# IT522 – Yazılım Mühendisliği 2021


PhD Furkan Gözükara, Toros University

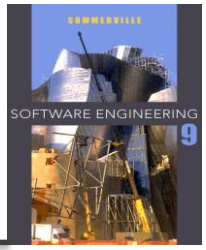
https://github.com/FurkanGozukara/Yazilim-Muhendisligi-IT522-2021

# Ders 13

# Güvenilirlik Mühendisliği

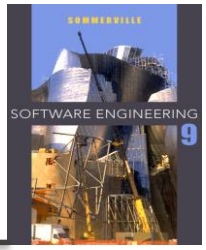

Kaynak : https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Presentations/index.html

# Bölüm 1'de İşlenmiş Konular



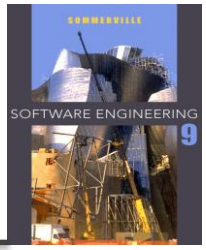

- Fazlalık ve çeşitlilik
  - Hata toleransı elde etmek için temel yaklaşımlar.
- Güvenilir süreçler
  - Güvenilir süreçlerin kullanımı güvenilir sistemlere nasıl yol açar?
- Güvenilir sistem mimarileri
  - Yazılım hata toleransı için mimari modeller
- Güvenilir programlama
  - Hataları önlemek için programlama yönergeleri.

# Yazılım Güvenilebilirliği



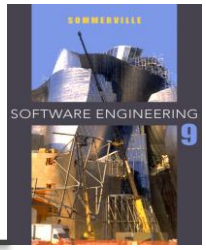

- Genel olarak, yazılım müşterileri tüm yazılımların güvenilir olmasını bekler. Ancak kritik olmayan uygulamalar için bazı sistem hatalarını kabul etmeye istekli olabilirler.
- Bazı uygulamaların (kritik sistemler) çok yüksek güvenilirlik gereksinimleri vardır ve bunu başarmak için özel yazılım mühendisliği teknikleri kullanılabilir.
  - Tıbbi sistemler
  - Telekomünikasyon ve güç sistemleri
  - Uzay sistemleri

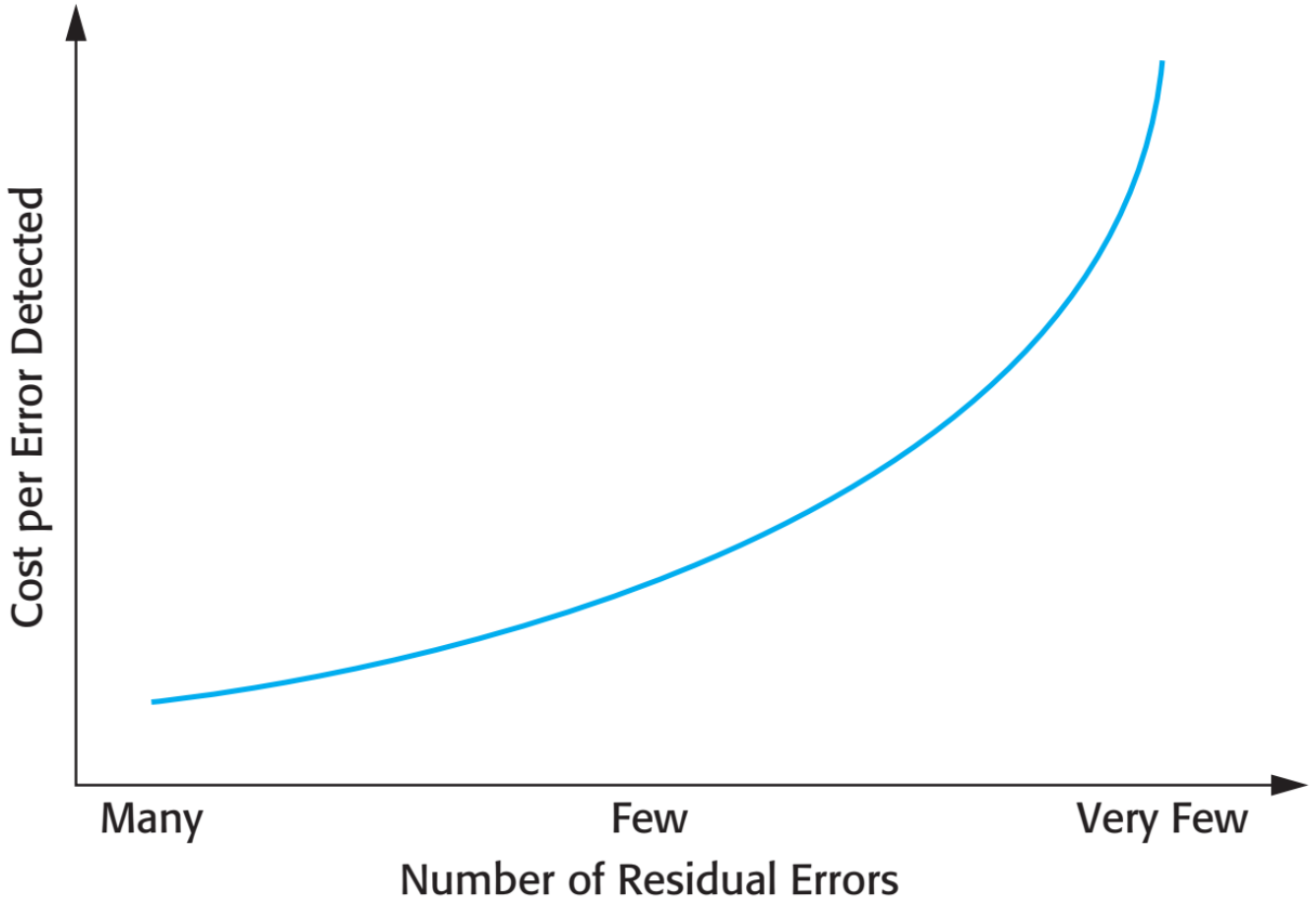
# Güvenilebilirlik Başarısı



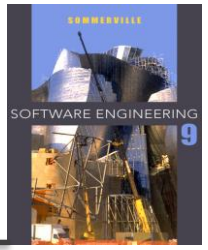

- Hata önleme
  - Sistem, insan hatasından kaçınılacak ve böylelikle sistem arızalarının minimuma indirileceği şekilde geliştirilmiştir.
  - Geliştirme süreci, sistemdeki arızaların müşteriye teslim edilmeden önce tespit edilip onarılması için düzenlenir.
- Arıza tespiti
  - Doğrulama ve onaylama teknikleri, devreye alınmadan önce bir sistemdeki hataları keşfetmek ve ortadan kaldırmak için kullanılır.
- Hata toleransı
  - Sistem, teslim edilen yazılımdaki hataların sistem arızasına neden olmayacak şekilde tasarlanmıştır.

# Artık Arıza Gidermenin Artan Maliyetleri



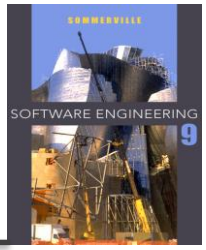



# Regüle Edilen Sistemler



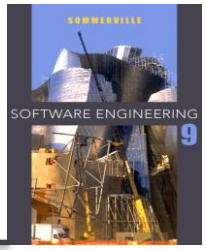

- Çoğu kritik sistem düzenlenmiş sistemlerdir, bu da kullanımlarının sistemler hizmete girmeden önce harici bir düzenleyici tarafından onaylanması gerektiği anlamına gelir.
  - Nükleer sistemler
  - Hava trafik kontrol sistemleri
  - Tıbbi cihazlar
- Bir güvenlik ve güvenilirlik durumu, düzenleyici tarafından onaylanmalıdır. Bu nedenle, kritik sistem geliştirme, bir düzenleyiciyi sistemin güvenilir, güvenli ve emniyetli olduğuna ikna etmek için kanıt oluşturmalıdır.

# Çeşitlilik ve Artıklık



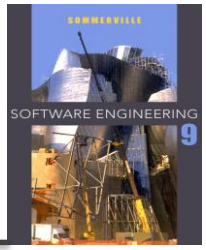

- Yedeklilik
  - Bir kritik bileşenin birden fazla sürümünü hazır bulundurun, böylece biri başarısız olursa bir yedek kullanılabilir.
- Çeşitlilik
  - Aynı işlevselliği farklı şekillerde sağlayın, böylece aynı şekilde başarısız olmazlar.
- Ancak, çeşitlilik ve fazlalık eklemek karmaşıklık katar ve bu da hata olasılığını artırabilir.
- Bazı mühendisler basitliği savunur ve kapsamlı V & V, yazılım güvenilirliği için daha etkili bir yoldur.

# Çeşitlilik ve Artıklık Örnekleri



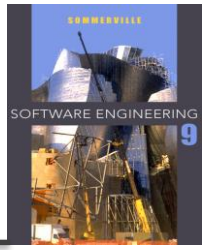

- Fazlalık. Kullanılabilirliğin kritik olduğu yerlerde (örneğin, e-ticaret sistemlerinde), şirketler normalde yedekleme sunucularını tutar ve bir arıza meydana gelirse bunlara otomatik olarak geçer.
- Çeşitlilik. Dış saldırılara karşı direnç sağlamak için, farklı işletim sistemleri (ör. Windows ve Linux) kullanılarak farklı sunucular uygulanabilir.

# Süreç Çeşitliliği ve Yedeklilik



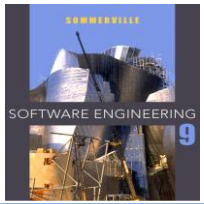

- Doğrulama gibi süreç faaliyetleri, sistemi doğrulamak için test etme gibi tek bir yaklaşıma bağlı olmamalıdır.
- Bunun yerine, birden fazla farklı süreç etkinliği birbirini tamamlar ve yazılımda hatalara yol açabilecek süreç hatalarını önlemek için çapraz kontrol yardımına izin verir.

# Güvenilir Süreçler



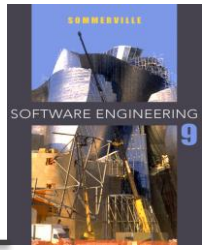

- Minimum sayıda yazılım hatası sağlamak için, iyi tanımlanmış, tekrarlanabilir bir yazılım sürecine sahip olmak önemlidir.
- İyi tanımlanmış tekrarlanabilir bir süreç, tamamen bireysel becerilere bağlı olmayan bir süreçtir; daha ziyade farklı kişiler tarafından canlandırılabilir.
- Düzenleyiciler, iyi bir yazılım mühendisliği uygulamasının kullanılıp kullanılmadığını kontrol etmek için süreç hakkındaki bilgileri kullanır.
- Hata tespiti için, süreç faaliyetlerinin doğrulama ve onaylamaya adanmış önemli çabayı içermesi gerektiği açıktır.

# Güvenilir Süreçlerin Nitelikleri



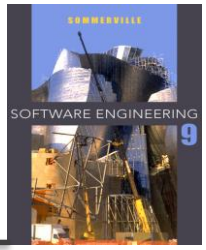

| İşlem karakteristiği | Açıklama |
|---|---|
| Belgelenebilir | Süreç, süreçteki faaliyetleri ve bu faaliyetler sırasında üretilecek belgeleri ortaya koyan tanımlanmış bir süreç modeline sahip olmalıdır. |
| Standartlaştırılmış | Yazılım üretimi ve dokümantasyonu kapsayan kapsamlı bir yazılım geliştirme standartları seti mevcut olmalıdır. |
| Denetlenebilir | Süreç, süreç standartlarının takip edildiğini kontrol edebilen ve süreç iyileştirme için önerilerde bulunan süreç katılımcıları dışında kişiler tarafından anlaşılabilir olmalıdır. |
| Çeşitli | Süreç, fazlalık ve çeşitli doğrulama ve onaylama faaliyetlerini içermelidir. |
| güçlü | Süreç, bireysel süreç faaliyetlerinin başarısızlıklarından kurtulabilmelidir. |

# Doğrulama Faaliyetleri



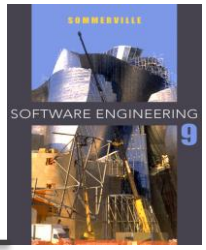

- Gereksinim incelemeleri.
- İhtiyaç Yönetimi.
- Biçimsel şartname.
- Sistem modelleme
- Tasarım ve kod incelemesi.
- Statik analiz.
- Test planlaması ve yönetimi.
- Ders 25'te tartışılan değişim yönetimi de önemlidir.

# Hata Toleransı



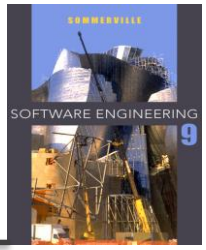

- Kritik durumlarda, yazılım sistemleri hataya dayanıklı olmalıdır.
- Arıza toleransı, yüksek kullanılabilirlik gereksinimlerinin olduğu veya sistem arızası maliyetlerinin çok yüksek olduğu yerlerde gereklidir.
- Hata toleransı, sistemin yazılım arızasına rağmen çalışmaya devam edebileceği anlamına gelir.
- Sistemin spesifikasyonuna uygun olduğu kanıtlanmış olsa bile, spesifikasyon hataları olabileceğinden veya doğrulama yanlış olabileceğinden hataya dayanıklı olmalıdır.

# Güvenilir Sistem Mimarileri



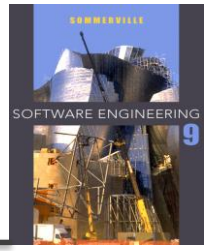

- Güvenilir sistem mimarileri, hata toleransının gerekli olduğu durumlarda kullanılır. Bu mimarilerin tümü genellikle fazlalık ve çeşitliliğe dayanır.
- Güvenilir mimarilerin kullanıldığı durumlara örnekler:
  - Sistem arızasının yolcuların güvenliğini tehdit edebileceği uçuş kontrol sistemleri
  - Bir kontrol sisteminin arızalanmasının kimyasal veya nükleer bir acil duruma yol açabileceği reaktör sistemleri
  - 7/24 kullanılabilirliğe ihtiyaç duyulan telekomünikasyon sistemleri.

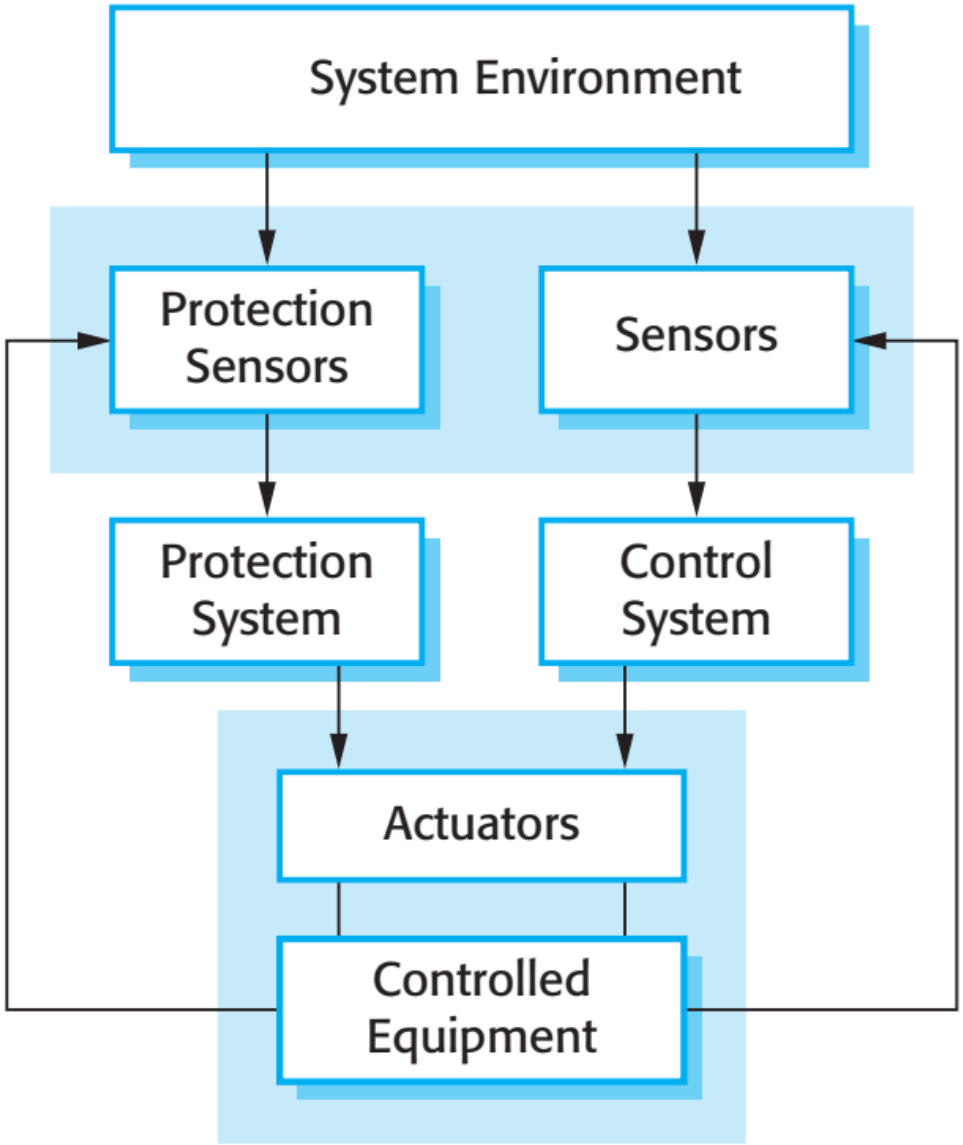
# Koruma Sistemleri



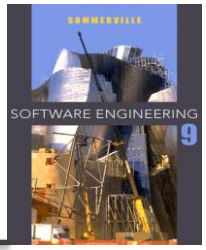

- Bir arıza meydana geldiğinde acil müdahale yapabilen başka bir kontrol sistemiyle ilişkili özel bir sistem.
  - Kırmızı ışıktan geçerse treni durduran sistem
  - Sıcaklık / basınç çok yüksekse reaktörü kapatan sistem
- Koruma sistemleri, kontrol edilen sistemi ve ortamı bağımsız olarak izler.
- Bir sorun tespit edilirse, sistemi kapatmak ve bir felaketten kaçınmak için acil eylem gerçekleştirme komutları verir.

# Koruma Sistemi Mimarisi



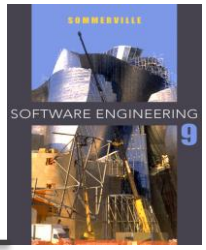



# Koruma Sistemi İşlevselliği



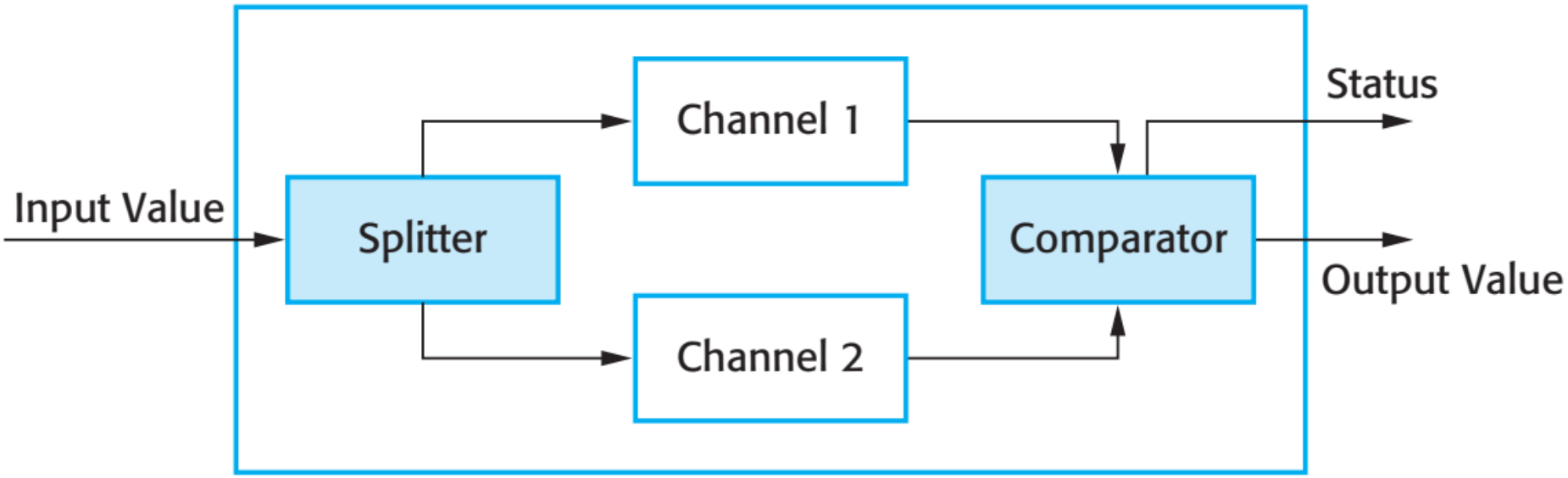

- Koruma sistemleri, kontrol yazılımındakileri kopyalayan izleme ve kontrol yeteneklerini içerdikleri için fazlalıktır.
- Koruma sistemleri çeşitli olmalı ve kontrol yazılımından farklı teknolojiler kullanmalıdır.
- Kontrol sisteminden daha basit oldukları için doğrulama ve güvenilirlik güvencesi için daha fazla çaba harcanabilir.
- Amaç, koruma sistemi için talep üzerine düşük bir arıza olasılığının olmasını sağlamaktır.

# Kendi Kendini İzleyen Mimariler

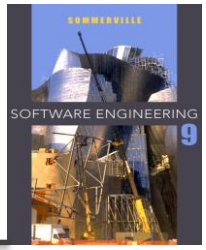



- Sistemin kendi işlemlerini izlediği ve tutarsızlıklar tespit edilirse harekete geçtiği çok kanallı mimariler.
- Her kanalda aynı hesaplama yapılır ve sonuçlar karşılaştırılır. Sonuçlar aynıysa ve aynı zamanda üretiliyorsa, sistemin doğru çalıştığı varsayılır.
- Sonuçlar farklıysa, bir arıza olduğu varsayılır ve bir arıza istisnası ortaya çıkar.

# Kendi Kendini İzleme Mimarisi

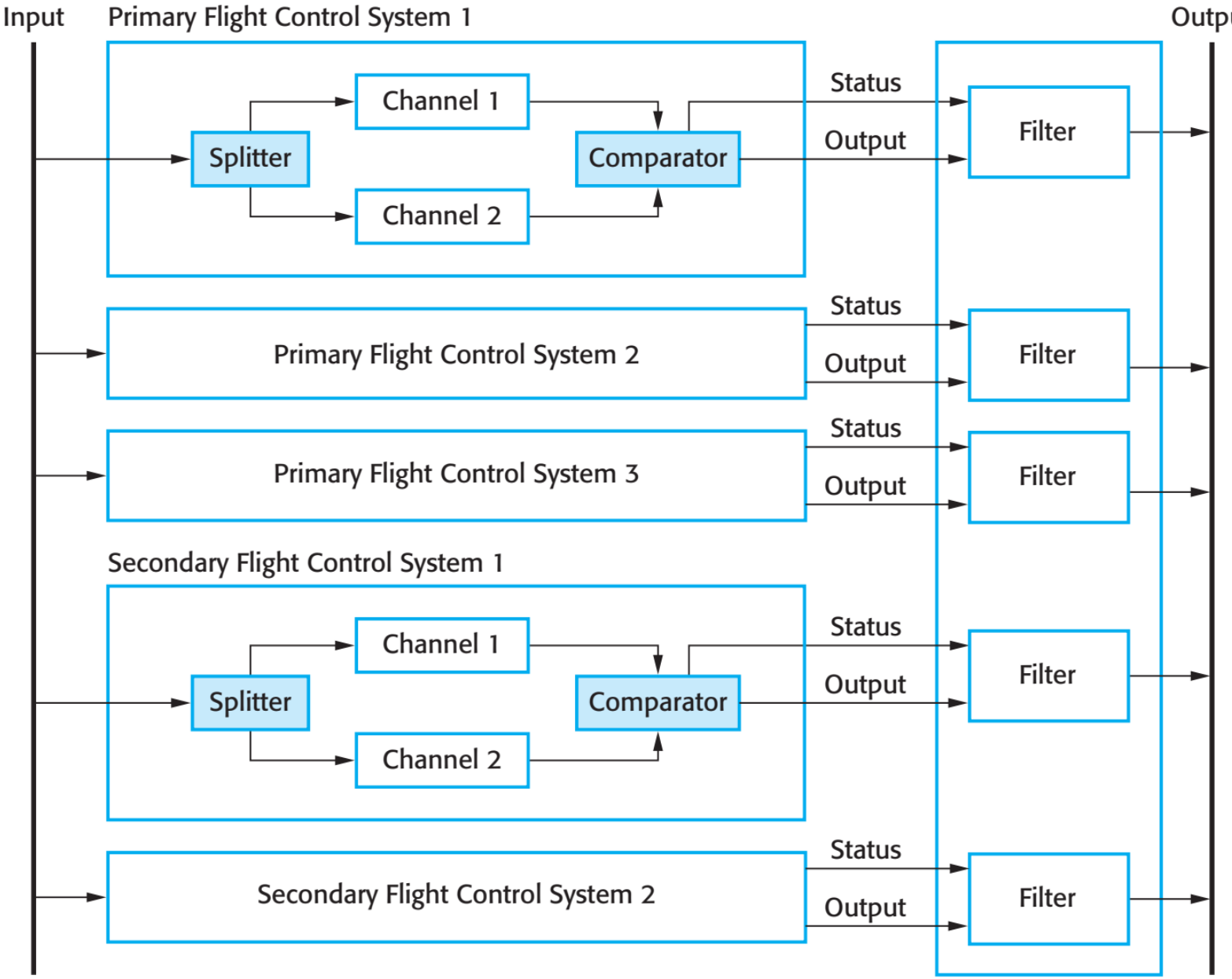


# Kendi Kendini İzleme Sistemleri

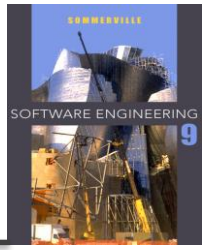



- Ortak mod donanım arızasının her kanalın aynı sonuçları üretmesine yol açmaması için her kanaldaki donanımın çeşitli olması gerekir.
- Her kanaldaki yazılım da çeşitli olmalıdır, aksi takdirde aynı yazılım hatası her kanalı etkileyecektir.
- Yüksek kullanılabilirlik gerekiyorsa, paralel olarak birkaç kendi kendini denetleme sistemini kullanabilirsiniz.
  - Bu, Airbus uçak ailesinde uçuş kontrol sistemleri için kullanılan yaklaşımdır.

# Airbus Uçuş Kontrol Sistemi Mimarisi

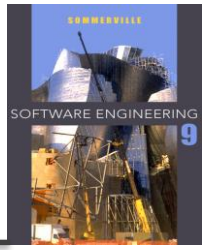



# Airbus Mimarisi Tartışması

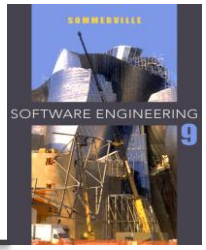



- Airbus FCS, herhangi biri kontrol yazılımını çalıştırabilen 5 ayrı bilgisayara sahiptir.
- Çeşitlilik kapsamlı bir şekilde kullanıldı
  - Birincil sistemler, ikincil sistemlerden farklı bir işlemci kullanır.
  - Birincil ve ikincil sistemler, farklı üreticilerin yonga setlerini kullanır.
  - İkincil sistemlerdeki yazılım, birincil sistemdekinden daha az karmaşıktır - yalnızca kritik işlevsellik sağlar.
  - Her kanaldaki yazılım, farklı ekipler tarafından farklı programlama dillerinde geliştirilir.
  - Birincil ve ikincil sistemlerde kullanılan farklı programlama dilleri.

# Bölüm 1 Anahtar Noktaları



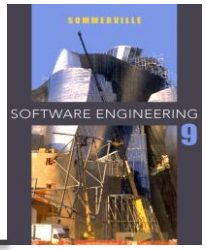

- Bir programdaki güvenilebilirlik, hataların ortaya çıkmasını önleyerek, sistem dağıtımından önce hataları tespit edip ortadan kaldırarak ve hata toleransı olanaklarını dahil ederek sağlanabilir.
- Donanım, yazılım süreçleri ve yazılım sistemlerinde yedeklilik ve çeşitliliğin kullanılması, güvenilir sistemlerin geliştirilmesi için gereklidir.
- Bir sistemdeki arızalar en aza indirilecekse, iyi tanımlanmış, tekrarlanabilir bir sürecin kullanılması çok önemlidir.
- Güvenilir sistem mimarileri, hata toleransı için tasarlanmış sistem mimarileridir. Hata toleransını destekleyen mimari stiller arasında koruma sistemleri, kendi kendini izleyen mimariler ve N-sürümü programlama bulunur.




# Ders 13 - Güvenilirlik Mühendisliği

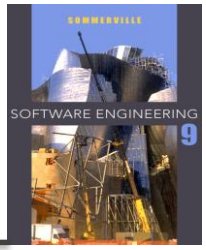
## Bölüm 2

# N-Sürüm Programlama



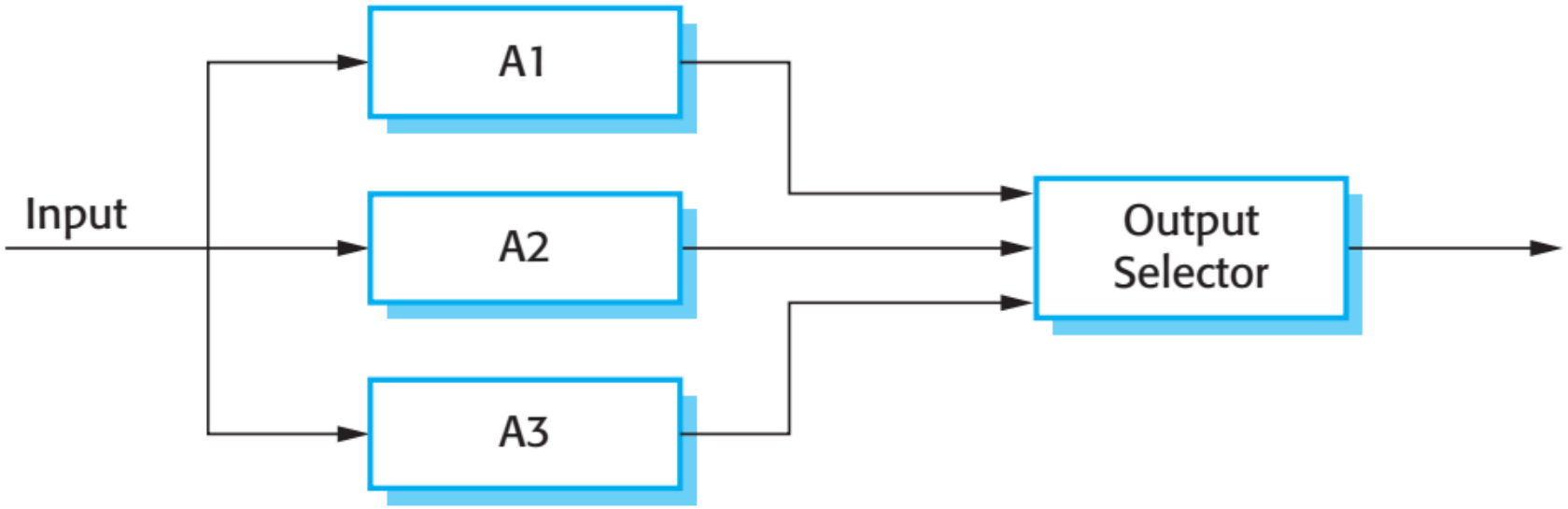

- Bir yazılım sisteminin birden çok sürümü aynı anda hesaplamalar yapar. İşin içinde tek sayıda bilgisayar olmalıdır, tipik olarak 3.
- Sonuçlar bir oylama sistemi kullanılarak karşılaştırılır ve çoğunluk sonucu doğru sonuç olarak alınır.
- Yaklaşım, donanım sistemlerinde kullanılan üçlü modüler artıklık kavramından türetilmiştir.

# Donanım Hatası Toleransı

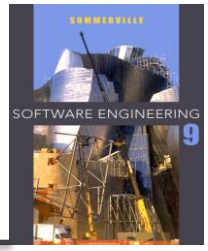


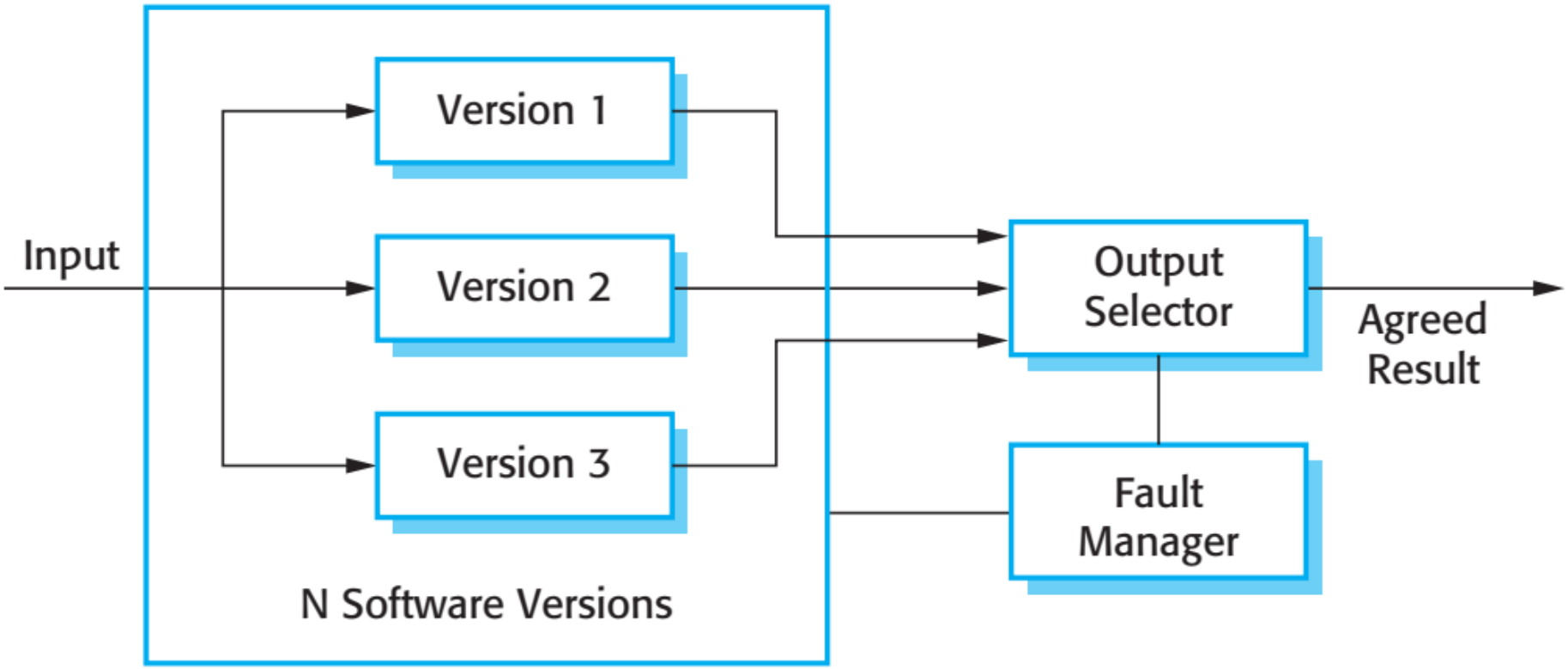

- Üçlü modüler yedekliliğe (TMR) bağlıdır.
- Aynı girdiyi alan ve çıktıları karşılaştırılan üç birbirinin aynısı bileşen vardır.
- Bir çıktı farklıysa, dikkate alınmaz ve bileşen arızası varsayılır.
- Tasarım hataları yerine bileşen arızalarından kaynaklanan çoğu hataya ve eşzamanlı bileşen arızası olasılığının düşük olmasına dayanır.

# Üçlü Modüler Artıklık

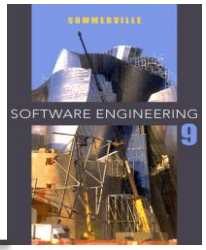


# N-Sürüm Programlama



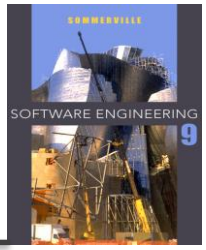



# N-Sürüm Programlama



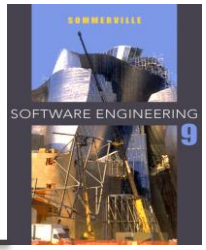

- Farklı sistem sürümleri, farklı ekipler tarafından tasarlanır ve uygulanır. Aynı hataları yapma olasılığının düşük olduğu varsayılmaktadır. Kullanılan algoritmalar farklı olmalıdır, ancak farklı olmayabilir.
- Ekiplerin genellikle spesifikasyonları aynı şekilde yanlış yorumladıklarına ve sistemlerinde aynı algoritmaları seçtiklerine dair bazı deneysel kanıtlar vardır.

# Yazılım Çeşitliliği



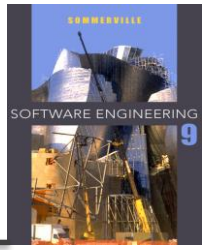

- Yazılım hata toleransına yönelik yaklaşımlar, aynı yazılım spesifikasyonunun farklı uygulamalarının farklı şekillerde başarısız olacağının varsayıldığı yazılım çeşitliliğine bağlıdır.
- Uygulamaların (a) bağımsız olduğu ve (b) yaygın hataları içermediği varsayılmaktadır.
- Çeşitliliğe ulaşmak için stratejiler
  - Farklı programlama dilleri
  - Farklı tasarım yöntemleri ve araçları
  - Farklı algoritmaların açık belirtimi

# Tasarım Çeşitliliği İle İlgili Sorunlar



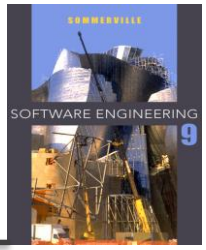

- Takımlar kültürel açıdan farklı değildir, bu nedenle sorunları aynı şekilde ele alma eğilimindedirler.
- Karakteristik hatalar
  - Farklı takımlar aynı hataları yapar. Bir uygulamanın bazı bölümleri diğerlerinden daha zordur, bu nedenle tüm ekipler aynı yerde hata yapma eğilimindedir;
  - Şartname hataları;
  - Spesifikasyonda bir hata varsa, bu tüm uygulamalara yansıtılır;
  - Bu, birden çok özellik gösterimi kullanılarak bir dereceye kadar ele alınabilir

# Şartname Bağımlılığı



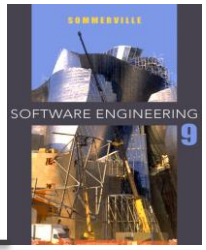

- Yazılım yedekliliğine yönelik her iki yaklaşım da spesifikasyon hatalarına açıktır. Spesifikasyon yanlışsa, sistem başarısız olabilir
- Bu aynı zamanda donanımla ilgili bir sorundur, ancak yazılım özellikleri genellikle donanım özelliklerinden daha karmaşıktır ve doğrulanması daha zordur.
- Bu, bazı durumlarda aynı kullanıcı spesifikasyonundan ayrı yazılım spesifikasyonları geliştirilerek ele alınmıştır.

# Pratikte İyileştirmeler



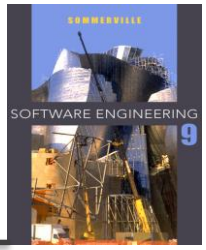

- Prensip olarak, çeşitlilik ve bağımsızlık elde edilebilirse, çok sürümlü programlama, güvenilirlik ve kullanılabilirlik açısından çok önemli gelişmelere yol açar.
- Uygulamada, gözlemlenen gelişmeler çok daha az önemlidir, ancak yaklaşım 5 ila 9 kat arasında güvenilirlik iyileştirmelerine yol açmaktadır.
- Kilit soru, bu tür iyileştirmelerin çok sürümlü programlama için önemli ekstra geliştirme maliyetlerine değip değmeyeceğidir.

# Güvenilebilir Programlama




- Program hatalarının görülme sıklığını azaltmaya yardımcı olan iyi programlama uygulamaları benimsenebilir.
- Bu programlama uygulamaları şunları destekler:
  - Hata önleme
  - Arıza tespiti
  - Hata toleransı

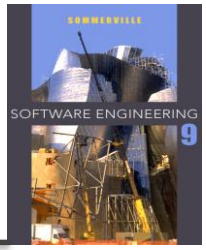
# Güvenilir Programlama İçin İyi Uygulama Yönergeleri




**Güvenilir programlama yönergeleri**

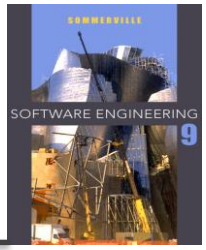
1. Bir programdaki bilgilerin görünürlüğünü sınırlayın
2. Tüm girdilerin geçerliliğini kontrol edin
3. Tüm istisnalar için bir işleyici sağlayın
4. Hataya açık yapıların kullanımını en aza indirin
5. Yeniden başlatma yetenekleri sağlayın
6. Dizi sınırlarını kontrol edin
7. Harici bileşenleri ararken zaman aşımlarını dahil edin
8. Gerçek dünya değerlerini temsil eden tüm sabitleri adlandırın

# Bir Programdaki Bilgilerin Görünürlüğünü Kontrol Edin



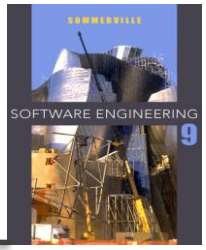

- Program bileşenlerinin yalnızca uygulanmaları için ihtiyaç duydukları verilere erişmesine izin verilmelidir.
- Bu, program durumunun bazı bölümlerinin bu bileşenler tarafından yanlışlıkla bozulmasının imkansız olduğu anlamına gelir.
- Veri sunumunun özel olduğu soyut veri türlerini kullanarak görünürlüğü kontrol edebilirsiniz ve verilere yalnızca get () ve put () gibi önceden tanımlanmış işlemlerle erişime izin verirsiniz.

# Geçerlilik İçin Tüm Girişleri Kontrol Edin



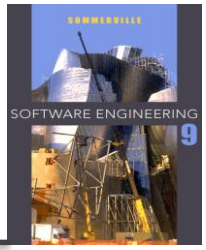

- Tüm programlar çevrelerinden girdi alır ve bu girdiler hakkında varsayımlarda bulunur.
- Bununla birlikte, program özellikleri, bir girdi bu varsayımlarla tutarlı değilse ne yapılacağını nadiren tanımlar.
- Sonuç olarak, birçok program olağandışı girdilerle sunulduğunda öngörülemez şekilde davranır ve bazen bunlar sistemin güvenliğine yönelik tehditlerdir.
- Sonuç olarak, bu girdiler hakkında yapılan varsayımlara göre işlemeden önce her zaman girdileri kontrol etmelisiniz.

# Geçerlilik Kontrolleri



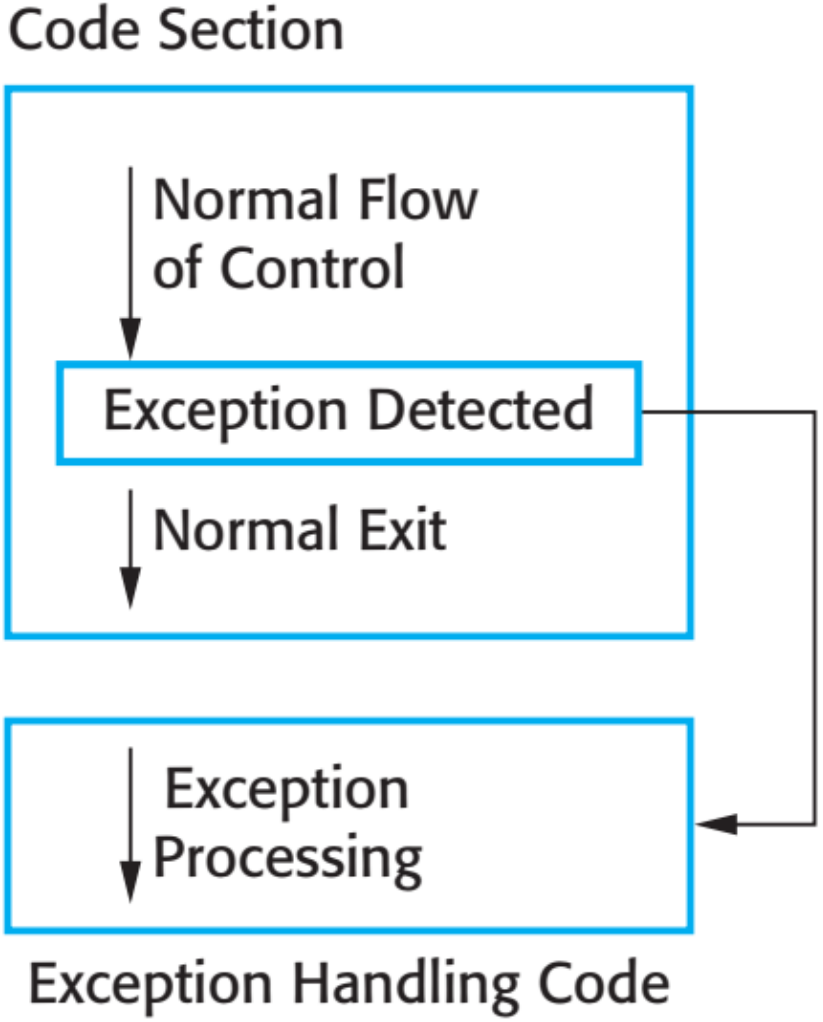

- Aralık kontrolleri
  - Girişin bilinen bir aralıkta olup olmadığını kontrol edin.
- Boyut kontrolleri
  - Girişin bazı maksimum boyutu aşmadığını kontrol edin, örneğin bir ad için 40 karakter.
- Temsil kontrolleri
  - Girişin temsilinin bir parçası olmaması gereken karakterler içermediğini kontrol edin, örneğin isimler rakam içermiyor.
- Makulluk kontrolleri
  - Aşırı bir değerden ziyade makul olup olmadığını kontrol etmek için girdi hakkındaki bilgileri kullanın.

# Tüm İstisnalar İçin Bir İşleyici Sağlayın

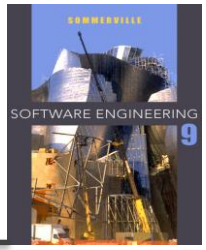



- Program istisnası, bir hata veya elektrik kesintisi gibi bazı beklenmedik olaylardır.
- İstisna işleme yapıları, istisnaları tespit etmek için sürekli durum kontrolüne gerek kalmadan bu tür olayların ele alınmasına izin verir.
- İstisnaları tespit etmek için normal kontrol yapılarını kullanmak, programa eklenecek birçok ek ifadeyi gerektirir. Bu, önemli bir ek yük ekler ve potansiyel olarak hataya açıktır.

# İstisna İşleme

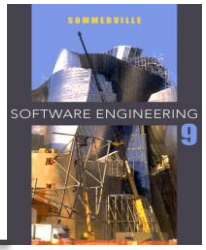


# İstisna İşleme



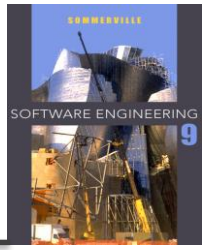

- Üç olası istisna işleme stratejisi
  - Çağıran bir bileşene bir istisnanın meydana geldiğini bildirir ve istisna türü hakkında bilgi sağlar.
  - İstisnanın meydana geldiği işleme bazı alternatif işlemler gerçekleştirin. Bu, yalnızca istisna işleyicinin ortaya çıkan sorundan kurtulmak için yeterli bilgiye sahip olması durumunda mümkündür.
  - İstisnayı işlemek için denetimi bir çalışma zamanı destek sistemine geçirin.
- İstisna işleme, bazı hata toleransı sağlayan bir mekanizmadır

# Hataya Açık Yapıların Kullanımını En Aza İndirin



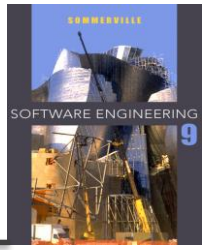

- Program hataları genellikle insan hatasının bir sonucudur çünkü programcılar sistemin farklı bölümleri arasındaki ilişkilerin izini kaybeder.
- Bu, doğal olarak karmaşık olan veya yapabildiklerinde hataları kontrol etmeyen programlama dillerindeki hataya açık yapılarla daha da kötüleşir.
- Bu nedenle, programlama yaparken, bu hataya açık yapılardan kaçınmaya veya en azından kullanımını en aza indirmeye çalışmalısınız.

# Hataya Açık Yapılar



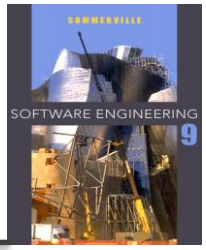

- Koşulsuz dal (goto) ifadeleri
- Kayan nokta sayıları
  - Doğası gereği belirsiz. Belirsizlik, geçersiz karşılaştırmalara yol açabilir.
- İşaretçiler
  - Yanlış bellek alanlarına atıfta bulunan işaretçiler verileri bozabilir. Takma ad, programların anlaşılmasını ve değiştirilmesini zorlaştırabilir.
- Dinamik bellek ayırma
  - Çalışma zamanı tahsisi bellek taşmasına neden olabilir.

# Hataya Açık Yapılar



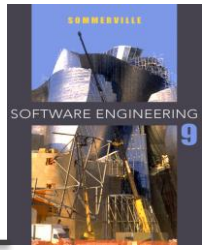

- Paralellik
  - Paralel süreçler arasındaki öngörülemeyen etkileşim nedeniyle ince zamanlama hatalarına neden olabilir.
- Özyineleme
  - Özyinelemedeki hatalar, program yığını doldukça bellek taşmasına neden olabilir.
- Kesmeler
  - Kesintiler, kritik bir işlemin sonlandırılmasına ve bir programın anlaşılmasını zorlaştırmasına neden olabilir.
- Miras
  - Kod yerelleştirilmedi. Bu, değişiklikler yapıldığında beklenmeyen davranışlara ve kodu anlamada sorunlara neden olabilir

# Hataya Açık Yapılar



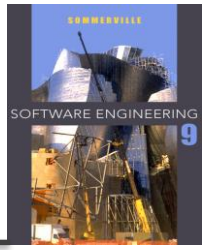

- Aliasing
  - Aynı durum değişkenine atıfta bulunmak için 1'den fazla ad kullanmak.
- Sınırsız diziler
  - Dizilerde bağlı denetim yoksa arabellek taşması hataları meydana gelebilir.
- Varsayılan giriş işleme
  - Girişten bağımsız olarak gerçekleşen bir giriş eylemi.
  - Varsayılan eylem denetimi programın başka bir yerine aktarmaksa, bu sorunlara neden olabilir. Yanlış veya kasıtlı olarak kötü niyetli giriş, bir program arızasını tetikleyebilir.

# Yeniden Başlatma Yetenekleri Sağlayın



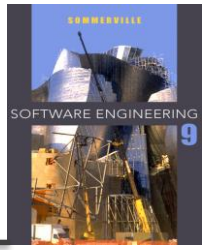

- Uzun işlemler veya kullanıcı etkileşimleri içeren sistemler için, her zaman, kullanıcıların yaptıkları her şeyi yeniden yapmak zorunda kalmadan sistemin arızadan sonra yeniden başlamasına izin veren bir yeniden başlatma özelliği sağlamalısınız.
- Yeniden başlatma, sistemin türüne bağlıdır
  - Formların kopyalarını saklayın, böylece kullanıcılar bir sorun olduğunda tekrar doldurmak zorunda kalmazlar.
  - Durumu periyodik olarak kaydedin ve kaydedilmiş durumdan yeniden başlatın

# Dizi Sınırlarını Kontrol Edin



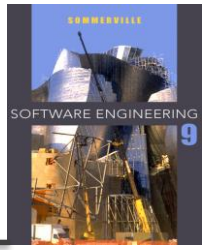

- C gibi bazı programlama dillerinde, bir dizi bildiriminde izin verilen aralığın dışındaki bir bellek konumunu adreslemek mümkündür.
- Bu, saldırganların bir dizideki en üst öğenin ötesine kasıtlı olarak yazarak yürütülebilir kodu belleğe yazdığı, iyi bilinen 'sınırlı arabellek' güvenlik açığına yol açar.
- Diliniz bağlı denetim içermiyorsa, bu nedenle her zaman bir dizi erişiminin dizinin sınırları içinde olup olmadığını kontrol etmelisiniz.

# Harici Bileşenleri Ararken Zaman Aşımlarını Dahil Et



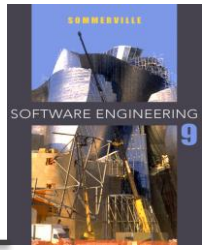

- Dağıtılmış bir sistemde, uzak bir bilgisayarın arızası 'sessiz' olabilir, böylece o bilgisayardan bir hizmet bekleyen programlar bu hizmeti veya bir arıza olduğuna dair herhangi bir göstergeyi asla alamaz.
- Bundan kaçınmak için, harici bileşenlere yapılan tüm aramalara her zaman zaman aşımı eklemelisiniz.
- Tanımlanmış bir süre yanıt verilmeden geçtikten sonra, sisteminiz arızayı varsaymalı ve bundan kurtulmak için gereken her türlü işlemi yapmalıdır.

# Gerçek Dünya Değerlerini Temsil Eden Tüm Sabitleri Adlandırın




- Sayısal değerlerini kullanmak yerine her zaman gerçek dünya değerlerini (vergi oranları gibi) yansıtan sabitler verin ve bunlara her zaman isimleriyle atıfta bulunun.
- Değer yerine ad kullanırken hata yapma ve yanlış değeri yazma olasılığınız daha düşüktür.
- Bu, bu 'sabitler' değiştiğinde (elbette, gerçekten sabit değiller), o zaman değişikliği programınızda yalnızca bir yerde yapmanız gerektiği anlamına gelir.

# Bölüm 2'nin Anahtar Noktaları




- Yazılım çeşitliliğine ulaşmak zordur çünkü yazılımın her sürümünün gerçekten bağımsız olmasını sağlamak neredeyse imkansızdır.
- Güvenilir programlama, girdilerin geçerliliğini ve program değişkenlerinin değerlerini kontrol etmek için bir programa fazlalığın dahil edilmesine dayanır.
- Goto ifadeleri, işaretçiler, özyineleme, kalıtım ve kayan nokta sayıları gibi bazı programlama yapıları ve teknikleri, doğası gereği hataya açıktır. Güvenilir sistemler geliştirirken bu yapılardan kaçınmaya çalışmalısınız.