IT522 – Yazılım Mühendisliği 2021

PhD Furkan Gözükara, Toros University

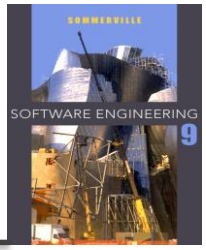
https://github.com/FurkanGozukara/Yazilim-Muhendisligi-IT522-2021

Ders 3

Çevik Yazılım Geliştirme

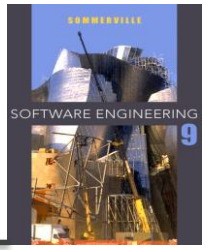
Kaynak : https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Presentations/index.html

Ders 3'de İşlenen Konular



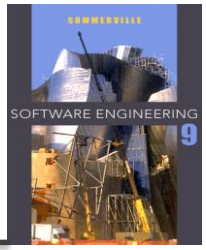
- Çevik yöntemler.
- Plan odaklı ve çevik geliştirme.
- Ekstrem programlama.
- Çevik proje yönetimi.
- Çevik yöntemleri ölçeklendirme.

Hızlı Yazılım Geliştirme



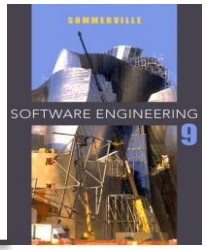
- Hızlı geliştirme ve teslimat, artık çoğu zaman yazılım sistemleri için en önemli gereksinimdir:
  - İşletmeler hızla değişen bir gereksinimle çalışır ve bir dizi kararlı yazılım gereksinimi üretmek pratik olarak imkansızdır.
  - Yazılım, değişen iş ihtiyaçlarını yansıtacak şekilde hızla gelişmelidir.
- Hızlı yazılım geliştirme:
  - Şartname, tasarım ve uygulama birbiriyle ilişkilidir.
  - Sistem, sürüm değerlendirmesine dahil olan paydaşlarla bir dizi sürüm olarak geliştirilmiştir.
  - Kullanıcı arayüzleri genellikle bir IDE ve grafik araç seti kullanılarak geliştirilir.

Çevik Yöntemler



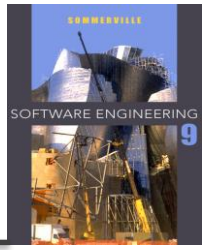
- 1980'lerin ve 1990'ların yazılım tasarım yöntemlerinde yer alan genel giderlerden memnuniyetsizlik, çevik yöntemlerin geliştirilmesini sağladı. Bu yöntemler:
  - Tasarım yerine koda odaklanın.
  - Yazılım geliştirme yinelemeli bir yaklaşıma dayanmaktadır.
  - Çalışan yazılımı hızlı bir şekilde sunması ve bunu değişen gereksinimleri karşılayacak şekilde hızla geliştirmesi amaçlanmaktadır.
- Çevik yöntemlerin amacı, yazılım sürecindeki genel giderleri azaltmak (örneğin, dokümantasyonu sınırlandırarak) ve aşırı yeniden çalışma yapmadan değişen gereksinimlere hızla yanıt verebilmektir.

Çevik Manifesto



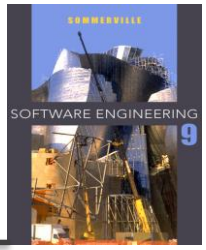
- *Bunu yaparak ve başkalarının yapmasına yardımcı olarak yazılım geliştirmenin daha iyi yollarını ortaya çıkarıyoruz. Bu çalışma sayesinde değer kazandık:*
  - *Süreçler ve araçlardan ziyade bireyler ve etkileşimler. Kapsamlı dokümantasyon yerine çalışan yazılım. Sözleşme müzakeresi yerine müşteri işbirliği. Bir planı takip etmek yerine değişime yanıt vermek.*
- *Yani sağdaki öğelerde değer varken soldaki öğelere daha çok değer veriyoruz.*

Çevik Yöntemlerin İlkeleri



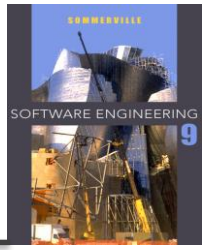
Prensip	Açıklama
Müşteri katılımı:	Müşteriler geliştirme süreci boyunca yakından ilgilenmelidir. Rolleri, yeni sistem gereksinimlerini sağlamak ve önceliklendirmek ve sistemin yinelemelerini değerlendirmektir.
Artımlı teslimat:	Yazılım, her bir artıma dahil edilecek gereksinimleri belirleyen müşteri ile aşamalı olarak geliştirilir.
Süreç yapmayan insanlar:	Geliştirme ekibinin becerileri tanınmalı ve kullanılmalıdır. Ekip üyeleri, kuralcı süreçler olmadan kendi çalışma yöntemlerini geliştirmeye bırakılmalıdır.
Değişikliği benimse:	Sistem gereksinimlerinin değişmesini bekleyin ve bu nedenle sistemi bu değişiklikleri karşılayacak şekilde tasarlayın.
Basitliği koruyun:	Hem geliştirilmekte olan yazılımda hem de geliştirme sürecinde basitliğe odaklanın. Mümkün olan her yerde, sistemdeki karmaşıklığı ortadan kaldırmak için aktif olarak çalışın.

Çevik Yöntem Uygulanabilirliği



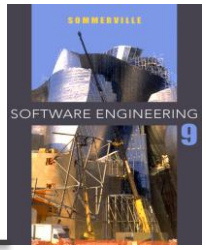
- Bir yazılım şirketinin satış için küçük veya orta ölçekli bir ürün geliştirdiği ürün geliştirme.
- Müşterinin geliştirme sürecine dahil olma konusunda açık bir taahhüdünün olduğu ve yazılımı etkileyen çok sayıda dış kural ve düzenlemenin olmadığı bir organizasyon içinde özel sistem geliştirme.
- Küçük, sıkı entegre ekiplere odaklandıkları için, çevik yöntemlerini büyük sistemlere ölçeklendirmede sorunlar var.

Çevik Yöntemlerle İlgili Sorunlar



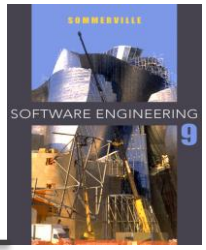
- Sürece dahil olan müşterilerin ilgisini korumak zor olabilir.
- Ekip üyeleri, çevik yöntemleri karakterize eden yoğun katılım için uygun olmayabilir.
- Birden fazla paydaşın olduğu yerlerde değişikliklere öncelik vermek zor olabilir.
- Sadeliği korumak ekstra çalışma gerektirir.
- Sözleşmeler, yinelemeli geliştirmeye yönelik diğer yaklaşımlarda olduğu gibi bir sorun olabilir.

Çevik Yöntemler Ve Yazılım Bakımı



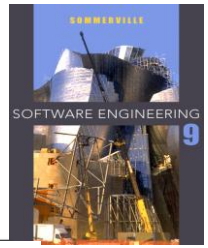
- Çoğu kuruluş, yeni yazılım geliştirmeye harcadıklarından daha fazlasını mevcut yazılımları korumak için harcamaktadır. Dolayısıyla, çevik yöntemler başarılı olacaksa, orijinal geliştirmenin yanı sıra bakımı da desteklemeleri gerekir.
- İki temel konu:
  - Biçimsel dokümantasyonu en aza indirme geliştirme sürecindeki vurgu göz önüne alındığında, çevik bir yaklaşım kullanılarak geliştirilen sistemler sürdürülebilir mi?
  - Müşteri değişim taleplerine yanıt olarak bir sistemi geliştirmek için çevik yöntemler etkili bir şekilde kullanılabilir mi?
- Orijinal geliştirme ekibinin yazılımın bakımını yapmayı sürdürmezse sorunlar ortaya çıkabilir.

Plan Odaklı Ve Çevik Geliştirme

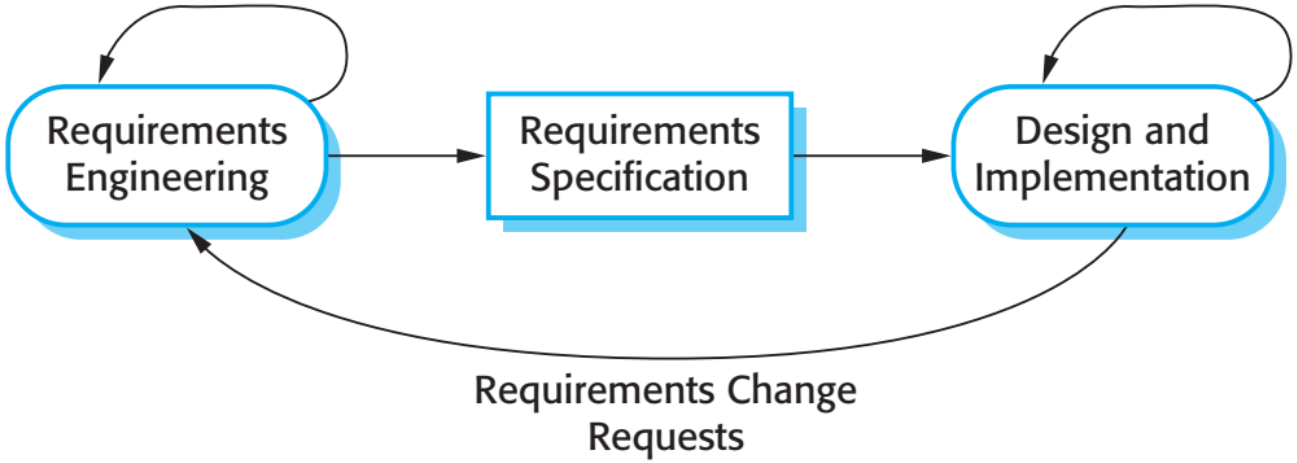


- Plan odaklı geliştirme:
  - Yazılım mühendisliğine yönelik plan odaklı bir yaklaşım, önceden planlanan bu aşamaların her birinde üretilecek çıktılarla ayrı geliştirme aşamalarına dayanmaktadır.
  - Şelale modeli olması gerekmez - plan odaklı, kademeli geliştirme mümkündür.
  - Yineleme, etkinlikler içinde gerçekleşir.
- Çevik geliştirme:
  - Spesifikasyon, tasarım, uygulama ve testler birbiri ardına bırakılır ve geliştirme sürecinden elde edilen çıktılara, yazılım geliştirme sürecinde bir müzakere süreci ile karar verilir.

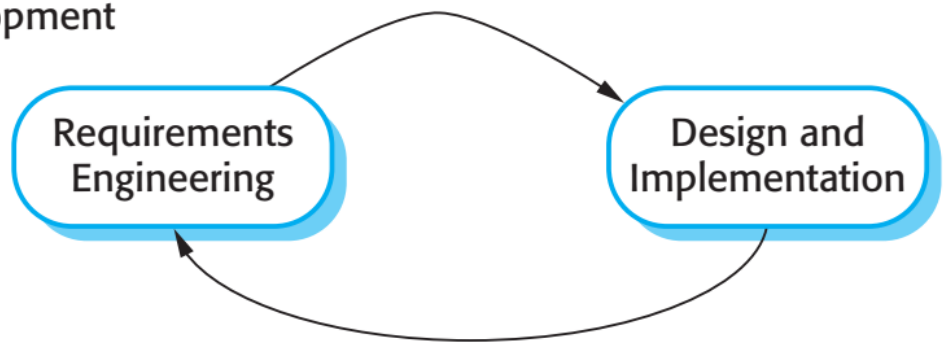
Plan Odaklı Ve Çevik Özellikler



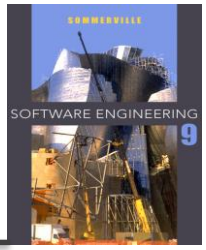
Plan-Based Development



Agile Development

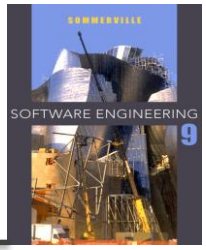


Teknik, İnsani, Organizasyonel Sorunlar



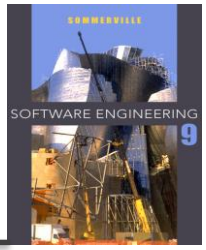
- Çoğu proje, plan odaklı ve çevik süreçlerin unsurlarını içerir. Dengeye karar vermek şunlara bağlıdır:
  - Uygulamaya geçmeden önce çok detaylı bir şartname ve tasarıma sahip olmak önemli mi? Öyleyse, muhtemelen plan odaklı bir yaklaşım kullanmanız gerekir.
  - Yazılımı müşterilere teslim ettiğiniz ve onlardan hızlı geri bildirim aldığınız aşamalı bir teslimat stratejisi gerçekçi mi? Öyleyse, çevik yöntemler kullanmayı düşünün.
  - Geliştirilmekte olan sistem ne kadar büyük? Çevik yöntemler, sistem, gayri resmi olarak iletişim kurabilen küçük bir ekiple geliştirilebildiğinde en etkilidir. Bu, daha büyük geliştirme ekipleri gerektiren büyük sistemler için mümkün olmayabilir, bu nedenle plan odaklı bir yaklaşımın kullanılması gerekebilir.

Teknik, İnsani, Organizasyonel Sorunlar



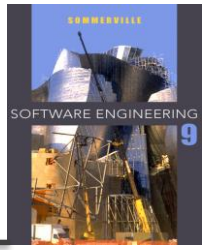
- Ne tür bir sistem geliştiriliyor?
  - Uygulamadan önce çok fazla analiz gerektiren sistemler için plan odaklı yaklaşımlar gerekli olabilir (örneğin, karmaşık zamanlama gereksinimleri olan gerçek zamanlı sistem).
- Beklenen sistem ömrü nedir?
  - Uzun ömürlü sistemler, sistem geliştiricilerinin orijinal niyetlerini destek ekibine iletmek için daha fazla tasarım dokümantasyonu gerektirebilir.
- Sistem geliştirmeyi desteklemek için hangi teknolojiler mevcuttur?
  - Çevik yöntemler, gelişen bir tasarımı takip etmek için iyi araçlara dayanır.
- Geliştirme ekibi nasıl organize edilir?
  - Geliştirme ekibi dağıtılıyorsa veya geliştirmenin bir parçası dış kaynak kullanılıyorsa, geliştirme ekipleri arasında iletişim kurmak için tasarım belgeleri geliştirmeniz gerekebilir.

Teknik, İnsani, Organizasyonel Sorunlar



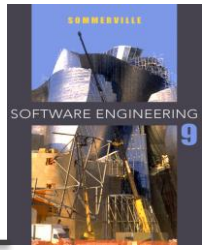
- Sistem gelişimini etkileyebilecek kültürel veya kurumsal sorunlar var mı?
  - Geleneksel mühendislik organizasyonları, mühendislikte norm olduğu için plan tabanlı bir geliştirme kültürüne sahiptir.
- Geliştirme ekibindeki tasarımcılar ve programcılar ne kadar iyi?
  - Bazen çevik yöntemlerin, programcıların ayrıntılı bir tasarımı koda dönüştürdüğü plan tabanlı yaklaşımlardan daha yüksek beceri seviyeleri gerektirdiği tartışılır.
- Sistem harici düzenlemeye tabi midir?
  - Bir sistemin harici bir düzenleyici tarafından onaylanması gerekiyorsa (örn. FAA, bir uçağın işletimi için kritik olan yazılımı onaylar), o zaman muhtemelen sistem güvenlik durumunun bir parçası olarak ayrıntılı belgeler üretmeniz istenecektir.

Ekstrem Programlama



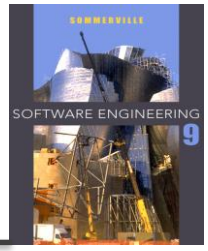
- Belki de en çok bilinen ve en çok kullanılan çevik yöntem.
- Ekstrem Programlama (Extreme Programming - XP), yinelemeli geliştirmeye "ekstrem" bir yaklaşım getirir.
  - Yeni sürümler günde birkaç kez oluşturulabilir;
  - Ürün sürümleri her 2 haftada bir müşterilere teslim edilir;
  - Tüm testler her derleme için çalıştırılmalıdır ve yapı yalnızca testler başarıyla çalıştırılırsa kabul edilir.

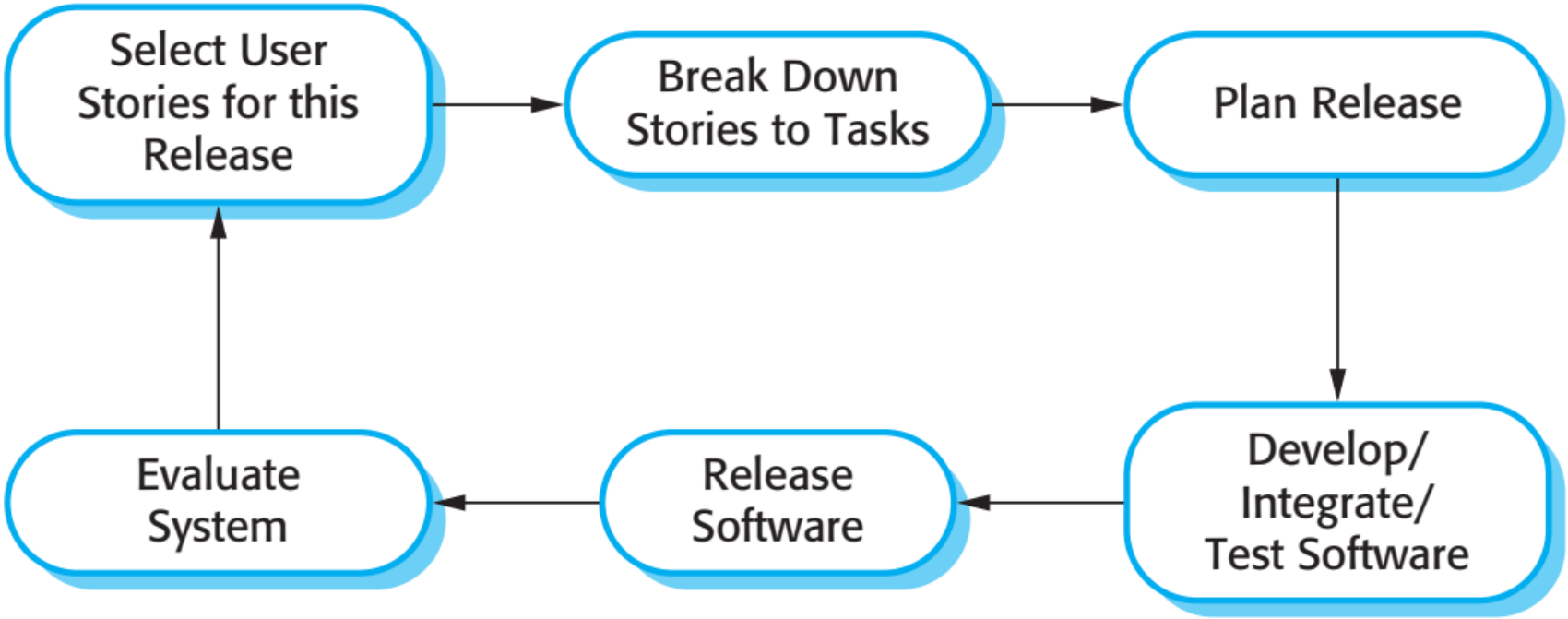
Ekstrem Programlama Ve Çevik İlkeler



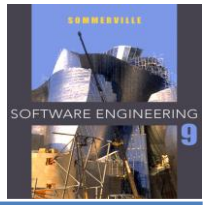
- Artımlı geliştirme, küçük, sık sistem sürümleriyle desteklenir.
- Müşteri katılımı, ekiple tam zamanlı müşteri etkileşimi anlamına gelir.
- İnsanlar ikili programlama, kolektif mülkiyet ve uzun çalışma saatlerinden kaçınan bir süreç yoluyla işlem yapmazlar.
- Düzenli sistem sürümleri aracılığıyla desteklenen değişiklik.
- Kodun sürekli yeniden düzenlenmesi yoluyla basitliği korumak.

Ekstrem Programlama Sürüm Döngüsü



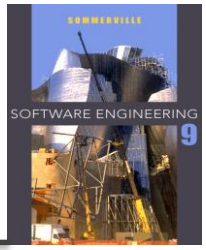


Ekstrem Programlama Uygulamaları (a)



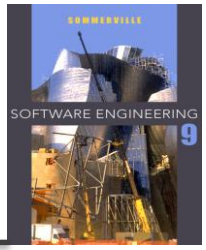
İlke veya Uygulama	Açıklama
Artımlı planlama:	Gereksinimler hikaye kartlarına kaydedilir ve bir güncellemeye/sürüme dahil edilecek hikayeler, mevcut zamana ve göreceli önceliklerine göre belirlenir. Geliştiriciler bu hikayeleri geliştirme 'Görevleri'ne böler. Şekil 3.5 ve 3.6'ya bakın.
Küçük sürümler:	İlk olarak iş değeri sağlayan minimal kullanışlı işlevsellik seti geliştirilir. Sistemin sürümleri sıktır ve ilk sürüme aşamalı olarak işlevsellik ekler.
Basit tasarım:	Mevcut gereksinimleri karşılamak için yeterli tasarım gerçekleştirilir daha fazlası değil.
Önce test geliştirme:	Otomatik bir birim testi framework'u, bu işlevselliğin kendisi uygulanmadan geliştirilmeden önce işlevsellik testini yazmak için kullanılır.
Yeniden düzenleme:	Tüm geliştiricilerin, mümkün olan kod iyileştirmeleri bulunur bulunmaz kodu sürekli olarak yeniden düzenlemeleri beklenir. Bu, kodu basit ve sürdürülebilir tutar.

Ekstrem Programlama Uygulamaları (b)



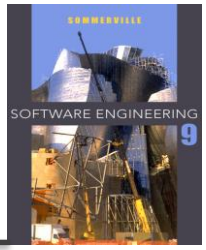
Çiftler Programı:	**Geliştiriciler çiftler halinde çalışır, birbirlerinin çalışmalarını kontrol eder ve her zaman iyi bir iş çıkarmak için destek sağlar.**
Toplu mülkiyet:	Geliştirici çiftleri sistemin tüm alanlarında çalışır, böylece hiçbir uzmanlık adası gelişmez ve tüm geliştiriciler tüm kodun sorumluluğunu alır. Herkes her şeyi değiştirebilir.
Sürekli entegrasyon:	Bir görev üzerindeki çalışma tamamlanır tamamlanmaz tüm sisteme entegre edilir. Bu tür bir entegrasyondan sonra, sistemdeki tüm birim testleri geçmelidir.
Sürdürülebilir hız:	Net etki genellikle kod kalitesini ve orta vadeli üretkenliği düşürdüğü için büyük miktarlarda fazla mesai kabul edilemez.
Yerinde müşteri:	Sistemin son kullanıcısının (müşteri) bir temsilcisi, XP ekibinin kullanımı için tam zamanlı hazır bulunmalıdır. Ekstrem bir programlama sürecinde, müşteri geliştirme ekibinin bir üyesidir ve uygulama için ekibe sistem gereksinimlerini getirmekten sorumludur.

Gereksinim Senaryoları



- XP'de, bir müşteri veya kullanıcı XP ekibinin bir parçasıdır ve gereksinimlerle ilgili kararlar vermekten sorumludur.
- Kullanıcı gereksinimleri, senaryolar veya kullanıcı hikayeleri olarak ifade edilir.
- Bunlar kartlara yazılır ve geliştirme ekibi bunları uygulama görevlerine böler. Bu görevler, zamanlama ve maliyet tahminlerinin temelidir.
- Müşteri, önceliklerine ve program tahminlerine göre sonraki sürüme eklenecek hikayeleri seçer.

Bir 'Reçeteli İlaç' Hikayesi



Prescribing Medication

Kate is a doctor who wishes to prescribe medication for a patient attending a clinic. The patient record is already displayed on her computer so she clicks on the medication field and can select current medication', 'new medication' or 'formulary'.

If she selects 'current medication', the system asks her to check the dose. If she wants to change the dose, she enters the dose and then confirms the prescription.

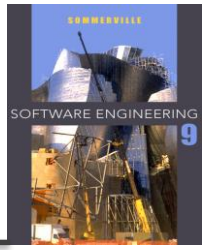
If she chooses 'new medication', the system assumes that she knows which medication to prescribe. She types the first few letters of the drug name. The system displays a list of possible drugs starting with these letters. She chooses the required medication and the system responds by asking her to check that the medication selected is correct. She enters the dose and then confirms the prescription.

If she chooses 'formulary', the system displays a search box for the approved formulary. She can then search for the drug required. She selects a drug and is asked to check that the medication is correct. She enters the dose and then confirms the prescription.

The system always checks that the dose is within the approved range. If it isn't, Kate is asked to change the dose.

After Kate has confirmed the prescription, it will be displayed for checking. She either clicks 'OK' or 'Change'. If she clicks 'OK', the prescription is recorded on the audit database. If she clicks on 'Change', she reenters the 'Prescribing medication' process.

Bir 'Reçeteli İlaç' Hikayesi



Kate, bir kliniğe giden bir hastaya ilaç yazmak isteyen bir doktordur. Hasta kaydı zaten bilgisayarında görüntülendiğinden, ilaç tedavisi alanına tıklar ve mevcut ilacı, "yeni ilacı" veya "formüler" i seçebilir.

"Mevcut ilacı" seçerse, sistem ondan dozu kontrol etmesini ister. Dozu değiştirmek isterse, dozu girer ve ardından reçeteyi onaylar.

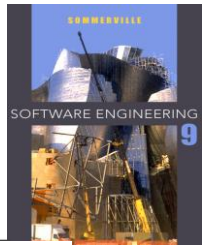
"Yeni ilaç" ı seçerse, sistem hangi ilacı reçete edeceğini bildiğini varsayar. İlaç adının ilk birkaç harfini yazar. Sistem, bu harflerle başlayarak olası ilaçların bir listesini görüntüler. Gerekli ilacı seçer ve sistem kendisinden seçilen ilacın doğru olup olmadığını kontrol etmesini isteyerek yanıt verir. Dozu girer ve ardından reçeteyi onaylar.

"Formüler" i seçerse, sistem onaylanmış formüler için bir arama kutusu görüntüler. Daha sonra gerekli ilacı arayabilir. Bir ilaç seçer ve ilacın doğru olup olmadığını kontrol etmesi istenir. Dozu girer ve ardından reçeteyi onaylar.

Sistem her zaman dozun onaylanmış aralıkta olup olmadığını kontrol eder. Değilse, Kate'den dozu değiştirmesi istenir.

Kate reçeteyi onayladıktan sonra kontrol için reçeteyi görecektir. Ya "Tamam" ı veya "Değiştir" i tıklar. "Tamam" ı tıklarsa, reçete denetim veritabanına kaydedilir. "Değiştir" i tıklarsa, "İlaç reçetesi yazma" sürecine yeniden başlar.

İlaç Reçetelemek İçin Görev Kartı Örnekleri



Task 1: Change Dose of Prescribed Drug

Task 2: Formulary Selection

Task 3: Dose Checking

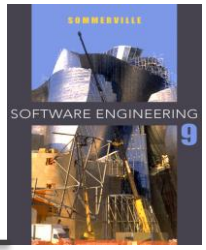
Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.

Using the formulary ID for the generic drug name, look up the formulary and retrieve the recommended maximum and minimum dose.

Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low.

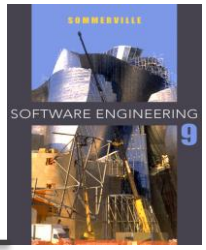
If within the range, enable the 'Confirm' button.

XP Ve Değişim



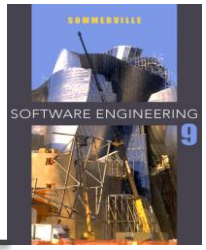
- Yazılım mühendisliğinde geleneksel akıl, değişim için tasarım yapmaktır. Daha sonraki yaşam döngüsünde maliyetleri düşürdüğü için değişiklikleri önceden tahmin etmek için zaman ve çaba harcamaya değer.
- Ancak XP, değişiklikler güvenilir bir şekilde öngörülemediği için bunun faydalı olmadığını savunuyor.
- Daha ziyade, uygulanmaları gerektiğinde değişiklikleri kolaylaştırmak için sürekli kod iyileştirme (yeniden düzenleme) önerir.

Yeniden Düzenleme



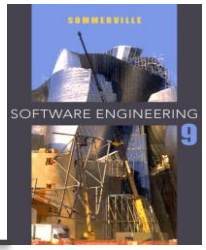
- Programlama ekibi olası yazılım iyileştirmelerini arar ve bu iyileştirmeleri acil ihtiyaç duyulmayan yerlerde bile yapar.
- Bu, yazılımın anlaşılırlığını artırır ve böylece dokümantasyon ihtiyacını azaltır.
- Kod iyi yapılandırılmış ve net olduğu için değişiklik yapmak daha kolaydır.
- Ancak, bazı değişiklikler mimarinin yeniden düzenlenmesini gerektirir ve bu çok daha pahalıdır.

Yeniden Düzenleme Örnekleri

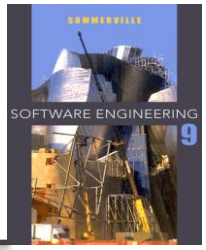


- Yinelenen kodu kaldırmak için bir sınıf hiyerarşisinin yeniden düzenlenmesi.
- Anlamalarını kolaylaştırmak için öznitelikleri ve yöntemleri toparlamak ve yeniden adlandırmak.
- Satır içi kodun, bir program kitaplığına dahil edilmiş yöntemlere yapılan çağrılarla değiştirilmesi.

Bölüm 1 Anahtar Noktalar



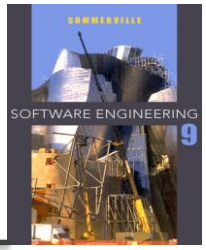
- Çevik yöntemler, hızlı geliştirmeye, yazılımın sık sürümlerine, işlem giderlerini azaltmaya ve yüksek kaliteli kod üretmeye odaklanan artımlı geliştirme yöntemleridir. Müşteriyi doğrudan geliştirme sürecine dahil ederler.
- Geliştirme için çevik veya plan odaklı bir yaklaşım kullanıp kullanmama kararı, geliştirilen yazılımın türüne, geliştirme ekibinin yeteneklerine ve sistemi geliştiren şirketin kültürüne bağlı olmalıdır.
- Ekstrem programlama, yazılımın sık sürümleri, sürekli yazılım iyileştirmesi ve geliştirme ekibine müşteri katılımı gibi bir dizi iyi programlama uygulamasını entegre eden iyi bilinen bir çevik yöntemdir.



Ders 3 - Çevik Yazılım Geliştirme

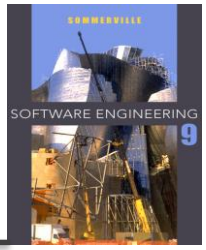
Bölüm 2

XP'de Test Etme



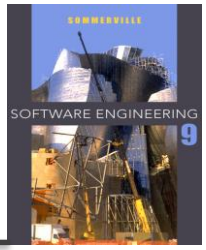
- Test, XP'nin merkezidir ve XP, her değişiklik yapıldıktan sonra programın test edildiği bir yaklaşım geliştirmiştir.
- XP test özellikleri:
  - Önce test geliştirme.
  - Senaryolardan artımlı test geliştirme.
  - Test geliştirme ve doğrulamada kullanıcı katılımı.
  - Otomatik test donanımları, yeni bir sürüm oluşturulduğunda her seferinde tüm bileşen testlerini çalıştırmak için kullanılır.

Önce Test Geliştirme



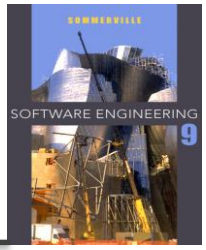
- Koddan önce testler yazmak, uygulanacak gereksinimleri açıklığa kavuşturur.
- Testler, otomatik olarak yürütülebilmeleri için veri yerine program olarak yazılır. Test, doğru şekilde yürütüldüğüne dair bir kontrol içerir.
  - Genellikle Junit gibi bir test çerçevesine dayanır.
- Tüm önceki ve yeni testler, yeni işlevsellik eklendiğinde otomatik olarak çalıştırılır, böylece yeni işlevin hatalara neden olup olmadığı kontrol edilir.

Müşteri Katılımı



- Müşterinin test sürecindeki rolü, sistemin bir sonraki sürümünde uygulanacak hikayeler için kabul testleri geliştirmeye yardımcı olmaktır.
- Ekibin bir parçası olan müşteri, geliştirme ilerledikçe testler yazar. Bu nedenle, tüm yeni kodlar, müşterinin ihtiyaç duyduğu şey olduğundan emin olmak için doğrulanır.
- Ancak, müşteri rolünü benimseyen kişilerin sınırlı zamanı vardır ve bu nedenle geliştirme ekibiyle tam zamanlı çalışamazlar. Gereksinimleri sağlamanın yeterli bir katkı olduğunu düşünebilirler ve bu nedenle test sürecine dahil olma konusunda isteksiz olabilirler.

Doz Kontrolü İçin Test Senaryosu Açıklaması



Test 4: Dose Checking

Input:

1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

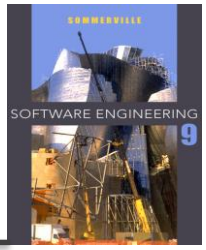
Tests:

1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose × frequency is too high and too low.
4. Test for inputs where single dose × frequency is in the permitted range.

Output:

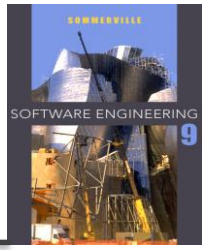
OK or error message indicating that the dose is outside the safe range.

Test Otomasyonu



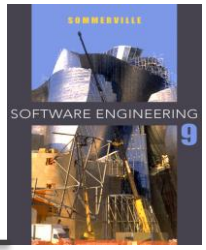
- Test otomasyonu, görev uygulanmadan önce testlerin yürütülebilir bileşenler olarak yazılması anlamına gelir:
  - Bu test bileşenleri bağımsız olmalı, test edilecek girdinin sunumunu simüle etmeli ve sonucun çıktı spesifikasyonunu karşılayıp karşılamadığını kontrol etmelidir. Otomatikleştirilmiş bir test çerçevesi (örn. Junit), yürütülebilir testler yazmayı ve yürütme için bir dizi test göndermeyi kolaylaştıran bir sistemdir.
- Test otomatikleştirildiği için, her zaman hızlı ve kolay bir şekilde yürütülebilecek bir dizi test vardır:
  - Sisteme herhangi bir işlevsellik eklendiğinde, testler çalıştırılabilir ve yeni kodun ortaya çıkardığı sorunlar anında yakalanabilir.

XP Test Zorlukları



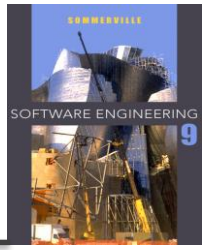
- Programcılar programlamayı teste tercih ederler ve bazen test yazarken kestirme yollar kullanırlar. Örneğin, oluşabilecek tüm olası istisnaları kontrol etmeyen eksik testler yazabilirler.
- Bazı testleri aşamalı olarak yazmak çok zor olabilir. Örneğin, karmaşık bir kullanıcı arayüzünde, ekranlar arasında 'görüntüleme mantığını' ve iş akışını uygulayan kod için birim testleri yazmak genellikle zordur.
- Bir dizi testin eksiksiz olup olmadığına karar vermek zordur. Çok sayıda sistem testiniz olsa da, test setiniz tam kapsam sağlamayabilir.

Çiftler Programı



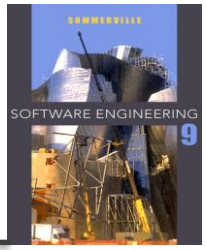
- XP'de programcılar kod geliştirmek için birlikte oturarak çiftler halinde çalışırlar.
- Bu, ortak kod sahipliği geliştirmeye yardımcı olur ve bilgiyi ekibe yayar.
- Her bir kod satırına 1'den fazla kişi tarafından bakıldığı için gayri resmi bir inceleme süreci olarak hizmet eder.
- Tüm ekip bundan faydalanabileceği için yeniden düzenleme yapmayı teşvik eder.
- Ölçümler, eşli programlama ile geliştirme üretkenliğinin bağımsız çalışan iki kişininkine benzer olduğunu göstermektedir.

Çiftler Programı



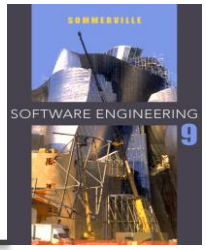
- Çift programlamada, programcılar yazılımı geliştirmek için aynı iş istasyonunda birlikte otururlar.
- Çiftler dinamik olarak oluşturulur, böylece tüm ekip üyeleri geliştirme sürecinde birbirleriyle çalışır.
- İkili programlama sırasında gerçekleşen bilgilerin paylaşımı, ekip üyeleri ayrıldığında bir projeye yönelik genel riskleri azalttığı için çok önemlidir.
- Çift programlama mutlaka verimsiz değildir ve birlikte çalışan bir çiftin, ayrı çalışan 2 programcıdan daha verimli olduğuna dair kanıtlar vardır.

Çift Programlamanın Avantajları



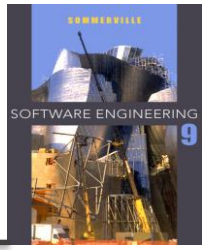
- Sistem için kolektif mülkiyet ve sorumluluk fikrini destekler.
  - Kod ile ilgili sorunlardan şahıslar sorumlu tutulamaz. Bunun yerine, ekibin bu sorunları çözmek için ortak sorumluluğu vardır.
- Her bir kod satırına en az iki kişi baktığı için gayri resmi bir inceleme süreci olarak hareket eder.
- Bir yazılım geliştirme süreci olan yeniden düzenlemeyi desteklemeye yardımcı olur.
  - Eşli programlama ve kolektif sahipliğin kullanıldığı yerlerde, diğerleri yeniden düzenleme işleminden hemen faydalanır, bu nedenle süreci desteklemeleri muhtemeldir.

Çevik Proje Yönetimi



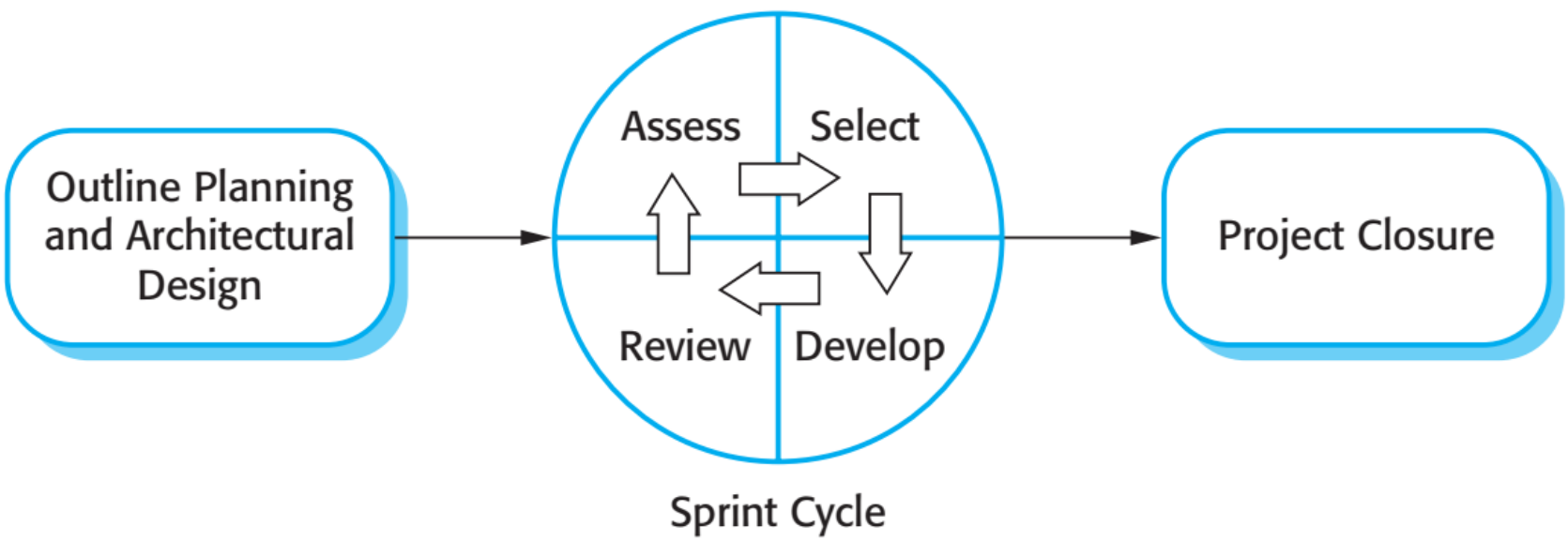
- Yazılım proje yöneticilerinin temel sorumluluğu, yazılımın zamanında ve proje için planlanan bütçe dahilinde teslim edilmesi için projeyi yönetmektir.
- Proje yönetimine yönelik standart yaklaşım, plan odaklıdır. Yöneticiler proje için neyin teslim edilmesi gerektiğini, ne zaman teslim edilmesi gerektiğini ve proje çıktılarının geliştirilmesi üzerinde kimin çalışacağını gösteren bir plan hazırlar.
- Çevik proje yönetimi, artan gelişime ve çevik yöntemlerin belirli güçlü yönlerine uyarlanmış farklı bir yaklaşım gerektirir.

Scrum

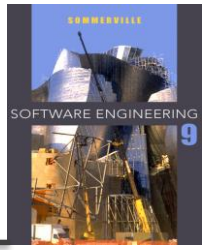


- Scrum yaklaşımı genel bir çevik yöntemdir ancak odak noktası, belirli çevik uygulamalardan ziyade yinelemeli geliştirmeyi yönetmektir.
- Scrum'da üç aşama vardır.
  - İlk aşama, proje için genel hedefleri belirlediğiniz ve yazılım mimarisini tasarladığınız bir taslak planlama aşamasıdır.
  - Bunu, her döngünün sistemin bir artışını geliştirdiği bir dizi sprint döngüsü izler.
  - Proje kapanış aşaması projeyi tamamlar, sistem yardım çerçeveleri ve kullanıcı kılavuzları gibi gerekli belgeleri tamamlar ve projeden öğrenilen dersleri değerlendirir.

Scrum Süreci

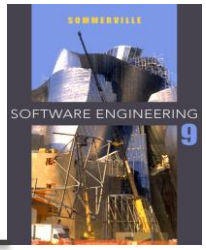


Sprint Döngüsü



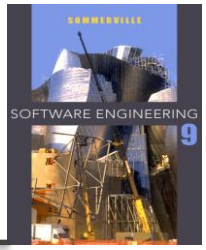
- Sprintler sabit uzunluktadır, normalde 2-4 haftadır. XP'de sistemin bir sürümünün geliştirilmesine karşılık gelirler.
- Planlama için başlangıç noktası, proje üzerinde yapılacak işlerin listesi olan ürün birikimidir.
- Seçim aşaması, sprint sırasında geliştirilecek özellikleri ve işlevleri seçmek için müşteriyle birlikte çalışan tüm proje ekibini içerir.

Sprint Döngüsü



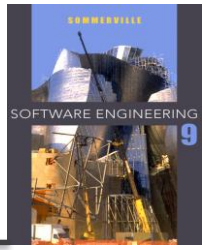
- Bunlar kararlaştırıldıktan sonra ekip, yazılımı geliştirmek için kendilerini organize eder. Bu aşamada, ekip müşteriden ve organizasyondan izole edilir ve tüm iletişimler sözde 'Scrum ustası' aracılığıyla gerçekleştirilir.
- Scrum ustasının rolü, geliştirme ekibini dış dikkat dağıtıcı unsurlardan korumaktır.
- Sprint sonunda yapılan iş gözden geçirilir ve paydaşlara sunulur. Bir sonraki sprint döngüsü başlar.

Scrum'da Takım Çalışması



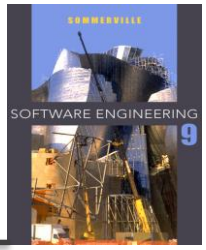
- 'Scrum ustası', günlük toplantılar düzenleyen, yapılacak işlerin birikimini izleyen, kararları kaydeden, iş yığınına göre ilerlemeyi ölçen ve ekip dışındaki müşterilerle ve yönetimle iletişim kuran bir kolaylaştırıcıdır.
- Tüm ekip, tüm ekip üyelerinin bilgi paylaştığı, son toplantıdan bu yana kaydettikleri ilerlemeleri, ortaya çıkan sorunları ve ertesi gün için planlananları anlattıkları kısa günlük toplantılara katılır.
  - Bu, takımdaki herkesin neler olup bittiğini bildiği ve sorunlar ortaya çıkarsa, bunlarla başa çıkmak için kısa vadeli çalışmaları yeniden planlayabileceği anlamına gelir.

Scrum Avantajları



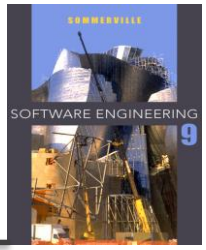
- Ürün, yönetilebilir ve anlaşılır parçalara bölünmüştür.
- Kararsız gereksinimler ilerlemeyi engellemez.
- Tüm ekip her şeyi görebilir ve sonuç olarak ekip iletişimi geliştirilir.
- Müşteriler, ürün artışlarının zamanında teslim edildiğini görür ve ürünün nasıl çalıştığına dair geri bildirim alır.
- Müşteriler ve geliştiriciler arasında güven tesis edilir ve herkesin projenin başarılı olmasını beklediği pozitif bir kültür oluşturulur.

Çevik Yöntemleri Ölçeklendirme



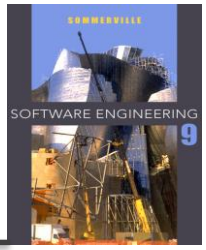
- Çevik yöntemlerin, küçük bir eş konumlu ekip tarafından geliştirilebilen küçük ve orta ölçekli projeler için başarılı olduğu kanıtlanmıştır.
- Bazen bu yöntemlerin başarısının, herkesin birlikte çalıştığı zaman mümkün olan gelişmiş iletişim sayesinde geldiği tartışılır.
- Çevik yöntemlerin ölçeğini büyütmek, bunların, belki de farklı yerlerde çalışan birden çok geliştirme ekibinin olduğu daha büyük, daha uzun projelerle başa çıkacak şekilde değiştirilmesini içerir.

Büyük Sistem Geliştirme



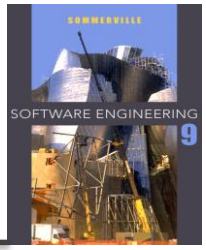
- Büyük sistemler genellikle, her sistemi ayrı ekiplerin geliştirdiği ayrı, iletişim halindeki sistemlerin koleksiyonlarıdır. Sıklıkla, bu ekipler farklı yerlerde, bazen farklı saat dilimlerinde çalışıyorlar.
- Büyük sistemler, 'kahverengi alan sistemleridir', yani mevcut bir dizi sistemi içerir ve bunlarla etkileşime girer. Sistem gereksinimlerinin çoğu bu etkileşimle ilgilidir ve bu nedenle kendilerini esnekliğe ve aşamalı geliştirmeye gerçekten adamazlar.
- Bir sistem oluşturmak için birkaç sistemin entegre edildiği yerlerde, geliştirmenin önemli bir kısmı orijinal kod geliştirmeden ziyade sistem konfigürasyonu ile ilgilidir.

Büyük Sistem Geliştirme



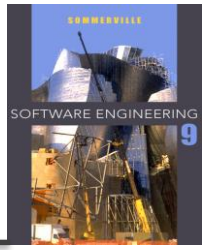
- Büyük sistemler ve geliştirme süreçleri, genellikle geliştirilme şekillerini sınırlayan dış kurallar ve düzenlemelerle sınırlandırılır.
- Büyük sistemler uzun bir tedarik ve geliştirme süresine sahiptir. İnsanların kaçınılmaz olarak başka işlere ve projelere geçmesi nedeniyle, o dönem boyunca sistemi bilen uyumlu ekipler oluşturmak zordur.
- Büyük sistemler genellikle çeşitli paydaşlara sahiptir. Tüm bu farklı paydaşları geliştirme sürecine dahil etmek neredeyse imkansızdır.

Ölçekleme Ve Büyütme



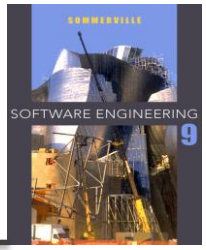
- 'Ölçek büyütme', küçük bir ekip tarafından geliştirilemeyen büyük yazılım sistemleri geliştirmek için çevik yöntemlerin kullanılmasıyla ilgilidir.
- 'Ölçeklendirme', çevik yöntemlerin uzun yıllara dayanan yazılım geliştirme deneyimine sahip büyük bir kuruluşta nasıl uygulanabileceğiyle ilgilidir.
- Çevik yöntemleri ölçeklendirirken çevik temelleri korumak çok önemlidir:
  - Esnek planlama, sık sistem sürümleri, sürekli entegrasyon, test odaklı geliştirme ve iyi ekip iletişimi.

Büyük Sistemlere Ölçekleme



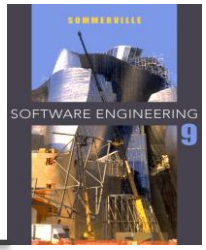
- Büyük sistem geliştirme için sadece sistemin koduna odaklanmak mümkün değildir. Daha fazla ön tasarım ve sistem dokümantasyonu yapmanız gerekir.
- Ekipler arası iletişim mekanizmaları tasarlanmalı ve kullanılmalıdır. Bu, ekip üyeleri arasında düzenli telefon ve video konferanslarını ve ekiplerin ilerleme konusunda birbirlerini güncelledikleri sık, kısa elektronik toplantıları içermelidir.
- Herhangi bir geliştirici bir değişikliği her kontrol ettiğinde tüm sistemin oluşturulduğu sürekli entegrasyon pratikte imkansızdır. Ancak, sık sistem kurulumlarının ve sistemin düzenli sürümlerinin sürdürülmesi çok önemlidir.

Büyük Şirketlere Ölçeklendirme



- Çevik yöntemler konusunda deneyimi olmayan proje yöneticileri, yeni bir yaklaşımın riskini kabul etme konusunda isteksiz olabilir.
- Büyük kuruluşlar genellikle tüm projelerin uyması beklenen kalite prosedürlerine ve standartlarına sahiptir ve bürokratik yapıları nedeniyle, bunlar büyük olasılıkla çevik yöntemlerle uyumsuzdur.
- Çevik yöntemler, ekip üyeleri nispeten yüksek bir beceri seviyesine sahip olduğunda en iyi şekilde çalışır. Bununla birlikte, büyük kuruluşlar içinde, çok çeşitli beceriler ve yetenekler olması muhtemeldir.
- Özellikle geleneksel sistem mühendisliği süreçlerini kullanma konusunda uzun bir geçmişe sahip olan kuruluşlarda, çevik yöntemlere kültürel direnç olabilir.

Bölüm 2 Anahtar Noktalar



- Ekstrem programlamanın belirli bir gücü, bir program özelliği oluşturulmadan önce otomatik testlerin geliştirilmesidir. Bir sisteme bir artış entegre edildiğinde tüm testler başarıyla yürütülmelidir.
- Scrum yöntemi, bir proje yönetimi çerçevesi sağlayan çevik bir yöntemdir. Bir sistem artışı geliştirildiğinde sabit zaman periyotları olan bir dizi sprint etrafında ortalanır.
- Büyük sistemler için çevik yöntemleri ölçeklendirmek zordur. Büyük sistemler, ön tasarıma ve bazı belgelere ihtiyaç duyar.