Test Yönetimi

Birçok projede test, tüm faaliyetlerin en büyük miktarda kaynağını tüketir. Durumda net bir stratejisi olmayan pullar gibi test durumlarını toplama eğilimindeyiz. Birçok şirket, kalite, görünürlük ve test ilerleme yönetimi konusunda yetersiz kalmaktadır. Yazar Panos Louridas test yönetimi tanıttı. Hem bu okuyuculardan hem de potansiyel yazarlardan bu sütun ve daha fazla bilgi edinmek istediğiniz teknolojiler hakkında bilgi almak için sabırsızlanıyorum.

MICROSOFT WROTE Office 2007 için 1 milyondan fazla test vakası. Bu rakam aşırı veya sadece leviathan ölçekli projeler için uygun görünebilir, ancak gerçek şu ki, test senaryoları geliştiricilerin sadece tehlikelerinde görmezden gelebilecekleri bir yazılım bileşeni teşkil ediyor. Test yönetimi nasıl yaptığımızla ilgilidir

• Test vakalarının çoğalması arasında gezinmek, onları organize etmek, onları test uzmanlarına atamak,

• Orkestrat testi,

• Test sonuçlarını toplamak ve ilerlemeyi ölçmek.

Bu sütunda anlatacağım test vakalarını yönetmek için son zamanlarda bazı ilginç teknolojiler getirildi. Katılan riskleri anlamak için, ne kadar çaba testi gerektirdiğini görmeye değer.

Test Effort

Test yönetimi, yüksek test maliyeti göz önüne alındığında anlamlıdır. Test vakalarının yönetilmesinin değeri doğrudan projede testin gösterdiği efor miktarından kaynaklanmaktadır. Steve McConnell ve Christof Ebert'in sanayi rakamları aşağıdakileri gösteriyor:

• Sistem testi, projenin büyüklüğüne bağlı olarak toplam çabanın yüzde 16 ila 29'unu gerektirir (bkz. Tablo 1).

• Test çalışması, projenin niteliğine (örneğin, güvenlik açısından kritik projeler basit bir aygıttan daha fazlasına ihtiyaç duyar), karmaşıklığa ve tabii ki süreçlere ve araçlara bağlı olarak değişir (bkz. Tablo 2). Örneğin, yetersiz girdi kalitesiyle test çabası hemen iki ya da üç kat artar.

• Sistematik test planlama, sık test kapsamı incelemeleri, artan gelişim, günlük yapımlar ve otomatik regresyonlu duman testleri sektördeki en büyük verimlilik kolları arasındadır.

Bu çaba seviyelerinde, testi mümkün olduğunca verimli kılmak proje yöneticileri ve geliştiricileri olmaya devam ediyor. Bu, daha sonra döneceğimiz ölçüm testlerini gerektirir.

Test Ölçümü

Test durumları, bir ürünün kalitesini artırmaya yardımcı olmasının yanı sıra, test sonuçlarını içeren bir dizi işlem ölçümünü ölçmek için kullanıldığında geliştirme sürecini iyileştirmede değerlidir. Hata giderme etkinliği (DRE) veya test etkinliği, test sırasında bulunan hata sayısının, bulunan toplam hata sayısına oranıdır. Örneğin, testler tarafından 90 böcek bulunursa ve diğer yollarla 10 böcek bulunursa (daha sonra ürün ömrü boyunca, kullanıcılar vb.) Test etkinliği 90 / (10 + 90) = 0,90'dır. Test verimliliği, literatürün test etkinliğiyle karıştırdığı bir ölçüdür - belki de etkili ve etkili olan semantik yakınlığı suçlamaktır. Fowler’ın Modern İngilizce Kullanımı, “kesin veya istenen bir etkiye sahip” olarak etkili ve “asgari atık veya eforla üretken” olarak etkili olarak tanımlar. Dolayısıyla, verimlilik, fizikteki güç gibi bir ölçüdür. Bu açıklamaya rağmen, test etkinliği, literatürdeki farklı tanımları kapsar. En geniş anlamıyla, onları bulmak için harcanan çabaya bölünerek bulunan kusurların sayısıdır. Bu, örneğin, test durumlarının sayısına bölünmüş bulunan kusur sayıları ile sayılabilir. Farklı bir tanım, hataları bulan testlerin yüzdesidir; aynı zamanda hangi testlerin böcek bulmakta daha etkili olduğunu da gösterebilir.

Geçiş oranı, toplam test sayısına geçen testlerin oranıdır.

Başarılı ve başarısız testler, başarısız olan test sayısına geçen testlerin oranıdır. Bu, çok sayıda test içeren projelerde başarısızlıkları göstermek için yararlı olabilir. Örneğin, 1 milyon test için, yüzde 95'lik bir geçiş oranı, 50.000 başarısız testine eşdeğerdir; yani, 19'luk başarısızlıklara karşı bir geçiş oranı.

Bir test ilerlemesi S eğrisi, x ekseninde zaman birimlerini (örneğin haftaları) çizen ve y eksenindeki her bir zaman biriminde başarıyla tamamlanan, denenen ve geçen testlerin sayısını gösteren bir grafiktir. Grafiğe S eğrisi adı verilir, çünkü bu şekli izler: çizilen test sayıları birikimlidir ve planlanan testlerin sayısı düşük başlar, sonra artar ve son olarak serbest bırakma zamanına yaklaştıkça yükselir (bkz. Şekil 1) . Test ilerleme grafiği, bir test çabasındaki (denenen test sayısı planlanan sayıdan az olduğunda) veya proje kalitesindeki (geçen test sayısı planlanan sayının oldukça gerisinde kaldığında) gecikmeleri vurgulamak için geliştirme çabalarına rehberlik etmek için kullanılır.

Zaman içerisinde hata varışlarının test edilmesi, x ekseni üzerindeki zaman birimlerini ve y ekseni üzerinde test edilerek keşfedilen hata varışlarını gösteren bir grafiktir. Grafik normalde kambur benzeri bir şekil olmalıdır: projenin başlangıcında, birkaç test raporları ve yine de kusurlar için birkaç fırsat olduğundan; proje ilerledikçe, hem testler hem de keşfedilen kusurlar yığılır; ancak sevkıyata yaklaştıkça daha az kusur ortaya çıkmalı.

Zaman içinde hata biriktirme testini tekrar gösteren bir grafik, zaman birimini x ekseni üzerinde gönderim tarihinden önce gösterir ve test edilerek keşfedilen ve henüz çözümlenemeyen kusurları y ekseni üzerinde gösterir. Metrik, nakliyeden önce kusur biriktirme hedefine yönelik belirli hedefler üzerinde anlaşarak süreç planlamasını yönlendirmek için kullanılabilir.

Test kapsamı, testlerin test edilecek kodu ne kadar kapsamlı kapsadığını ölçer. Farklı şekillerde ölçülebilir - örneğin, koddaki tüm fonksiyonları test etmek için gereken test vakalarının sayısına veya yürütülen toplam test sayısına göre test edilen dalların sayısına bölünerek yapılan test vakalarının oranı ile programda şubeler. Test kapsamı, programda kör nokta kalmadığından emin olmak için kullanışlıdır, ancak proje süreci ilerlemesiyle ilgili diğer ölçümleri türetmek için de faydalıdır.

Test Vakası Yönetim Araçları

Bir test durumunun yararlı olması için şunları içermesi gerekir:

• Hedef, test etmek istediğini kısaca tanımlayarak;

• Bunun neden önemli olduğunu açıklayan gerekçe;

• Ön koşul, çalışması için yürürlükte olması gereken çevre üzerindeki kısıtlamaları listelemek;

• girişler;

• Kodda uygulanmadıkça, numaralandırılmış sıradaki adımlar;

• beklenen sonuçlar;

• ne sıklıkta ve ne zaman çalışması gerektiği hakkında talimatlar; ve

• altında çalışması gereken konfigürasyonlar.

Test durumlarını yönetmenin en basit yolu, aşağıdaki bilgileri içeren bir e-tablo iledir:

• Test grubunun adı ve her test paketi için testin adı.

• Test yürütme için sıraya alınmışsa, her test vakasının durumu, değerler, başarılı, başarısız veya değeri yok.

• Test durumu için kullanılan sistem yapılandırması için bir tanımlayıcı.

Tanımlayıcı, konfigürasyon detaylarını ayrı bir e-tabloda aramak için kullanılır.

• Test tarafından keşfedilen hatanın tanımlayıcısı. Test birden fazla hata bulduysa, tek bir hücrede virgüllerle veya farklı satırlarda (diğer satırlar yukarıdaki satırdan kopyalanmış olarak) listelenebilir.

• Test cihazının baş harfleri.

Bu basit gibi görünse de, bir elektronik tablo kullanmadan kaç test yapıldığını hesaba katabilirsiniz. Elektronik tabloyu iyileştirmenin ve geliştirme sürecinin bir parçası haline getirmenin yollarını düşünmek kolaydır. Kişisel bir elektronik tablo olarak kalması için bir neden yok - Google Dokümanlar veya bir ekip wiki gibi bir ekip ortamına yerleştirdi.

Açık ve kapalı kaynaklı ticari olanların yanı sıra, testlerin yönetimine yönelik daha sofistike yaklaşımların yanı sıra bir dizi araç bulunmaktadır. (Bir noktada, Microsoft'ta MS Windows'ta en az yarım düzine test senaryosu yönetim sistemi vardı, ancak daha sonra ekipler Visual Studio Team Sistemindeki test senaryosu yönetim araçlarına geçmeye başladılar). Hangisini seçeceğinizi seçerken dikkat etmeniz gerekenler şunlardır:

• Bir hata veya sorun takip sistemiyle birlikte çalışabilirlik.

• İhtiyaç izleme sistemleriyle birlikte çalışabilirlik. İdeal olarak, bir test senaryosu yönetim aracı, fonksiyonel gereksinimler, test senaryoları ve hatalar arasında üç yönlü bir bağlantı sunmalıdır. Bu şekilde, hangi test durumlarının bir gereksinime karşılık geldiğini bilebilir, test koşullarına sahip olmayan gereksinimleri belirleyebilir veya belirli gereksinimlere yönelik hataları haritalandırabiliriz.

• Her test durumu için tutulan bilgiler. Bu bilgiler, bu bölümün başında listelediğimiz materyali içermeli ve ayrıca testin yapıldığı tarih, test yazarı, ilgili test durumlarına bağlantılar ve izleme günlükleri gibi bilgileri de içermelidir.

• Otomasyon derecesi. Temizlik test çalışmaları ve sonuçları (elektronik tablo çözümü ya da wiki tabanlı basit bir uygulama gibi) için aracı kullanabilir ya da test planlarını isteyerek tüm test sürecini yönlendirebilirsiniz.

• Raporlama yetenekleri ve ölçümleri hesaplama. Ölçümler daha önce sunulanları içerebilir ve sonuçların raporlarda sunulması gerekir. Daha da iyisi, metriklerin sonuçları ve temel alınan veriler bir API üzerinden elde edilebilir.

• Kimlik doğrulama sistemleri ile birlikte çalışabilirlik. İdeal olarak, her bir proje katılımcısı tek bir kimliğe sahip olmalı ve bu kimliği her şey için kullanmalı, kodlama, hata bildirme, deneme testleri yapma ve hatta proje ile ilgili posta listelerinde yazarken kullanmalıdır.

• Kurulum gereksinimleri. Genellikle, bu araçlar bir sunucuya dağıtılır, böylece Web üzerinden geliştirme ekibinin kullanımına açık olur. Popüler Web geliştirme çerçeveleri (Ruby on Rails gibi) üzerine inşa edilebilirler, ancak genel olarak, kendi araç ekosisteminiz ve altyapınızla ne kadar iyi bütünleştiklerini kontrol etmeye değer.

• Maliyet ve lisans.

Test senaryosu yönetim araçları tipik olarak, test testlerini tanımlamanın bir yoluyla, örneğin, kullanıcının test testini oluşturan alanları doldurduğu bir dizi şablonla başlar (bkz. Şekil 2). Ayrıca, test durumları arasında ilişki kurmanın, koşullarını yerine getirmelerine (örneğin, test durumu B başarılı olursa test yapılmalıdır) ve değişkenlere (sistem gibi) bağlı olarak test durumlarının çeşitlerini otomatik olarak oluşturmanın mümkün olduğunu kanıtlayabilir. konfigürasyonu). Test vakaları daha sonra kullanıcılara atanır. Genel ilerleme ve uygulamaya genel bakış, test sonuçlarını hem grafiklerde hem de sayılarda sunan bir gösterge panosu aracılığıyla sağlanır. Bu sayılara dayanarak test ölçümlerini hesaplayabiliriz; araçlar ayrıca yönetim amaçları için yararlı olabilecek raporlar hazırlar.

Bir aracın en önemli özelliği teknik kabiliyeti değil, ne kadar etkili kullanıldığı ve geliştirme sürecinde ne kadar iyi kullanıldığı ve ne kadar iyi kullanıldığı (bkz. “Daha Okuma” kenar çubuğuna bakınız). Bir kusur önleme ve kalite güvencesi kültürünü desteklemek, Othello'nun “modern görünüm” olarak nitelendirdiği herhangi bir araçtan daha önemlidir.