BİÇİMSEL YÖNTEMLER

Biçimsel yöntemler, hem yazılım hem de elektronik donanımdaki sistemlerin belirtilmesi, doğrulanması ve hatasız olarak geliştirilmesi için matematiksel modellemelerin kullanılmasıdır.

Neden kullanılır? Avantajları ve dezavantajları nelerdir?

Avantajlar

* Sistem özelliklerinin belirsizliğini gidermeye ve örtük varsayımları ifade etmeye yardımcı olurlar
* Sistem gereksinimlerindeki kusurları ortaya çıkarırlar ve titizlikleri sorunun daha iyi anlaşılmasını sağlar
* Biçimsel bir dil kullandıkları için, birçok meslektaş verileri, özellikleri bağımsız olarak doğrulayabilir ve böylece geliştirme sürecinin başlarında hataları çözebilir

Dezavantajlar

* Küçük matematiksel gösterimlerle İngilizce dilinden tamamen resmi dil gösterimleri ve kurallarına kadar çeşitli seviyelerde karmaşıklık alabilir.
* Sistem işlevselliğinin belirtilmesi ve modüllerin açıklanmasında ayrık matematik kullanılabildiği için yetersiz kalabileceğinden dolayı sistem ve donanım kısıtlamalarını dışarıda bırakabilir.
* Bu nedenle standart kalite güvence yöntemlerinin yerini alamazlar.

Bu nedenle sistemde sadece tamamlayıcı tekniktirler.

-Peki, bariz bir şekilde işlevsel olarak karmaşık bir problem içeren durumlarda neden biçimsel yöntemlere başvuralım?

Cevap: Biçimli metotların devi olan Forster tarafından veriliyor:

=> “Yirminci yüzyıl mantığının en büyük iç görülerinden biri şuydu:

Formüllerin taşıdıkları anlamları nasıl taşıyabileceklerini anlamak için önce onları tüm bu anlamlardan arındırın ki sembolleri kendileri olarak görebilelim...

O zaman onlara sistematik bir şekilde anlamlar yükleyebiliriz…

Bu, ne tür anlamların olduğuna dair teoremleri kanıtlamayı mümkün kılar.

Bu, sembollerden oluşturulmuş diller tarafından karşılanabilir.

Biçimsel Yöntem Türleri

1-) Model Tabanlı Diller

Model tabanlı diller, sistemin davranışını belirlemek için sistemin matematiksel bir modelini oluşturur.

* Model, temel bir durumdan (veri) ve bu durum üzerindeki bir işlemler yığınından oluşur.
* Durum modeli, ilişkiler, kümeler, diziler ve fonksiyonlar gibi matematiksel varlıklar yardımıyla oluşturulur.
* İşlemler, ön ve son koşullar açısından verilen yüklemlerle de tanımlanır.

Model tabanlı diller geliştirmek için en yaygın kullanılan gösterimler Viyana Geliştirme Metodu (VDM), Zed (Z) ve B'dir.

2-) Cebirsel Gösterim

 Soyut cebirden(vektör uzayı, modüller, alanlar, halkalar) türetilen yöntemleri kullanarak sistem davranışını belirtmek için kullanılır.

* Soyut veri türlerinin ve arayüzün tanımlanması için tasarlanmıştır
* En yaygın kullanılan gösterimler LARCH, ASL ve OBJ'dir.

3-) Süreç Odaklı Gösterim

Temel olarak üstü kapalı bir model oluşturarak eşzamanlı sistemi tanımlamak için kullanılır.

* Bu dillerde süreçler ifadelerle belirtilir ve temel ifadelerin yardımıyla oluşturulur.
* Temel ifadeler, daha karmaşık işlemler elde etmek için işlemleri birleştiren işlemlerle birlikte basit işlemleri tanımlar.
* En yaygın olarak kullanılan süreç odaklı dil CSP ve CCS dir.

Yazılım Yaşam Döngüsünde Biçimsel Diller

Yaşam döngüsünde biçimsel diller gereksinim analizi ve test aşamalarında kullanılır.

Gereksinim Analizinde:

* Z, VDM ve Larch sıralı sistemlerin davranışını belirlemek için kullanılırken, CSP, CCS, Durum çizelgeleri, Temporal Logic, Lamport ve I/O otomatları gibi diğer biçimsel yöntemler, eşzamanlı sistemlerin davranışını belirlemeye odaklanır . RAISE, zengin durum alanlarını işlemek için kullanılır ve LOTOS, eşzamanlılık nedeniyle karmaşıklığı ele alan dillerden biridir.

Test Aşamasında:

1. Model kontrolünde, sistemin sonlu durum modeli oluşturulur ve durum uzayı mekanik olarak incelenir.

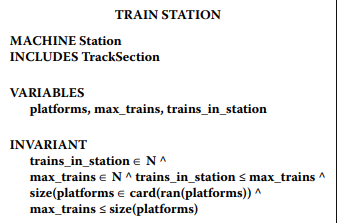
* Yaygın olarak kullanılar eşdeğer model denetleyicisi NuSMV ve SPIN’dir

1. Sistemin bir modeli matematiksel bir dilde tanımlanır ve modelin istenen özellikleri bir teorem ispatlayıcısı ile ispatlanabilir.  Bir teorem ispatlayıcısı tarafından kontrol edilecek belirtim, matematiksel bir gösterimle yazılır. Z bunun iyi bilinen bir örneğidir.

Örnek:

B Tren İstasyonu’nun Biçimlendirilmesi

Şekil 6.1 -) Tren istasyonu için kurulum özellikleri.



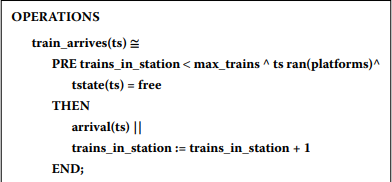
Şekil 6.2 -) Tren İstasyonu için başlatma şartı



Şekil 6.3 -) İstasyon için tren varış davranışı.

Tren İstasyondan hareket etmeden önce yapılacak kontroller;

İstasyonda tren için yer var mı? Platform hazır mı?

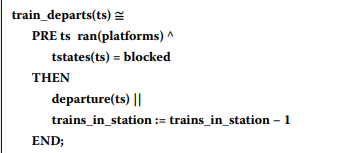


Eğer şartlar sağlanıyorsa, tren geliyor olarak işaretlensin ve istasyondaki tren sayısı artsın.

Şekil 6.4 -) İstasyon için tren kalkış davranışı.

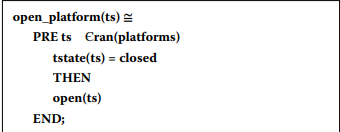
Tren platform sırasında ve hareket etmiyorsa;

Platformdaki tren yola çıkmış olarak işaretlenir ve platformdaki tren sayısı 1 eksilir.



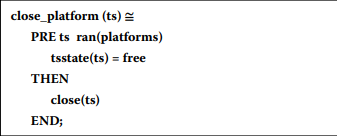
Şekil 6.5 -) İstasyon için platform açma davranışı.

Platform istasyon için diğer platformlarla ardışık düzendeyse ve mevcut platform kapalıysa; platform işaretlenir ve açılır.



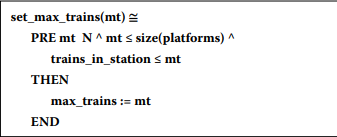
Şekil 6.6 -) İstasyon için platform kapatma davranışı.

* Platform istasyon için diğer platformlarla ardışık düzendeyse ve mevcut platform açıksa; platformu işaretlenir ve kapatılır.



Şekil 7.7 -) Bir tren istasyonu için maksimum tren sayısını belirleme davranışı.

* Maksimum tren sayısı doğal sayı olmalı ve platformların sayısını geçmemeli.
* Bir istasyon için maksimum tren sayısı, mevcut tren sayısından az olamaz.



* Sonuç olarak matematiksel gösterim, doğal dil versiyonundan daha kesindir. Fakat daha az ayrıntılıdır.
* B İstasyonu için yapılan biçimsel gösterim C++ veya Ada gibi proglamlama dil kodlarına dönüştürülebilir.

***MurΦ Kullanarak Uzay Mekiği Uçuş Yazılımının Resmileştirilmesi***

Bir sonraki resmileştirme örneği, prototip doğrulama sistemi (PVS)

dilini, uzay mekiği uçuş yazılımının çeşitli yönlerini modellemek ve kanıtlamak için kullanır.

aşağıdaki örnek, Uzay mekiği işlevselliği için PVS'de başlatma kodudur.

Kaynak: (Crow, J., and Di Vita, B., ACM Trans. Softw. Eng. Methodol., 7: 296–332, 1998.)

Crow ve Di Vita (1998) sistemine göre; PVS durum yönelimli bir dildir. Demek ki davranışı yönlendirmek için sonlu durum makinesi tasarlar.

Bir sonlu durum makinesi modeli (M), Kabul edilebilir durumlardan oluşan küme (S), bir girdiler seti (I), bir çıktılar seti (O) ve bir durum geçiş fonksiyonu, şu denklem yoluyla açıklanıyor;

M : I × S → [O × S]

Başlangıç ve son durumların da tanımlanması gerekir.

PVS'deki uygulama kodunun bir parçası yukarıda gösterilmiştir. Kodu yorumlamadan, sonlu durum makinesini modellemedeki anlamlılığına dikkat çekmek ilgi çekicidir.

Burada ***Principal\_function (Asıl İşlev)***, davranışı yönlendirmek için gereken girdileri (pf\_inputs) ve durumları (pf\_state) içeren bir arabirim tanımlar. Çıktı ise bir çıktı ifadesi ve makinedeki sonraki durumdur.

Davranış belirtimi, sonsuz bir durum geçişleri (fonksiyonel dönüşümler)

dizisi olarak ve her durumla (burada küçük kontrollü jet roketlerinin (vernier roketler) ateşlenmesiyle ilgili durum ile) ilişkili bir dizi fonksiyon olarak devam eder. PVS'nin sözdizimini (C, C++ veya Java'ya benzediği için)

tamamıyla anlamadan bile Sistemin bu davranışının nasıl açıkça tanımlanabileceğini görebileceğinizi ummaktayız.

Açıklık ve ekonomiye ek olarak, davranışı tanımlamak için PVS kullanmanın bir başka avantajı, etkileşimli bir ispat denetleyicisi ve Stanford Üniversitesi'nin MurΦ'sinin ("Murphy“ (murfi) olarak telaffuz edilir) teorem ispatı (specification verification) (spesifikasyon doğrulaması) için kullanılabilmesidir.

***Kategori Teorisi Kullanılarak Bir Enerji Yönetim Sisteminin Resmileştirilmesi***

Kategori teorisi (CT), işlevsel gösterimin soyut nesneleri modellemek için kullanılmasına izin verir ve bu nedenle bu nesnelerin biçimsel olarak belirtilmesinde yararlıdır.

Aşağıdaki örnek, Uzay Mekiği Sistemindeki sürmeli roketlerin davranışı;

Kaynak: (Crow, J., and Di Vita, B., ACM Trans. Softw. Eng. Methodol., 7, 1998.)

finite\_sequences[n: posnat, t: TYPE]: THEORY BEGIN

finite\_seq: TYPE = [below[n] -> t]

END finite\_sequences

requirements: THEORY

BEGIN

IMPORTING finite\_sequences

.. .

jet: TYPE

rot\_jet: TYPE FROM jet

jet\_bound: posint

jet\_count: [rot\_jet -> below[jet\_bound]]

jet\_count\_ax: AXIOM injective?(jet\_count)

finite\_number\_of\_jets: LEMMA .. .

vernier?: pred[rot\_jet]

vernier\_jet: TYPE = (vernier?)

there\_is\_a\_vernier: AXIOM (EXISTS j: vernier?(j)

primary\_rot?(j): bool = NOT vernier?(j)

primary\_rot\_jet: TYPE = (primary\_rot?)

there\_is\_a\_primary: AXIOM

(EXISTS j: primary\_rot?(j))

downfiring?: pred[vernier\_jet]

.. .

END requirements

Bu biçimselleştirme, iyi tasarımı kolaylaştırır ve kullanıcı arayüzü tasarımına yardımcı olabilir. Ve sistem kullanıcılarının tasarım kararlarını etkileyen kısıtlamaları öğrenirken kendi alan bilgilerini geliştirmelerine izin verir. Bu tasarım kararları daha sonra kurallar gereksinimlerden tasarıma ve kullanıcı arayüzü işlevselliğine kadar uygulandıkça belirginleşir. Son olarak, tasarımcı ve programcılar için bakım ve sorun giderme kolaylığı için, CT kullanımı bir hatanın nerede bulunabileceğini gösterebilirken, işlev grubunun giriş bazında analiz edilmesine ve sorun gidermeye izin verir.

Aşağıda, yazılım spesifikasyonu için CT'ye kısa bir giriş yer almaktadır;

A, B ve C gibi soyut nesne sınıflarımız olduğunu varsayalım. Ayrıca, f ve g fonksiyonlarının olduğunu, öyle ki f : A → B ve g : B → C olduğunu varsayalım. Bir kategori, kategorinin nesnelerinden ve kategori morfizmalarından oluşur. g ◦ f'nin f ve g'nin bileşimi olduğu aşağıdaki örnekte temel bir fonksiyonel ilişki görülebilir;

CT, yapıları ve zaman içindeki dönüşümlerini modellemek için yaygın olarak kullanılır (sonlu veya sonsuz fonksiyonel bileşim dizileriyle temsil edilir). Örneğin, yukarıda verilen son örnekte f ve g arasındaki birleştirilmiş ilişkilerin sırası f, fg, fgf vb. ile verilecektir.

Bu diziler ayrıca kategorileri temsil eder. Örneğin, PVS gösterimini kullanarak sonsuz bir durum geçişleri dizisi olarak temsil edilen uzay mekiği davranışsal belirtimini az önce gördük.

Bir sistemin bileşimi, daha büyük bir sistemin modülerliğini ve ara bağlantılarını modellemek için de uygundur. Harika bir resmileştirme ve test etme aracı olan bir doğruluk tablosuna kıyasla, CT, ilgili öğeleri ilişkilendirebilir ve ardından her birini daha düşük düzeyde incelenebilecek bir şeye ayrıştırabilir. Örneğin, iş mantığı, sezgisel veya mimari olarak bir kategoriye yerleştirilebilir.

***Örnek: Bir Enerji Yönetim Sistemi***

Tanımlamakta olduğumuz örnek, bir elektrik şirketinin enerji yönetim sistemini (EMS) kısmi olarak resmileştirmesidir. Herhangi bir enerji üretim kuruluşu, yüksek düzeyde otomasyon, hata toleransı ve yedeklilik içeren karmaşık, gerçek zamanlı bir sistemle uğraşmak zorundadır.

Bir EMS'de bir açık erişim ağ geçidi (OAG), bilgisayar sistemleri ve uzak terminal birimleri (RTU'lar) arasındaki birçok iletişim protokolü için bir iletişim bağlantısı görevi görür. OAG ayrıca, şirket içi iletişim protokolü (ICCP) aracılığıyla tesis dijital arabirimi (PDI) ile ve aşağıda olan şekil 1’de gösterilen ara bağlantı şeması aracılığıyla diğer kuruluşlarla iletişim kurmak için kullanılır.

Şekil 1'i CT kullanarak resmileştirmek için, önce varlıkları Şekil 2‘de gösterilen stenografi sembollerini kullanarak çeviriyoruz.

Ortaya çıkan yeniden etiketlenmiş sistem Şekil 3'te gösterilmektedir. Şekil 3'e işlevsel gösterimler eklenmiştir, ancak bunların daha fazla tanımlanması gerekir.

Şekil 3'teki resmileştirmenin küçük bir bölümünü tartışalım. EMS A ve EMS B ile başlıyoruz. A1 = EMS A ve B1 = EMS B ise, f1'in EMS A ve EMS B arasında CfgCtrl ve CfgMoni arasındaki iletişimi tanımlayan çift yönlü bir eşleme olduğunu varsayalım. . Eşlemenin bir tarafını alarak, f1'imiz olduğunu varsayalım: A1 → B1. Daha sonra, f1'in tüm A1'de ve f1 ∈ B1'in tüm değerlerinde tanımlı olduğunu iddia ederiz. Yani, aralık(f1) ⊆ B1.

Devam edersek, EMS A ve EMS B arasında g1 adlı başka bir çift yönlü eşleme olduğunu varsayalım. g1 : A1 → D1'in bir tarafını tanımlarken, birleşik g1 ∘ f1 : A1 → C1 ∩ D1 fonksiyonuna sahibiz, yani (g1 ∘ f1) (a1) = g1(f1(a1)) ve a1 ∈ A1.

Oluşturulan fonksiyon, aşağıdaki gibi birleştiricidir: j : C1 → A2 ve j ∘ g1 ve g1 ∘ f1 oluşturan başka bir fonksiyon varsa, o zaman j ∘ g1 ∘ f1 ve j ∘ (g1 ∘ f1)'i karşılaştırabiliriz.

Şekil 3'te gösterildiği gibi, bu iki fonksiyonun her zaman aynı olduğu ortaya çıkmaktadır.

Bu noktada, hem EMS A hem de B sunucularında C1 ve D1'in CfgMoni olduğu A1 ∩ B1 ⊂ C1 ∪ D1'den gelen tüm giriş ve çıkışların kümesine dikkat edilmelidir. C1 ∩ D1 ⊂ A1 ∪ B1'den gelen girdiler ve çıktılar seti, burada A1 ve B1, EMS A ve B sunucularındaki CfgCtrl'dir. CfgCtrl, hem EMS hem de OAG sunucularında kendisiyle iletişim kurar. j(g) : C1 → A2 ∪ B2 işlevi, EMS CfgMoni'den OAG CfgCtrl'ye yapılandırmayı gösterir. OAG ayrıca ev sunucusunda CfgCtrl ile iletişim kurar, yani g1(g) : C1 → A1 işlevi vardır. Bu özellik, CfgCtrl'nin her yedek sunucu ile iletişim kurması nedeniyle gereklidir, böylece CfgMoni, izlemekte olduğu sunucudaki güncel durum için CfgCtrl'ye durum gönderir ve alırken kontroller verilebilir.

Bunu EMS ve OAG konfigürasyonlarıyla ilişkilendirmek için A1 seti EMS CfgCtrl A'dır, B1 seti EMS CfgCtrl B'dir, ve j(g) : (C1 ∪ D1) → (A2 ∪ B2), EMS A veya B'den gelen CfgMoni'nin OAG A veya B üzerindeki CfgCtrl ile eşleştiğini gösterir. Bu resmileştirmeye devam edebiliriz, ancak şimdiye kadar fark etmiş olmalısınız ki, bu tür bir belirtimi oluşturmak için ayrıntılara yüksek düzeyde dikkat ve titizlik gereklidir, bu da potansiyel olarak önemli arayüz tanımı ve diğer sorunlardan kaçınmaya yardımcı olur.

Örneğin, güç yönetim sistemlerinde güvenlik açığı büyük bir endişe kaynağıdır; bu, belirli kamu hizmeti şirketinin bilgisayar sistemlerine yapılan ve elektrik kesintilerine neden olan siber saldırılarda gösterilen bir gerçektir. Saldırılar, üzerinde birçok EMS'nin çalıştığı (burada açıklananlar dahil) SCADA (denetimsel kontrol ve veri toplama) sistemlerindeki güvenlik açıklarına dayanıyordu. Resmi yöntemler kullanılarak sistemin kapsamlı bir analizinin güvenlik açıklarını tanımlaması mümkündür.

**Gereksinim Doğrulama**

Gayri resmi teknikleri kullanarak gereksinimleri doğrulamayı Bölüm 5'te zaten tartışmıştık, ancak resmi teknikler, örneğin gereksinimlerin tutarlı olduğundan emin olmak için gereksinimleri test etmek için özellikle uygundur. Tutarlı bir gereksinimin yerine getirilmesinin, memnuniyeti engelleme diğini kastediyoruz.

**Doğruluk Tablolarını Kullanarak Tutarlılık Kontrolü**

Tutarlılığı resmi olarak kanıtlamanın bir yolu bir doğruluk tablosu kullanmaktır. Bu durumda yeniden yazıyoruz. Her gereksinimi bir Boole önermesi olarak. Ardından, bu gereksinimler koleksiyonunun Boole değişkenleri için bazı değer kombinasyonları olduğunu gösteriyoruz. Tüm önermelerin (gereksinimlerin) doğru olduğu gibi.Örneklemek için, evcil hayvan mağazasının bir bölümünden aşağıdaki gereksinimleri göz önünde bulundurun

POS sistemi:

1.1 Sistem yazılımı hata ayıklama modundaysa, kullanıcılarınveritabanına erişin.

1.2 Kullanıcıların veritabanı erişimi varsa, yeni kayıtları kaydedebilirler.

1.3 Kullanıcılar yeni kayıtları kaydedemiyorsa sistem hata ayıklama modunda değildir.

Gereksinimleri bir dizi Boole önermesine dönüştürüyoruz. p, q ve r olsun

Boole değişkenleri olun ve izin verin

p: "sistem yazılımı hata ayıklama modunda" ifadesi olsun

q: "kullanıcılar veritabanına erişebilir" ifadesi olsun

r: “kullanıcılar yeni kayıtlar kaydedebilir” ifadesi olsun

Açıkça, spesifikasyonlar aşağıdaki önermelere eşdeğerdir:

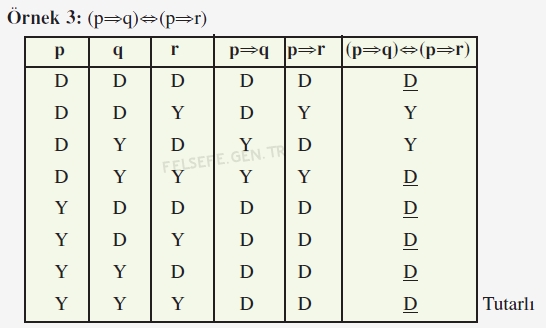
1,1 p ⇒ ¬q

1,2 q ⇒ r

1,3 ¬r ⇒¬p

Tablo 6.1 Bir Koleksiyon için Doğruluk Tablosu Tutarlılık Kanıtı

Gereksinimler



Şimdi bir doğruluk tablosu oluşturuyoruz (Tablo 6.1) ve bu satırlardan herhangi birinin olup olmadığını belirleyelim.önerme 1.1, 1.2 ve 1.3'e karşılık gelen sütunlarda tüm "T"'lere sahiptir, yani gereksinimlerin her biri karşılanmıştır. Böyle bir satır bulabilirsek, gereksinimler tutarlıdır.Üç önermenin tümünün temsil ettiği dört satır olduğunu görüyoruz.gereksinimler, Boole değerlerinin tüm kombinasyonları için geçerlidir. Bu nedenle, bu setgereksinimleri tutarlıdır.

Gereksinim 1.3'ün muhtemelen mantıklı olmadığına dikkat edin. eğer kullanıcılaryeni kayıtlar kaydedilemiyorsa, muhtemelen sistemin hatalı olduğunu söylemek amaçlanmıştır.hata ayıklama modunda (hata ayıklama modunda değil). Bu mantıksal olarak sorgulanabilir gereksinimi bir noktaya değinmek için kasıtlı olarak yerinde bıraktık - bu gereksinimler sistemi tutarlıdır, ancak mutlaka doğru değildir (müşterininamaçlanan). Bu tutarlılık kontrol süreci, niyet sorunlarını ortaya çıkarmayacaktır.(geçerlilik)—bunlar, gereksinimler gibi diğer yollarla ortaya çıkarılmalıdır.

**Muayene ile Tutarlılık Kontrolü**

Bazen, bir dizi gereksinim, tutarlılık için aşağıdakileri kullanmadan denetlenebilir:doğruluk şeması. İşte bir örnek. Bagaj için bu üç gereksinimi göz önünde bulundurunsürüş sistemi:

3.4.1 Kırmızı acil durum butonuna basılırsa sistem kapatılacaktır.hemen.

3.7.2 Sistem için ekran aydınlatması LED teknolojisini kullanacaktır.

3.11.4 Sesli alarmlar, 65 desibeli aşmayacak bir çıkışa sahip olacaktır.

Bu gereksinimler için tutarlılığı kanıtlamak için bir doğruluk tablosuna ihtiyacınız var mı? Numara,çünkü bu ifadelerin hiçbirinin mantıksal olarak bağlantılı olmadığı açık olmalıdır—hepsi bağımsız olarak doğru veya yanlış olabilir. Bu nedenle tutarlılık yoksorunlar. Bu alternatif analiz yaklaşımı, özellikle büyük bir küme varsa yararlıdır.çok büyük ve hantal bir gerçeği gerektiren - dikkate alınması gereken gereksinimlerintablo. Diğerleriyle mantıksal bağlantısı olmayan gereksinimleri atlayabilirsiniz.analizden, böylece doğruluk tablosu boyutunu azaltır.

**Tutarlılık Kontrolü Sınırlamaları**

Boolean tatmin edilebilirliğini kullanarak tutarlılık kontrolü güçlü bir araçtır. Yine de,süreç otomatikleştirilebilse de, sorun alanı çok hızlı bir şekilde büyür.Gereksinim kümesindeki n mantıksal değişken için sorun O(2n)'dir. Bu türsorun, süper bilgisayarlar için bile hızla çözülemez hale gelir - bu iyi bilinen bir sorundur.NP-tam problem, Boolean tatmin edilebilirlik problemi. bazı özel vardırBoolean tatmin edilebilirlik probleminin yapılandırılabileceği durumlarbilgisayar programları kullanılarak verimli bir şekilde çözülebilir. Öyle olsa bile, muhtemelen bunu kullanırdıkyalnızca çok kritik mantık durumları için gereksinim doğrulaması için teknik,örneğin, bir silah için fırlatma/fırlatmama karar mantığı, dozaj yönetimibir tür tıbbi cihaz için mantık veya bir nükleer santral için kapatma mantığıdır.

**Hoare Mantığı**

Hoare mantığı, basit matematiksel gösterim ve ispat tekniklerine sahiptir. Bankacılık gibi çeşitli kritik uygulamalarda gizlilik gereksinimleri test takımlarını temsil etmek ve doğrulamak için kullanılmıştır (Agrafiotis et al. 2011).

Aşağıdakilerin çoğu Gries ve Gries'te (2005) bulunan sunumu takip eder. Bir gereksinim belirtimini sistem davranışını belirleyen bir dizi mantıksal ifade olarak gördüğümüzü varsayalım. 1969 yılında C.A.R. (Tony) Hoare notasyonunu tanıttı (Hoare üçlüsü olarak adlandırılır):

P {S} Q

Burada P ve Q iddialardır (sırasıyla ön ve son koşullar) ve S bir sistem davranışsal segmentidir. O halde önkoşul, deyim, sonkoşul üçlüsü şu anlama gelir: true önkoşuluyla başlayan deyimin yürütülmesi\* sona ereceği garanti edilir ve sona erdiğinde sonkoşul doğru olur.

Amaç, S'nin yürütülmesinden önce P'nin doğru olması,ardından S'nin yürütülmesinden sonra Q'nun doğru olmasıdır. S sonlandırmazsa Q'nun yanlış olduğuna dikkat edin.

Hoare'nin mantık sistemi, verilen iddialar altında sistem segmentinin (şartname) doğru olduğunu göstermek için kullanılabilir. Örneğin, aşağıdakilere sahip olduğumuzu varsayalım:

//{x = 5} bir iddia (önkoşul)

z = x + 2;

//{z = 7} bir iddia (sonkoşul)

İspat: Diyelim ki {x = 5} doğru, sistem çalışmaya başladıktan sonra z = 5 + 2 = 7. Yani sistem kısmen doğru. Yürütülecek başka ifade olmadığından sistemin sona erdiği açıktır. Dolayısıyla doğruluğu kanıtlanmıştır. Hoare sırayla iki kural için bir çıkarım kuralı ekledi. Bunu şöyle yazıyoruz:

P {S1} Q

Q {S2} R

P {S1; S2} R

\* "Yürütme", yazılım dışı davranış segmenti durumunda, program yürütme veya sistem davranışında etkili değişiklik anlamına gelir.

Bu çıkarsama kuralı, eğer ilk sistem bölümünün (S1) son koşulu (Q), bir sonraki bölümün öncülü (ön koşulu) ise, sonraki bölümün (R) son koşulu, birleştirilmiş bölümlerin sonucudur.

Yatay çizgi “bu nedenle” veya kelimenin tam anlamıyla “sonuç olarak” anlamına gelir.

Örneklemek için, verilen iddialar kapsamında aşağıdaki spesifikasyonun doğru olduğunu gösteriyoruz:

//{x = 1 ∧ y = 2} bir iddia (önkoşul)

x = x + 1;

z = x + y;

//{z = 4} bir iddia (sonkoşul)

Kanıt: Farz edin ki {x = 1} doğrudur, sonra sistem x = 2 olan ilk komutu yürütür. Ardından, {y = 2}'nin doğru olduğunu varsayalım. Daha sonra ikinci ifadenin yürütülmesinden sonra z = 2 + 2 = 4. Dolayısıyla, son ifadenin yürütülmesinden sonra z = 4 ve son iddia doğrudur, dolayısıyla sistem kısmen doğrudur. Yürütülecek başka ifade olmadığından sistemin sona erdiği açıktır.

Koşulları işlemek için bir çıkarım kuralına ihtiyaç vardır. Bu kural şunu gösteriyor

(P ∧ koşulu){S}Q

(P ∧¬ koşulu) ⇒ Q

––––––––––––––––––––

∴ P {eğer koşul ise S}Q

Örneklemek için, verilen iddialar altında spesifikasyon bölümünün doğru olduğunu gösteriyoruz:

/{Herhangi biri} her zaman doğru olan bir iddia (önkoşul)

x > y ise

y = x;

//{x ≤ y} bir iddia (sonkoşul)

Kanıt: Bölüme girişte x ≤ y ise, if ifadesi başarısız olur ve hiçbir talimat yürütülmez ve açıkça son iddia doğru olmalıdır. Öte yandan, girişte x > y ise, y = x ifadesi yürütülür. Ardından y = x ve nihai x ≤ y iddiası doğru olmalıdır. Yürütülecek başka ifade olmadığından sistemin sona erdiği açıktır.

Başka bir çıkarım kuralı, if then-else durumlarını ele alır, yani,

(P ∧ koşulu){S1}Q

(P ∧¬ koşulu){S2}Q

––––––––––––––––––––––––––

∴ P {eğer koşul ise S1 değilse S2}Q

Örneklemek için, verilen iddialar altında spesifikasyon bölümünün doğru olduğunu gösteriyoruz:

//{Herhangi biri} her zaman doğru olan bir iddia (önkoşul)

x < 0 ise

abs = –x;

başka

abs = x;

//{abs = |x|} bir iddia (sonkoşul)

Kanıt: x < 0 ise, abs = –x ⇒ abs, pozitif x olarak atanır. x ≥ 0 ise, abs'e ayrıca x'in pozitif değeri atanır. Bu nedenle abs = |x|. Yürütülecek başka ifade olmadığından sistemin sona erdiği açıktır.

Son olarak, "while" ifadelerini işlemek için bir kurala ihtiyacımız var:

(P ∧ koşulu){S}P

––––––––––––––––––––––––––––––

∴ (P ∧ iken koşul){S}(¬koşul ∧ P)

Bu kuralı birazdan örnekle açıklayacağız.

Şimdiye kadar kanıtlanmış olan belirtim parçacıkları (bunların gerçekten "kod" parçacıkları oldukları söylenebilir), bir gereksinim belirtiminde bulunması muhtemel olmayan çok basit ama düşük düzeyli, ayrıntılı davranışı temsil eder. Yoksa durum bu mu? Kritik davranış için bu ayrıntı düzeyinde spesifikasyonlar sağlamanın gerekli olabileceğini hayal etmek o kadar zor değil. Örneğin, bir insülin pompasının (veya kontrollü radyasyon tedavisi veren makinenin) dozlama mantığı için gereken davranış, böyle bir mantığı gerektirebilir. Bir füze fırlatma, uzay istasyonundaki yaşam destek sistemini kontrol etme veya bir nükleer santrali kapatma kararı basit ama kritik bir mantığa dayanabilir. Bu mantık, tesadüfen, tamamen donanımda uygulanabilirdi.

Çalışan örneklerimiz bile resmi kanıt gerektiren kritik bir mantığa sahip olabilir. Örneğin, bagaj taşıma sistemi için bagaj sayma mantığı veya belirli bir envanter kontrol mantığı aşağıdaki davranışı gerektirebilir.

/{toplam = 0 ∧ sayım = n ≥ 0}

sayım > 0 iken

{

toplam = toplam + sayı;

say = say--;

}

//{toplam = n(x + 1)/2}

Ve bu nedenle, verilen iddialar altında spesifikasyon segmentinin doğru olduğunu göstermek gerekli olacaktır. Bunu kanıtlamak için tümevarım kullanıyoruz.

**Temel**

İddiada verildiği gibi toplam = 0 ve n = 0 olduğunu varsayalım. Döngü korumasının test edilmesinden sonra değer false olur ve sistem sonlandırılır. Bu noktada, toplamın değeri, son koşulu sağlayan sıfırdır. Sistemin sona erdiği az önce gösterildi.

**Tümevarım Hipotezi**

Program, sayım =n değeri için doğrudur. Yani, toplam = n(n + 1)/2 değerini üretir ve sistem sonlanır.

**İndüksiyon Adımı**

Önkoşulda toplam = 0 ve sayım = n+1 değerinin olduğunu varsayalım.

Döngüye girdikten sonra, toplam = n + 1 değerini atadık ve ardından sayı = n olarak ayarladık. Şimdi, tümevarım hipotezinden, say = n için toplam = n(n + 1)/2 olduğunu biliyoruz. Bu nedenle, sistemin yürütülmesine devam edildiğinde, toplam (n + 1) + n(n + 1)/2 ile sonuçlanacaktır, bu da sadece (n + 1)(n + 1 + 1)/2'dir ve sistem kalıcıdır.

Tümevarım hipotezi tarafından, döngü sayısının n'inci yinelemesi tarafından n kez azaltıldığından (sayım n'ye başlatıldığında döngüden çıkıldığından beri). Tümevarım adımında, sayım n + 1 olarak başlatıldı, bu nedenle n'inci yinelemede bir veya 1 değerine sahip olur. Böylece n + 1. yinelemeden sonra 0'a düşürülecek ve bu nedenle sistem döngüden çıkar ve sona erer.

Bu yaklaşım gerçekten doğruluğu kanıtlıyor mu? Öyle ama inanmıyorsanız, yanlış bir belirtim segmentinin doğru olduğunu kanıtlamak için Hoare mantığını kullanmayı deneyin, örneğin:

//{toplam = 0 ∧ sayım = n ≥ 0}

sayarken < n

{

toplam = toplam + sayı;

sayı = sayı++;

}

//{toplam = n(n + 1)/2}

Bu belirtim yanlıştır çünkü n toplamın dışında bırakılır ve bu nedenle doğruluk kanıtı elde edilemez.

Bir for döngüsünün while döngüsüne benzer bir ispat tekniği kullandığını göstermek kolaydır. Aslında, Böhm ve Jacopini (1966) tarafından, sadece ilk iki çıkarım kuralından doğrulama kanıtları oluşturabileceğimizi biliyoruz. Özyinelemeli prosedürler için, tümevarımı tümevarım adımında n + 1. yinelemeye uygulayan ve gerekirse güçlü tümevarım kullanan benzer, tümevarımlı, kanıtlama tekniğini kullanırız.

**Model Kontrolü**

Resmi bir belirtim verildiğinde, bir model denetleyicisi, belirli özelliklerin belirtimin teoremleri olduğunu otomatik olarak doğrulayabilir. Model denetimi, geleneksel olarak donanım tasarımlarının denetlenmesinde kullanılmıştır (örneğin, mantık diyagramlarının kullanımı yoluyla), ancak aynı zamanda yazılımla da kullanılmıştır. Model kontrolünün yazılım spesifikasyonlarını kontrol etmedeki kullanışlılığı, birleşimsel durum patlaması nedeniyle ve değişkenler Boolean olmayan değerler alabildiğinden her zaman sorunlu olmuştur. Yine de Edmund M. Clarke, E. Allen Emerson ve Joseph Sifakis, özellikle çip tasarımında model kontrolünü daha pratik hale getiren çalışmalarıyla 2007 A. M. Turing Ödülü'nü kazandı. Model kontrolünde çok sayıda ilginç ve önemli araştırma devam ediyor ve pratik uygulamalar mevcut.

**Entegre Araçlar**

Resmi belirtim araçlarını model denetleyicileri, tutarlılık denetleyicileri ve kod oluşturucularla birleştiren birkaç entegre araç seti vardır.

Örneğin, ilk olarak 1970'lerde tanıtılan, yazılım maliyetini düşürme (SCR) adı verilen gereksinim odaklı resmi bir yöntemi düşünün. SCR, sistem davranışını paydaşların anlayabileceği bir şekilde temsil etmek için bir tablo gösterimi (doğal olarak resmi) kullanır. Ardından, spesifikasyonun tutarlılığını, eksiksizliğini ve doğruluğunu kontrol etmek için bir dizi resmi araç kullanılabilir. Ek olarak, araç seti bir model denetleyicisi, bir PVS ön ucu, bir değişmez oluşturucu, bir özellik denetleyicisi ve bir test senaryosu oluşturucu içerir.

Lockheed Martin dahil birçok şirket, uzun yıllardır SCR kullanıyor. Kayda değer başarılar arasında, gömülü bir yazılım cihazının güvenlik açısından kritik bir modülünün veri ayrımını zorunlu kıldığının onaylanması ve NASA sistemleri için güvenlik açısından kritik üç yazılım modülünün gereksinimlerinin belirtilmesi sayılabilir (Heitmeyer 2007).

itirazlar, mitler ve sınırlamalar

Resmi yöntemlere, örneğin kullanımının çok zor veya pahalı olduğu, sosyalleştirilmesi zor olduğu veya çok fazla eğitim gerektirdiği gibi bir dizi “standart” itiraz vardır. Bu itirazlara yönelik çürütmeler basit, ancak durumsaldır. Örneğin, çok fazla eğitim ne kadardır? belirli kuruluşlar, belirli yetenek olgunluk modeli (CMM) seviyelerine ulaşmak için milyonlarca dolar harcarlar, ancak resmi yöntemlerin kullanımı için eğitime önemli ölçüde daha az para yatırmaktan kaçınırlar. Her durumda, naif noktalara dayanan resmi yöntemlerin elden çıkarılmasına karşı dikkatli olun. Ancak implante edilebilir bir insülin pompasına veya defibrilatöre ihtiyacınız olsaydı, ne tür gereksinimlerin doğrulanması ve onaylanmasını istersiniz? Muhtemelen resmi teknikler de dahil olmak üzere her türlü doğrulama ve doğrulamanın mevcut olmasını istersiniz. Aynı şey evinizin yakınındaki bir nükleer santral, ihtiyaç duyduğunuz radyasyon dozlama makinesi, otomobiliniz için güvenlik ve kısıtlama sistemi vb. için de geçerlidir. Akıllı çamaşır makinesi gibi nispeten sıradan ev cihazlarında bile yüksek doğruluk gereksinimleri gereklidir ve bunlar yalnızca resmi ve resmi olmayan tekniklerin bir kombinasyonu ile elde edilebilir. Resmi yöntemler her durum için değildir, ancak görev açısından kritik durumlarda gerçekten dikkate alınmaları gerekir.

İtirazlar ve Efsaneler

İtirazlar ve Mitler Biçimsel yöntemlerin kullanılmasına yönelik bazı itirazlar, hataya açık olabilecekleri (tıpkı matematiksel teorem kanıtlama veya bilgisayar programlama gibi) ve bazen bazı sistemler için gerçekçi olmadıkları gerçeğini içerir. Bu itirazlar bazen geçerlidir. Ancak resmi yöntemlerin tek başına kullanılması amaçlanmamıştır ve testin yerini almazlar. Resmi yöntemler, diğer kalite güvencesi yaklaşımlarını tamamlayıcı niteliktedir. Biçimsel yöntemlere yönelik diğer “standart” itirazlardan bazıları mitlere dayanmaktadır. Hall ve Bowen ve Hinchey'nin makaleleri bu yanlış anlamaları yakalamaya ve çürütmeye yardımcı oluyor.

İlk mit grubu aşağıdaki gibidir (Hall 1990)

* 1. Efsane: Resmi yöntemler, yazılımın mükemmel olduğunu garanti edebilir. Gerçek: Hiçbir şey yazılımın mükemmel olacağını garanti edemez. Resmi yöntemler, yazılım niteliklerini, özellikle güvenilirliği artıran birçok teknikten
* 2. Efsane: Resmi yöntemler tamamen program kanıtlama ile ilgilidir. Gerçek: Biçimsel yöntemlerin yalnızca program kanıtlamadan daha fazlasını içerdiğini ve gereksinimleri kesinlik ve tasarrufla ifade etmeyi, gereksinimleri doğrulamayı ve model kontrolünü içerdiğini gösterdik.
* 3. Efsane: Resmi yöntemler yalnızca güvenlik açısından kritik sistemler için faydalıdır. Gerçek: Resmi yöntemler, yüksek kaliteli yazılımların istendiği her yerde yararlıdır.
* 4. Efsane: Resmi yöntemler, yüksek eğitimli matematikçiler gerektirir. Gerçek: Matematik eğitimi, ifadenin nüanslarında ustalaşmada çok yardımcı olsa da, resmi yöntemleri kullanmak gerçekten bir veya başka bir resmi dili öğrenmek ve dili tam olarak nasıl kullanacağınızı öğrenmekle ilgilidir. Gereksinim mühendisliği, resmi veya başka türlü, dilin kesin kullanımına dayanmalıdır.
* 5. Efsane: Resmi yöntemler geliştirme maliyetini artırır. Gerçek: Resmi bir yöntem programının uygulanmasıyla ilgili maliyetler olsa da, bununla bağlantılı faydalar da vardır. Bu faydalar, yazılım yaşam döngüsünde - projede düzeltmeler yapmak için daha pahalı bir aşamada - gerekli yeniden çalışmanın azaltılmasını içerir.
* 6. Efsane: Resmi yöntemler kullanıcılar tarafından kabul edilemez. Gerçek: Kullanıcılar sağladıkları faydaları anlarlarsa kötü yöntemleri kabul edeceklerdir.

7. Efsane: Resmi yöntemler gerçek, büyük ölçekli sistemlerde kullanılmaz. Gerçek: Resmi yöntemler, NASA'daki yüksek profilli projeler de dahil olmak üzere çok büyük sistemlerde kullanılmaktadır

Hall'un mitlerinden beş yıl sonra, Jon Bowen ve Mike Hinchey bize Biçimsel Yöntemlere İlişkin Yedi Efsane Daha Verdiler (Bowen ve Hinchey 1995a)

1. Efsane: Resmi yöntemler geliştirme sürecini geciktirir. Gerçek: Yazılım geliştirme yaşam döngüsüne düzgün bir şekilde dahil edilirse, resmi yöntemlerin kullanılması geliştirme sürecini geciktirmeyecek, aksine hızlandıracaktır.

2. Efsane: Resmi yöntemlerde araçlar yoktur. Gerçek: Biçimsel yöntemleri destekleyen bir dizi araç olduğunu görmüştük: düzenleme araçları, çeviri araçları, derleyiciler, model denetleyicileri vb. Kullanılabilir araç seti, kullanılan resmi yönteme bağlıdır.

3. Efsane: Resmi yöntemler, geleneksel mühendislik tasarım yöntemlerinin yerini alır. Gerçek: Resmi yöntemler geleneksel tasarımı geliştirir ve zenginleştirir.

4. Efsane: Resmi yöntemler yalnızca yazılım için geçerlidir. Gerçek: Donanım sistemleri için birçok resmi yöntem geliştirildi, örneğin dijital tasarımda Petri ağlarının kullanımı.

5. Efsane: Resmi yöntemler gereksizdir. Gerçek: Bu ifade, yalnızca yüksek bütünlüklü sistemler gereksiz olduğunda doğrudur.

6. Efsane: Resmi yöntemler desteklenmez. Gerçek: Birçok kuruluş resmi yöntemleri kullanır, geliştirir ve destekler.

7. Efsane: Formel metotlar, insanlar her zaman formel metotları kullanırlar. Gerçek: “Resmi metotlar insanlar” ne gerekiyorsa onu kullanırlar. Bu genellikle resmi yöntemleri kullanmak anlamına gelir, ancak aynı zamanda resmi olmayan ve “geleneksel” teknikleri de kullanmak anlamına gelir.

* Parnas (2010), geleneksel biçimsel yöntemlere karşı bir başka zorlayıcı itirazda bulunarak, "Finansman sağlayan kuruluşlar genellikle daha büyük araştırma fonlu projelerin endüstriyel kuruluşlarla bazı işbirliğini içermesini ve 'gerçek' örnekler üzerinde bir yaklaşımın pratikliğini göstermesini şart koşar. Yazarlar bu tür çabaları rapor ettiklerinde,başarılı olduklarını belirtmektedir. Paradoksal olarak, bu tür başarı öyküleri, endüstrinin resmi yöntemleri standart prosedürler olarak benimsemedeki başarısızlığını ortaya koymaktadır; Bu yöntemlerin kullanımı rutin olsaydı, başarılı kullanımı açıklayan makaleler yayınlanmayacaktı." Analizine itiraz etmek zordur - gerçek endüstriyel deneme gücü sistemlerinde resmi yöntemlerin kullanımı hala nispeten nadirdir. Ancak Parnas'ın önerdiği panzehir,ana akım mühendisleri tarafından uyarlanması daha muhtemel olan yeni bir dizi resmi yöntem icat etmek. Bugüne kadar böyle bir panzehir bulunamadı.

Bowen ve Hinchey'in Tavsiyesi

* Biçimsel yöntemlerin kullanımına ilişkin bazı son tavsiyeler, Bowen ve Hinchey (1995b) tarafından "On Emir Biçimsel Yöntemler"den uyarlanmıştır.
* 1. Uygun gösterimi seçeceksiniz.Yorum: İster B, Z, VDM, PVS veya her neyse kullansanız, duruma en uygun resmi gösterimi seçin. Herhangi bir resmi dili veya diğerini savunmuyoruz. Bir durum için neyin uygun olduğu, büyük ölçüde kuruluşun en rahat kullandığı şeye ve mevcut destekleyici araçlara bağlı olacaktır. Bununla birlikte, bir kuruluş tipik olarak birden çok resmi dilde akıcı değildir ve bir şirketin gösterimi bir durumdan diğerine değiştirmesi önerilmez. Tercih edilen aracı kullanabilecek yeterli sayıda mühendis yoksa, resmi bir dil kullanımı eğitimi dikkate alınmalıdır.

3. Maliyetleri tahmin edeceksiniz. Yorum: Modern proje yönetimi uygulaması, yolun her adımında maliyet tahmini gerektirir.

4. Çağrı sırasında resmi bir yöntem uzmanınız olacak. Yorum: Resmi yöntemler konusunda her zaman bir uzmana erişim sağlanması tavsiye edilir. Bu uzman, kurum içi bir uzman veya danışman olabilir.

5. Geleneksel geliştirme yöntemlerinizden vazgeçmeyeceksiniz. Yorum: Biçimsel yöntemlerin doğası gereği artırıcı olduğu sürekli olarak belirtilmiştir.

6. Yeterince belgeleyeceksin. Yorum: Belgeleme, tüm sistem mühendisliği faaliyetlerinde kritik öneme sahiptir.

7. Kalite standartlarından ödün vermeyeceksiniz. Yorum: Burada herhangi bir itiraz var mı?

* 8. Dogmatik olmayacaksın. Yorum: Dogmatizm, resmi yöntemlerin faydalarını tanıtmanın yolu değildir. Resmi yöntemlere yaklaşımımızda dengeli olmaya çalıştık.
* 9. Test edecek, test edecek ve tekrar test edeceksiniz. Yorum: Resmi yöntemler testin yerini almaz, testin tamamlayıcısıdır. Modern sistem mühendisliği için uygun bir yaşam döngüsü test programı gereklidir.
* 10. Yeniden kullanacaksınız. Yorum: Biçimsel yöntemlerin gücünden yararlanın ve bunların en büyüklerinden biri yeniden kullanıma olan güvendir. Resmi olarak doğrulanmış (ve geleneksel anlamda test edilmiş) bir yazılım modülü, güvenilir olduğu için yeniden kullanım için iyi bir aday olacaktır.
* Son olarak, resmi yöntemleri zorlamaya veya aşırı kullanmaya çalışmayın. Resmi yöntemlerin o kadar yararlı olmadığı durumlarda değerlerini abartmak, direnişe ve başarısızlığa yol açabilir.(Bowen ve ark. 2010).

Son Başarı Resmi Yöntemleri Kullanma

* Gereksinim mühendisliği için resmi yöntemlerin kullanıldığı birkaç kamuya açık başarı olmuştur (yayınlanmamış başka başarılar da olabilir). İlk örnek, resmi olarak Quark'ı içerir.GMail, Facebook ile uyumlu doğrulanmış Web tarayıcısı,ve diğer iyi kullanılan sosyal medya ve ticaret sitelerinin yanı sıra Amazon. Kod Python, C/C++ ve Occam'da ve bir gereksinim belirtimini temsil eden kodla yazılmıştır. Bir Web tarayıcısı çok fazla kod içerdiğinden, yaklaşım, "tüm bileşenleri, bileşenlerin yalnızca izin verilen davranışları sergilemekle sınırlandırılmasını sağlayan küçük, hafif bir şim aracılığıyla iletişim kurmaya zorlamayı içeriyordu. Resmi altlık doğrulaması, yalnızca bir kişinin altlık hakkında akıl yürütmesini gerektirir, böylece bir prova yardımcısında büyük, karmaşık bileşenlerin doğrulanması gibi son derece pahalı veya olanaksız görevi ortadan kaldırır."

Egzersizler

* 1. Müşterilerin kendilerine resmi yöntemler anlatılıp sonra kullanılırsa kendilerini daha güvende hissetmeleri daha olası mı ?
* 2- Yazılım geliştirme süreci yaşam döngüsünün neresinde formal yöntemler en fazla faydayı sağlar ?
* 3-yazılım geliştirme sürecinde en çok sıkıntı yaşanılan kısım hangisidir ?
* 4-Yazılım Geliştirmede Güvenlik sıkıntısı azaltılması için Yönetim ekibi şartmıdır ? Şart ise neden ?
* 5-Yazılım Geliştirmede Kriterlerden ödün verilebilirmi ?