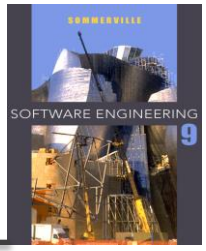
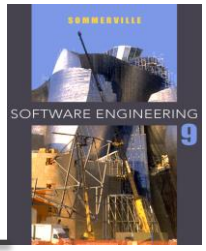


Bölüm 3 – Çevik (Agile) Yazılım Geliştirme

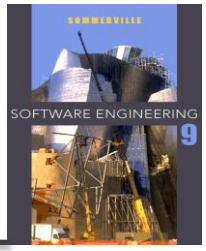
Ders 1

Konular



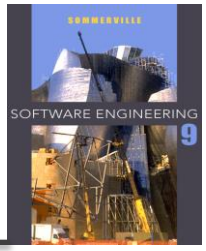
- ✧ Çevik Yöntemler
- ✧ Plan Tabanlı ve Çevik Geliştirme
- ✧ Extreme Programlama
- ✧ Çevik Proje Yönetimi
- ✧ Çevik Yöntemleri Ölçeklendirme

Hızlı (Rapid) yazılım geliştirme



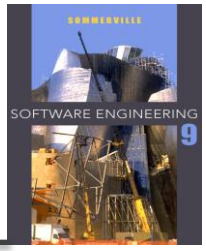
- Hızlı geliştirme ve teslim, yazılım sistemleri için en önemli ihtiyaç
  - İş hayatı hızla değişir. Dolayısıyla, değişmeyen yazılım ihtiyaçlarına sahip olması imkansız.
  - Yazılımlar hızlı bir biçimde, değişen ihtiyaçlara göre güncellenebilmeli
- Hızlı yazılım geliştirme
  - Tanımlama, tasarım ve gerçekleştirim iç içe geçmiştir
  - Sistem, müşterinin de katkıda bulunduğu sürümler serisi olarak geliştirilir.
  - Kullanıcı arayüzleri genellikle grafiksel araçlar veya bütünleşik geliştirme ortamları kullanılarak oluşturulur.

Çevik yöntemler



- 1980'ler ve 90'ların yazılım geliştirme yöntemlerinden kaynaklanan müşteri memnuniyetsizliği çevik yöntemlerin gelişmesine öncülük etmiştir. Bu yöntemler:
  - Tasarımdan ziyade koda odaklanmak
  - İteratif olarak yazılım geliştirme yaklaşımında bulunmak
  - İhtiyacı karşılayacak bir yazılımı çabucak teslim etmek ve ihtiyaçlara göre bu yazılımı güncellemek
- Çevik yöntemlerin amacı yazılım geliştirme sürecindeki yükü azaltarak (ör: dokümantasyonu sınırlandırarak) değişen ihtiyaçlara hızlıca ve çok fazla yeniden çalışma yükü getirmeden cevap verebilmektir.

Çevik manifesto

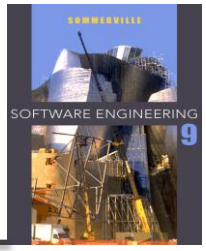


- Bizler daha iyi yazılım geliştirme yollarını uygulayarak ve başkalarının da uygulamasına yardım ederek ortaya çıkartıyoruz. Bu çalışmaların sonucunda:

  - Süreçler ve araçlardan ziyade bireyler ve etkileşimlere
    Kapsamlı dökümantasyondan ziyade çalışan yazılıma
    Sözleşme pazarlıklarından ziyade müşteri ile işbirliğine
    Bir plana bağlı kalmaktan ziyade değişime karşılık vermeye
    değer vermeye kanaat getirdik.

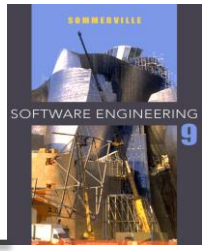
- Özetle, sol taraftaki maddelerin değerini kabul etmekle birlikte, sağ taraftaki maddeleri daha değerli bulmaktayız.
  - Fazlası için: http://www.agilemanifesto.org/iso/tr/

Çevik Yöntemlerin Prensipleri



Prensip	Açıklama
Müşterinin katılımı	Müşteriler geliştirme sürecinin her aşamasına katkı sağlamalı ve öncelikleri belirlemeli.
Artırımlı teslim	Yazılım küçük sürümler ile ve her sürümde müşterinin istediği eklemeler yapılarak geliştirilmeli.
İnsan odaklı olmak (süreç odaklı değil)	Geliştirme ekibinin yeteneklerinin farkına varılmalı ve kullanmaları sağlanmalı. Ekip üyelerinin, kendi çalışma yöntemlerine izin verilmeli.
Değişiklikleri benimse	Sistem ihtiyaçlarının değişeceği gerçeğini kabul et ve sistemini buna göre tasarla.
Basitliği koru	Yazılım geliştirilirken ve bu süreç yönetilirken basitliği koru.

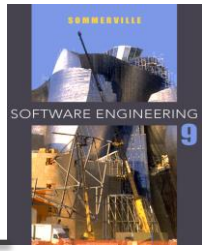
Çevik yöntemlerin uygulanabilirliği



- Küçük veya orta ölçekli yazılımlar üreten bir firmada.
- Bir kuruluş için özel bir sistem geliştirirken:

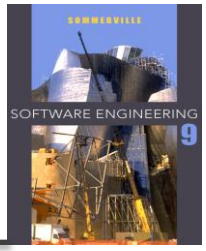
  geliştirme sürecinin müşteriden alınan taahhüt ile gerçekleştirilebileceği ve harici kuralların/kamu regülasyonlarının yazılımı çok fazla etkilemediği durumlarda.
- Çevik yöntemler küçük ve birbirleri ile yakın çalışan takımlara odaklandığı için bu yöntemlerin geniş ölçekli sistemler için kullanılması problemler yaratabilir.

Çevik yöntemlerdeki problemler



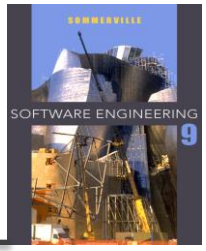
- Müşterilerin sürece katkıda bulunmalarını sağlamak zor olabilir.
- Takım üyeleri, iş arkadaşları ile sıkı bir biçimde çalışmaya yatkın olmayabilir.
- Birden fazla paydaş varsa, istekleri önceliklendirmek zor olabilir.
- Basitliği korumak fazladan efor ister.
- İmzalanan anlaşmalar, diğer iteratif geliştirme yöntemlerinde olduğu gibi sıkıntı çıkarabilir.

Çevik yöntemler ve yazılım bakımı



- Çoğu kuruluş yeni bir yazılım sistemi yapmak için harcayacağından daha fazlasını var olanın bakımını yapmak için harcar (bu eğilimdedir). Eğer çevik yöntemler başarılı olmak istiyorsa, yeni bir yazılım geliştirmenin yanı sıra var olanın bakımını yapma konusunda destek sunmalı.
- İki temel konu:
  - Çevik yaklaşımla geliştirilen sistemler sürdürülebilir mi (bakımı yapılabilir mi)
  - Çevik yaklaşım, müşterinin değişen isteklerine cevap vermek üzere geliştirilen bir sistemde efektif olarak kullanılabilir mi?
- Geliştirme ekibi parçalandıysa problem var.

Plan tabanlı ve çevik geliştirme



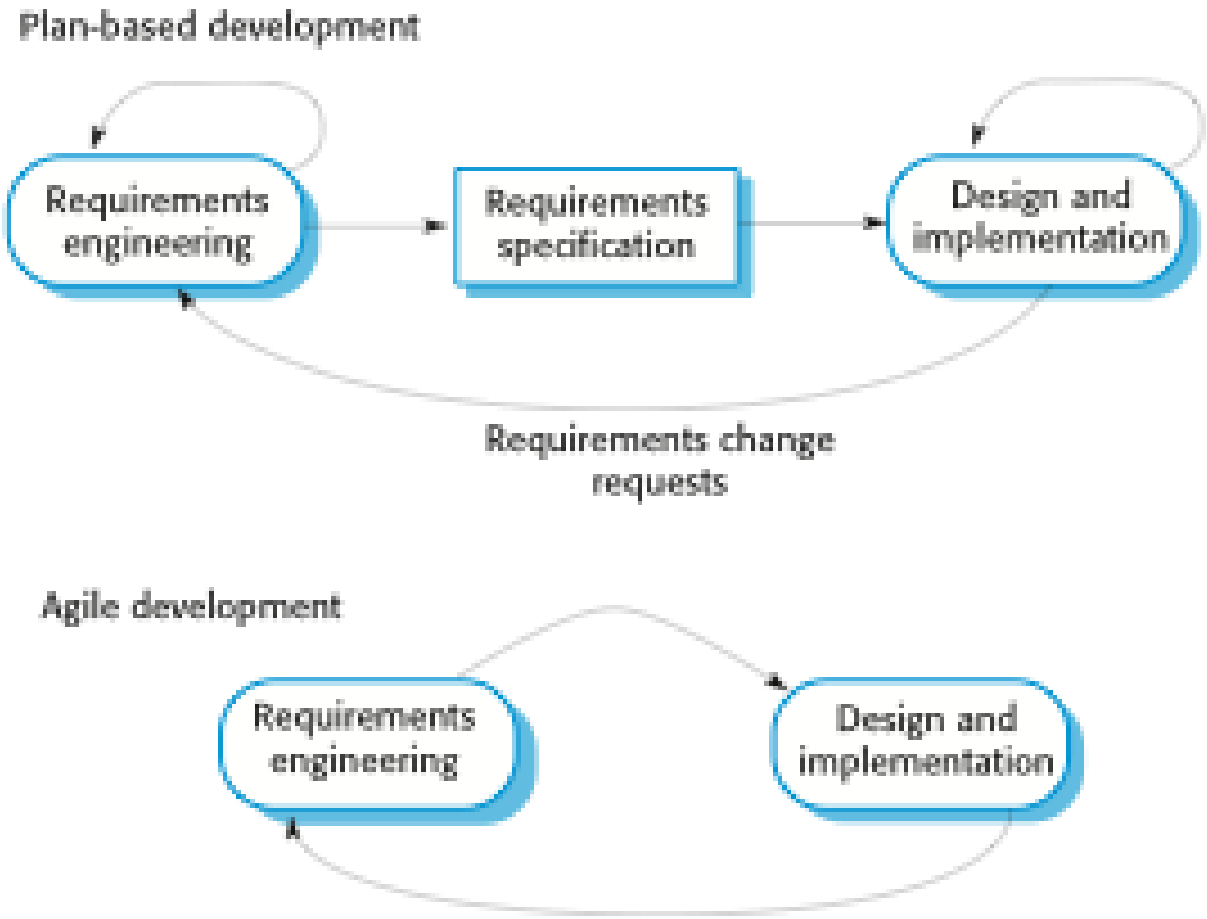
✧ Plan tabanlı geliştirme

- Ayrı ayrı geliştirme aşamaları vardır. Her bir aşamada ne yapılacağı bellidir.
- Yalnızca çağlayan modeli değil. Artırımlı model de olabilir.

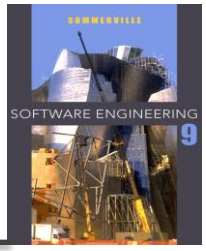
✧ Çevik geliştirme

- Tanımlama, tasarım, geliştirme ve test aşamaları iç içe geçmiştir ve geliştirme sürecinin çıktıları, yazılım geliştirme esnasında karşılıklı müzakere ile belirlenir.

Plan tabanlı ve Çevik Tanımlama

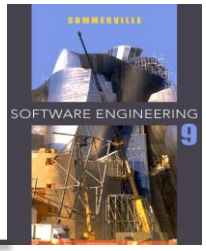


Teknik, insani ve organizasyonel konular



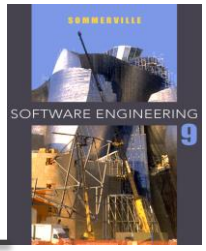
- Çoğu proje plan tabanlı ve çevik süreçlerden unsurlar içerir. Bunlar aşağıdakilere bağlıdır:
  - Gerçekleştirme aşamasından önce çok detaylı tanımlama ve tasarım gerekliyse plan tabanlı yaklaşım kullanılabilir.
  - Yazılımı küçük sürümler halinde geliştirmek ve müşteriden geri dönüşleri almak mümkünse çevik tabanlı yaklaşım kullanılabilir.
  - Geliştirdiğiniz sistemin büyüklüğü ne kadar? Çevik yöntemler, birbirleri ile yakın ilişki içerisinde olan ve bir arada bulunan geliştirme ekipleri için uygundur. Büyük çaplı projelerde bu mümkün olmayabilir.

Teknik, insani ve organizasyonel konular



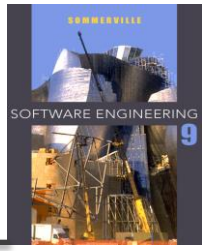
- Hangi tip sistem geliştiriliyor?
  - Plan tabanlı yaklaşım, gerçekleştirimden önce çok detaylı analiz gerektiren durumlarda kullanışlıdır. (karmaşık zamanlama gerektiren gerçek zamanlı sistemler gibi)
- Sistemin yaşam süresi ne kadar olacak?
  - Uzun ömürlü sistemler daha detaylı tasarım dokümanlarına gerek duyabilir.
- Geliştirme takımının organizasyonu nasıl?
  - Eğer geliştirici takım dağıtık ise veya geliştirme işinin bir kısmı dışarıdan hizmet olarak alınıyorsa bu ekibin birbirlerini anlayabilmeleri için detaylı tasarım dokümanlarına ihtiyaç olur.

Teknik, insani ve organizasyonel konular



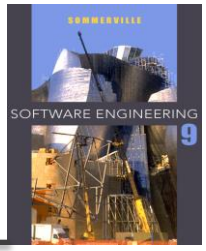
- Kültürel ve organizasyonel konular sistem geliştirmesini etkiler mi?
  - Geleneksel mühendislik firmaları plan tabanlı geliştirme kültürüne sahiptir.
- Geliştirme ekibindekiler ne kadar iyi?
  - Çevik yaklaşımın uygulandığı durumlarda ekibin programlama becerisinin daha yüksek olması gerektiği tartışılır. Çünkü, plan tabanlı yaklaşımda olduğu gibi detaylı tasarımı kodlama gibi basit bir işten daha fazlası gerekir.
- Sistem, kanunlar ve düzenlemelerden etkilenecek mi?
  - Geliştirdiğiniz sistem bir otorite tarafından denetlenecekse çok daha detaylı dokümantasyona ihtiyacınız olacaktır.

Extreme programlama



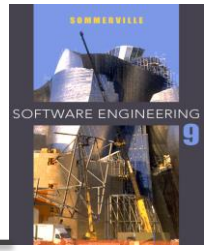
- Muhtemelen en iyi bilinen ve en sık kullanılan çevik yöntem.
- Extreme Programlama (XP) artırımlı geliştirmede «extreme» bir yol izler
  - Günde birkaç tane yeni sürüm geliştirilebilir;
  - Her iki haftada bir yeni sürüm müşteriye teslim edilir;
  - Her bir sürüm için bütün testler yapılır ve ancak bütün testler başarılı olursa sürüm teslim edilir.

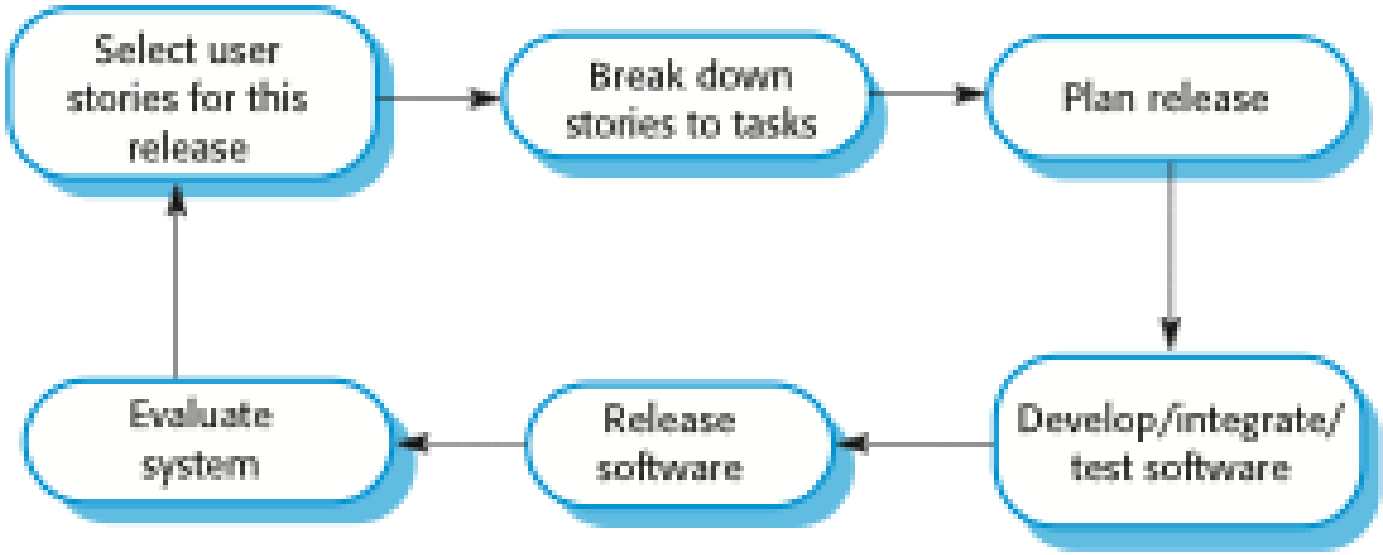
XP ve çevik prensipler



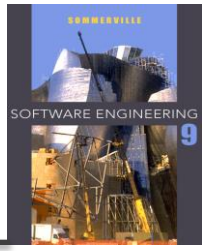
- Küçük ve sık sistem sürümleri sayesinde artırımlı yöntem benimsenir.
- Müşterinin geliştirme sürecine katılımının anlamı müşterinin tam zamanlı olarak ekibe dahil olmasıdır.
- Süreç değil insan desteklenir. Bunun için pair programming (eş olarak program geliştirme) yöntemi kullanılır.
- Değişiklikler, rutin sistem sürümleri ile gerçekleştirilir.
- Kodun basitliği refactoring ile korunur.

Extreme programlama sürüm döngüsü



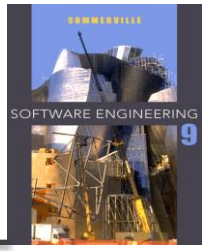


Extreme programlamanın uygulanması



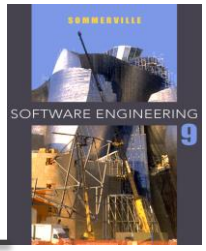
Prensip veya uygulama	Açıklama
Artırımlı planlama	İhtiyaçlar hikaye kartları şeklinde tutulur ve eldeki zamana göre ve ihtiyaçların önceliklerine göre bu hikayeler sürüme dahil edilir. Geliştiriciler bu hikayeleri geliştirme aşamasında «görevlere» böler
Küçük sürümler	İş görecek kadar fonksiyonellik sağlayan kısımlar ile geliştirilir. Sonraki sürümler sık sık ver artırımlı biçimde yeni fonksiyonellikler ekler.
Basit tasarım	Mevcut ihtiyaçları karşılayacak kadar tasarım yeterlidir.
Önce-test yaklaşımı ile geliştirme	Birim kod parçası (unit) geliştirilmeden önce onu test edecek kod geliştirilir. Böylece, geliştirme başlamadan önce ihtiyaçların iyi anlaşıldığından emin olunur.
Refactoring (sistemin çıktılarını ve işlevlerini değiştirmeden iç yapısının yeniden düzenlenerek geliştirilmesi)	Bütün geliştiricilerden sürekli olarak mümkün olduğu zaman kodları iyileştirmek üzere yeniden yazmaları istenir. Bu da kodu basit ve sürdürülebilir yapar.

Extreme programlamanın uygulanması



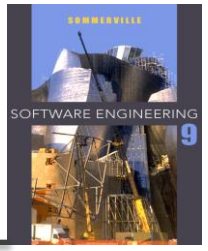
Eş programlama (Pair programming)	Geliştiriciler her zaman iki kişi olarak çalışırlar ve birbirlerinin kod geliştirmelerine yardımcı olurlar.
Birlikte sahiplenme	Geliştirici eşler sistemin bütün alanlarında çalışırlar. Böylece bütün geliştiriciler kodun tamamı üzerinde sorumluluk sahibi olur ve herhangi biri herhangi bir şeyi kolaylıkla değiştirebilir.
Sürekli entegrasyon	Bir görev tamamlandığı zaman hemen entegrasyonu sağlanmalı. Entegrasyondan sonra sistemdeki bütün birim testleri gerçekleştirilmeli.
Sürdürebilir hız	Fazla mesai kabul edilebilir değildir. Uzun vadede kod etkinliğini azaltır ve işleri karmaşık hale getirir.
Yerleşik müşteri	Son kullanıcıları temsil edebilecek biri tam zamanlı olarak geliştirme ekibi ile birliktedir. Bu kişi geliştirme ekibinin bir parçasıdır ve ihtiyaçları geliştirme ekibine getirmek ile sorumludur.

İhtiyaç senaryoları



- Kullanıcı ihtiyaçları senaryo ya da kullanıcı hikayesi olarak kartlara yazılır.
- Geliştirme ekibi bu kartları kullanarak ihtiyaçları görevlere böler. Bu görevler zamanlama ve maliyet tahminine temel teşkil eder.
- Müşteri, hikayeler arasından sonraki sürümde olmasını istediğini seçer.

Bir «reçete yazma» hikayesi



Prescribing medication

The record of the patient must be open for input. Click on the medication field and select either 'current medication', 'new medication' or 'formulary'.

If you select 'current medication', you will be asked to check the dose; If you wish to change the dose, enter the new dose then confirm the prescription.

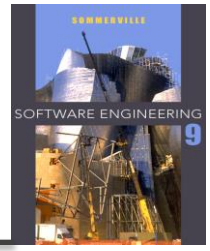
If you choose, 'new medication', the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose 'formulary', you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click 'OK' or 'Change'. If you click 'OK', your prescription will be recorded on the audit database. If you click 'Change', you reenter the 'Prescribing medication' process.

Reçete yazma hikayesi için oluşturulan örnek görev kartları



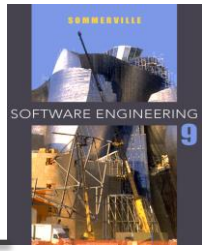
Task 1: Change dose of prescribed drug

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, lookup 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

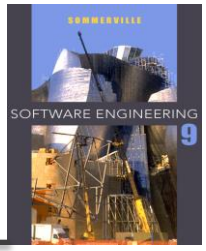


- Yazılım mühendisliğindeki geleneksel yaklaşım «değişim için tasarla» yaklaşımıdır. Değişim için gerekli olan eforu harca, böylece yazılım yaşam süresince daha az maliyetli olur.

- Ancak XP böyle düşünmez. XP için öngörülemeyen değişimler için zaman harcamak boşunadır.

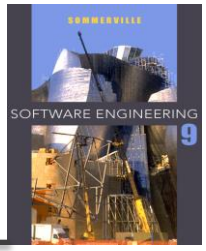
- Bunun yerine refactoring ile ilgilenir. Böylece değişiklik gerekli olduğunda yapması daha kolay olur.

Refactoring
(sistemin çıktılarını ve işlevlerini değiştirmeden iç yapısının yeniden düzenlenerek geliştirilmesi)



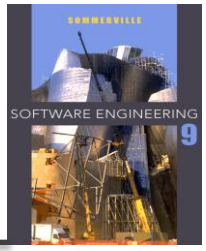
- Programlama ekibi mümkün iyileştirmeler için kodu inceler. O anda gerekli olmasa bile bu iyileştirmeyi yaparlar.
- Bu, programın anlaşılırlığını arttırır ve dokümantasyon ihtiyacını azaltır.
- Değişiklikleri yapmak kolaydır çünkü kod iyi yapılandırılmış ve anlaşılabilirdir.
- Ancak bazı değişiklikler mimari boyutunda refactoring ister. Bunu yapmak biraz pahalı olabilir.

Refactoring örnekleri



- Sınıf hiyerarşisini yeniden organize ederek tekrar eden kodları elemek.
- Fonksiyonları ve özellikleri yeniden adlandırarak daha anlaşılır hale getirmek.
- Satır içi kodları, bağımsız fonksiyonlar haline getirip kütüphane oluşturmak.

Özet

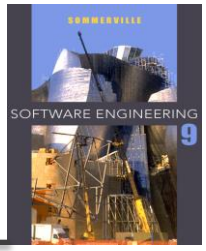


- Çevik yöntemler hızlı geliştirmeye odaklı, sık sık sürümler oluşturan, süreç yüklerini azaltan ve yüksek kaliteli kod üreten yöntemlerdir. Müşteriyi doğrudan geliştirme sürecinin bir parçası yaparlar.

- Plan tabanlı ya da çevik bir yöntemin seçimine karar vermek geliştirilecek yazılımın tipine, geliştirme ekibinin yeteneklerine, kurum kültürüne bağlıdır.

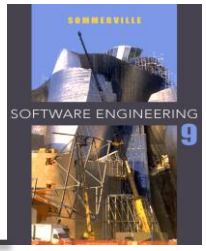
- XP en iyi bilinen çevik yöntemdir ve sık sık sürüm geliştirme, kodların sürekli iyileştirilmesi ve müşterinin geliştirme takımına katılımı gibi birçok uygulamayı (pratik yaklaşımı) barındırır.

XP’de test



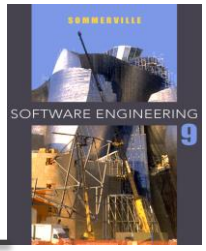
- ✧ Test XP’de işin merkezindedir.
- ✧ XP test özellikleri:
  - Önce-test yaklaşımlı geliştirme
  - Artırımlı test geliştirme.
  - Test ve doğrulama esnasında müşterinin katılımı.
  - Yeni bir sürüm üretmeden önce bütün bileşenleri otomatik test araçları ile test etmek.
    - Fazlası için: Junit, HP QuickTest Professional (QTP), Rational Functional Tester, Telerik TestStudio…

Önce-test yaklaşımlı geliştirme



- Koddan önce testi yazmak ne istediğinin iyi anlaşılmasını sağlar.
- Testler program olarak yazılır. (veri değil)
- Yazılıma yeni fonksiyonellikler eklendikçe eski ve yeni bütün testler çalıştırılır.

Müşteri Katılımı

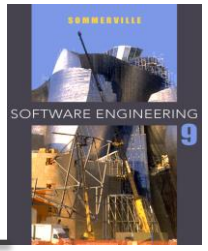


- Müşterinin testlere katılması kabul testinin geliştirilmesine yardımcı olur.

- Müşteri, testi yazan ekibin bir parçasıdır. Böylece, geliştirilen her yeni kodun müşteri ihtiyaçlarını karşıladığından emin olunur.

- Ancak insanların müşteri rolünü kabul edebilmeleri için sınırlı süreleri olabilir ve tam zamanlı çalışamayabilirler. Ayrıca, ihtiyaçların listesini vermenin yeterli olduğunu düşünebilirler ve test aşamasına katılmak istemeyebilirler.

Doz ayarlaması için test durumu tanımlaması



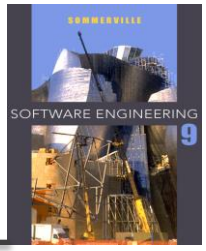
Test 4: Dose checking

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.

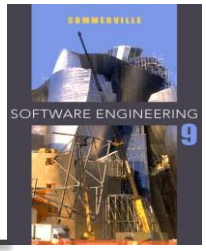
Test otomasyonu



- Test otomasyonunun anlamı testlerin uygulanmadan önce çalıştırılabilir (executable) olarak geliştirilmeleridir.

- Bir test otomatikleştirildiğinde bir test kümesi kolay ve hızlı biçimde çalıştırılabilir.

XP’de test zorlukları

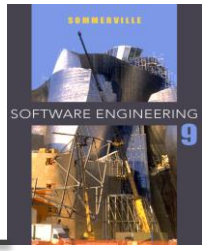


- Programcılar test programı geliştirirken bazı şeyleri eksik bırakabilirler. Örneğin, olası bütün durumları kontrol etmeyebilirler.

- Bazı durumlarda testleri artırımlı olarak yazmak çok zordur. Örneğin, karmaşık bir kullanıcı arayüzüne sahip bir sistem için iş akışlarını gösteren birim testlerini yazmak gibi.

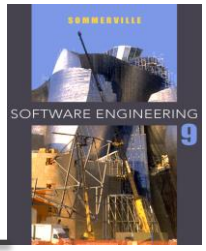
- Test edilen durumların eksiksiz olduğuna hüküm vermek biraz zordur. Birçok sistem testiniz olmasına rağmen bu testler olası bütün durumları kapsamıyor olabilir.

Eş programlama (Pair programming)



- XP'de programcılar eş olarak oturur ve kod geliştirirler.
- Böylece ortak sahiplenme artar ve bilgi ekibe yayılır.
- Ayrıca her bir satır kodun en az iki kişi tarafından kontrol edilmesini de garanti eder.
- Bütün takım faydalanacağı için refactoring'i teşvik eder.
- Ölçümler, üretim verimliliğinin ayrı ayrı program geliştiren iki kişi ile aynı olduğunu göstermektedir.

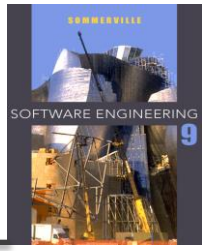
Eş programlama (Pair programming)



- Eşler dinamik olarak oluşturulurlar. Böylece bütün takım üyeleri birbirleri ile çalışmış olur.

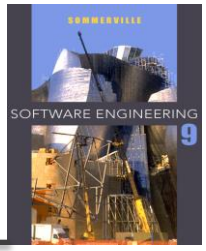
- Eş programlama esnasında bilginin yayılımı, ekip üyelerinden birinin ekipten ayrılması durumunda projenin başarısız olmaması için çok önemlidir.

Eş programlamanın avantajları



- ✧ Sistem üzerindeki ortak sahiplenme ve sorumluluk duygusunu geliştirir.
- ✧ Bir kod satırının en az iki programcı tarafından kontrol edilmesini sağlar.
- ✧ Yazılım iyileştirmeyi (refactoring) destekler.

Çevik Proje yönetimi

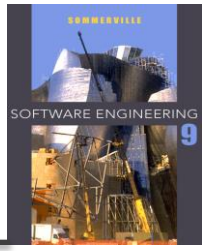


- Bir yazılım proje yöneticisinin temel amacı yazılımın zamanında ve planlanan bütçe içerisinde teslim edilebilmesidir.

- Proje yönetimindeki temel yaklaşım plan yapmaktır. Yöneticiler, neyin, ne zaman teslim edileceği ve bu işi kimin geliştireceğini gösteren planlar yapar.

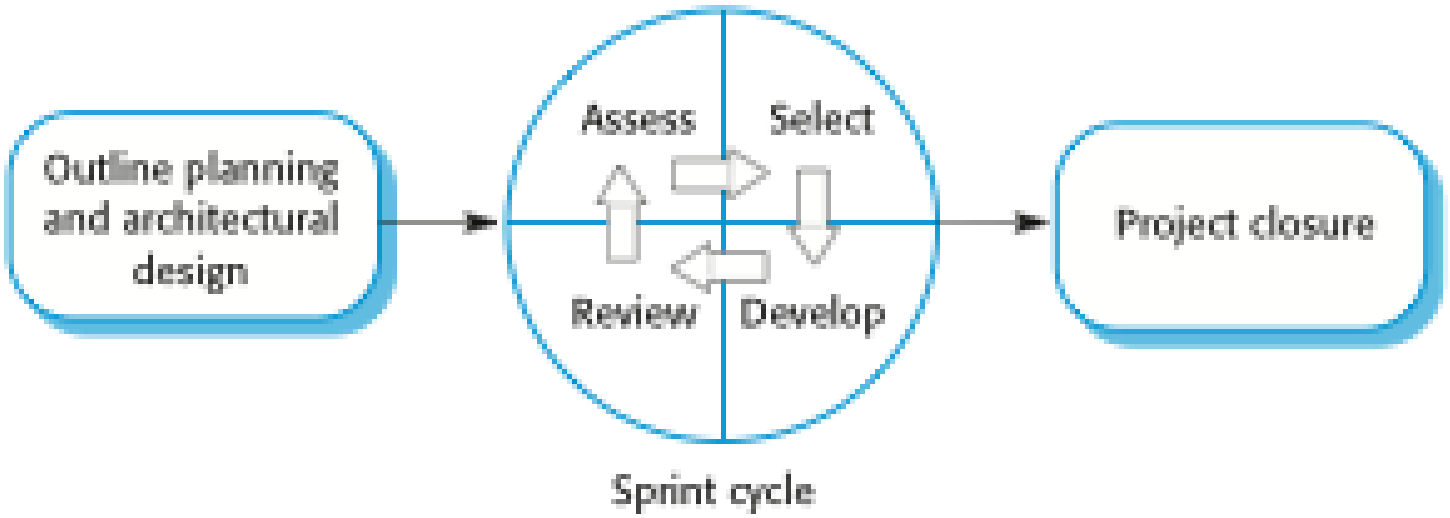
- Çevik proje yönetimi daha farklı bir yaklaşım ister. Bu yaklaşım artırımlı geliştirmeye ve çevik yöntemlerin güçlü yönlerine uygun olmalıdır.

Scrum

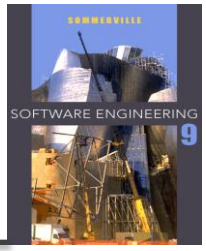


- Scrum yaklaşımı genel bir çevik yöntemdir ancak odak noktası proje geliştirme pratiklerinden çok artırımlı geliştirmenin yönetilmesidir.
- Scrum'da üç aşama vardır.
  - **İnitial** - Başlangıç aşamasında projenin genel amaçlarını belirten proje taslağı oluşturulur ve yazılım mimari yapısı belirlenir.
  - **Sprint** - Bunun ardından hızlı döngüler gelir. Her bir döngüne sistemin bir artırımı (fonksiyonellik barındıran yeni sürümü) geliştirilir.
  - **Closure** - Proje kapanış aşamasında proje bitirilir; kullanıcı klavuzları gibi dokümanlar tamamlanır ve projeden öğrenilen dersler değerlendirilir.

Scrum süreci



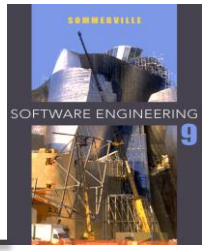
Sprint – hızlı döngüler aşaması



- Döngüler sabit uzunluktadır. Normalde 2-4 hafta. Her bir döngü bir sürümün geliştirilmesinden sorumludur.

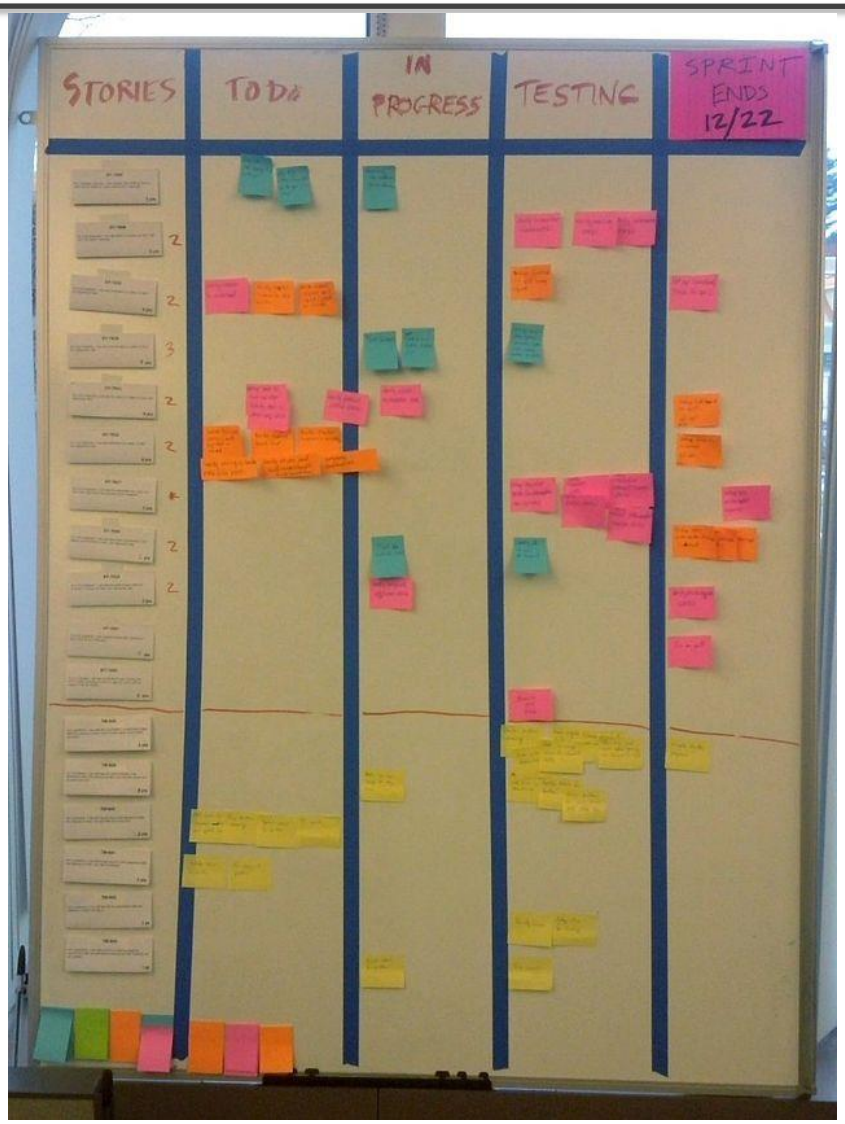
- Seçme aşaması, proje ekibi ve müşterinin katılımı ile bu döngü içerisinde hangi özelliklerin ve fonksiyonelliklerin geliştirileceğinin seçildiği aşamadır.

Sprint – hızlı döngüler aşaması

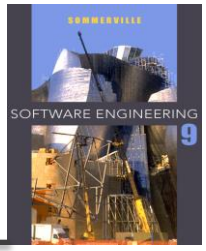


- Bir önceki madde kararlaştırıldıktan sonra takım yazılımı geliştirmek için kendi kendine organize olur.
- Bu süreçte takım müşteriden ve kurumun geri kalanından izole edilir. Bütün haberleşme, scrum-master (takımda yer almayan, ancak yürütücülük ve organizatörlük görevi olan kişi) aracılığı ile yapılır.
- Scrum-master'ın rolü, ekibin dış kaynaklı dikkat dağınıklığına uğramasını engellemektir.
- Döngünün sonunda iş gözden geçirilir ve paydaşlara sunulur. Ardından yeni döngü başlar.

Sprint – hızlı döngüler aşaması



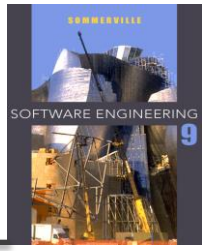
Scrum’da takım çalışması



- «Scrum master» günlük görüşmeleri ayarlar, tamamlanan işleri takip eder, ilerlemeyi ölçer ve yöneticiler ve müşteri gibi takım dışı kişilerle iletişimi sağlar.

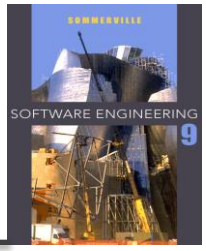
- Günlük kısa toplantılara bütün ekip üyeleri katılır; bilgi paylaşır; son görüşmeden beri kaydettiği ilerlemeyi anlatır; karşılaştığı problemleri tanımlar ve o gün için ne yapacağından bahseder.

Scrum'ın faydaları



- Ürün, yönetilebilir ve anlaşılabilir küçük parçalara bölünür.
- Bütün ekip her şeyi görür ve takımın iletişimi gelişir.
- Müşteri, her bir küçük sürümün zamanında teslim edildiğini görür ve geri bildirimde bulunur.
- Müşteri ile geliştiriciler arasındaki güven temin edilmiş olur ve projenin başarılı olacağına olan inanç herkeste oluşur.

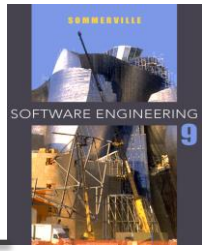
Çevik yöntemlerin ölçeklenebilirliği



- Çevik yöntemler küçük ve orta ölçekli projelerde küçük ve yakın çalışan bir ekip ile başarılı olduklarını ispatlamıştır.

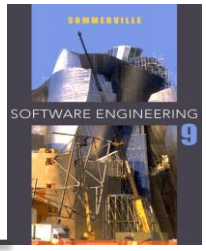
- Daha uzun süreli ve daha geniş projelerde birden fazla ekibin muhtemelen farklı yerlerde çalışması gerekebilir.

Geniş sistemlerin geliştirilmesi



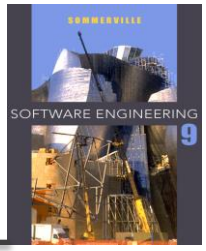
- Geniş sistemler genellikle birden fazla geliştirme takımının geliştirdiği sistemlerdir ve geliştirme takımları farklı zaman dilimlerinde bile olabilir.
- Sıklıkla geniş sistemler, zaten kullanılmakta olan başka sistemlere entegre edilmek zorundadırlar. Sistem ihtiyaçlarının büyük bir kısmı var olan sistemlerle etkileşmek ile ilgilidir. Bu sistemler bir müşteri gibi bilgi vermezler ve artırımlı geliştirme sürecini zora sokarlar.
- Birden fazla sistemin tek bir sistem oluşturmak için entegre edilmesi gerektiğinde, geliştirme eforunun büyük kısmı kod geliştirmekten ziyade sistem konfigürasyonuna harcanır.

Geniş sistemlerin geliştirilmesi



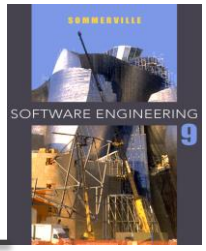
- Geniş sistemler genellikle başka bir kurumun düzenlemesi altındadır.
- Geniş sistemlerin tedarik ve geliştirme süreleri uzundur. Bu nedenle, uyumlu bir ekip kurmak zordur. Kaçınılmaz bir biçimde kişiler diğer işlere veya projelere yönlendirilirler.
- Geniş sistemlerin genellikle çeşitli paydaşları vardır. Bu paydaşların geliştirme sürecine katkıda bulunmaları pratik olarak imkansız olabilir.

Geniş sistemlerin geliştirilmesi



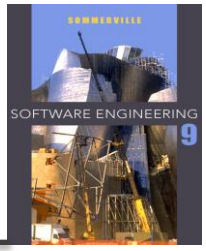
- Geniş sistemlerin geliştirilmesi için yalnızca koda odaklanmak imkansızdır. Öncesinde daha fazla tasarım ve dokümantasyona ihtiyaç vardır.
- Geliştirme ekipleri arası iletişimi sağlayacak mekanizmalar kurulmalıdır. Düzenli telefon görüşmeleri, video konferans yapmak her bir takım üyesinin diğer takımlar ile ilgili bilgi sahibi olmasını sağlar.
- Sürekli entegrasyon ve bir geliştiricinin bütün sistemi kontrol etmesi imkansızdır. Ancak, mümkün olduğu kadar sık ve düzenli sürümler üretmek faydalı olacaktır.

Geniş sistemlerin geliştirilmesi



- Çevik yöntemler ile ilgili bilgi sahibi olmayan yöneticiler, yeni bir yaklaşım kullanma riskini almak istemeyebilirler.
- Geniş kuruluşların sıklıkla kalite standartları vardır. Bu bürokratik süreçler çevik yöntemlerle uyumsuz olabilir.
- Çevik yöntemler, geliştirme ekibinin yüksek kabiliyete sahip olduğu durumlarda en iyi şekilde çalışır. Ancak geniş organizasyonlardaki çalışanların yetenekleri farklı olabilir.
- Uzun süre geleneksel yöntemler ile yazılım geliştirmiş bir kuruma çevik yazılım geliştirme yöntemlerini benimsetmek zor olabilir.

Özet



- XP'nin kuvvetli bir yönü, bir program parçası geliştirilmeden önce o parça için otomatikleştirilmiş test programının yazılmasıdır.
- Scrum yöntemi, bir proje yönetimi çerçevesi oluşturan çevik bir yöntemdir.
- Çevik yöntemlerin geniş sistemlere uygulanması zordur. Geniş sistemler açık tasarımlar ve bazı dokümanlar gerektirir.