UML Software Architecture and Design Description

1997’deki girişinden bu yana, Birleşik Modelleme Dili birçok kurum ve uygulayıcıyı kendine çekti. UML şimdi yazılım geliştirme için fiili modelleme dilidir. Bazı özellikleri popülaritesini açıklamaktadır: standartlaştırılmış bir gösterimdir, ifade açısından zengindir - UML 2.0, birkaç farklı görünümün ve soyutlama seviyesinin modellenmesini sağlayan 13 diyagram türü sunar. Ayrıca, UML kalıplaşmış ve etiketlenmiş değerleri kullanarak alana özgü uzantıları destekler.

Son olarak, birkaç vaka aracı UML modellemesini kod oluşturma ve koddan tersine mühendislik modelleri gibi diğer görevlerle birleştirir. Çoğu projede, UML modelleri sistematik olarak yazılım mimarisini temsil eden ilk eserlerdir.1 Daha sonra geliştirme sürecinde değiştirilmiş ve rafine edilmişlerdir. Model odaklı mimari metodolojisinin ortaya çıkması ile önemi arttı (www.omg.org/mda). Bununla birlikte, UML'nin projelerde gerçekte nasıl kullanıldığı ve UML tabanlı geliştirmenin tuzakları hakkında çok az şey bilinmektedir. Uygulayıcıların UML'yi nasıl kullandıklarını bulmak için, yazılım mimarlarını Web tabanlı anonim bir anket doldurmaya davet ettik. Ayrıca, 14 endüstri vaka çalışmasında UML modellerini analiz ettik ve analitik sonuçları uygulayıcı raporları ile karşılaştırdık. Çalışmamız, gösterim veya dil olarak yeterliliği yerine, gerçek projelerde UML kullanımı ve model kalitesine odaklanmıştır.

**UML kUllaNIMINDaKI UygUlayICI yaNsIMalaRI**

İki aylık bir sürede 80 mimar katıldı. Arka plandaki sorular, katılımcılar arasında aşağıdaki ana sorumlulukları ortaya koydu:

■ analiz (yüzde 66),

■ tasarım (yüzde 66),

■ şartname (yüzde 61) ve

■ programlama (yüzde 52).

Yanıt verenler farklı uygulama alanlarından geldi. Çoğu bilgi sistemlerinde çalıştı (yüzde 61); Yüzde 28'i gömülü sistemlerde, bir kısmı da bir araç ve işletim sistemi geliştirmede çalışıyordu. Yüzde altmışı beş kişiden fazla projelerde çalıştı.

**Mimari görünümlerin kullanımı**

UML’nin Philippe Kruchten’in “4 + 1 Görüş Modeli” nde öne sürdüğü farklı mimari görüşlerin tanımlanmasındaki popülerliğini araştırdık. 3 Ancak, çoğu uygulayıcı için iyi bilindiği için, Rational Rose vaka aracı terminolojisini benimsedik. Şekil 1, katılımcıların kullanım durumlarında UML kullanımı, mantıksal görünümler, bileşen görünümleri, dağıtım ve senaryolar hakkındaki anket sonuçlarını göstermektedir.

**UML şartnamesine bağlılık**

UML belirtimi (www.uml.org), dil gösterimini, sözdizimini ve (enformel) semantiği tanımlar. Şekil 2, uygulayıcıların bu şartnameye uyduklarını nasıl kesin olarak düşündüklerini göstermektedir. Bu öz değerlendirmelere dayanarak, şartnameye bağlılık oldukça gevşek. Bu, UML’nin biçimsel anlambilim eksikliğinin ve uygulamasında büyük bir serbestlik derecesinin bir sonucu olabilir. Öte yandan, gayrı resmi kullanımın birçok uygulayıcı için “yeterince iyi” olabilir.

**Modelleme için durma kriterleri**

Modelleme faaliyetlerinin ne zaman biteceğini belirlemek için uygulayıcıların uyguladıkları kriterler ile ilgilendik. En belirgin kriter, uygulayıcıların yüzde 52,5'inin seçtiği tamlıktı. Bununla birlikte, bir modelin tamamlandığını veya somut bir kalite anlayışını karşıladığını doğrulamak için hiçbir objektif kriter yoktur. En öne çıkan ikinci kriter, uygulayıcıların yüzde 33,8'inin seçtiği bir inceleme veya incelemeden geçiyordu.

Programlama veya son tarih, modellemeden uygulamaya geçiş için bir kriter haline geldiğinde, genellikle sistematik bir gözden geçirme için kötü bir alternatif ve ürün kalitesi için olumsuz bir göstergedir. Bununla birlikte, uygulayıcıların yüzde 32,8'i, durdurma kriteri olarak son tarihten bahsetti - endişe verici derecede yüksek bir yüzde. Yüzde 17,5'i, durma kriteri olarak harcanan emeğin miktarını belirtti.

Katılımcı demografik özellikleri durdurma ölçütleri sorusuyla ilişkilendirirken, ölçütlerin farklı proje boyutlarına göre değiştiğini gördük. En çarpıcı olan, son tarih ile bütünlük arasındaki karşılaştırma idi. Şekil 3, son tarihin daha küçük projelerde daha durma kriteri olduğunu gösterirken, 99 kişiden daha büyük olan tüm projeler, durma kriteri olarak eksiksizlik bildirmiştir. Bunun nedeni, hem üründe hem de projede artan karmaşıklığın aynı şekilde sistematik bir yaklaşıma olan ihtiyacı arttırmasıdır.

**Eksik modellerle ilgili sorunlar**

Uygulayıcıların modellemeyi durdurmaya karar vermede birincil kriter olarak eksiksiz olduklarını bildirmeleri nedeniyle, tam bir model olmadan bir sonraki proje aşamasına geçmenin etkilerini araştırdık.

Şekil 4, tamamlanmamış modellerin bir sonucu olarak karşılaşılan dört problemi göstermektedir. Yarısından fazlası, proje eksik modeller ile ilerlerken proje üyeleriyle paydaşlar arasında bir iletişimsizlik olduğunu bildirdi. Ayrıca, uygulanan ürünün kalitesizliğinden de bahsettiler; belirtilen şartlara uymamak anlamında verilen “yanlış” bir ürün; ve yüksek miktarda test etme çabası.

**UML açıklamaları ile ilgili sorunlar**

Hem anketlerde hem de ek görüşmelerde, mimarlar karşılaştıkları sorunları UML gibi çok diyagramlı bir gösterimde kullandıklarını belirtti. Cevaplarına dayanarak dört ana problem sınıfı belirledik:

■ Dağınık bilgi. Tasarım seçenekleri çoklu görünümlere dağılmış durumda. Örneğin, bazı bağımlılıklar mantıksal görünümde, bazıları ise işlem görünümünde görünebilir.

■ Eksiklik. Birçok proje bilerek mimari görüşlerin sadece bir alt kümesini tamamlar. Mimarlar, önemli olduğunu düşündükleri şeye odaklanır.

■ Orantısızlık. Mimarlar, daha karmaşık olduğunu düşündükleri sistem parçaları için daha fazla ayrıntı bulabilir. Artan ayrıntı tutarlı bir şekilde artan kritiklik veya karmaşıklık gösteriyorsa, uygulayıcılar bu bilgiyi test kaynaklarını tahsis etmek için kullanabilirler.

■ Tutarsızlık. UML tabanlı yazılım geliştirme kaçınılmaz olarak tutarsızdır. Endüstriyel sistemler genellikle ekipler tarafından geliştirilir. Farklı ekipler, sistemin farklı anlayışlarına ve farklı modelleme stillerine sahip olabilir ve bu da tutarsız modellere yol açabilir. Bazı UML araçları, diyagramlar arasındaki tutarlılık konusunda temel kontroller sağlasa da, bu kontrollerin kapsamı sınırlıdır ve tutarsızlıkların görünümler arasında kalması için çok fazla alan bırakmaktadır.

Eksiklik, orantısızlık ve tutarsızlık, yazılım mimarisi modellerinde kaynak kodundan çok daha fazla ortaya çıkmaktadır. Bunun nedeni kaynak kodunun form için resmi kriterleri (gramer) ve bu kriterleri kontrol etmek için araçları (derleyicileri) içermesidir.

Diğer problem sınıfları aşağıdakileri içerir:

■ Diyagram kalitesi. UML, mimarilerin bir tasarımı farklı şekillerde temsil etmelerini sağlar. Örneğin, çok fazla eleman içeren bir diyagramı birkaç küçük diyagrama ayırabilirler. Diyagramları düzenlemenin farklı yolları asıl tasarımı değiştirmese de, modelin ne kadar kolay anlaşıldığını ve nasıl yorumlandığını etkileyebilir.

■ Gayri resmi kullanım. Mimarlar bazen UML'yi çok kabataslak bir şekilde kullanırlar. Örneğin, bir sistemi daha iyi anlamak veya mimariyi açıklamak için genel çizim yazılımı kullanabilir, hatta kağıt üzerinde eskizler yapabilirler. Bu diyagramlar, resmi UML sözdiziminden saparak anlamlarını belirsizleştirir.

■ Modelleme sözleşmelerinin eksikliği. Örnek olaylarımız, mühendislerin bireysel alışkanlıklarına göre UML kullandıklarını göstermektedir. Bu alışkanlıklar, yerleşim kuralları, yorum yapma, yöntem ve işlemlerin görünürlüğü ve diyagramlar arasındaki tutarlılığı içerebilir. Programlamada, kodlama standartları kalite güvence tekniklerinin standart repertuarına aittir. Benzer şekilde, mimarlık aşamasındaki “modelleme standartları” daha düzenli UML kullanımı oluşturulmasına yardımcı olacaktır.

UML'nin farklı kullanımları doğal olarak ortaya çıkar ve tasarım kararları ve tasarım süreci boyunca hem miktar hem de karmaşıklıkta detaylar artar. İlk fikirleri yakalamak için bir çizim yeterli olabilir, oysa önemli detaylar uygulamaların kaynak kod üretmek için kullanacağı bir sistem mimarisine girer. Martin Fowler, üç tip UML kullanımını ayırt eder: bir taslak, bir taslak ve bir programlama dili olarak.

UML'nin bu kadar geniş bir amaca hizmet etmesini sağlayan genellik ve özgürlük de zayıflığının kaynaklarıdır. UML'nin resmi bir semantiği yoktur. Farklı insanlar bir UML modeli kullandığında bu bir sorun teşkil eder; ve UML’nin temel amaçlarından biri bir tasarım hakkında iletişim kurmak olduğundan, UML'yi kullanmanın farklı yolları, iletişim sorunlarının olası nedenleridir.

Örneğin, bir model için yalnızca en önemli sınıfları, yöntemleri ve nitelikleri belirten bir mimarı düşünün - kendi için basit olduğunu düşündüğü yardımcı sınıfları çıkarır. Farklı bir ofiste veya ülkede çalışan meslektaşlar, sistemi uygulamak için mimarın modelini kullanmalıdır. Modelin her ayrıntıyı tanımladığını varsayarlarsa, yanlış yorumlayacaklardır.

Bu sorunlara rağmen, geniş UML kullanıcıları topluluğu, genel olarak faydasının kanıtıdır.

**ENDUsTRIyel UML MODELLERINDE HATALAR**

Anket yoluyla elde edilen öznel izlenime ek olarak, endüstriyel UML modellerinin kalitesiyle ilgili objektif ölçümler yaptık. Bu amaçla MetricView (www.win.tue.nl/ empanada / metricview) adlı bir model uygulayarak kusurları, uygunsuzlukları ve yanlış anlama risklerini kontrol ettik. Aracı, çeşitli kuruluşlardan ve uygulama alanlarından farklı büyüklükteki çeşitli endüstriyel vaka çalışmalarına uyguladık (bkz. Tablo 1). Durumlarda çeşitli UML CASE araçları kullanılır.

**Örnek olaylarda bulunan kusurlar**

Vaka çalışmaları, UML modelleme kurallarının ihlal edildiğini ortaya koydu. Tüm bu kuralların herhangi bir projede geçerli olmasını beklememekle birlikte, tasarım kalitesi ve proje ilerlemesi önlemlerimizde model tutarsızlıkları ve eksik noktalar olduğunu hesapladık.

Dizi diyagramlarında çağrılmayan yöntemler. Sınıf etkileşimlerini göstermek için, bir sınıfın bir sınıf diyagramında sağladığı genel yöntemlerin sıra diyagramlarında çağrılması gerekir. Çağrılmayan yöntemler etkileşimlerde kullanılamaz; alternatif olarak, etkileşimli işlevleri açısından yeterince açıklanamayabilirler.

Analiz ettiğimiz modellerde kullanılan yöntemlerin yüzde 40 ila 78'i dizi diyagramında çağrılmadı.

Dizi diyagramlarında oluşmayan sınıflar. Bir tasarım, bir sıralama diyagramı nesnesi olarak oluşmayan bir sınıf içeriyorsa, sınıf gereksizdir (sıralama diyagramlarının tüm işlevselliği tanımladığını varsayarsak) veya tasarım sınıflarının etkileşimi, sıralama diyagramları kullanılarak tamamen açıklanmaz.

Analiz ettiğimiz modellerde, sınıfların yüzde 35 ila 61'i nesne olarak gerçekleşmedi.

İsimsiz nesneler. Bir dizi diyagramındaki her nesnenin bir adı olmalıdır. Adlandırılmış nesneler adsız olanlardan daha anlamlı ve anlaşılırdır. Nesnelerin isimlendirilmesi esastır, özellikle aynı türden birkaç nesne tek bir dizi diyagramında gerçekleştiğinde önemlidir.

Örnek çalışmalarımızda, tüm nesnelerin yüzde 25 ila 92'sinin isimlerinin olmadığı tespit edilmiştir.

Mesajlar yöntemlere karşılık gelmiyor. Bir dizi diyagramındaki bir nesnenin aldığı her mesaj, nesnenin sınıf arayüzündeki bir yönteme karşılık gelmelidir; Aksi takdirde, mesajın anlamı belirsizdir.

Analiz edilen modellerde mesajların yüzde 8 ila 59'u yöntemlere uymuyordu.

Yöntemsiz sınıflar. Yöntemi olmayan bir sınıf nesneye yönelik paradigmayı, özellikle de kapsülleme kavramını ihlal eder. Yöntemleri olmayan bir sınıf diğer sınıflarla etkileşime giremez ve bu nedenle eksiktir. Bir sınıfı tamamlamak için, tasarımcı bir sınıfın yöntemlerini tanımlamalı ve etkileşimlerini bir dizi diyagramında tanımlamalıdır. Yalnızca erken modelleme aşamalarında, yöntemleri tanımlamaksızın sınıflar oluşturabilirsiniz.

Analiz aşamasının dışındaki tüm vaka çalışmalarında, tüm sınıfların yüzde 20 ila 60'ı yöntemlerden yoksundu.

**UML model kusurlarının etkileri**

CASE araçlarının yardımına rağmen, yazılım geliştirmede kusurlar yaygındır. Fakat UML modelindeki kusurların etkileri nelerdir? Sonunda, birisi modeli uygulama için bir şartname olarak kullandığında tespit edildi mi? Değilse, bir kusur gerçekten de farklı okuyucuların modeli farklı şekillerde yorumlamasına neden oluyor mu?

Bu soruları cevaplamak için 110 öğrenci ve 48 uygulayıcı ile kontrollü bir deney yaptık. Sonuçlar, model sistemi uygulamak için dikkatlice okunsa bile, kusurların sıklıkla tespit edilmediğini gösterdi. Örneğin, okuyucuların yüzde 61'i, sınıf diyagramında karşılık gelen bir yöntemi olmayan bir dizi diyagramı mesajı tespit etmedi. Sınıf şartnamesi olmayan bir nesne için sonuçlar daha kötü: yüzde 82'si bu hatayı tespit etmedi.

Bu düşük tespit oranları, kusurların farklı yorum riskini arttırıp arttırmadığı sorusunu gündeme getirmektedir. Deneysel sonuçlar, 0'dan 1'e kadar yanlış yorumlama için çeşitli riskler gösterdi; burada 0, yorumlardaki en geniş yayılımı ve 1, bir yorumdaki toplam anlaşmayı temsil ediyor. Örneğin, bir sıralama şeması ile tanımlanmayan bir kullanım durumunun, yanlış yorumlama için yüksek bir riski temsil eden 0.44 değeri vardır.

**UML kullanımını geliştirme fırsatları**

Bu kusurları gözlemledikten sonra, bunları önlemek ve ortadan kaldırmak için birçok fırsat gördük.

**Arıza kontrolü ve geri bildirim**

Mevcut araçlar hata kontrolü için sadece sınırlı destek sağlar. UML’nin gayrı resmi yapısı resmi kontrolleri desteklemese de, hataları otomatik olarak algılamak için araçlar genişletilmelidir.

UML profilleri, belirli mimari stilleri ve kalıpları, örneğin telekomünikasyon sistemlerinde kullanım için müşteri-sunucu kalıplarını tanımlamak için kullanılmıştır. Kalıp, model geliştiricinin hangi yapı taşlarını kullanabileceğini ve bloklar arasında hangi tür ilişkilerin olabileceğini tanımlar. Model geliştiricisi tasarıma rehberlik etmek için bu kalıpları yapıcı olarak kullanabilir. Geliştirme sırasında, araçlar ayrıca mimarinin, tasarımın ve uygulamanın bu kalıba uygun olduğunu doğrulayabilir.

**UML metrikleri**

Yazılım ölçümleri, uygulama sırasında kaynak kod kalitesini yönetmek için iyi bilinen bir tekniktir. Metrikler, mimarlık ve tasarımın kalitesini de yönetebilir. Şekil 5, anket katılımcılarının her bir geliştirme aşamasındaki ölçümler için belirtilen ölçütlerin yanı sıra, ölçümlerin ne kadar yararlı olacağını düşündüklerini göstermektedir. İstenen metrik kullanımı, gerçek kullanımdan iki ila dört kat daha yüksektir. Bu sonuçlar UML modellerinde ölçüm talebini vurgulamaktadır. UML modellerinin çoklu görünümleri kaynak kodunda bulunmayan bilgileri içerir. Bu, mimari düzeyindeki ölçümler için bir temel oluşturur. Projeler, bu ilk ölçütleri tasarım kılavuzlarına ve sezgisellere uymayı veya bunlara aykırı olduğunu belirtmek için kullanabilir. Aşağıda bazı mimari metrikleri açıklanmaktadır:

■ Sınıf dinamikliği. Bu ölçüm, bir sınıfın karmaşıklığını gösterir. Bir sınıfın birçok farklı gelen ve giden mesajı varsa ve birkaç farklı dizi diyagramında görünüyorsa, sınıfın sistemde kritik bir rol oynadığını ve inceleme ve test sırasında daha fazla dikkat edilmesini hak ettiğini varsayıyoruz. Dinamik eşiğin üstünde ise, sezgisel sınıfın davranışını modellemek için bir durum makinesi kullanmanızı önerebilir.

■ Her kullanım durumu için sınıf sayısı. Bu metrik, ilgili işlevselliğin bir tasarımın birçok bölümüne yayılmış olup olmadığını gösterir. Daha yüksek bir değer düşük bakım kolaylığı ve yeniden kullanılabilirliği gösterir. Sınıfı olmayan vakaları kullanmak, geliştiricilerin henüz bir uygulama öngörmediğini, bu durumun da bir uygulama ihmalini gösterebileceğini belirtir.

■ Sınıf başına birkaç kullanım durumu. Birçok kullanım durumunda uygulanan bir sınıfın düşük uyumu olabilir ve bu nedenle büyük miktarda ilgisiz işlevsellik sunar. Ayrıca sistem için merkezi olabilir, bu da sistemdeki hataların birçok sistem özelliğinin zarar görmesine neden olacağı anlamına gelir.

Mevcut araçlar genellikle metrikleri ve hataları tablo halinde gösterir. Bu, tasarımcıların tabloları üzerinde çalıştıkları diyagramlarla ilişkilendirmelerini gerektirir. Bir UML diyagramında metrik değerlerin ve kusurların grafiksel olarak karşılık gelen yerde sunulması daha doğrudan geri bildirim oluşturur. Şekil 6'da, MetricView aracımız UML diyagramları üzerinde görselleştirilen ölçümleri ve hataları gösterir.

UML uygulamaları bunun için geliştirme araçlarında artan yeteneklerle gelişmelidir. Birkaç alanda iyileştirme gerekiyor:

Tasarım kusurlarının, ihmallerin ve tutarsızlıkların tespiti. Yazılım ölçümleri, UML model özelliklerinin bir değerlendirmesini desteklemek için hızlı, objektif bir teknik sunarak kusurları tespit etmede rol oynayabilir. Ayrıca, mimarinin yapısını iyileştirmek için yeniden yapılanma fırsatlarının belirlenmesine yardımcı olurlar.

■ Modellemede daha fazla homojenlik. Standartları ve bunları kontrol etmek için geliştirme araçlarını modellemek bu amaca ulaşabilir.

■ Etki alanına veya projeye özgü referans mimarileri ve kalıpları. UML profilleri bu çalışmayı desteklemektedir. UML 2.0, bu tür profilleri tanımlamak için çeşitli olanaklar sağlar, ancak bunları kontrol etmek için çok az destek sağlar.

■ UML modelleri ve sistem gereksinimleri ile uygulamalar arasında daha fazla tutarlılık. İzlenebilirlik ve gidiş dönüş mühendisliği için daha iyi mekanizmalar bu sorunların azaltılmasına yardımcı olacaktır.

■ UML modelleri için tanımlanmış kalite hedefleri. Kalite hedefi, bir modeli geliştirmek için bir teşviktir ve tanımlanan hedefe ulaşmak için iyileştirilmesi gereken noktaları belirler. Mevcut UML geliştirme araçları ölçülebilir kalite hedeflerinin tanımını desteklememektedir ve bu işlev gereklidir.

Burada bildirilen örnek olay incelemelerine uyguladığımız MetricView aracı, analizi otomatikleştirerek uygun maliyetlidir. Çoğu durumda, konuştuğumuz mimarlar analiz bulgularında değer gördü ve aracın modellerde zayıf noktaları belirlediğini kabul etti. Çoğu durumda, geribildirimimiz, tespit edilen sorunları gidermek için model değişikliklerine yol açtı. Bazı vaka çalışmaları için ardışık model versiyonlarını analiz ettik ve yeniden işlemden sonra model kalitesini iyileştirdik.