Yazılım Süreç Modelleri (..devam)

BBS-651 Yazılım Mühendisliği

Hafta #2 (13 Ekim 2010)

Geleneksel Yazılım Süreç Modelleri

- Çağlayan (“waterfall”) modeli
- Evrimsel (“evolutionary”) model
- Bileşen-tabanlı (“component-based”) model
- Artırımlı (“incremental”) model
- Döngüsel (“spiral”) model

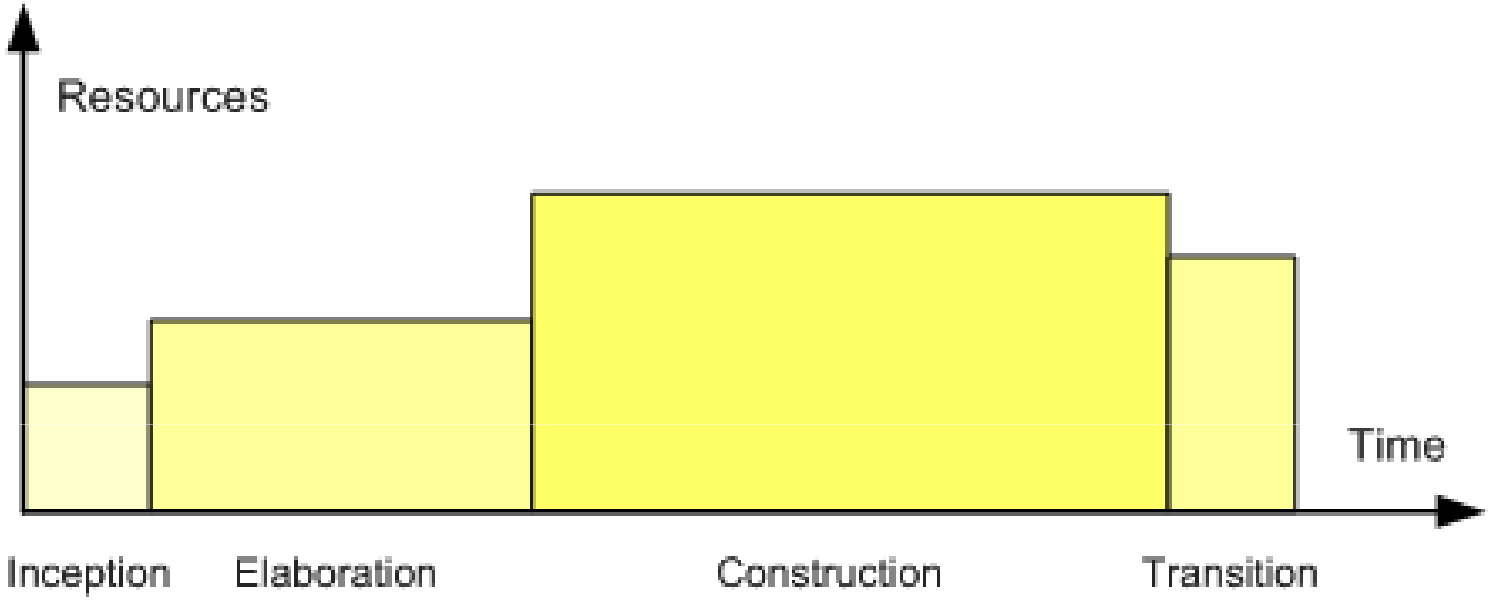
Tümleşik Süreç (“Unified Process”)

- Bir süreç çatısıdır.
  - Tekrarlı (“iterative”) ve artırımlı (“incremental”)
  - “Use-case” esaslı
  - Mimari merkezli (“architecture centric”)
  - Risk odaklı

- Statik yapısına ek olarak, kurumların ve projelerinin özelliklerine göre uyarlanabilir bir yapıya sahiptir (“tailorable”).

- Referans:
  - “*The Unified Software Development Process”,* ISBN 0-201-57169-2, 1999. Ivar Jacobson, Grady Booch, James Rumbaugh.

Tümleşik Süreç – Yaşam Döngüsü



"Inception" – Projenin kapsamını tanımla ve iş durumunu ("business case") geliştir

"Elaboration" – "Projeyi planla, özelliklerini tanımla, mimariyi dayanağı ("baseline") oluştur

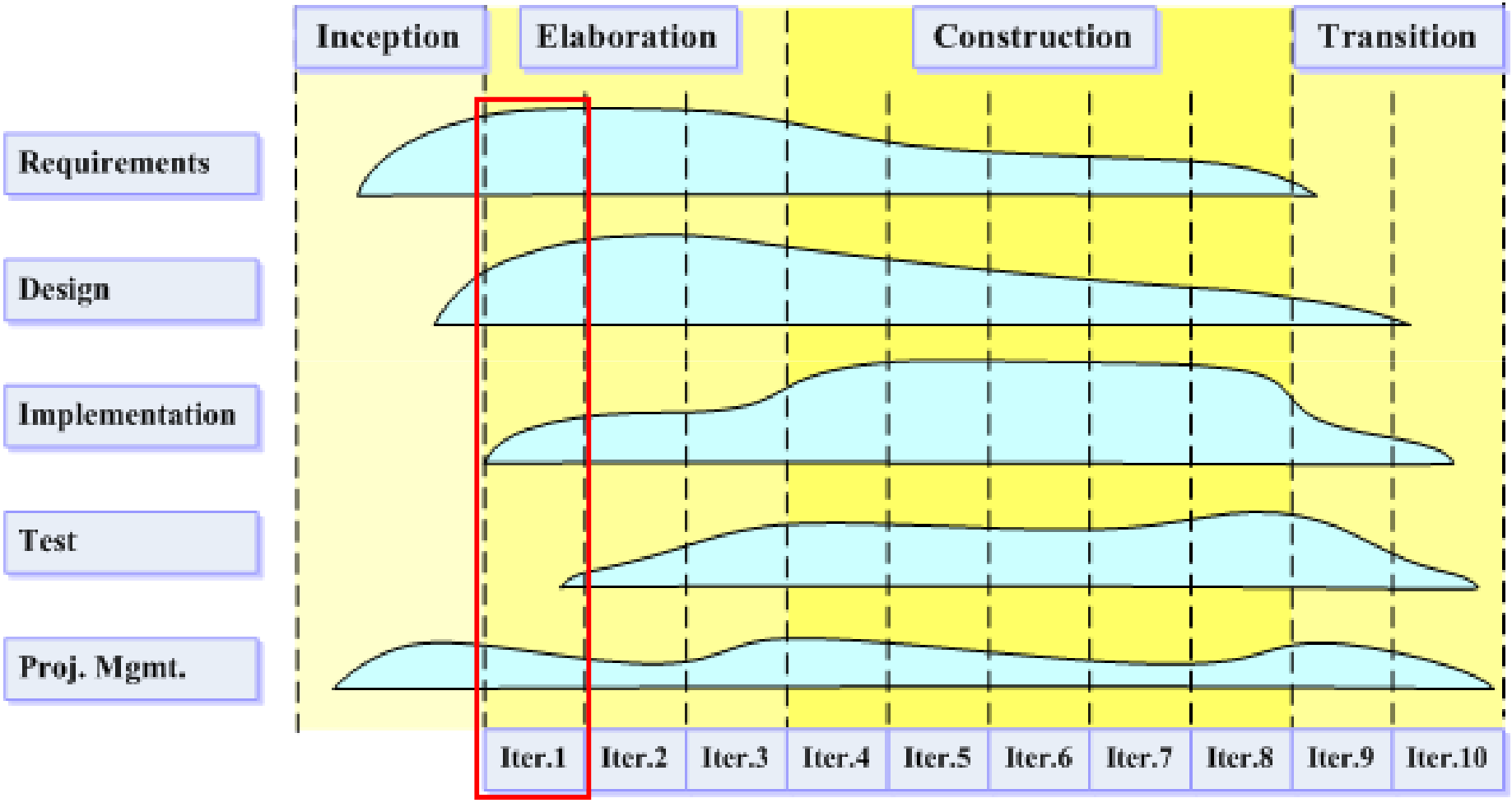
"Construction" – Ürünü gerçekleştir

"Transition" – Ürünü kullanıcılarına teslim et

Tümleşik Süreç – İlkeler

- Yazılımı tekrarlı (“iteratively”) geliştir
- Gereksinimleri yönet
- Bileşen tabanlı (“component-based”) mimari kullan
- Yazılımı görsel modelle
- Yazılım kalitesini sürekli doğrula (“verification”)
- Yazılımla ilgili değişiklikleri kontrol et

Tümleşik Süreç – Yapı



Bugünkü İçerik

- Yazılım Süreç Modelleri (..devam)
  - Rational Unified Process
  - Çevik (“agile”) modeller
    - Uçdeğer (“Extreme”) programlama, Scrum, Özellik Güdümlü Geliştirme (“Feature Driven Development), Çevik Tümleşik Süreç (“Agile Unified Process”)
  - CASE (“Computer-Aided Software Engineering”)

- Yaşam Döngüsü Süreçleri
  - Yaşam Döngüsü Süreçleri için standartlar
    - ISO/IEC 15288 (IEEE Std 15288-2008)
      - Sistem ve Yazılım Mühendisliği – Sistem Yaşam Döngüsü Süreçleri
    - ISO/IEC 12207 (IEEE Std 12207-2008)
      - Sistem ve Yazılım Mühendisliği – Yazılım Yaşam Döngüsü Süreçleri
  - Yaşam Döngüsü için süreç referans modeli
    - Tümleşik Yetenek Olgunluk Modeli (Capability Maturity Model Integration – CMMI)
      - Sistem ve yazılım geliştirme için süreç değerlendirme ve iyileştirme çatısı

Rational Unified Process

- http://ftp.cs.hacettepe.edu.tr/pub/dersler/BBS6XX/BBS651_YM/kaynaklar/RationalUnifiedProcess.zip

Dosyayı açın ve index.html'i çalıştırın.

Süreci inceleyin.

Çevik (“Agile”)
Yazılım Süreç Modelleri

Çevik (“Agile”) Yazılım Süreç Modelleri

- Çevik modeller, mevcut geleneksel modellere alternatif olarak, 1990’larda ortaya çıkmaya başlamıştır.
  - Çevik: *Kolaylık ve çabuklukla davranan, tetik, atik*. [TDK]
  - 1950’lerdeki üretim alanında verimliliğin artırılması için geliştirilen yalın yaklaşımların, yazılım sektöründe bir uzantısı olarak ortaya çıkmıştır.
- Çevik modeller kapsamında; yazılım sistemlerini etkili ve verimli bir şekilde modellemeye ve belgelendirmeye yönelik, pratiğe dayalı yöntemler yer alır.
- Bu modelleme biçiminin; kapsadığı değerler, prensipler ve pratikler sayesinde geleneksel modellemelere metotlarına göre yazılımlara daha esnek ve kullanışlı biçimde uygulanabileceği savunulmaktadır.

Çevik Yazılım Geliştirme Manifestosu

- 2001 yılında, dünyanın önde gelen çevik modellerinin temsilcileri, ortak bir zeminde buluşabilmek adına bir araya gelerek "çevik yazılım geliştirme manifestosu" nu yayınlamışlardır. Bu manifestoya göre;

  - Bireyler ve aralarındaki etkileşim, kullanılan araç ve süreçlerden;
  - Çalışan yazılım, detaylı belgelerden;
  - Müşteri ile işbirliği, sözleşmedeki kesin kurallardan;
  - Değişikliklere uyum sağlayabilmek, mevcut planı takip etmekten;

  daha önemli ve önceliklidir.

  (Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas)

Çevik Yazılım Geliştirme Prensipleri

- Çevik yazılım geliştirme manifestosu maddelerinin altında yatan temel prensipler şunlardır:
  - İlk öncelik, sürekli, kaliteli yazılım teslimatıyla müşteri memnuniyetini sağlamaktır.
  - Proje ne kadar ilerlemiş olursa olsun değişiklikler kabul edilir. Çevik yazılım süreçleri değişiklikleri müşteri avantajına dönüştürür.
  - Mümkün olduğunca kısa zaman aralıklarıyla (2–6 hafta arası) çalışan, kaliteli yazılım teslimatı yapılır.
  - Analistler, uzmanlar, yazılımcılar, testçiler, vs. tüm ekip elemanları bire bir iletişim halinde, günlük olarak birlikte çalışır.
  - İyi projeler motivasyonu yüksek bireyler etrafında kurulur. Ekip elemanlarına gerekli destek verilmeli, ihtiyaçları karşılanarak proje ile ilgili tam güvenilmelidir.
  - Ekip içerisinde kaliteli bilgi akışı için yüz yüze iletişim önemlidir.
  - Çalışan yazılım, projenin ilk gelişim ölçütüdür.
  - Çevik süreçler mümkün olduğunca sabit hızlı, sürdürülebilir geliştirmeye önem verir.
  - Sağlam teknik alt yapı ve tasarım çevikliği arttırır.
  - Basitlik önemlidir.
  - En iyi mimariler, gereksinimler ve tasarımlar kendini organize edebilen ekipler tarafından yaratılır.
  - Düzenli aralıklarla ekip kendi yöntemlerini gözden geçirerek verimliliği arttırmak için gerekli iyileştirmeleri yapar.

Çevik Yazılım Süreç Modelleri

- Uçdeğer Programlama (“Extreme Programming – XP”)
- Scrum
- Özellik Güdümlü Geliştirme (“Feature-Driven Development – FDD”)
- Çevik Tümleşik Süreç (“Agile Unified Process – AUP”)

Uçdeğer Programlama (“Extreme Programming - XP”)

- Sınırsal programlama, Kent Beck tarafından 1999 yılında bir yazılım geliştirme disiplini olarak ortaya çıkarılmıştır.
- 4 prensip etrafında toplanır: Basitlik, İletişim, Geri bildirim, Cesaret
- 12 temel pratiğin birlikte uygulanmasını gerektirir:
  - Planlama oyunu
  - Kısa aralıklı sürümler
  - Müşteri katılımı
  - Yeniden yapılandırma (“refactoring”)
  - Önce test (“test first”)
  - Ortak kod sahiplenme
  - Basit tasarım
  - Metafor
  - Eşli programlama (“pair programming”)
  - Kodlama standardı
  - Sürekli entegrasyon (“continuous integration”)
  - Haftada 40 saat çalışma

Scrum

- Jeff Sutjerland ve Ken Schawaber tarafından 1990'ların ortalarında geliştirilen, proje yönetimi ve planlama ile ilgili yöntemlere odaklı olan ve mühendislik detayları içermeyen bir modeldir.
- Yazılımı küçük birimlere (sprint) bölerek, yinelemeli olarak geliştirmeyi öngörür.
- Bir yinelemenin tanımlanması 30 günden fazla sürmemekte ve günlük 15 dakikalık toplantılarla sürekli iş takibi yapılır.
- Karmaşık ortamlarda adım adım yazılım geliştiren küçük ekipler (<20) için uygundur.
- Yüksek seviyede belirsizlik arz eden projelerde, son yıllarda daha yaygın biçimde uygulandığı ve başarılı sonuçlar ürettiği bildirilmiştir.

Özellik Güdümlü Geliştirme (“Feature Driven Development - FDD”)

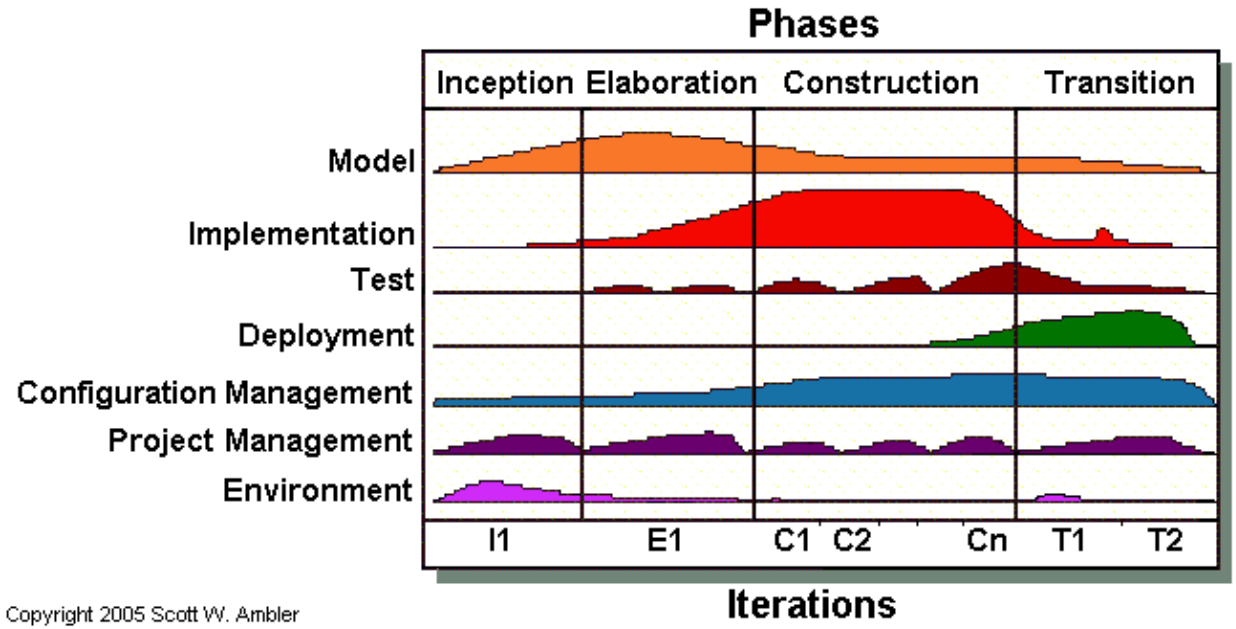
- Jeff De Luca ve Peter Coad tarafından 1997’de geliştirilen ve özellik tabanlı gerçekleştirimi esas alan bir modeldir.

- Süreç beş ana basamaktan oluşur; her bir basamak için çıkış şartları tanımlanır ve bunlara uyulur.
  - Genel sistem modelinin geliştirilmesi,
  - Özellik listesinin oluşturulması,
  - Özellik güdümlü bir planlama yapılması,
  - Özellik güdümlü tasarımın oluşturulması,
  - Özellik güdümlü geliştirmenin yapılması.

- Sisteme yeni bir özellik kazandırılmadan önce, detaylı bir tasarım çalışması yapılarak bu özelliği kapsayan mimari yapı oluşturulur.

Çevik Tümleşik Süreç (“Agile Unified Process”)

- Çevik Tümleşik Süreç (Agile Unified Process – AUP”); Rational Unified Process (RUP)’un, çevik yaklaşıma göre adapte edilerek basitleştirilmiş halidir.
- Test-güdümlü geliştirme (“test driven development – TDD”), çevik değişiklik yönetimi, veritabanını yeniden yapılandırma gibi çevik pratikleri uygulatır.
- RUP’dan farklı olarak, 7 adet disiplin içerir.



Bilgisayar Destekli Yazılım Mühendisliği (“Computer-Aided Software Engineering -- CASE”)

“CASE (Computer-Aided Software Engineering)” Nedir? (Bilgisayar Destekli Yazılım Mühendisliği)

- Yazılım süreci etkinlikleri için otomatik destek sağlamak üzere geliştirilmiş yazılım araçlarıdır.
- CASE sistemleri genelde yöntemleri desteklemek için kullanılır.
  - “Upper-CASE”
    - Analiz ve tasarım gibi erken geliştirme etkinliklerini destekleyen araçlardır.
  - “Lower-CASE”
    - Programlama, hata ayıklama ve test gibi geliştirme etkinliklerini destekleyen araçlardır.

“CASE” Ne Yapar, Ne Yapamaz?

- CASE teknolojisi, yazılım sürecini desteklemek üzere gelişme göstermiştir.
  - Sistem modelini geliştirme için grafik editörler
  - Tasarım öğelerini yönetmek için veri sözlüğü
  - Kullanıcı arayüzü oluşturmak için grafik kullanıcı arayüzü yazılımları
  - Programdaki hataları bulmak için hata ayıklayıcılar
  - Detay tasarımdan kod oluşturan dönüştürücüler

- Ancak bu gelişmelerden hiçbiri, yazılım sürecinin insana bağımlı unsurlarını adresleyemez.
  - Yazılım mühendisliği bilişsel algılama ve ifade gerektirir.
  - Yazılım geliştirme ekip işidir ve özellikle büyük kapsamlı projelerde, proje zamanının önemli kısmı iletişimle geçer.

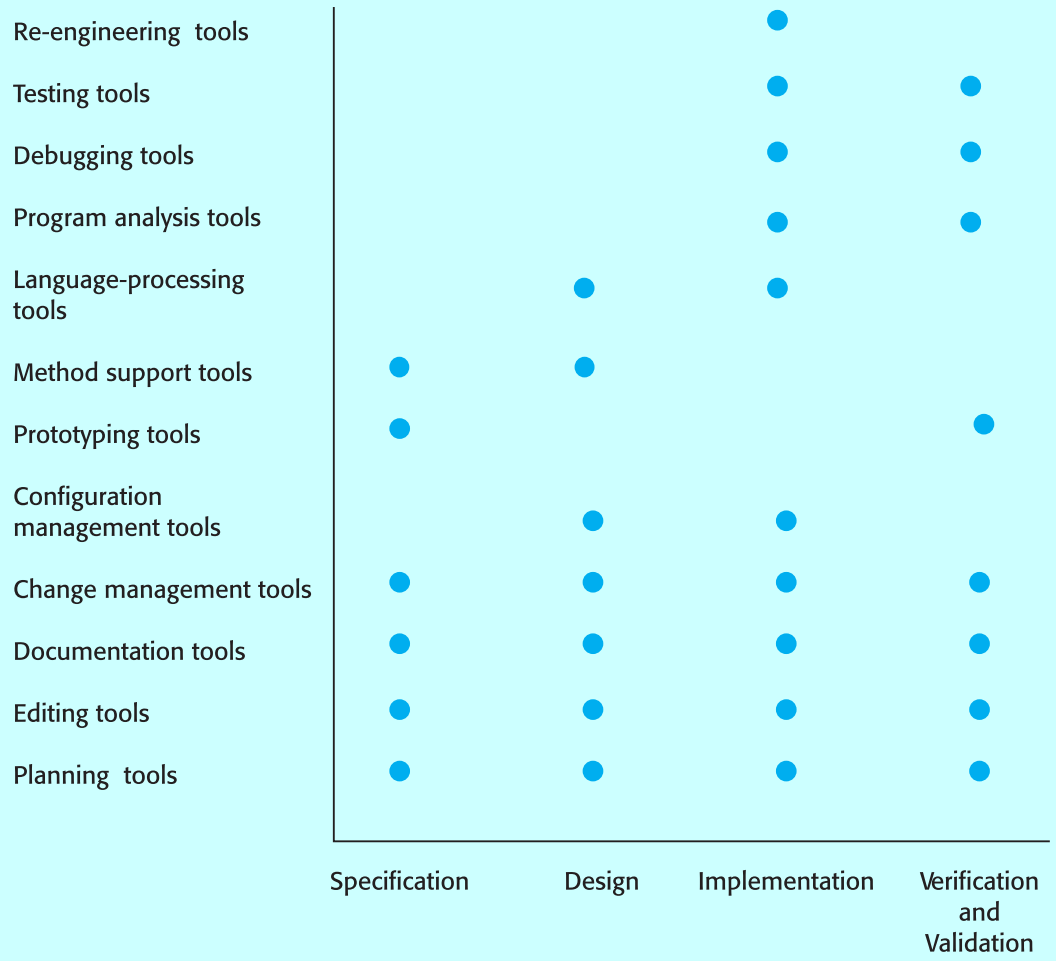
“CASE” Sınıflandırma

- CASE araçlarını sınıflandırmak, farklı tiplerdeki araçları ve yazılım sürecini nasıl desteklediklerini anlamayı kolaylaştırır:
  - İşlevsel bakış açısı (“functional perspective”)
    - İşleve göre sınıfandırma
  - Süreç bakış açısı (“process perspective”):
    - Desteklediği süreç etkinliğine göre sınıflandırma
  - Tümleştirme bakış açısı (“integration perspective”):
    - Tümleştirme yapısına göre sınıflandırma

İşlevsel CASE Araçları

Tool type	Examples
Planning tools	PERT tools, estimation tools, spreadsheets
Editing tools	Text editors, diagram editors, word processors
Change management tools	Requirements traceability tools, change control systems
Configuration management tools	Version management systems, system building tools
Prototyping tools	Very high-level languages, user interface generators
Method-support tools	Design editors, data dictionaries, code generators
Language-processing tools	Compilers, interpreters
Program analysis tools	Cross reference generators, static analysers, dynamic analysers
Testing tools	Test data generators, file comparators
Debugging tools	Interactive debugging systems
Documentation tools	Page layout programs, image editors
Re-engineering tools	Cross-reference systems, program re-structuring systems

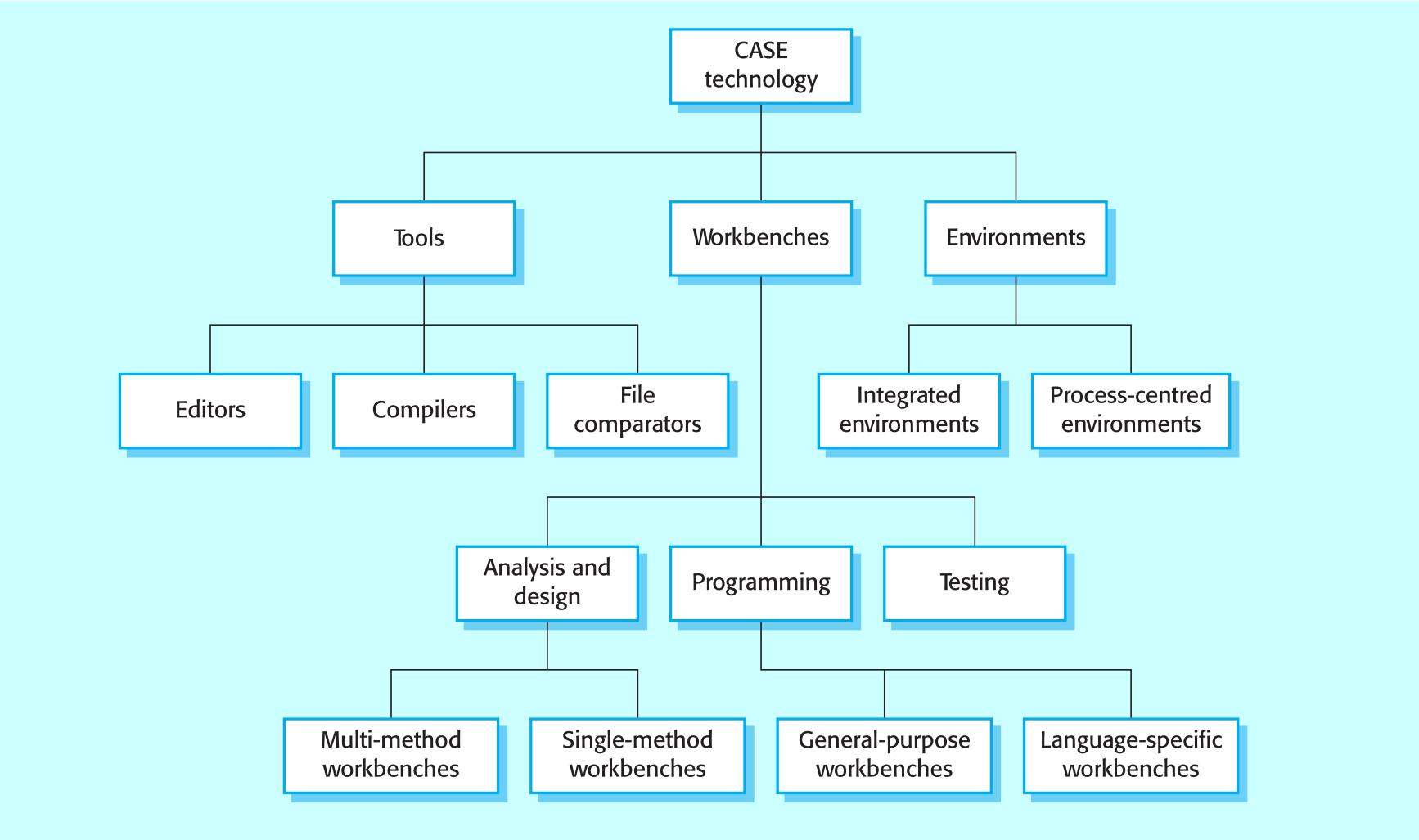
Etkinliğe-Bağlı CASE Araçları



Tümleştirme CASE Araçları (1)

- "Tools"
  - Tasarım, tutarlılık kontrolü, metin yazma gibi alt adımları destekler.
- "Workbenches"
  - Tanımlama ve tasarım gibi süreç aşamalarını destekler ve birden çok aracı içerir.
- "Environments"
  - Yazılım sürecinin tümünü veya bir bölümünü destekler ve birden çok "workbench" içerir.

Tümleştirme CASE Araçları (2)



Yaşam Döngüsü Süreçleri

ISO/IEC 12207 (IEEE Std 12207-2008)

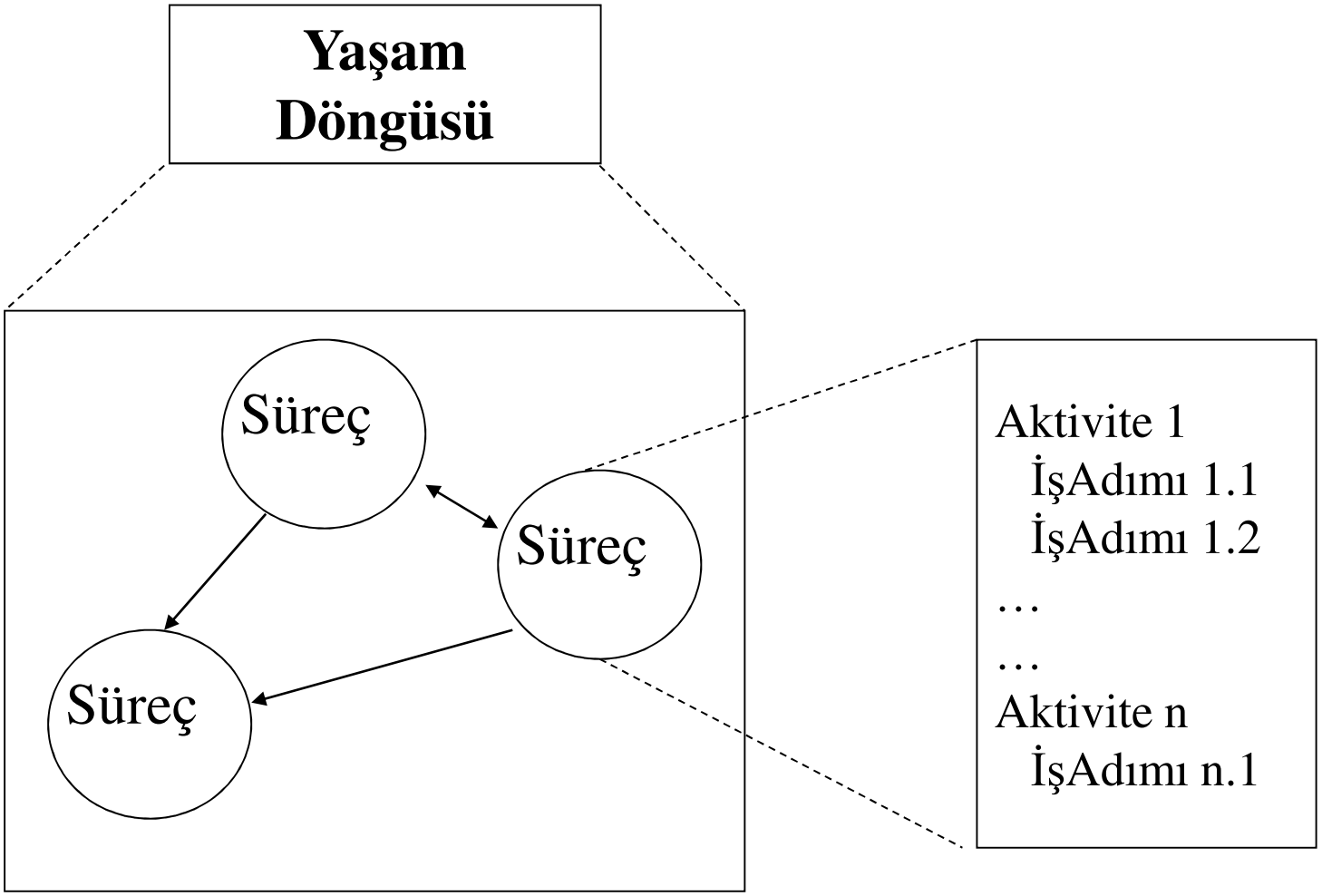
- Sistem ve yazılım mühendisliği –

  Yazılım yaşam döngüsü süreçleri

- 1995'den bu yana sektörde yaygın olarak kullanılmaktadır.
  - 2008 revizyonu, 1995 tarihli standart ile standardın 2002 ve 2004 yıllarında yayınlanan iki ekini birleştirmiştir.

- Kurumun veya projenin ihtiyaçlarına göre uyarlanabilir.
  - Standartta tanımlanan uyarlama kuralları ve önerileri dikkate alınmalıdır.
  - Tam uyumluluk / Seçerek uyarlama

ISO/IEC 12207 Genel Yapısı



ISO/IEC 12207 Kapsamı

- Daha büyük bir sistemin parçası olarak veya tek başına *yazılım ürünü veya hizmeti* için, bir grup yaşam döngüsü süreci ile bunların altındaki etkinlikleri ve görevleri tanımlar.

- Yazılım ürününün veya hizmetinin satınalınması, tedariği, geliştirilmesi, işletilmesi, bakımı ve elden çıkarılması boyunca referans alınabilir.

- ISO/IEC 15288 standardı ile yapı, terimler ilişkili süreçler açılarından tam olarak uyumludur.
  - ISO/IEC 15288-2008: Sistem ve yazılım mühendisliği – Sistem yaşam döngüsü süreçleri

ISO/IEC 12207 Kısıtları

- Herhangi bir yaşam döngüsü modelinin, geliştirme yaklaşımının veya yönteminin kullanılmasını öngörmez.

- Her sürecin beklenen çıktıları üreterek amacına ulaşması için uygulanacak etkinlikleri ve görevleri tanımlar.
  - Süreci, yöntem ve teknikleri içerecek şekilde detaylandırmaz.
  - Sürecin amacına ulaşması için neler yapılması gerektiğini tanımlar.
  - Süreç kapsamındaki etkinlikler ve görevler doğal bir sırada tanımlanmış olsa da bunlar, sürecin nasıl uygulanacağı konusunda yönlendirme vermez.

- Bilgi ürünlerinin (belgelerin) adı, biçimi, içeriği ve materyali ile ilgili detayları tanımlamaz.

Sistem Yaşam Döngüsü Süreçleri (15288)

Anlaşma Süreçleri

- Satınalma Süreci (6.1.1)
- Tedarik Süreci (6.1.2)

Proje-Sağlayıcı Süreçler

- Yaşam Döngüsü Modeli Yönetimi Süreci (6.2.1)
- Altyapı Yönetimi Süreci (6.2.2)
- Proje Portföy Yönetimi Süreci (6.2.3)
- İnsan Kaynakları Yönetimi Süreci (6.2.4)
- Kalite Yönetimi Süreci (6.2.5)

Proje Süreçleri

- Proje Planlama Süreci (6.3.1)
- Proje Değerlendirme ve Kontrol Süreci (6.3.2)
- Karar Yönetimi Süreci (6.3.3)
- Risk Yönetimi Süreci (6.3.4)
- Konfigürasyon Yönetimi Süreci (6.3.5)
- Bilgi Yönetimi Süreci (6.3.6)
- Ölçme Süreci (6.3.7)

Teknik Süreçler

- Paydaş Gereksinimlerini Tanımlama Süreci (6.4.1)
- Sistem Gereksinimleri Analizi Süreci (6.4.2)
- Sistem Mimari Tasarımı Süreci (6.4.3)
- [illegible] 7.1.1
- Sistem Tümleştirme Süreci (6.4.5)
- Sistem Nitelik Testi Süreci (6.4.6)
- Yazılım Kuruluş Süreci (6.4.7)
- Yazılım Kabul Destek Süreci (6.4.8)
- Yazılımı İşletme Süreci (6.4.9)
- Yazılım Bakım Süreci (6.4.10)
- Yazılımı Elden Çıkarma Süreci (6.4.11)

Yazılım Yaşam Döngüsü Süreçleri

Yazılım Gerçekleş-tirme Süreçleri

- Yazılım Gerçekleştirme Süreci (7.1.1)
- Yazılım Gereksinimleri Analizi Süreci (7.1.2)
- Yazılım Mimari Tasarımı Süreci (7.1.3)
- Yazılım Detaylı Tasarım Süreci (7.1.4)
- Yazılım İnşa Süreci (7.1.5)
- Yazılım Tümleştirme Süreci (7.1.6)
- Yazılım Nitelik Testi Süreci (7.1.7)

Yazılım Destek Süreçleri

- Yazılım Belgelendirme Yönetimi Süreci (7.2.1)
- Yazılım Konfigürasyon Yönetimi Süreci (7.2.2)
- Yazılım Kalite Güvence Süreci (7.2.3)
- Yazılım Doğrulama Süreci (7.2.4)
- Yazılım Geçerleme Süreci (7.2.5)
- Yazılım Gözden Geçirme Süreci (7.2.6)
- Yazılım Denetleme Süreci (7.2.7)
- Yazılım Problem Çözümleme Süreci (7.2.8)

Yazılımı Tekrar Kullanma Süreçleri

- Alan Mühendisliği Süreci (7.3.1)
- Tekrar Kullanma Varlık Yönetimi Süreci (7.3.2)
- Tekrar Kullanma Program Yönetimi Süreci (7.3.3)

Sistem Kapsamlı Süreçler

Anlaşma Süreçleri

- Satınalma Süreci (6.1.1)
- Tedarik Süreci (6.1.2)

Proje-Sağlayıcı Süreçler

- Yaşam Döngüsü Modeli Yönetimi Süreci (6.2.1)
- Altyapı Yönetimi Süreci (6.2.2)
- Proje Portföy Yönetimi Süreci (6.2.3)
- İnsan Kaynakları Yönetimi Süreci (6.2.4)
- Kalite Yönetimi Süreci (6.2.5)

Proje Süreçleri

- Proje Planlama Süreci (6.3.1)
- Proje Değerlendirme ve Kontrol Süreci (6.3.2)
- Karar Yönetimi Süreci (6.3.3)
- Risk Yönetimi Süreci (6.3.4)
- Konfigürasyon Yönetimi Süreci (6.3.5)
- Bilgi Yönetimi Süreci (6.3.6)
- Ölçme Süreci (6.3.7)

Teknik Süreçler

- Paydaş Gereksinimlerini Tanımlama Süreci (6.4.1)
- Gereksinim Analizi Süreci (6.4.2)
- Mimari Tasarım Süreci (6.4.3)
- Gerçekleştirme Süreci (6.4.4)
- Tümleştirme Süreci (6.4.5)
- Doğrulama Süreci (6.4.6)
- Geçiş Süreci (6.4.7)
- Geçerleme Süreci (6.4.8)
- İşletme Süreci (6.4.9)
- Bakım Süreci (6.4.10)
- Elden Çıkarma Süreci (6.4.11)

Yazılıma Özel Süreçler

Yazılım Gerçekleş-tirme Süreçleri

- Yazılım Gerçekleştirme Süreci (7.1.1)
- Yazılım Gereksinimleri Analizi Süreci (7.1.2)
- Yazılım Mimari Tasarımı Süreci (7.1.3)
- Yazılım Detaylı Tasarım Süreci (7.1.4)
- Yazılım İnşa Süreci (7.1.5)
- Yazılım Tümleştirme Süreci (7.1.6)
- Yazılım Nitelik Testi Süreci (7.1.7)

Yazılım Destek Süreçleri

- Yazılım Belgelendirme Yönetimi Süreci (7.2.1)
- Yazılım Konfigürasyon Yönetimi Süreci (7.2.2)
- Yazılım Kalite Güvence Süreci (7.2.3)
- Yazılım Doğrulama Süreci (7.2.4)
- Yazılım Geçerleme Süreci (7.2.5)
- Yazılım Gözden Geçirme Süreci (7.2.6)
- Yazılım Denetleme Süreci (7.2.7)
- Yazılım Problem Çözümleme Süreci (7.2.8)

Yazılımı Tekrar Kullanma Süreçleri

- Alan Mühendisliği Süreci (7.3.1)
- Tekrar Kullanma Varlık Yönetimi Süreci (7.3.2)
- Tekrar Kullanma Program Yönetimi Süreci (7.3.3)

Örnek: Yazılım Geçerleme Süreci (7.2.5)

- Amaç:
  - Yazılım iş ürününün, amaca özel kullanımına yönelik gereksinimleri karşıladığını onaylamak
- Çıktılar:
  - Geçerleme stratejisi geliştirilir ve uygulanır.
  - Gereken yazılım iş ürünlerinin tümü için geçerleme kriterleri belirlenir.
  - Gereken geçerleme etkinlikleri uygulanır.
  - Hatalar tespit edilir ve kaydedilir.
  - Geliştirilen yazılım iş ürünlerinin, amaçlanan kullanıma uygun olduğuna dair kanıtlar sağlanır.
  - Geçerleme etkinliklerinin sonuçları, müşterinin ve işe dahil olan diğer birimlerin erişimine açılır.

Yazılım Geçerleme Süreci – Etkinlikler ve Görevler

7.2.5.3.1 – Süreç uygulaması

1. Projenin geçerleme işgücünü garanti edip edemeyeceği ve bu işgücünün ne derecede kurumsal bağımsızlık gerektirdiği belirlenecektir.

2. Proje geçerleme işgücünü garanti ediyorsa, bir geçerleme süreci oluşturulacaktır. Geçerleme görevleri; ilişkili yöntemler, teknikler ve araçlarla birlikte, seçilecektir.

3. Proje bağımsız bir geçerleme işgücünü garanti ediyorsa, geçerlemeyi gerçekleştirmekten sorumlu olacak yetkin bir birim seçilecektir. Birimin geçerlemeyi yapacak bağımsızlığa ve yetkiye sahip olduğu güvence edilecektir.

4. Bir geçerleme planı geliştirilecek ve belgelendirilecektir.

5. Geçerleme planı uygulanacaktır. Geçerleme sırasında belirlenen problemler ve uygunsuzluklar, Yazılım Problem Çözümleme sürecine girdi olacaktır. Geçerleme etkinliklerinin sonuçları, müşterinin ve işe dahil olan diğer birimlerin erişimine sunulacaktır.

Yazılım Geçerleme Süreci – Etkinlikler ve Görevler (.. devamı)

7.2.5.3.2 – Geçerleme

1. Seçilen test gereksinimleri, test durumları ve test tanımları hazırlanacaktır.
2. Tanımlanan test gereksinimlerinin, test durumlarının ve test tanımlarının, amaçlanan kullanıma özel gereksinimleri yansıttığı güvence edilecektir.
3. Tanımlanan testler gerçekleştirilecektir.
4. Yazılım ürününün amaçlanan kullanımı karşıladığı geçerlenecektir.
5. Yazılım ürünü, uygun olduğu ölçüde, amaçlanan ortamın seçilen alanlarında test edilecektir.

Geçerleme Planı

- Geçerlemeye esas öğeler
- Gerçekleştirilecek geçerleme görevleri
- Kaynaklar
- Sorumluluklar
- Takvim
- Geçerleme raporlarının satın alan kuruma veya işe dahil olan diğer birimlere iletilmesi için prosedürler

Tümleşik Yetenek Olgunluk Modeli (“Capability Maturity Model Integration – CMMI”)

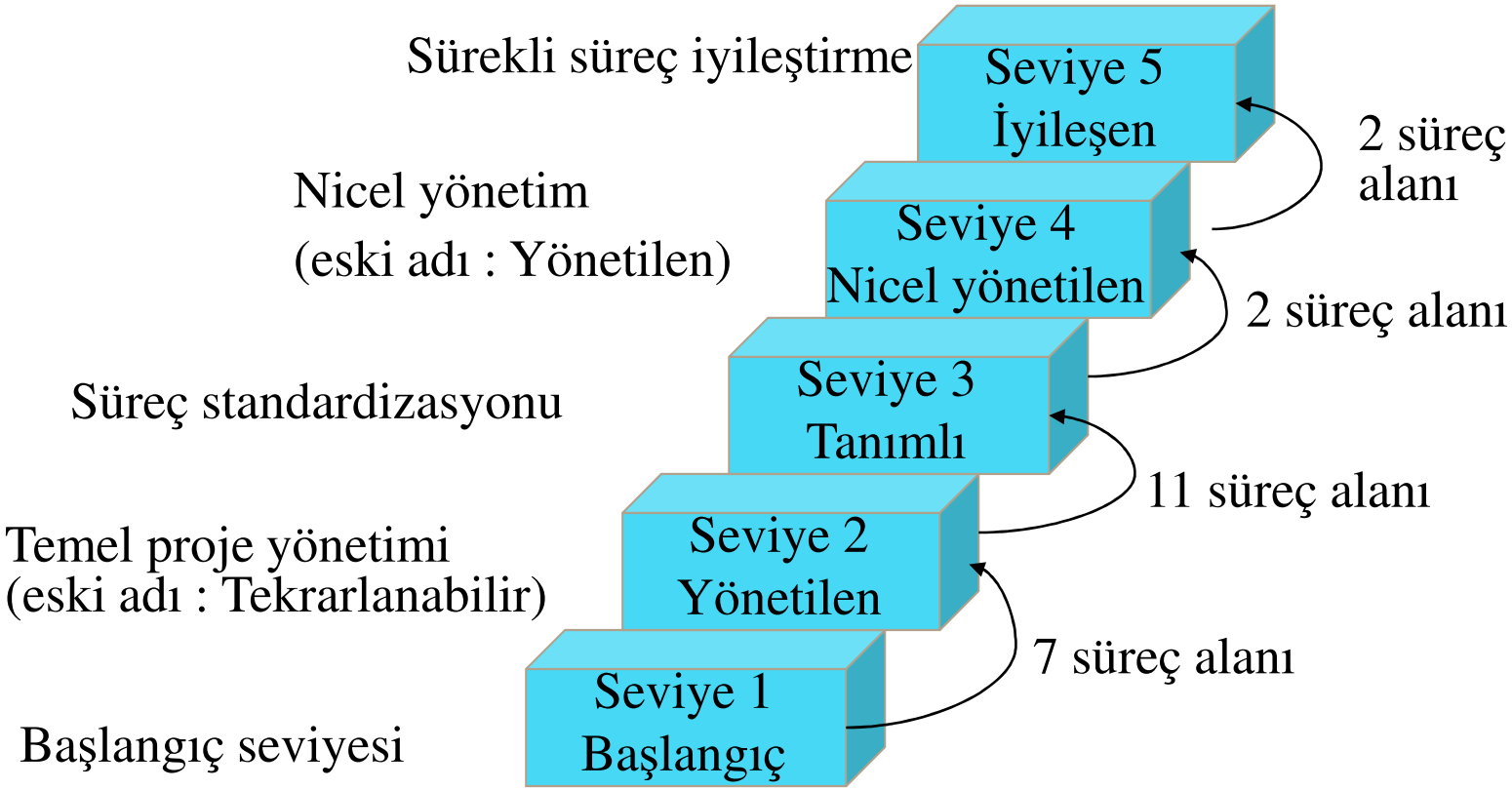
Capability Maturity Model (CMM)

- Toplam kalite yönetimi kavramlarının yazılım geliştirme alanına uyarlanması
  - "Crosby Maturity Grid" temel alınmıştır
  - Sisteminin kalitesini, üretimde ve bakımda kullanılan süreçlerin kalitesi belirler
  - Yazılım sektöründe yaşanılan deneyimler esas alınmıştır (1989-2006)
- ABD Savunma Bakanlığı için Software Engineering Institute (SEI) tarafından geliştirildi
  - Açık standart değil, tüm hakları SEI'de
- Güvenilir ve tutarlı yazılım süreç değerlendirme için altyapı sağlar
  - Değerlendirme yöntemi: The *Standard CMMI Appraisal Method for Process Improvement* (*SCAMPI*)
- Kurumsal iyileştirme ve gelişim için bir model sağlar
  - Süreç iyileştirme sistematik olarak küçük adımlar ile gerçekleşir
  - Kuruluş bir olgunluk seviyesinde diğerine yükselir

CMMI Genel Bakış

- Faklı süreç modellerini birleştirmek
  - Bütünleşik model oluşturmak
- Deneyimler ve en iyi uygulamalar çerçevesinde CMM yaklaşımını yenilemek
  - Sistem mühendisliği ve yazılım mühendisliğinin işbirliğini destekler
  - Ölçme ve analizin iyileştirme çalışmalarının başlarında ele alınmasına önem verir
  - Tüm tasarım ve ürün geliştirme yaşam döngüsünü kapsar
- CMMI Modelleri
  - Software engineering (SW)
  - System engineering + software engineering (SE/SW)
  - System engineering + software engineering + Integrated product and process development (SE/SW/IPPD)
  - System engineering + software engineering + Integrated product and process development + Supplier sourcing (SE/SW/IPPD/SS)
- CMMI Gösterimleri
  - Basamaklı (Staged)
  - Sürekli (Continous)

CMMI Olgunluk Seviyeleri



CMMI – Basamaklı Model Süreç Alanları

Seviye 2 **Yönetilen** *Temel Proje Yönetimi*	■ Gereksinim yönetimi ■ Proje planlama ■ Proje izleme ve kontrolü ■ Tedarikçi anlaşması yönetimi ■ Ölçme ve analiz ■ Ürün ve süreç kalite güvencesi ■ Konfigürasyon yönetimi
Seviye 3 – **Tanımlı** *Süreç Standardizasyonu*	■ Gereksinim geliştirme ■ Teknik çözüm ■ Ürün entegrasyonu ■ Doğrulama ■ Geçerleme ■ Kurumsal süreç odağı ■ Kurumsal süreç tanımı ■ Kurumsal eğitim ■ Bütünleşik proje yönetimi (IPPD) ■ Risk yönetimi ■ Bütünleşik takım kurma ■ Bütünleşik tedarikçi yönetimi ■ Karar analiz ve çözümleme ■ Bütünleştirme için kurumsal ortam
Seviye 4 - **Nicel Yönetilen**	■ Kurumsal süreç performansı ■ Nicel proje yönetimi
Seviye 5 – **İyileşen**	■ Kurumsal yenilik ve yayılma ■ Sebep analizi ve çözümleme

Süreç Alanı (“Process Area”)

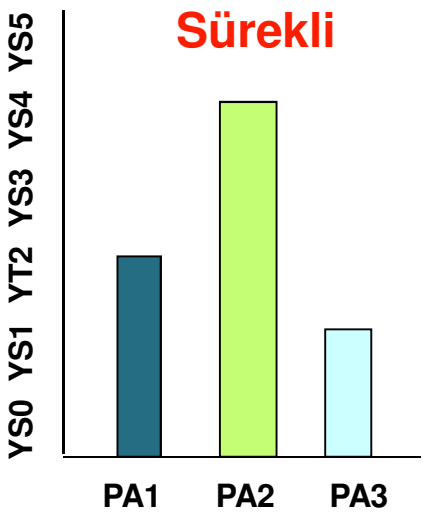
- Birlikte gerçekleştirildiklerinde, belirli bir alanda, süreç iyileştirme için önemli olan hedeflere ulaşılmasını sağlayacak bir grup uygulama
  - Her süreç alanının amacı “beklenen davranışı” tanımlar
    - Beklenen davranış
  - Uygulamalar süreç alanı hedeflerine ulaşılması için atılması gereken adımlardır
  - CMMI’da süreç alanları sürekli ve basamaklı gösterim için aynıdır
  - Süreç alanı süreç tanımı veya prosedür değildir

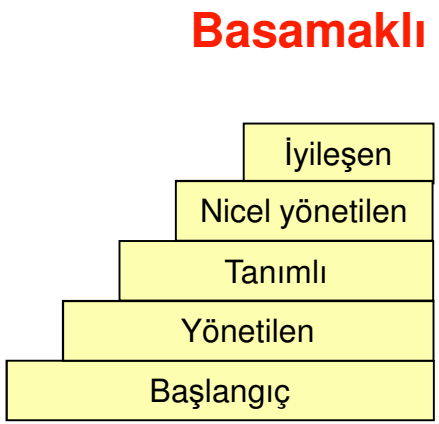
CMMI – Sürekli Model

Süreç Kategorisi	Süreç Alanları	
Proje yönetimi	Proje planlama	Proje izleme ve kontrol
	Risk yönetimi	Entegre proje yönetimi
	Nicel proje yönetimi	Tedarikçi anlaşma yönetimi
	Bütünleşik proje yönetimi	Bütünleşik tedarikçi yönetimi
Mühendislik	Gereksinim yönetimi	Gereksinim geliştirme
	Teknik çözümleme	Ürün entegrasyonu
	Doğrulama	Geçerleme
Destek	Konfigürasyon yönetimi	Ölçme ve analiz
	Süreç ve ürün kalite güvencesi	Bütünleştirme için kurumsal ortam
	Sebep analizi ve çözümleme	
	Karar analiz ve çözümleme	
Süreç yönetimi	Kurumsal süreç odağı	Kurumsal süreç tanımı
	Kurumsal eğitim	Kurumsal süreç performansı
	Kurumsal yenilik ve yayılma	

CMMI Model Gösterimleri

- CMMI, süreç iyileştirme çalışmaları için iki farklı yol sunar
  - Sürekli
    - Kurum tarafından seçilen süreç(ler) odaklı iyileştirme
  - Basamaklı
    - Önceden tanımlı bir grup ilgili süreci birlikte ele alarak iyileştirme





Basamaklı Gösterim

- Yapısal ve sistematik bir yaklaşım sunar
  - İyileştirme adım adım gerçekleştirilir
  - Her basamak bir sonraki basamak için temel oluşturur
- Süreç iyileştirme çalışmaları için denenmiş bir yönlendirme sağlar
  - Her süreç alanının ne zaman ele alınması gerektiği belli
  - Süreç iyileştirme yolunu belirler
- Süreç alanları olgunluk seviyeleri altında gruplandırılmış
- SW-CMM'den geçiş kolaydır
- Nereden başlayacağınızı bilmiyorsanız iyi bir seçim

Sürekli Gösterim

- Daha esnek bir yaklaşım sunar
- İş hedefleriniz ile uyumlu olarak iyileştirme yapmak istediğiniz öncelikli alanları kendiniz tanımlarsınız
  - Tek bir süreçte odaklanma olasılığı
  - Birbirleri ile ilişkili süreç grubu
  - Farklı süreçleri farklı hızlarda iyileştirme şansı
- Her süreç alanında tanımlanan yetenek seviyeleri iyileştirme çalışmaları için sıralı bir yaklaşım sağlar
- ISO 15504 (SPICE) modeli ile uyumludur
- Gösterimin esnekliği biraz daha bilinçli bir yaklaşımı gerekli kılar
  - Kullanılan süreçlerde nelerin iyileştirilmesi gerektiğini bilmek
  - CMMI'da tanımlanan süreçler arasındaki bağımlılıkları bilmek

Örnek Süreç Alanı: Geçerleme

- Amaç:
  - Ürünün hedeflenen çalışma ortamında hedeflenen kullanımı sağladığını göstermek
- Özel Hedefler:
  - SG1. Geçerleme için hazırlan
    - SP1.1. Geçerli kılınacak iş ürünlerini seç
    - SP1.2. Geçerleme ortamını oluştur
    - SP1.3. Geçerleme prosedürlerini ve kriterlerini oluştur
  - SG2. Ürünü veya ürün parçalarını geçerle
    - SP2.1. Geçerleme yap
    - SP2.2. Geçerleme sonuçlarını analiz et

GG2. Yönetilen Süreci Kurumsallaştır

Genel uygulamalar	Geçerli kılma süreç alanı açılımları/Örnekler
GP2.1. Kurumsal politikayı oluştur	-
GP2.2. Süreci planla	**Geçerleme planı**
GP2.3. Kaynakları sağla	**Test durumu yaratan araçlar, simülatörler, test yönetim araçları vs.**
GP2.4. Sorumlulukları ata	-
GP2.5. Kişileri eğit	**Sistemin çalışacağı ortam, geçerleme yöntemleri**
GP2.6. Konfigürasyonu yönet	**Gereksinimler, gereksinim izlenebilirlik matrisi, vs.**
GP2.7. Paydaşları belirle ve katılımı sağla	**Geçerleme sonuçlarının gözden geçirmesi, vs.**
GP2.8. Süreci izle ve kontrol et	**Açılan ve kapatılan geçerli kılma raporlarının sayısı, problem raporlarının açık kalma zamanı**
GP2.9. Uyumluluğu tarafsız değerlendir	**Geçerleme stratejisi, prosedürleri**
GP2.10. Durumu üst yönetimle gözden geçir	-