12.1.4. Özel Sistemlerin Testleri

10. Hft
2. Grup
Başlıyor

Bilgisayar tabanlı sistemlerin uygulama alanları değiştikçe onların test şekilleri de değişiklik gösterir. Sistemlerin hem donanımları hem de yazılımları uygulama alanının isterlerini karşılayacak şekilde test edildikten sonra kullanıma sunulmalıdır. Şimdi bu tür sistemleri ve test yöntemlerini gözden geçirelim.

- **Gömülü sistemler**
  Bu tür sistemlerin yazılımları donanıma kumanda ettiği için testlerin donanımla birlikte yapılması gereklidir. Özellikle, dış dünyadan koşutzamansız olarak

girdi alan ve bu girdilere tepki vermek zorunda olan yazılımlar, bu girdilerin ne zaman ve hangi çalışma kipi sırasında geleceği belirli olmadığından gerekli önlemleri almalı, hata yaratıp çökmemelidir. Örnek olarak bir lazer yazıcının çalışmasını denetleyen yazılımı ele alalım. Yazıcı açıldıktan sonra kendi testlerini yaparak Çevrimiçi durumuna geçer. Bu durumdayken bilgisayardan gelen komutlara ve verilere uyarak baskı türünü belirler, verileri alır, kağıt durumunu kontrol eder ve baskıya geçer. Basım işi bitince de yine Çevrimiçi durumuna döner. Denetim yazılımı bu işleri yaparken dışarıdan gelebilecek girdileri de gerektiği gibi değerlendirmek zorundadır. Veri alışı sırasında kağıt tepsisi çıkarıldığında ya da bakım kapağı açıldığında, ilgili algılayıcıdan gelen işaretle veri alışı kesilerek uyarı verilmeli, Çevrimdışı durumuna geçmelidir. Kağıt tepsisi yerine konduğunda tekrar Çevrimiçi duruma gelmeli ve baskı süreci devam etmelidir. Bu durumların herbiri, olabilecek her koşul için test edilmelidir. Çevrimiçi durumdayken kağıt tepsisinin çıkarılması testi başarılı geçebilir. ancak, veri alışı sırasında çıkarılması da bir hataya yol açmamalıdır. Buna benzer sistemlerin girdi-çıktı sayısı ve işlevleri arttıkça bu tür olayların olma olasılıklarının da tek tek değerlendirilmesi ve test edilmesi gereklidir. Bu işlevlere bir de gerçek zamanlılık eklendiğinde testin önemi daha ön plana çıkar.

- **Gerçek zamanlı sistemler**

  Daha önce de açıkladığımız gibi, bir sistemin gerçek zamanlı sayılabilmesi için hızlı yanıt vermesi değil çok küçük zaman sınırları içinde işini tamamlama zorunluluğu bulunmasıdır. Bu tür sistemlerin de ne zaman gerçek zamanlı tepki gerektiren bir olayla karşılaşacağı her zaman belirli olmaz. Bu nedenle, gerçek zamanlı sistemlerin yazılımı ya çok özel benzetim ortamlarında sınanır ya da gerçek sistem üzerinde, laboratuvarda, olabildiğince gerçek koşullar altında donanımla birlikte test edilir. Ancak bazı durumların test edilmesi hiç mümkün olmayabilir; onun yerine, en yakın durum için işlevsellik testi yapılarak varsayım yoluna gidilir. Gerçek zamanlı sistemlerin en yaygın örneklerinden biri olan bilgisayarlı kontrol sistemleri içinde uçuş kontrol sistemleri, güç santralı denetimi, hava trafik kontrolü, silah atış kontrol sistemleri, uzay araçları yer almaktadır. Bir savaş uçağının otomatik uçuş kontrol sistemini dikkate alacak olursak, sistemin anlık yükseklik ve hız bilgilerini almasının, ivme ölçerlerinden gelen bilgileri kullanmasının, yatay ve dikey dümenlere uygun komutlar göndermesinin ve sürekli içsel işlevsellik testi yapmasının gerekli olduğunu görürüz. Bu sistemin yazılımı önce benzetim ortamlarında test edilir, sonra uçağın üzerindeki sistemlerle birlikte uçak yerdeyken test edilir. Bir sonraki aşama rüzgar tüneli ya da gerçek uçuş koşullarıdır. Uygulanabilen tüm senaryolarda testler geçilmiş olmasına rağmen, belirli bir sürat ve manevra sırasında bir yazılım parçasının hatalı davranması olasıdır. O yüzden, bu tür sistemler kullanıma verilmeden önce çok uzun süreli testlerden geçirilirler. Bazı testler ise yalnızca varsayımda kalmak zorundadır (uçak bir isabet aldığında hasar derecesinin uçuş kontrol sistemine ne kadar zarar vereceği ve sistemin durumu koruyup koruyamadığı gerçekten yaşanmadan test edilemeyecektir).

- **Büyük veritabanı sistemleri**
  Bankacılık, personel ve envanter bilgileri gibi çeşitli kayıtlar artık bilgisayarlı sistemlerde tutulmakta ve her türlü erişim işlemi onun üzerinden yapılmaktadır. Bu sistemler için geliştirilecek yazılımlar tam olarak kullanıma sunmadan önce çok ayrıntılı ve hassas testlerden geçirilmeli, ondan sonra gerçek bilgilerle kullanıma sunulmalıdır. Bir bankacılık sisteminin veritabanının yenilendiğini düşünelim. Bazı aykırı durumların iyi kotarılmaması müşteri hesaplarının karışmasına yol açabilir. Yanlış kişilere yönlendirilen para veya fon aktarımları birçok kişinin ve de bankanın maddi hasar görmesine yol açabilir. Benzer şekilde bir envanter kontrol sistemine yeni yüklenen hatalı bir yazılım birimi üzerinde işlem yapılan ürün ya da nesnenin farklı sayıda gözükmesine, dolayısıyla da hesaplarda açığa yol açabilir. Böyle sistemlerin hatalarını bulup düzeltmek çok güç olabilir; hatta bazen geri dönüşü mümkün olmayabilir.
- **Güvenlik sistemleri**
  En dikkatli şekilde test edilmesi gereken sistemlerden biri olan güvenlik sistemleri, çeşitli algılayıcılar ve alarmlardan oluşur. Bu tür sistemler hiç kesintisiz çalışmalı, sürekli devrede kalmalı, hem her türlü olağan dışı olayı tespit edebilmeli hem de yanlış alarm üretmemelidirler. Her türlü bakım ve onarım işleri sistem devrede iken yapılabilmelidir. Bu sistemlerin testleri işlevsellik yanında her türlü olasılığın dikkate alınmasıyla yapılmalıdır.
- **Geniş paket yazılımlar**
  Büyük çapta kullanıcısı olan paket yazılımlar için yapılacak testler, bir kusurun çok sayıda kullanıcıyı etkileyebileceği, istenmeyen sonuçlara yol açabileceği göz önünde bulundurularak yoğun testler yapıldıktan sonra ürün sürümü gerçekleştirilmelidir. Daha ileride göreceğimiz alfa ve beta testleri bu tür yazılımların test edilebilmesi için yaygın olarak kullanılan bir yöntemdir.

12.1.5. Otomatik Test Araçları

Yazılım testleri çok uzun sürdüğü ve yüksek maliyet gerektirdiği için bu işin bir kısmını kolayca ve otomatik olarak yapabilecek araçlar kullanmak mümkündür. Bu tür test araçlarını şöyle sıralayabiliriz:

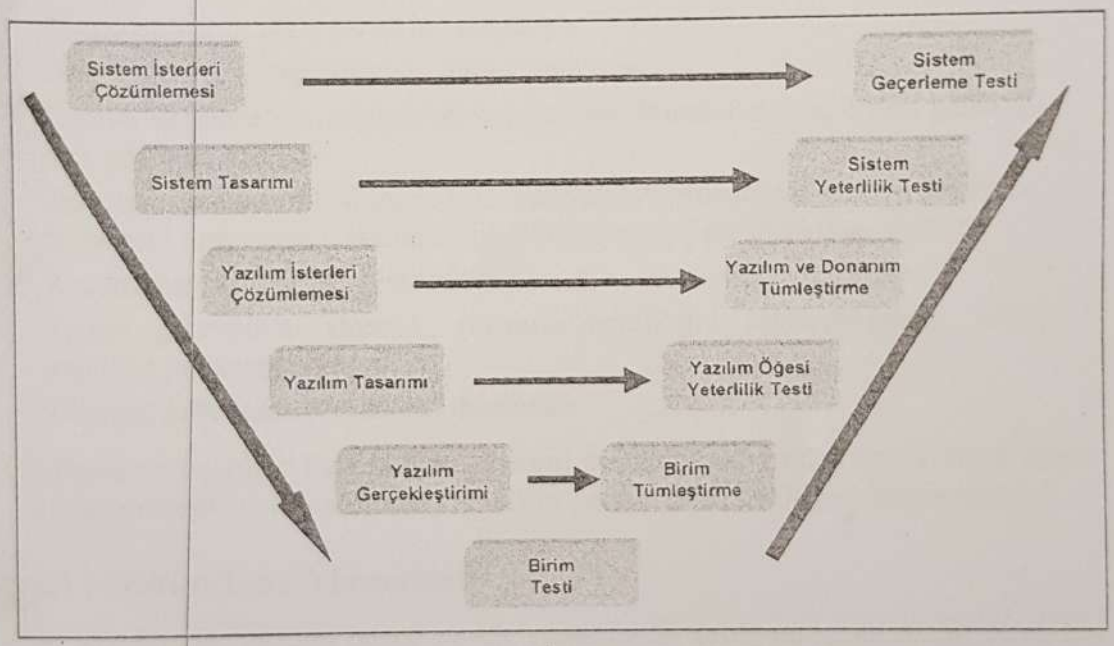
- **Akıllı derleyiciler**
  Bazı derleyiciler kaynak kodu tarayıp fazlaca tip denetimi yapmadan makine kodu üretirler. Bazıları ise kaynak kodu çok sıkı denetimden geçirdikten sonra kod üretirler. Tarama sırasında hatalı olarak değerlendirdikleri her durum asıl yürütme anında bir çöküşe neden olabilir. Bu nedenle, hassas tip denetimi yapan, kod içindeki bazı sakıncalı durumları yakalayıp uyarı verebilen derleyiciler sonuçta daha güvenilir kod üretirler.
- **Durağan çözümleyiciler**
  Bu araçlar, yazılımın kaynak kodunu inceleyerek yapısındaki zayıf noktaları bulur ve uyarı verirler. Kodlayıcı bu uyarı veya önerileri dikkate alarak kodu

daha güvenilir hale getirir. Bazı araçlar da kodu biçim bakımından inceleyerek çeşitli süzgeçlerden geçirirler ve daha güvenilir kaynak kod elde edilmesini sağlarlar.

- **Benzetim ortamları**
  Bir yazılımın gerçek ortamda çalıştığı takdirde nasıl davranacağını denemek üzere onu sanal bir işlemci üzerinde çalıştırarak test etmek mümkündür. Bu maksatla, merkezi işlem birimi, donanım mimarisi ve işletim sisteminin özellikleri dikkate alınarak özel sistemler geliştirilir. Bu sisteme ait derleyiciler kullanılarak kaynak kodun sanal donanım üzerinde çalışması sağlanır. Özellikle gerçek zamanlı ve paralel görevcikleri olan yazılımlar bu şekilde test edilebilirler. İşlemcinin iş sıralamasını nasıl yaptığı, tahsisli süre içinde görevin tamamlanıp tamamlanmadığı gibi zamana yönelik testler bu şekilde yapılabilir.
- **Girdi dosyaları**
  Bazı yazılımlar çeşitli dosyalardan veri okuyarak işlem yapmak üzere geliştirilirler. Bu tür yazılımları test edebilmek üzere uygulama alanında kullanılabilecek türden verileri önceden üretip uygun biçimde dosyalara yazan araçlar kullanılır. Bu araçların bazılarının işlevleri piyasada hazır olarak bulunan yazılım paketleri ile karşılanabilir; bazıları içinse özel yazılım geliştirilmesi gereklidir.
- **Test yazılımları**
  Test edilmesi gereken bir yazılım birimine gerçek ortamdaki gibi veri veya olay girişi sağlamanın bir yolu da test edilecek yazılıma girdi sağlayan diğer yazılım ya da donanımların yerine geçebilecek yazılımlar kullanmaktır. Bu şekilde, test donanımı ya da gerçek donanım üzerinde test edilecek yazılımı çalıştırıp ona girdi sağlayan yazılımlar yardımıyla test senaryoları oluşturulur.
- **Simgesel testler**
  Bu testler sayısal değerler yerine cebirsel işlemlerle yapılır. Bazı yazılımlar belirli algoritmalar çalıştırarak giriş değerlerine göre çıkış değerleri üretirler. Bunu algoritmaları test edebilmek için girişte cebirsel denklemler kullanan ve çıkışı da yine cebirsel olan özel yazılım gereçleri kullanılır. Bu testler yapıldıktan sonra algoritmalar kodlanır ve program testleri ayrıca yapılır.
- **Çevre benzeticileri**
  Özellikle gömülü, tahsisli ve gerçek zamanlı kontrol sistemlerini gerçek çalışma koşullarında dinamik olarak test edebilmek üzere çevreden gelen etkileri benzetim yoluyla sisteme girebilen ve sistem çıkışlarını alabilen benzeticiler kullanılır. Bunlar, belirli arayüz donanımlarına sahip elektronik cihazlar olabileceği gibi başka bilgisayarlar üzerinde çalışan yazılımlar da olabilir.
- **Sergileme yazılımları**
  Zaman aralıklı ölçümler, hesaplamalar ve denetimler yapan sistemlerin hesaplama sonuçlarını grafik üzerinde göstermek, elde edilen değerleri iki ya da üç boyutlu grafik nesneler halinde sergilemek yapılan testlerin sonuçlarının daha çabuk değerlendirilmesini ve anlatılmasını sağlar.

12.2. Test Stratejileri (12.2.)

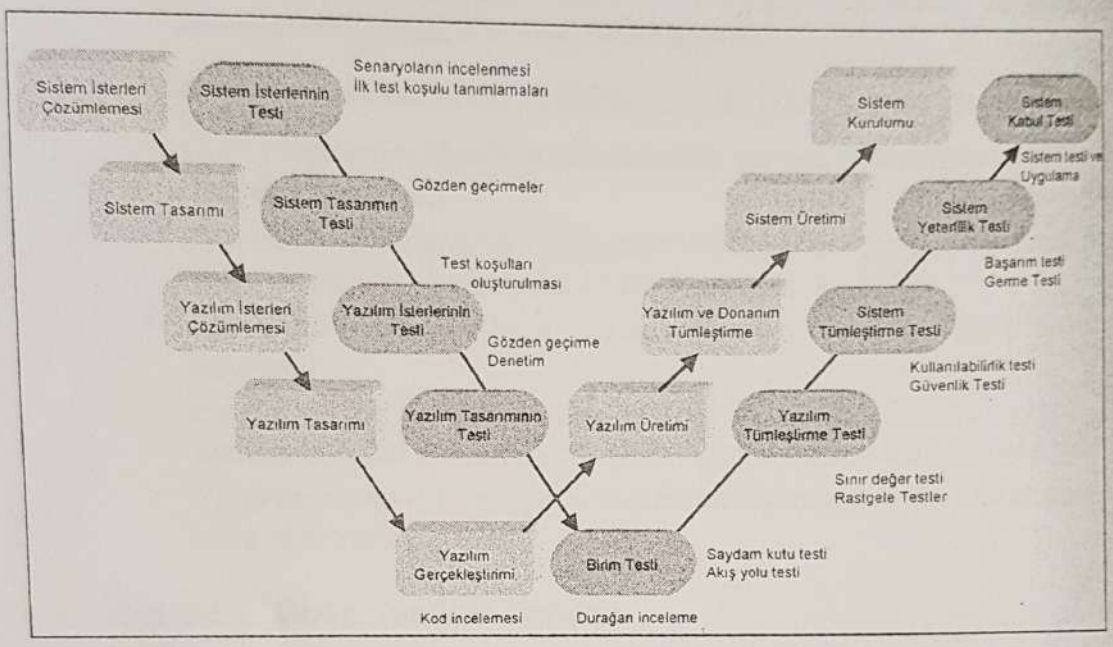
Buraya kadar açıkladığımız yöntemlerle yazılım testlerinin nasıl yapılabileceğini ortaya koyduk. Ancak, yazılım bir sistem olarak geliştiriliyorsa bu sistemin testlerinin de belirli bir sırada, belirli şekillerde, belirli stratejilere göre yapılması gereklidir. Bilgisayar tabanlı sistemler genellikle birden fazla yazılım ve donanım öğesinden meydana gelir. Bu öğelerin herbirinin ayrı alt bileşenleri ve birimleri olabilir. Sistemin bütününün iyi bir nitelik düzeyine ulaşabilmesi için bunların herbirinin testinin yapılması gereklidir. Test stratejilerinden en önemlilerinden biri büyük sistemler için uygulanan bir geliştirme modeli olan "V Modeli"dir. Şekil-8.3 te gösterilen bu modelde, sistem mühendisliği ve isterler çözümlemesinden sonra yapılan yazılım tasarım ve kodlama aşamalarının herbirinin karşıtı olan bir test aşaması bulunmaktadır:



Şekil-8.3. Sistem geliştirmede "V" Modeli.

Bu modelde geliştirme ve test paralel bir şekilde yapılır. Her aşama sonunda test edilecek ürün, bağımsız bir test grubu tarafından sınanır, onay verildikçe bir sonraki aşamaya geçilir. Örneğin, çözümleme aşamasının testi İsterler Belirtimi'nin gözden geçirilmesidir. Her aşamada yapılan test bir tür doğrulama sağlar. Her test aşaması da geçerleme sağlar. Günümüzde halen kullanılmasına karşın, bu model yerine, özellikle ilk kez yapılan geliştirme projelerinde, evrimsel veya spiral modeller daha fazla seçilmektedir.

Yine de, "V" şeklinin sol tarafındaki belgeler ile sağ tarafındaki belgeler arasında her zaman bire bir örtüşme sağlanamaz. Ayrıca, V-Modeli durağan testleri de içermemektedir. Bunlar arasında gözden geçirmeler, denetimler, kod incelemeleri sayılabilir. Bu nedenle, V-Modelini W-Modeline dönüştüren Şekil-8.4 teki ara etkinlikler ve test aşamaları ortaya konmuştur:



Şekil-8.4. Sistem geliştirmede "W" Modeli.

Bu modelde, ürünün en riskli görülen kısımlarına daha fazla eğilecek şekilde testler konmaktadır. Bunların bir kısmı durağan, bir kısmı da dinamik testlerdir. Geliştirme sürecinde yer alan aşamalar uygulama alanı ve ürünün özelliklerine göre değiştirilebileceği gibi, her aşamanın karşılığı olan test etkinlikleri de gereksinime göre değiştirilebilir.

12.2.1. Birim Testi

Bir bilgisayar yazılımının en küçük birimi yürütülebilir bir programdır. Bir sistem, bir ya da daha fazla yürütülebilir programdan oluşabilir. Özellikle çok programlı sistemlerde önce birimlerin ayrı ayrı test edilmeleri gereklidir. Bu şekilde, yazılım mühendisinin veya kodlayıcının elinden çıkan kodun hatalarının önce kendi içinde düzeltilmesi sağlanır. Daha çok saydam kutu yönteminin kullanıldığı *birim testi* (unit testing) ile düzgün ve hatasız çalıştığına kanaat getirilen bir birim artık tümleştirme testi için hazır hale gelmiş olur.

Birim testi sırasında ayrıntılı tasarım temel alınarak o birim sınırları içinde kalan önemli denetim yollarının ve veri yapılarının içinde bulunabilecek hataların yakalanmasına ve giderilmesine çalışılır. Tasarım ne kadar iyi olursa olsun yine de kodlamayı bir insan yapmak zorundadır. Her satır kodun yazılıma potansiyel olarak hata sokabileceği bir gerçektir. Öbekler halinde veya bir bütün olarak yazılım birimi test edilir, hataların yakalanmasına çalışılır. Test koşullarının karmaşıklığı ve yakalanabilecek hataların miktarı test edilen birimin içsel yapısıyla beraber diğer birimlerle olan ilişkisine bağlıdır.

12.1.1. Birim Gerçekleştiriminde Hatalar

Gerçekleştirim, yani kodlama sırasında derleyici denetiminden geçen ancak çalışma sırasında ortaya çıkabilen yaygın hatalar arasında şunları sayabiliriz:

- Deyimleri simgesel olarak göstermedeki yanlışlıklar
- İlk değerlerin atanmasında yanlışlık ya da eksiklik
- Farklı veri tiplerinin karşılaştırılması ve yordam parametrelerinde uyuşmazlık (kuvvetli tip kontrolü yapmayan diller için)
- Kayan nokta değerleri arasında karşılaştırma yapıldığında hassasiyet değerinin karşılaştırmayı yanıltması
- Aritmetik operatörlerin önceliklerinin yanlış kullanılması
- Parantezlerin kullanımında hata yapılması
- Döngüden yanlış çıkış ya da hiç çıkmama
- Döngü kontrol değişkenlerinin yanlış kullanımı
- Tasarım sırasında hiç olasılık verilmeyen bir dallanmada yapılması gerekenlerin göz ardı edilmesi
- Yakalanan hatanın düzeltilmesi sırasında yapılan yanlışlıklar (durum değişkenlerine eski değerlerinin atanmaması, akışın yanlış yere aktarılması gibi)
- Koşutzamansız iletişim yönteminde oluşan hataların kotarılmaması
- Paralel görevcikler (thread) arasında eşanlılığın sağlanamaması, kritik yapıların korunamaması
- Ölümcül kilitlenme (deadlock) durumları

Günümüz derleyicileri hala birçok denetimi insana bıraktığından kodlayıcılık deneyimi, gerçekleştirim aşamasındaki en önemli etken olma özelliğini korumaktadır.

12.1.2. Birim Testi Yöntemleri

Test edilebilecek yazılım birimi ya bir yürütülebilir programdır ya da bir sınıf, bir paket veya bir kütüphane gibi pasif bir yazılım modülüdür. Bu birimlerin testleri de aktif ya da pasif olmalarına göre farklı şekillerde yapılır. Yazılım birimi olarak bir sınıf, bir paket veya bir kütüphane kabul edilirse, bir pasif birimin testi şu şekilde yapılır:

- Bir test programı geliştirilerek test edilecek birimi kullanması sağlanır.
- Test programının ana yordamı içinden birimin tüm arayüzlerini denemek üzere çağrılar yapılır.
- Birime gönderilen ve alınan değerler ya ekrana ya da bir dosyaya yazdırılır.
- Test edilecek birim başka birimleri kullanıyorsa bu birimler de test programı içine dahil edilmelidir.
- Birim içindeki verileri, durum değişkenlerini incelemek için ya klasik yöntem olan ekrana yazdırma kullanılır, ya da iyi bir hata ayıklayıcı araç (debugger) ile yürütme sırasında inceleme yapılır.

- Ne tür testler yapılacağı önceden bir belgeye yazılmalı, bu belge ve test sonuçları saklanmalıdır (Test Tanımlaması).

Yazılım birimi kendi başına yürütülen, ayrı bir aktif program ise şu şekilde test edilir:

- Önce programın çalışabilmesi için gerekli altyapı, ilklendirme ve yapılandırma dosyaları hazırlanır.
- Test edilecek programın etkileşimde olduğu diğer programlar hazır ise önce onlar çalıştırılır, sonra da test edilecek program başlatılır.
- Eğer diğer programlar hazır değilse onların yerine geçecek birer test programı hazırlanır.
- Programın ilk çalışma durumuna geçmesi gerekli iletilerin veya verilerin üretilmesi başka programlar ya da altyapı yardımcı araçlarıyla sağlanır.
- Programın gelen iletilere ya da verilere tepkisi sonucu ortaya çıkan yeni iletiler veya yeni veriler incelenir.

12.2.1.3. Birim Testinin Yapılışı

Bir birim testi yapılırken aşağıdakilere dikkat edilmelidir:

- Birimin arayüzü test edilerek bilgi giriş/çıkışlarının uygun ve yeterli şekilde yapıldığı kontrol edilir. Örneğin, programa giren ve çıkan tüm iletilerin doğru tipte oldukları gerçeklenir.
- Yerel veri yapıları inceleme altına alınarak algoritmanın çalışması boyunca ya da yordamların çağrılması sırasında verilerin saklandığı yerin bütünlüğünün bozulup bozulmadığı test edilmelidir. Veri yapıları, özelikle dinamik şekilde büyüyüp küçülen yapılar, yanlış işaretçi kullanımı nedeniyle en yüksek hata üretme olasılığına sahiptirler. Veri yapılarının kapasiteleri ve erişim hızları kadar erişilen veri kayıtlarının kullanım şekli de hataya düşürmeyecek şekilde gerçekleştirilmelidir.
- Yazılım birimleri işlevsel kısıtlamalar ve sayısal sınırlar içinde çalışırlar. Bu şekilde verilerin ve işlevlerin korunması sağlanır. Örneğin, veritabanına girmesi gerekli kayıt adedi belirli bir sayıyı geçmemelidir. Birim, bir iletiye yanıt beklerken başka bir iletiye işlem yapmamalıdır. Bir hesaplama sonucu göndermesi gereken ileti içindeki veriler belirli sayısal değerleri aşmamalıdır (örneğin, hesaplanan hız 0.0 ile 100.0 m/s arasında olacaktır).
- Sınır durumlarının en düşük ve en yüksek tam değerleri, bu değerlerin biraz altı ve biraz üstü kullanılarak sınanmalıdır.
- Birim içindeki birbirinden bağımsız tüm çalışma yolları, tüm dallanmalar tek tek sınanmalıdır.
- Birim içindeki önemli değişkenlerin ve veri yapılarının ilk değerleri kod içinde kontrol edilmeli, çalışma sırasında bunların doğruluğu gerçeklenmelidir.
- Birim, bir donanım aygıtıyla iletişim içindeyse, açma-kapama tutamaçları (handle), hata yakalama ve rapor etme şekilleri, ileti türleri, iletilerin alanlarının

tipleri ve uzunlukları, adresler, bit sıraları gibi ayrıntılar kontrol edilmeli, belirli noktalar sınanmalıdır.

- Birim içindeki hata yakalayıcılar birer birer denenmelidir. Bir hata durumu yakalandığında aşağıdakilerin varlığı da test edilmelidir:
  - Yakalanan hataya uygun bir uyarı iletisinin verilmesi
  - Hatanın durdurup yeniden başlatmaya neden olup olmadığı
  - Hatanın kotarılmasının uygun şekilde yapılması
  - Hata iletisinin hatanın meydana geldiği yeri tam olarak belirli etmesi.

12.2.2. Tümleştirme Testi (12.2.2.)

Bir yazılım paketi çeşitli birimlerden oluşabilir. Bu paket uygun bir donanım üzerinde çalıştırılır. Yazılım ve donanım birimlerinin kendi başlarına sorunsuzca çalışıyor olmaları bir araya geldiklerinde de sorunsuz çalışacakları anlamına gelmez. Çünkü, birimler arası uyumu tam olarak sağlayabilmek oldukça güçtür ve çok test gerektirir. Birden fazla yazılım biriminin bir araya getirilerek uyumlu bir şekilde ve hatasız çalışması, herbirinin tek tek değil de bir bütün içinde, tasarımda belirtildiği şekilde, kendi üzerlerine düşen görevleri yerine getirip getirmedikleri *tümleştirme testi* (integration test) ile kontrol edilir. Çoğu zaman, bireysel olarak doğru çalıştığı sanılan yazılım birimleri, bir araya getirildiklerinde daha önceden fark edilemeyen veya öngörülemeyen davranışlarda bulunabilirler. Bu kusurlu davranışları yakalayabilmek için yapılan tümleştirme testi, birimler arasındaki arayüzlerden kaynaklanan kusurları ortaya çıkarabilmek üzere yapılan testlerle beraber program yapısını oluşturmak için uygulanan sistematik bir tekniktir. Amaç, birim testlerini geçmiş modülleri alıp tasarımda belirtilen program yapısını ortaya çıkarmaktır. Tümleştirme testinin yapılma nedenlerini şöyle özetleyebiliriz:

- Bir birimin çalışması bir başka birimin çalışmasını etkileyebilir.
- Birimler arasında koşutzamanlılığın sağlanması gereklidir.
- Bütün bir işlevi oluşturan alt işlevlerin herbiri belirli birimler tarafından doğru olarak yerine getirilse bile hepsi bir araya getirildiğinde beklenen sonucu vermeyebilir.
- Birimler arasındaki arayüzler arasında verilerin kaybolması olasılığı vardır (özellikle dağıtık mimarilerde)
- Bir birim için kabul edilebilir sınırlar içinde olan kesinlik değerleri birden fazla birimin devreye girmesi durumunda kabul edilemeyecek değerlere ulaşabilir.
- Birimler arasında paylaşılan evrensel veri yapıları sorun çıkarabilir.
- Birimlerin ağ üzerinde dağıtılması durumunda çalışma sırası, ileti aktarımı, koşutzamanlı iletişim sorunları, geri kazanma gibi konularda sorun yaşanabilir.
- Bir birimin gönderdiği iletilerin karşı tarafa ulaşma sıraları yürütme ortamındaki belirsiz durumlardan ötürü farklılık gösterebilir. Tasarım sırasında öngörülen iletilerin gönderilme ve alınma sıraları gerçek yürütme sırasında değişik olabilir.

Tümleştirme testleri genellikle yazılım geliştirenler tarafından değil de test için görevli ekipler tarafından kara kutu yöntemleriyle yapılır. Tabii ki test işlemleri belirli bir eşgüdüm içinde yürütülür.

Bazı geliştiriciler bir yazılım paketinin tüm birimlerini aynı anda bir araya getirerek test etme yöntemini seçerler. Daha önce iyice test edildiğinden emin olunan, kendini ispatlamış birimlerle bunu yapmak mümkün olabilir. Ancak, yeni geliştirilen birimlerden oluşan yazılım paketinin bütününün testi sırasında karşılaşılan kusurları düzeltmek son derece güçtür. Öncelikle sorunun hangi birimden kaynaklandığını belirlemek, sonra da o sorunun birim içindeki tam yerini ve nedenini bulmak gereklidir. Sorunu çözmek için yapılan işlemin başka birimleri de olumsuz şekilde etkilemesi mümkündür.

Bir anda tümleştirme yerine, yazılım paketini küçük parçalara bölerek birimler halinde artımlı olarak tümleştirmek daha sağlıklı sonuç verdiği için yaygın olarak kullanılan bir yöntemdir. Artımlı tümleştirme yönteminde değişik stratejiler kullanılabilir. Şimdi bunları görelim.

12.2.2.1. Yukarıdan Aşağı Tümleştirme

Yazılımı oluşturan birimlerin denetim sıradüzeni içinde, birimler ya *derinlik* tabanlı ya da *genişlik* tabanlı olarak tümleştirilirler. Önce ana denetim biriminin testi yapılır, sonra ona en yakın düzeydeki birimlerden biri ile beraber test yapılır. Bir sonraki aşamada ya aynı düzeydeki bir başka birimin testi yapılır; ya da aynı birimin bir alt düzeyindeki birimle tümleştirmesi ve testi yapılır. Bu işlemler sırasında, tümleştirmesi daha sonra yapılacak olan birimin geçici olarak yerine geçecek şekilde basit kod parçaları veya sürücüler kullanılır. Test sırası geldikçe bu geçici kod parçaları gerçekleriyle değiştirilir. Herbir eklemede daha önce yapılan testlerin bir kısmı ya da tamamı tekrar yapılarak yeni bir kusur eklenip eklenmediğinden emin olunur.

Bu test stratejisi, yazılımda denetim ve akış sorunları varsa bunların öncelikli olarak belirlenmesini, tasarımda öngörülen önemli karar verme noktalarının erkenden gerçeklenmesini sağlar. Eğer derinlik tabanlı yöntem kullanılırsa, bir işlevin tamamının gerçekleştirilmesi ve test edilmesi sağlanabilir.

Yukarıdan aşağı tümleştirme stratejisinin en büyük sorunu geçici yazılım birimlerinin (stub) uygun şekilde hazırlanmasıdır. Gerekli tüm testleri yapabilmek için oldukça yetenekli kod parçalarının geliştirilmesi gereklidir. Bu nedenle, ya yeterli veri giriş/çıkışını sağlayabilecek kod geliştirilmeli ya da ayrıntılı testler sonraya ertelenmelidir.

12.2.2.2. Aşağıdan Yukarıya Tümleştirme

Bu stratejide tümleştirme atomik birimlerin çalıştırılıp test edilmesiyle başlar. Alt düzey birimler birleştirilerek kümeler haline getirilir. Bu küme, test senaryosu gereğince geliştirilen bir sürücü ana programla veri giriş/çıkışı bakımından test edilir. Benzer şekilde, diğer alt düzey birimler de kümeler halinde test edilir. Daha sonra sürücü programlar ayrılarak yazılım paketinin yapısı içinde bulunan bu kümelerin

birleştirilmesinden oluşan daha üst düzeyde yeni kümeler meydana getirilir. Bu kümeler de yine sürücü programlarla test edilir. Bu şekilde en üstte bulunan ana birime kadar ulaşılır. Tümleştirme üst düzeylere doğru çıktıkça test sürücülerine olan gereksinim azalır. Bu stratejinin en büyük sorunu da tümleştirme sonuçlanmadan önce ana yazılım paketini oluşturmanın mümkün olmamasıdır.

Bazı dağıtık sistem yazılımları büyük kümelerden oluşan yazılım öğeleri şeklinde geliştirilir ve test edilirler. Herbir yazılım öğesi ya bütün olarak ya da aşağıdan yukarı yöntemle tümleştirilir ve test edilir.

Hangi stratejinin seçileceği yazılımın özelliğine, yazılım geliştirme deneyimlerine ve bazen de proje planlamasına bağlıdır. Her iki stratejinin üstünlüklerinden yararlanabilmek için üst düzeydeki birimlerin yukarıdan aşağı, alt düzeydeki birimlerin de aşağıdan yukarı tümleştirilmesi en yararlı yöntem olarak benimsenmektedir. Testler sırasında en kritik olarak değerlendirilen birimlere özel önem verilerek test senaryolarının onları iyice kapsaması sağlanmalıdır.