YAZILIM GEÇERLEME VE SINAMA

White-Box Testi

White-Box

White-box testing, aynı zamanda "clear box testing", "glass box testing", "transparent box testing" ve "code-based testing" olarak da bilinir. Bu test yöntemi, yazılımın iç yapılarına, yani kod bloklarına, iç işleyişine ve program akışına odaklanır. Testçi, yazılımın nasıl çalıştığını ve belirli girdilerle nasıl tepkiler verdiğini anlamak için yazılımın kodunu inceleyerek testler gerçekleştirir.

White-Box

Yazılımın dahili mantığını, akışını ve yapısını test etmek için kullanılır. Test eden kişi, belirtilen gereksinimleri karşıladıklarından emin olmak için kod yollarını ve mantık akışlarını incelemek üzere test senaryoları oluşturur.

White-Box

Beyaz kutu test yöntemleri kullanılarak, şu test senaryoları gerçekleştirilir:

(1) Bir modül içindeki tüm bağımsız yolların en az bir kez sınanması

(2) Tüm mantıksal kararların doğru ve yanlış taraflarında sınanması

(3) Tüm döngülerin sınır değerlerinde sınanması

(4) Geçerliliklerini sağlamak için iç veri yapılarının sınanması

Ana Kavramlar ve Yaklaşımlar

1. **Kodun İç Yapısına Odaklanma:** Testçi, yazılımın kaynak koduna erişim sağlar ve kodun işleyişini anlar.
2. **Kod Kapsamı (Code Coverage):** Testlerin, yazılım kodunun hangi kısımlarını kapsadığını belirlemek önemlidir. Amacı, mümkün olduğunca kodun daha fazla kısmını test etmektir.

Ana Kavramlar ve Yaklaşımlar

3. **Kontrol Akışı Analizi:** Yazılımın kontrol yapılarını (if-else blokları, döngüler vb.) test ederek, farklı yollar ve durumlar üzerinden test senaryoları oluşturulur.
4. **Hata Ayıklama (Debugging):** Kod içindeki potansiyel hatalar, mantık hataları ve işlevsel sorunlar belirlenir ve düzeltilir.

White-Box Test Türleri

Birim Testi

- Uygulamanın her bir parçasının veya işlevinin doğru çalışıp çalışmadığını kontrol eder.
- Uygulamanın geliştirme sırasında tasarım gereksinimlerini karşıladığından emin olur.

Entegrasyon Testi

- Uygulamanın farklı parçalarının birlikte nasıl çalıştığını inceler.

White-Box Test Türleri

- Bileşenlerin hem tek başlarına hem de birlikte iyi çalıştığından emin olmak için birim testinden sonra yapılır.

Regresyon Testi

- Değişikliklerin veya güncellemelerin mevcut işlevselliği bozmadığını doğrular.
- Uygulamanın güncellemelerden sonra hala tüm mevcut testleri geçtiğinden emin olur.

White-Box Test Teknikleri

İfade Kapsama (Statement Coverage)

Dal Kapsama (Branch Coverage)

Koşul Kapsama (Condition Coverage)

Yol Kapsama (Path Coverage)

White-Box Test Teknikleri

İfade Kapsama (Statement Coverage)

Bir uygulama içindeki her kod satırının en az bir test senaryosu tarafından test edilmesini sağlar. Bir akış şeması durumunda, her düğüm en az bir kez taranmalıdır. Kodun bazı bölümlerinin kullanılmadığını veya erişilemez olduğunu belirlemeye yardımcı olabilir; bu, programlama hataları, güncellemeler vb. nedeniyle olabilir. Bu ölü kodu belirlemek, geliştiricilerin yanlış koşullu ifadeleri düzeltmesini veya uygulama performansını ve güvenliğini iyileştirmek için gereksiz kodu kaldırmasını sağlar.

White-Box Test Teknikleri

Dal Kapsama (Branch Coverage)

Koşullu ifadeler, farklı girdiler farklı yürütme yollarını izleyebildiğinden bir uygulamanın yürütme kodunda dallar oluşturur. Bu test, bir uygulama içindeki her dalın birim testi tarafından kapsanmasını sağlar. Bu, az kullanılan kod yollarının bile düzgün bir şekilde doğrulanmasını sağlar.

Koşul Kapsama (Condition Coverage)

Koşul ifadesinde tanımlanan her ifade hem doğru hem de yanlış sonuçlara göre değerlendirilir. Sonuç olarak, bir karar ifadesindeki her iki dalın da test edilmesini sağlar. Bir karar ifadesinin OR ve AND olmak üzere çeşitli koşulları içermesi durumunda, koşul kapsama testi, koşulların tüm çeşitli kombinasyonlarının test durumlarına dahil edildiğini doğrular.

White-Box Test Teknikleri

Yol Kapsama (Path Coverage)

Bir yürütme yolu, bir uygulamanın başladığı andan sonlandığı ana kadar yürütülebilecek komutların sırasını tanımlar. Bu test, bir uygulama boyunca her yürütme yolunun kullanım durumları tarafından kapsandığından emin olur. Bu, tüm yürütme yollarının işlevsel, verimli ve gerekli olduğundan emin olmaya yardımcı olabilir.

White-Box Test Teknikleri

Temel Yollar Testi(Basic Paths Testing)

Yazılımın çalışması esnasında geçebileceği bağımsız yolları ortaya çıkarır. Amaç, bir programdaki tüm olası yürütme yollarının en az bir kez test edilmesini sağlamaktır. Bu bağımsız yolların saptanması için önce programın çizgesel bir şekilde ifade edilmesi gerekmektedir. Bu çizgesel ifade akış çizgesi kullanılarak yapılır. Akış çizgesinde her program parçasının bir başlangıç ve bir bitiş noktası vardır.

Temel Yollar Testi

Başlangıç ve bitiş noktaları düğümler ile ifade edilir. Bunların arasında her koşul veya döngü deyimi için ayrıca bir düğüm kullanılır. Düğümler arasında ise düğümleri birbirlerine bağlayan kenarlar yer alır. Koşullara karşı gelen düğümler dallanma oluştururlar; bu ayrışmalar, ilgili koşul akışlarının sonunda yeniden birleşimi sağlayacak ek düğümlerle sonlandırılmalıdır.

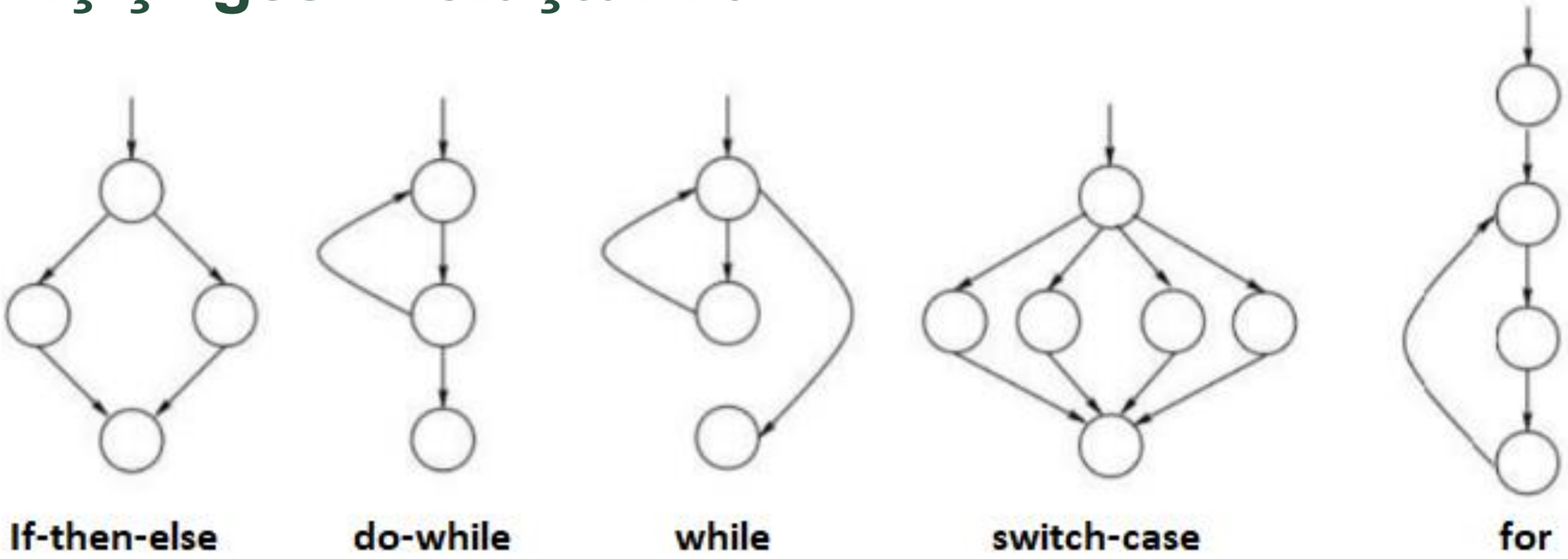
Temel Yollar Testi

Test adımları:

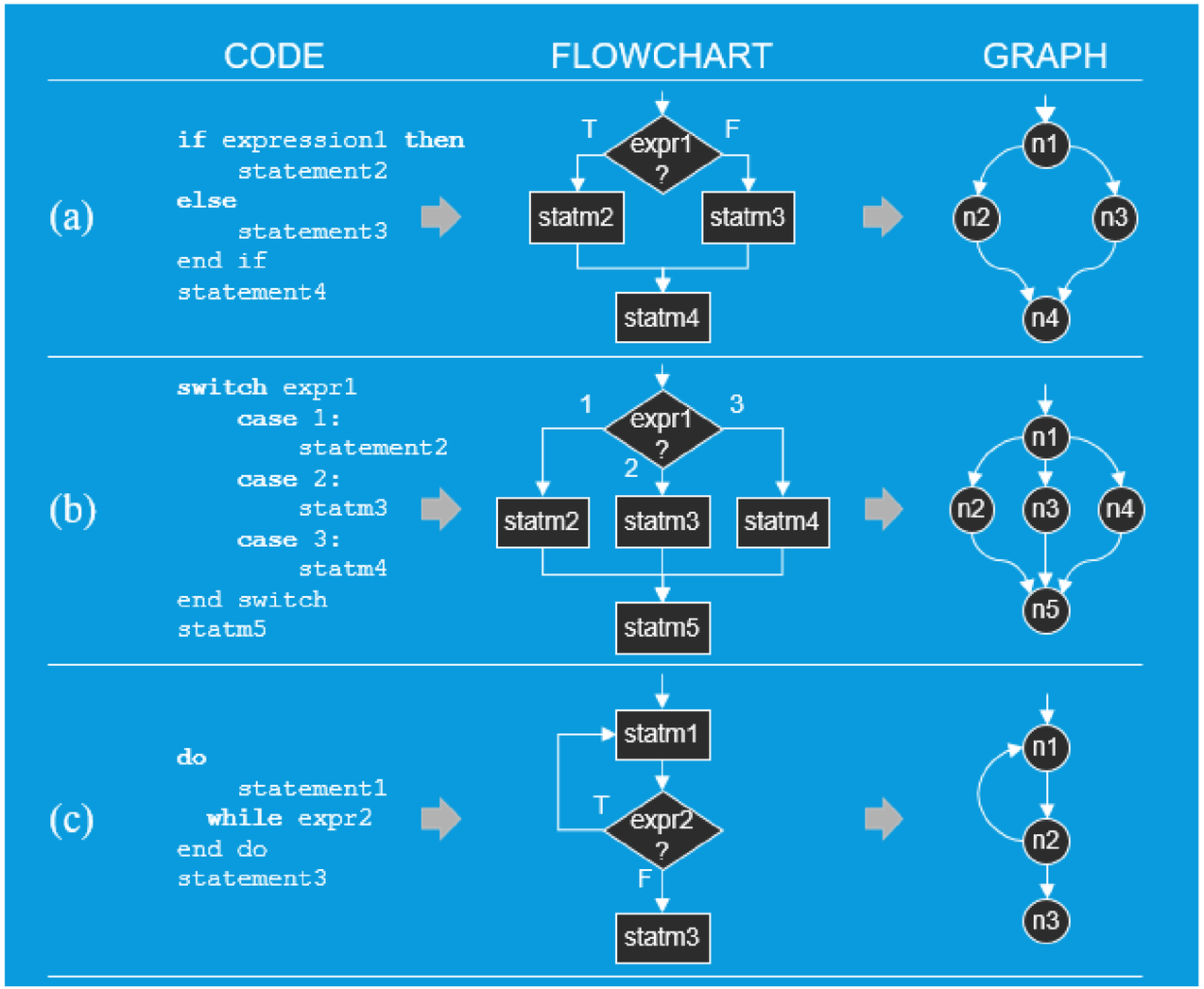
1. Akış çizgesini oluşturma
2. Çevrimsel karmaşıklığı hesaplama
3. Bağımsız program yollarını bulma
4. Test senaryolarını türetme

Temel Yollar Testi

1. Akış çizgesini oluşturma

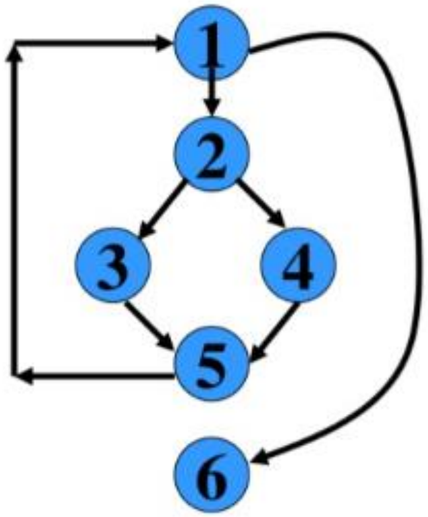


Temel Yollar Testi

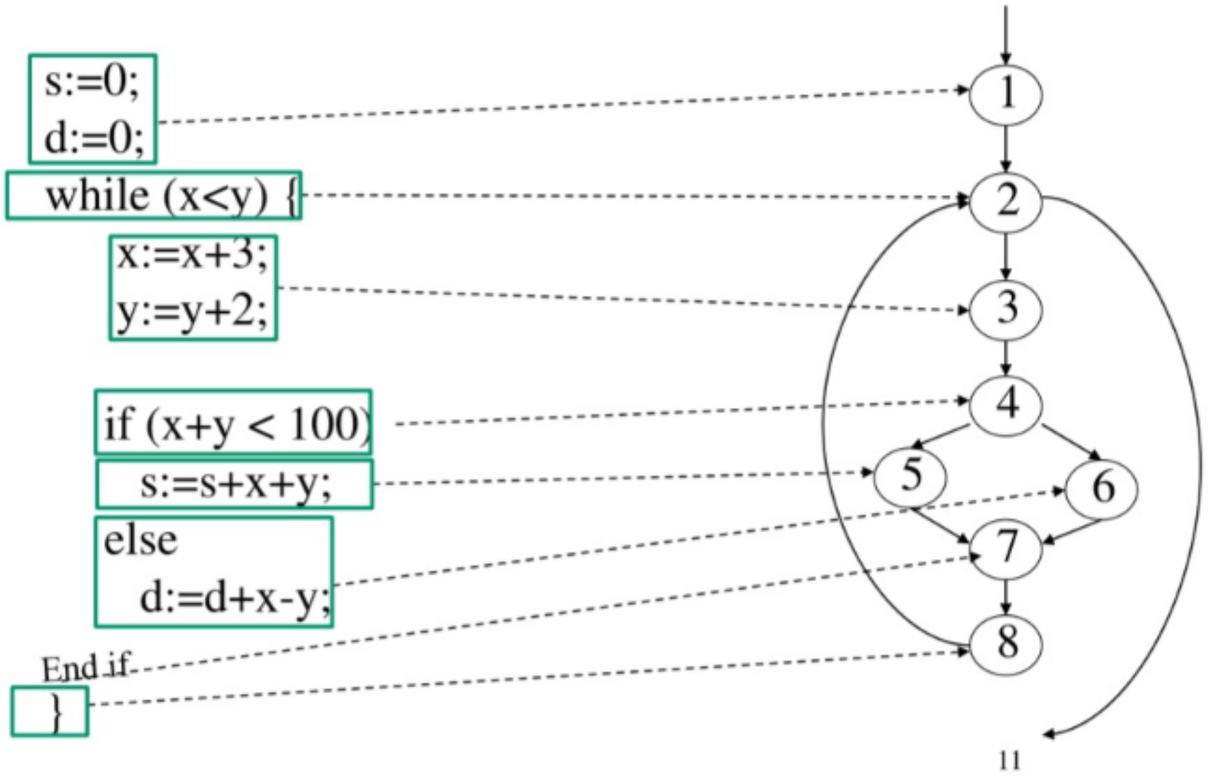


Temel Yollar Testi

```
int f1(int x,int y){
1 while (x != y){
2    if (x>y) then
3          x=x-y;
4    else y=y-x;
5 }
6 return x;        }
```



Temel Yollar Testi



2. Çevrimsel karmaşıklığı hesaplama

Çevrimsel Karmaşıklık(Cyclomatic complexity): Bir programın veya fonksiyonun karmaşıklığını ölçen bir metrik olarak tanımlanır.

- Bir programın mantıksal karmaşıklığının nicel bir ölçüsünü sağlar.
- Temel kümedeki bağımsız yolların sayısını tanımlar.
- Tüm ifadelerin en az bir kez yürütüldüğünden emin olmak için gerçekleştirilmesi gereken test sayısı için bir üst sınır sağlar.

Temel Yollar Testi

$$V(G) = E - N + 2$$

E : Akış çizgesindeki kenar sayısı

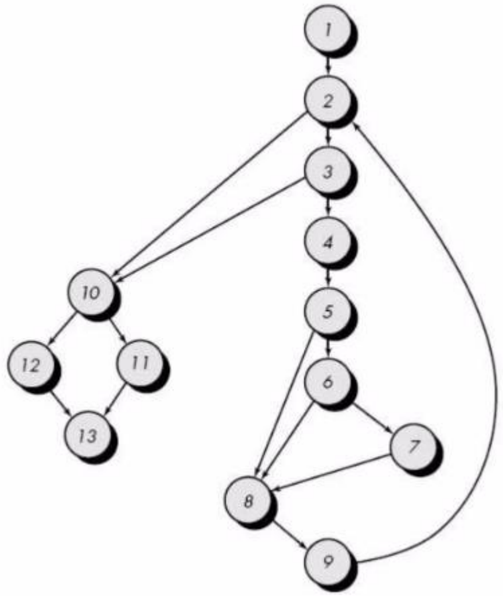
N : Akış çizgesindeki düğüm sayısı

Çevrimsel Karmaşıklık	**Anlamı**
1 - 20	Risk taşımayan, basit bir program
11 - 20	Daha karmaşık, riskli bir program
21 – 50	Karmaşık ve çok riskli bir program
> 50	Bakımı yapılamayacak kadar risk taşıyan bir program

3. Bağımsız program yollarını bulma

Doğrusal olarak bağımsız yolların bir temel kümesi belirlenir. V(G) değeri, program kontrol yapısı boyunca doğrusal olarak bağımsız yolların sayısını sağlar.

Yol 1 : 1-2-10-11-13
Yol 2 : 1-2-10-12-13
Yol 3 : 1-2-3-10-…
Yol 4 : 1-2-3-4-5-8-9-2-…
Yol 5 : 1-2-3-4-5-6-8-9-2-…
Yol 6 : 1-2-3-4-5-6-7-8-9-2-…



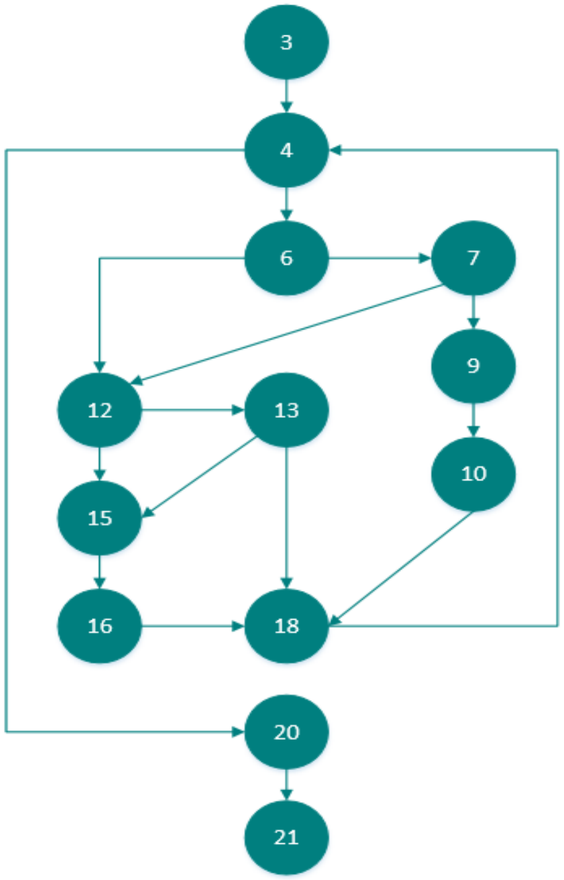
V(G) = 17-13+2=6

4. Test senaryolarını türetme

Temel kümedeki her bir yolun yürütülmesini zorlayacak test senaryoları hazırlanır. Veriler, her bir yol test edilirken tahmin düğümlerindeki koşulların uygun şekilde ayarlanması için seçilmelidir. Her bir test senaryosu yürütülür ve beklenen sonuçlarla karşılaştırılır. Tüm test senaryoları tamamlandıktan sonra, test eden kişi programdaki tüm ifadelerin en az bir kez yürütüldüğünden emin olabilir.

Temel Yollar Testi

```
1   int functionZ(int y)
2   {
3   int x = 0;
4   while(x <= (y * y))
5       {
6       if ( (x % 11 == 0) &&
7            (x % y == 0) )
8           {
9           printf("%d", x);
10          x ++;
11          } // end if
12      else if ( (x % 7 == 0) ||
13                (x % y == 1) )
14          {
15          printf("%d", y);
16          x = x + 2;
17      } // end else
18      printf("\n");
19      } // end while
20  printf("End of list\n");
21  return 0;
22  } // end functionZ
```



V(G) = 17 – 13 + 2 = 6

3,4,20,21
3,4,6,12,15,16,18,4,20,21
3,4,6,12,13,15,16,18,...
3,4,6,7,12,15,16,18,...
3,4,6,7,12,13,18,...
3,4,6,7,9,10,18,...

Bir Yazılım Projesinde Hem Black-Box Hem de White-Box Testlerinin Uygulanmasının Avantajları

Hata Tespitinde Etkinlik: Black-box testi, kullanıcı perspektifinden hataları ve eksiklikleri tespit ederken, white-box testi kod seviyesindeki hataları ve mantık hatalarını ortaya çıkarır. Bu çift yönlü yaklaşım, yazılımın daha güvenilir ve hatasız olmasını sağlar.

Bir Yazılım Projesinde Hem Black-Box Hem de White-Box Testlerinin Uygulanmasının Avantajları

Kullanıcı Beklentileri ve Teknik Gerekliliklerin Dengelemesi: Black-box testi, kullanıcı beklentilerine odaklanırken, white-box testi teknik gereklilikleri ve performansı değerlendirir. Bu denge, yazılımın hem son kullanıcılar hem de teknik standartlar açısından tatmin edici olmasını sağlar.

Bir Yazılım Projesinde Hem Black-Box Hem de White-Box Testlerinin Uygulanmasının Avantajları

Risk Yönetimi: Her iki test türü de farklı risk alanlarını ele alır. Black-box testi, kullanıcı deneyimi ve işlevsellikle ilgili riskleri, white-box testi ise kod bütünlüğü ve güvenlikle ilgili riskleri azaltır.

Bir Yazılım Projesinde Hem Black-Box Hem de White-Box Testlerinin Uygulanmasının Avantajları

Geliştirme Sürecinin İyileştirilmesi: White-box testi, geliştirme aşamasında hataların erken tespit edilmesine yardımcı olurken, black-box testi, ürünün piyasaya sürülmeden önce son kullanıcı beklentilerini karşıladığını doğrular. Bu, geliştirme sürecinin daha verimli ve etkili olmasını sağlar.

Bir Yazılım Projesinde Hem Black-Box Hem de White-Box Testlerinin Uygulanmasının Avantajları

Esneklik ve Uyarlanabilirlik: Farklı test yöntemlerinin kullanılması, projenin değişen gereksinimlerine ve hedeflerine uyum sağlamayı kolaylaştırır. Bu esneklik, projenin başarısını artırır.

Bir Yazılım Projesinde Hem Black-Box Hem de White-Box Testlerinin Uygulanmasının Avantajları

Kalite Güvencesi: Her iki test türünün birleşimi, yazılımın kalitesini artırır ve kullanıcıların güvenini kazanır. Bu, ürünün piyasada daha iyi bir konum elde etmesine yardımcı olur.

Bir Yazılım Projesinde Hem Black-Box Hem de White-Box Testlerinin Uygulanmasının Avantajları

Sonuç olarak, bir yazılım projesinde hem black-box hem de white-box testlerinin uygulanması, yazılımın hem kullanıcı beklentilerini karşılamasını hem de teknik açıdan sağlam olmasını sağlar. Bu, yazılımın genel kalitesini ve başarısını artırır.

Sonuç

Her iki test yöntemi de yazılım geliştirme sürecinin kritik parçalarıdır ve birbirlerini tamamlayıcı niteliktedir. **Black-box testi**, kullanıcı deneyimi ve yazılımın işlevsel doğruluğu üzerine odaklanırken, **White-box testi** yazılımın iç yapısını ve kod bütünlüğünü sağlamak için kullanılır.

Etkili bir test stratejisi, her iki yöntemin dengeli bir şekilde kullanılmasını gerektirir. Bu, yazılımın hem kullanıcı beklentilerini karşılamasını hem de yüksek kalite ve güvenilirlik standartlarını sağlamasını garanti eder.

Sorular

1. Black-Box Testing Nedir?
2. White-Box Testingin Ana Farkı Nedir?
3. Black-Box Testingin Avantajları ve Dezavantajları Nelerdir?
4. White-Box Testingin Kod Kapsamı (Code Coverage) İle İlişkisi Nedir?
5. Bir White-Box Test Senaryosu Nasıl Oluşturulur?

Sorular

6. Kullanılabilirlik Testinin Önemi Nedir?
7. Entegrasyon Testinin Amacı Nedir?
8. Fonksiyonel Testin Özellikleri Nelerdir?
9. Arayüz Testi Hangi Durumlarda Önem Kazanır?
10. Test Otomasyonunun Yazılım Geliştirme Sürecine Katkıları Nelerdir?
11. Bir Yazılım Projesinde Hem Black-Box Hem de White-Box Testlerinin Uygulanmasının Avantajları Nelerdir?