ve müşteri arasında bir anlaşmaya varılmalıdır.

13.3.3. İsterlerin ve Tasarımın Denetlenmesi

Ortaya çıkan ürünün kararlılığı ve bütünlüğünün sağlanabilmesi için isterlerin ve tasarımın denetlenmesi gereklidir. Denetim, proje örgütü içinde, yöneticinin yetki verdiği kişiler ya da bir nitelik güvence grubu tarafından yapılmalıdır. Denetlemenin ne şekilde yapılması gerektiğini şöyle özetleyebiliriz:

- Geliştirilen yazılım ürünleri düzenleşim yönetim sistemi altına alınmadan önce resmi bir test ve denetimden geçirilmelidir. Bundan sonra bu ürünler başka ürünlerin geliştirilmelerinde temel alınırlar.
- Denetim, yazılım geliştirme planında belirtildiği şekilde, o ürün için geçerli olabilecek kabul esaslarına dayalı bir süreci şeklinde yapılmalıdır.
- Denetim sırasında belirli bir takım metrikler toplanmalı ve izlenmelidir. Bu metrikler arasında, yazılım kusurları, kusurların bulunduğu yerler, bu kusurları gidermek için gerekli gayretler ve denetim için harcanan özkaynakların miktarı bulunabilir.
- Denetimler, kavram tanımlamasıyla başlar, mühendislik etkinliklerinin tamamlanmasıyla son bulur.
- Program veya proje için ayrılan mali kaynaklar denetim ve sorun giderme için gerekli harcamaları da içermelidir.
- Denetimlerin başarılı sonuçlanması ürünün veya tanımlanmış işin resmi bitişini göstermelidir. Buna göre hakedişlerin ödenmesi, ürünün düzenleşim denetiminde bir sürümünün yayınlanması, bir aşama noktasının geçilmesi gibi değerlendirmeler yapılabilir.

- Denetim ve gözden geçirme süreci ürünün tanımlanması ve ilk isterlerinin belirlenmesi ile başlar, mimari, tasarım, kodlama, tümleştirme, test ve belgelendirme aşamalarında devam eder.

13.3.4. Gerçekleştirim

Gerçekleştirim yani kodlama sırasında dikkat edilmesinde yarar gördüğümüz öneriler şunlardır:

- Yordamlarınızı (fonksiyonlar) 20 satırdan fazla uzatmamaya çalışın. Toplu bir işi bir yordama yaptırmak, bu yordamı bir başka yordamın içinde çağırmak belirli düzeylerde soyutlama sağlayacağı için kodun anlaşılabilirliğini, takibini, hata ayıklamasını ve genişletilebilir olma özelliğini arttırır.
- Bir kod dosyasının çok fazla uzamamasına gayret gösterin. Binlerce satır uzunluğundaki tek bir dosyayı geliştirme sırasında kullanmak oldukça güçtür. Çok uzun dosyaların tekrarlı derleme süresini de arttırdığını unutmayın.
- Büyük dosyalar yerine modüler bir dosya yapısı oluşturun. Ancak, dosya sayısını aşırı miktarda attırmayın.
- Kod yazarken küçük ve mantıklı bölümler halinde yazıp sık sık derleme yapın. Herbir derleme arasında ne kadar fazla kod yazarsanız hata çıkma olasılığı da o ölçüde artar.
- Kodunuzda değişiklik yapacağınız zaman her seferinde bir tek şey değiştirin. Eğer iki şey değiştirirseniz ve yazılım çalışmazsa, o zaman değiştirdiğiniz yerlerden birini iptal ederek önce hatanın yerini bulmak ve sorunu ondan sonra çözmek zorunda kalabilirsiniz.
- Kullanılan bilgisayar mimarisinin özelliklerini dikkate alın. Bazı programlama dilleri donanıma doğrudan kumanda edebilmeye olanak tanır. Örneğin, gerçek zamanlılığın sağlanmasının çok önemli olduğu hallerde ve hedef sistemin mikroişlemcisi çok sayıda saklayıcıya sahipse kısa ömürlü bazı değişkenlerin bir C++ anahtar sözcüğü olan `register` ile bildirilmesi yürütme hızını artırır.
- Kod içinde sıkça kullanacağınız sayı değerlerini (dizi boyu, önemli bir katsayı, matematiksel sabitler, çevirme çarpanları, yapı tanımlayan sabitler gibi) sabit değerler olarak yazılımın tümü tarafından erişilebilecek bir yerde belirleyin.
- İlke olarak, kod içinde hiçbir şekilde, hiçbir sayı düzenine göre rakam veya sabit sözcük katarı (string) kullanmayınız.
- Yordamlara parametre geçirirken mümkün olan yerlerde nesnelerin adreslerini kullanın. Nesnenin tümünü sistem yığınına kopyalamak yerine yalnızca o nesneyi gösteren bir işaretçi geçirmek genel çalışma hızını artırır. Dikkat edilmesi gereken nokta alıcının bu nesne üzerinde değişiklik yapıp yapmamasıdır. Bu tür parametreler bazı dillerin özelliklerine göre güvence sağlayacak şekilde kullanılabilirler (örneğin, C++'ta `const` sözcüğü, ADA'da `in` veya `out` parametreler).
- Kodlamada kullanılan programlama dili desteklese dahi `goto` deyimini kullanmaktan özenle kaçının.

- Kod yazarken zor gelse dahi yaptığınız işi, gerekiyorsa her satır için, mutlaka açıklama satırları kullanarak anlaşılır hale getirin. Yordamların ne gibi işler yaptıklarını, dilin anlamsal özelliği içinde bulunmayan kodla ilgili açıklamaları kodun o kısmına eklemeyi unutmayın; bu size ileride mutlaka yardımcı olacaktır. Tabii ki kaynak kodu bir tasarım belgesi haline getirmeyin.
- Kodun okunabilirliğini artırmak için metinin şekilsel düzenine dikkat edin. Satırbaşı ve içeriden başlama kurallarını belirli bir sisteme göre uygulayın ve tüm yazılım kapsamında buna bağlı kalın. Taşınabilirlik amacıyla satırbaşı aralıkları için sekme (tab) kullanmaktan kaçınınız, onun yerine boşluk karakteri kullanınız.
- Kodlamada ilerlerken yazdıklarınızın bir hata sonucu tamamen kaybolmaması için sık sık değişik ortamlarda yedekleme yapın.
- Çok sayıda kişinin çalıştığı ve paylaşılır dosyaların kullanıldığı geliştirme ortamlarında, erişimi iyi denetlenen ortak bir çalışma alanı oluşturunuz. Kişilerin kendi yerel alanlarında geliştirme yapmaları yerine bu ortak alanları düzenleşim yönetim sistemi desteğiyle kullanmalarını sağlayınız. Bu şekilde, gereksiz kopyaların oluşmasını engelleyebilirsiniz.
- Dizin isimlerini mantıklı ve disiplinli bir şekilde oluşturunuz.
- Yazılımı denerken hata oluştuğunda (bilgisayarın kilitlenmesi gibi) hemen kullandığınız bilgisayar sistemini suçlamayın. Sürekli tekrarlanan hatalarda işletim sistemi veya donanım arızası aranması yerine çalıştırılmak istenen yazılımda belirli bir mantık sırasında göre hata aranması çözüm için harcanacak toplam zamanı azaltır.
- Hiçbir zaman hatalara teslim olmayın. Bazen küçük bir yazılımda karşılaşılan hataları bulmak çok uzun zaman alabilir. Böyle durumlarda sabırla ve hatanın kökenine inmeye çalışarak sorunun giderilmesine gayret gösterin. Bu tür hataların deneyiminizi arttırdığını da göreceksiniz.
- Hiçbir yazılım geliştirici hata ayıklama işleminden kaçamaz. Fakat iyi bir geliştirici dikkatli, korunmalı ve olası hataların yerini bildirecek şekilde kod yazmayı bir alışkanlık haline getirerek hataları en aza indirgemeyi sağlayabilir.
- Kod yazmak bir sanattır. O nedenle tasarımın iyi olması yanında, ancak kodlayıcının bilgi ve deneyimi ile nitelikli bir yazılım elde edilebilir.
- Aynı amaca ulaşan çok sayıda yol vardır. Ancak, en nitelikli şekilde bu amaca ulaşmak için mutlaka daha önce denenmiş ve kabul edilmiş yöntemler kullanılmalıdır. Tasarım kalıpları ve kütüphane kullanımı bunun için örnektir.

13.3.5. Sürekli Test

Sistem oluşturmak üzere geliştirilen her yazılım birimi tek başına test edilmeli, sonra da tümleştirmesi ve genel sistem testleri yapılmalıdır. Yönetim altında yürütülen bu testler projenin başarısı açısından oldukça önemlidir. Bu konudaki önerilerimiz şunlardır:

- Kritik bileşenler saydam kutu yöntemiyle test edilmelidir.

- Her türlü test işlemi önceden üzerinde karar kılınmış ilkelere göre yapılmalı ve belirli bir plan izlenmelidir. Bu amaç için gerekli olan harcamalar önceden bütçelendirilmelidir.
- Düzenleşim yönetim sistemi altına alınıp sabitlenecek her ürün ya da her öğe mutlaka bir test sürecinden geçirilmelidir.
- Testler yalnızca ortalama sistem durumlarını değil, aynı zamanda anormal ve arıza durumlarını, aşırı yüklemeleri de içermelidir.
- Test için kullanılacak her türlü araç ve gereç, test durumları, senaryolar, kabul koşulları, beklenen değerler denetim altına alınmalı ve düzenleşim yönetim sistemi altında tutulmalıdır.
- Her test için giriş ve çıkış değerleri, izlenecek yollar ve kabul esasları tanımlanmalı, sonunda "geçti" veya "kaldı" gibi bir değerlendirme yapılmalıdır.
- Son kabul testleri, bilgisayarlı sistemin son kullanım alanında, olası en gerçekçi senaryolarla yapılmalıdır.
- Test planı belirli amaçlar içermelidir. Üretilen ve hazır olarak kullanılan öğelerle oluşturulan sistemin doğru çalıştığının gösterilmesi en büyük amaç olmalı, bu gerçekleştikten sonra ürün kullanıma sürülmelidir.
- Test işlemlerinin olabildiğince kendiliğinden yapılması, çıkan hataların ivedi olarak düzeltilip tekrar test yapılabilmesini sağlar. Bu test çevrimi ne kadar hızlı olursa testler için kullanılması gereken toplam süre de o kadar düşer.
- Bağımsız bir nitelik güvence grubu, test yordamlarını, yürütme şeklini ve sonuç raporlarını gözlemleyerek denetlemelidir.

13.3.6. Sık Derleme ve Test

Tasarımın koda dönüşümü yapılırken en büyük yardımcı derleyicidir. Elle yazılan kodun sözdizim hatalarından arındırılması derleyici tarafından sağlanır. Aslında derleyici, önce kodu ayrıştırarak dilin sözdizim kurallarının tam olarak uygulanıp uygulanmadığını kontrol eder, ondan sonra da nesne kodunu üretir.

En küçük derleme birimi bir dosya olduğundan kodlama aşamasına iyi bir dosya yapısı kurarak başlanmalıdır. Her kaynak kod dosyasının bir an önce derleyiciden geçirilmesi hedef olmalıdır. Çünkü, derleyici, insanlardan çok daha dikkatlidir. Sık derleme ve ortaya çıkacak ilk yürütülebilir kodu test etmek, olası hataların büyümeden yakalanması için zorunludur. Şimdi bu konudaki önerilerimizi sıralayalım:

- Ana sistem testleri tamamlandıktan sonra hedef sistem testleri olabildiğince kısa sürede yapılmalıdır. Bu şekilde, sık derleme ve üretme kullanılarak olası hataların daha çabuk bulunması sağlanır.
- Test sistemi üzerinde en küçük yazılım yapısı oluşur oluşmaz testlere başlanmalıdır. Bu maksatla kullanılacak yazılımlar sık sık derlenip üretilmelidir.
- Test için kullanılacak tüm yazılımlar düzenleşim yönetim sisteminden alınarak kurulmalıdır. Yeni bir sürüm ortaya çıkmışsa başka bileşenlerin testlerinde kullanılmadan önce bu yeni sürüm testlerden geçirilmelidir.

- Eklenen bir özellik veya tümleştirilen bir bileşen için gerekli testler ancak ana sistemin sabitlenmesi ve testlerinin başarılı olmasından yapılmalıdır.
- Testler sırasında bulunan tüm kusurlar tanımlanmalı ve belgelendirilmelidir. Aksi durumda, bir sonraki testlerde yine aynı kusurların tekrarlanmasından kaçınılamaz.
- Testlerin sonuçları tüm proje personeline açık olmalı, hatta sıkça sorulan sorular belirli bir yerde toplanmalı ve bilgiler başkaları ile paylaşılmalıdır.
- Asıl kullanılacak sisteme en yakın bir sistem düzeneği, altsistemler veya onların benzeticileri (simülatör) geliştirme yapılan yerde kurularak testlerin olabildiğince gerçekçi ortamda yapılması sağlanmalıdır.

14.3.7. Hata Ayıklama

Kavramsal olarak yöntemler bilinse dahi pratik uygulamasının nasıl yapıldığı asıl önemli olan noktadır. Bu nedenle şimdi de hata ayıklama sürecinde kullanılabilecek bazı pratik önerilerde bulunacağız:

- Test aşaması için kapsamlı test senaryoları hazırlanmalı ve her senaryo mutlaka sonuna kadar çalıştırılmalıdır. Bu şekilde, olası yazılım kusurlarının varlığının belirlenmesine çalışılmalı, kusur belirlendikten sonra temizleme aşamasına geçilmelidir.
- Testler kötümser bir yaklaşımla yapılmalıdır ki kusurlar ortaya çıksın. Yakalanan her kusur ileride olası bir yazılım hatasına engel olacaktır.
- Hata ayıklama sırasında yeri belirlenen yazılım kusurunun giderilmesi sırasında yeni bir kusur katılması yüksek bir olasılıktır. Bu nedenle, kusurlar birer birer ele alınmalı, çok fazla değişiklik yaparak birden fazla kusuru bir defada gidermeye çalışılmamalıdır.
- Çoklu programlama ortamlarında hata ayıklama son derece güçtür. Bu tür ortamlar tasarlanırken, modüllerin ayrı ayrı test edilebilmelerine olanak tanınması çok önemlidir. Bir hata durumunda, önce hataya neden olduğu düşünülen modülün girdi ve çıktıları tek tek sınanmalıdır. Sonra da yine bu modülün girdileri üzerinde hatanın oluştuğu durumu tekrarlayabilmek üzere basit senaryolar yaratılarak testler yapılmalı ve hatanın tam yeri aranmalıdır.
- Birden fazla modülde hata bulunması durumunda kusurlar bir sıraya göre düzeltilmeli, birden fazla modül bir anda değiştirilmemelidir.
- Kodlama sırasında, herbir yordam içine hata yakalama tuzakları yerleştirilmeli, yakalanan hatalar nedenleriyle birlikte standart bir hata raporlama düzeneğine bildirmelidir. Bu şekilde, yürütme sırasında oluşan bir hataya neyin neden olduğunu ve ne zaman oluştuğunu bulmak kolaylaşır.
- Dağıtık ortamlarda, yazılımda meydana gelen bir hatayı bulmak için hangi düğümde hangi modüllerin çalıştığının takip edilmesi gereklidir. Bu tür ortamlarda, küçük ölçekli de olsa mutlaka bir ara katman kullanılmalı, kodlama, hata yakalama ve raporlamada belirli düzenekler bulunmalıdır.

- Testler sürerken hata düzeltme işlemi sürdürülmemelidir. Her hata durumunda da testi durdurup hatayı düzeltmek, sonra teste kaldığı yerden devam etmek de her zaman doğru değildir. Eğer testlerin gidişini etkileyecek derecede büyük hata varsa, test durdurulmalı, ileriki bir tarihte tekrar edilmelidir.

13.3.8. Ölçme Süreci Uygulaması

Yazılım geliştirme etkinliklerinden ölçme sürecinin uygulanması sırasında dikkat edilecek bazı önerilerimiz aşağıdadır:

- Örgüt içindeki herkesin ölçme sürecinin yeteneklerini ve kısıtlarını anlaması sağlanmalıdır.
- Metriklerin bir anlam ifade edebilmesi için herkesin ölçülen verilerin ne anlama geldiği hakkında aynı düşünceye sahip olması gereklidir.
- Önce küçük hedeflerle işe başlanmalı, yalnızca birkaç temel ölçüm kullanılmalı ve etkileri kontrol edilmelidir.
- Yalnızca tanımlanan ve kullanılacak olan metrikler toplanmalı, veriler kullanılmayacaksa boşuna toplanmamalıdır.
- Ölçme için bir kişi görevlendirilmeli, metrikleri doğru şekilde toplaması ve birleştirmesi için geliştiricilerle yakın temasta olması sağlanmalıdır.
- Metrik toplamak ve saklamak için basit de olsa bir araç kullanılmalıdır.
- Ölçümlere dayanarak belirli bir grubu ya da kişiyi değerlendirme yoluna gidilmemeli, kimsenin de böyle düşünmesine izin verilmemelidir.
- Ölçüm sonuçları iyi ya da kötü olsun, mutlaka raporlanmalıdır.