



BM306

Yazılım Mühendisliği



DERS 1

Giriş

*Tanımlar, tarihçe, kapsam
Yazılım Sürec Modelleri*

Dr. Öğr. Üyesi Hasan BADEM
hbadem@ksu.edu.tr



Ders Kitapları ve Yardımcı Kaynaklar

- ❖ **YAZILIM MÜHENDİSLİĞİ - Software Engineering (10. baskı çeviri):**

Ian Sommerville, Nobel Yayınevi

- ❖ Dr. Ayça TURHAN, Hacettepe Üniversitesi

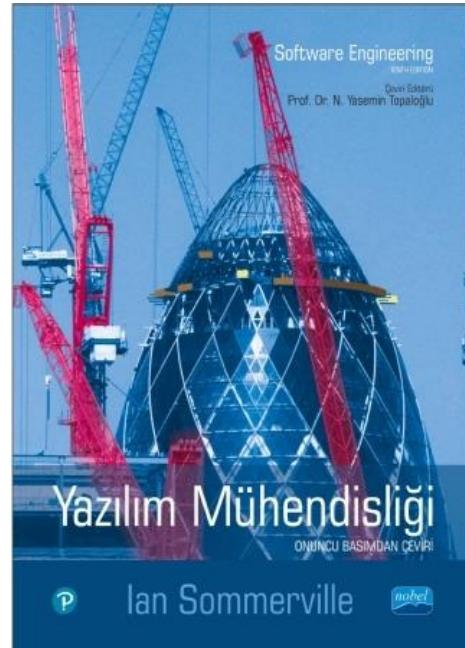
- **Yazılım Mühendisliği Ders Notları**

- ❖ Dr. Hacer KARACAN, Gazi Üniversitesi

- **Yazılım Mühendisliği Ders Notları**

- ❖ Ian Sommerville, Orjinal Sunumlar.

- **Yazılım Mühendisliği**



- ❖ Ayrıca internet üzerinden çok sayıda kaynağa ulaşabilirsiniz.

■ Yazılım Mühendisliği

- ▶ Tanımlar, tarihçe, kapsam

■ Yazılım Süreç Modelleri

- ▶ Yazılım süreci ve süreç modeli nedir?
- ▶ Geleneksel modeller
 - ◆ Çağlayan (“Waterfall”), Evrimsel (“Evolutionary”), Bileşen-Tabanlı (“Component-Based”), Artırımlı (“Incremental”), Döngüsel (“Spiral”)
 - ◆ Rational Software'in Tümleşik Süreçi (“Rational Unified Process”)

Yazılım Mühendisliği

- Tanımlar, Tarihçe, Kapsam -

Yazılım Nedir?

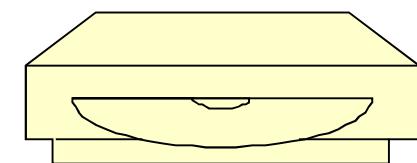
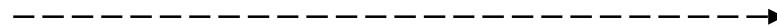
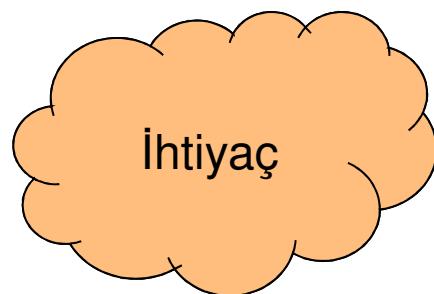
■ Bilgisayar programları ile bunlarla ilişkili yapılandırma ve belgelerin bütünüdür.

► Genel yazılım

- ◆ Özellikleri pazardaki genel ihtiyaca göre belirlenerek geliştirilir.
- ◆ “Generic software”, “commercial-off-the-shelf (COTS) software”
- ◆ Örnek: MS Office yazılımı

► Müşteriye özel yazılım

- ◆ Özellikleri belirli bir müşterinin ihtiyaçına göre belirlenerek geliştirilir.
- ◆ “Custom software”
- ◆ Örnek: Hava trafiği kontrol yazılımı



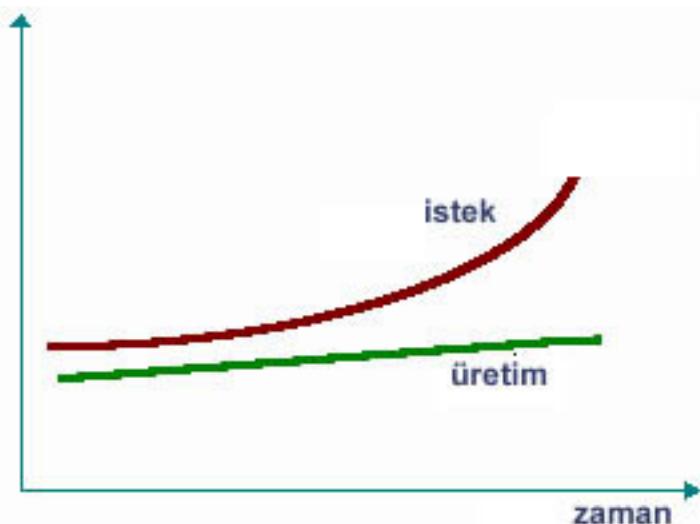
Yazılım

Yazılımın Gelişimi

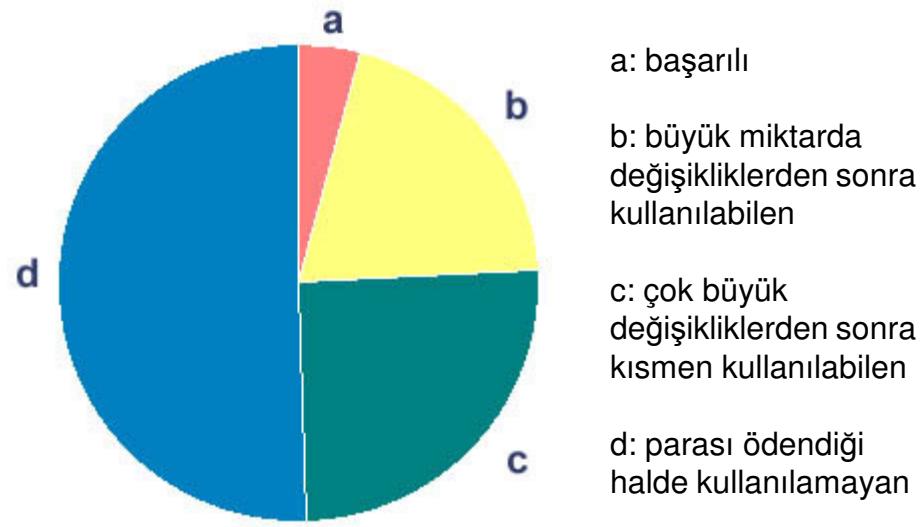
- 1940'lar:
 - ▶ Elle yazılan basit makine kodları
- 1950'ler:
 - ▶ Üretkenlik ve kaliteyi artırmak için yazılan makro birleştiriciler ve yorumlayıcılar
 - ▶ İlk kuşak derleyiciler
- 1960'lar:
 - ▶ Üretkenlik ve kaliteyi artırmak için yazılan ikinci kuşak derleyiciler
 - ▶ Yazılım Mühendisliği NATO Konferansı, 1968: "Yazılım mühendisliği" kavramının tartışılması
 - ▶ İlk büyük yazılım projeleri (1000 programcı)
 - ▶ Büyük iş alanları için ana-bilgisayarlar ve ticari yazılımlar
- 1970'ler:
 - ▶ UNIX ve kod kütüphaneleri için araçlar
 - ▶ Mini-bilgisayarlar ve küçük iş alanları için yazılımlar
- 1980'ler:
 - ▶ Kişisel bilgisayarlar ve iş istasyonları
 - ▶ Ticari yazılımlar
- 1990'lar:
 - ▶ Avuç-içi bilgisayarlar ve web teknolojileri
 - ▶ Teknolojideki gelişmeler sebebiyle düşen fiyatlar ve karmaşıklaşan iş talepleri
- 2000'ler:
 - ▶ Globalleşme sebebiyle artan talepler
 - ▶ Bütünleşik geliştirme ortamları

Yazılım Krizi

- 1960'lardan itibaren yazılım ürünlerine artan talepler karşısında otomasyonu da içeren değişik çözümler uygulandıysa da yeterli üretim kapasitesine erişilemedi. Bu yetersizlik, "yazılım krizi" söylemiyle dünya literatüründe yerini aldı.
- Yazılım krizinin temel sebebi, üstel olarak artan talebe karşılık doğrusal hızla artış gösteren üretim kapasitesidir. Diğer yandan yazılım projelerindeki başarısızlık oranı da şaşırtıcı derecede yüksek olup bu krizi beslemektedir.



Yazılım ürününe arz ve talep artışları



Toplam maliyete göre yazılım başarısı

Yazılım Ürünlerine Talep

■ Yazılım ürünlerine olan talep ve yazılım ürünlerinden bekleneler çok hızlı artıyor

▶ Boeing 777

- ◆ A.B.D. ve Japonya'da 1700 iş istasyonu
- ◆ 4,000,000 Kod Satırı
- ◆ “Kanatları olan yazılım”

▶ Beyaz eşyalar, cep telefonları, otomobiller

▶ Akıllı ev ve ofis sistemleri

▶ ...

Bazı Dehşet Hikayeleri

■ Denver havaalanı otomatik bagaj sistemi

- Açılmış 2 yıl gecikti
- \$27 milyon maliyet aşımı
- \$360 milyon gecikme maliyeti

■ Hava trafik kontrol (FAA modernizasyon)

- \$5.6 milyar maliyet aşımı
- 8 yıl gecikme
- 4 sistemden 2 tanesi iptal edildi, üçüncü sistemin gereksinimlerinin %52'si karşılandı

■ Comanche Helikopteri

- 10 yılda maliyeti 10 kat arttı, \$34.4 milyon
- Gereksinimleri %74 azaltıldı



Ürün Büyüklüğü – Başarı İlişkisi

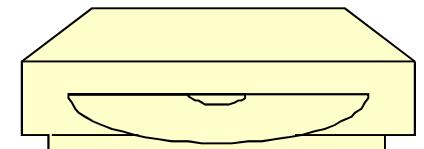
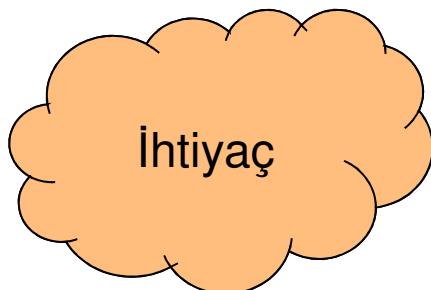
Kestirim (C Satır)	Önce	Zamanında	Gecikme	İptal
13.000	6.06%	74.77%	11.83%	7.33%
130.000	1.24%	60.76%	17.67%	20.33%
1.300.000	0.14%	28.03%	23.83%	48.00%
13.000.000	0.0%	13.67%	21.33%	65.00%

Referans:
“Patterns of Software Failure and Success”,
C. Jones

*Büyük kapsamlı yazılım ürünleri için,
mühendislik yaklaşımı zorunlu hale gelmiştir!*

Yazılım Mühendisliği Nedir?

- Yazılım üretiminin tüm etkinliklerini kapsayan mühendislik disiplinidir.
 - ▶ Sistem tanımlama gibi erken aşamalardan sistemin kullanımı esnasındaki bakımına kadar uzanan etkinlikleri kapsar.
 - ▶ Sadece teknik etkinlikleri değil, aynı zamanda yönetim etkinliklerini de içerir.



Yazılım

Mühendislik Yaklaşımı !

- Yazılım Mühendisliği; yazılım ürününün geliştirilmesi, işletilmesi ve bakımı için uygulanan; sistematik, disiplinli ve ölçülebilir yaklaşımdır.
[IEEE, 1990]

- ▶ Mühendislik, herhangi bir bilim alanındaki bilgi birikimini sistematik olarak pratiğe geçirmeyi hedefler; bilimi ve matematiği kullanır.
 - ◆ Yönetim parametreleri: İşlev, maliyet, zaman
 - ◆ Kalite parametreleri: Dayanıklılık, bakım kolaylığı, güvenlik, kullanım kolaylığı, vb.
- ▶ Tekrarlanabilir başarılar için mühendislik yaklaşımı şarttır.
 - ◆ Mühendislik öğretisi ile bir yöntem uygulandığında, benzer sonuçları her zaman elde etme güvenliği vardır.

Yazılım Mühendisliği - Bazı Tanımlar (1)

■ “The practical application of scientific knowledge in the design and construction of computer programs and the associated documentation to develop, operate and maintain them.”

[Boehm, 1976]

■ “... the technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost estimates”

[Fairley, 1985]

■ “The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software”

[IEEE, 1990]

Yazılım Mühendisliği - Bazı Tanımlar (2)

- Software engineering is concerned with the definition, refinement and evaluation of principles, methods, techniques and tools to support:
 - ▶ Individual aspects of software development and maintenance (design, coding, etc.)
 - ▶ Planning of software development projects
 - ▶ Performing development, project management and quality assurance activities according to the plan
 - ▶ Assessing the performance of the development and improving products, methods, techniques and tools.

[Rombach and Verlage, 1995]

Yazılım Mühendisliği Neleri Kapsar?

- Yazılım Gereksinim Analizi
- Yazılım Tasarım
- Yazılım Gerçekleştirme  *Programlama*
- Yazılım Test
- Yazılım Bakım
- Yazılım Mühendislik Yönetimi
- Yazılım Konfigürasyon Yönetimi
- Yazılım Mühendisliği Süreçleri
- Yazılım Mühendisliği Araç ve Yöntemleri
- Yazılım Kalitesi

*Yazılım
Geliştirme
Kapsamı*

[“Guide to the Software Engineering Body of Knowledge”, 2004]

“İyi Yazılım” Ne Demektir?

■ Yazılım ürünü, müşterinin beklediği işlevsellik ve performansın yanı sıra, aşağıdaki özellikleri de taşımalıdır.

- ▶ Bakım-yapılabilirlik (“maintainability”)
 - ◆ Yazılım, değişen ihtiyaçlara göre değişim olmalıdır.
- ▶ Güvenilirlik (“dependability”)
 - ◆ Yazılım, güvenilir olmalıdır.
- ▶ Etkinlik (“efficiency”)
 - ◆ Yazılım, sistem kaynaklarının israfına sebep olmamalıdır.
- ▶ Kabul-edilebilirlik (“acceptability”)
 - ◆ Yazılım, kullanıcıları tarafından kabul edilebilmelidir.
(Diğer bir deyişle; anlaşılabilir ve kullanılabilir olmalı, diğer sistemlerle uyumlu çalışabilmelidir.)

Bilgisayar Bilimleri ve Yazılım Mühendisliği Arasındaki İlişki Nedir?

- Bilgisayar Bilimleri; bilgisayar ve yazılım sistemlerinin temelinde yatan teoriler ve yöntemlerle ilgilenir.
 - ▶ Algoritmalar ve veri yapıları, programlama dilleri, mimari, bilimsel hesaplama, işletim sistemleri, bilgi ve veri yönetimi, grafik görüntüleme ve çoklu-ortam, bilgisayar ağları, akıllı sistemler, vb.
(Fizik bilimlerinin, Elektrik ve Elektronik Mühendisliğinin temelinde yattığı gibi)
- Yazılım Mühendisliği; yazılım üretmenin pratik problemleriyle ilgilenir.
 - ▶ Bunu yaparken Bilgisayar Bilimlerinin sunduğu kavramsal altyapayı kullanır.
 - ▶ Bir yazılım mühendisinin, ilişkili uygulama alanına göre (örneğin veri-işleme, animasyon, vb.), altta yatan bilgisayar bilimine hakim olması zorunludur.

Sistem Mühendisliği ve Yazılım Mühendisliği Arasındaki İlişki Nedir?

- Sistem Mühendisliği; yazılımın önemli rol oynadığı, karmaşık bilgisayar sistemlerinin geliştirilmesi ve idamesi ile ilgilenir.
 - ▶ Örnek: [ATM sistemi](#)
 - ▶ Sistem Mühendisliği; sistemin genel çatısıyla ilgiliidir.
 - ◆ Sistemin kullanılacağı alana ilişkin süreçlerin analiz edilmesi, sistem gereksinimlerinin tanımlanması, sistem mimarisinin oluşturulması, sistem bileşenlerinin tımlaştırılması gibi aşamaları içerir.
 - ▶ Söz konusu bilgisayar sistemi; donanım, ağ, yazılım bileşenlerinden oluşur. Sistem Mühendisliği, bileşenlere ilişkin mühendislik etkinliklerinin detaylarıyla pek ilgilenmez.
 - ◆ Bu tür sistemler için Yazılım Mühendisliği (kapsamındaki tüm etkinliklerle birlikte), Sistem Mühendisliği altında ve onun bir parçası olarak uygulanır.
 - ▶ Sistem Mühendisliği, Yazılım Mühendisliğine kıyasla çok daha eski bir mühendislik disiplinidir.

Donanım Mühendisliği ve Yazılım Mühendisliği Arasındaki Farklar Nelerdir?

- Donanım mühendislikleri ile yazılım mühendisliğinin en belirgin farkı ürünlerindedir.
 - ▶ Yazılım ürünü diğer mühendislik ürünlerine oranla daha soyuttur.
 - ▶ Yazılım projesi geliştirme ile sonlanırken donanım projelerinde ek olarak imalat safhası vardır.
 - ▶ Seri üretim, yazılım geliştirme içerisinde neredeyse hiçten ibarettir.
 - ▶ Donanım ürünleri kullanıldıkça aşınır; yazılım ürünlerinde ise aşınma olmaz. Yalnızca baştan beri gizli bulunan hatalar, yazılım kullanıldıkça ortaya çıkar.
- Donanım mühendisliklerinde maliyet odağı seri üretim ve yıpranmayken, yazılım mühendisliğinde maliyet odağı geliştirmedir.

Yazılım Mühendisliğinde Maliyetlerin Dağılımı Nedir?

- Kabaca söylersek, yazılım maliyetinin;
 - %60 : Geliştirme maliyeti,
 - %40 : Test maliyetidir (test maliyetine bulunan hataları düzeltmenin maliyeti de dahildir.)
- Müşteriye özel üretilen yazılımlar için idame (bakım) maliyeti, geliştirme maliyetinin birkaç katına çıkmaktadır.

Yazılım Mühendisliğinde Hedefler / Zorluklar

■ Heterojenlik (“heterogeneity”)

- ▶ Yazılım geliştirme için, heterojen platformları ve çalışma ortamlarını destekleyecek teknikler geliştirmek

■ Teslim (“delivery”)

- ▶ Yazılımın zamanında teslimi için teknikler geliştirmek

■ Güven (“trust”)

- ▶ Yazılımın kullanıcıları tarafından güvenilebileceğini gösteren teknikler geliştirmek

Yazılım Mühendisliğinde Profesyonel ve Etik Gereklilik

■ Yazılım mühendisliği, teknik yetkinliğin uygulanmasının yanında, aşağıdaki konularda sorumluluk gerektirir:

- ▶ Gizlilik
 - ◆ Örnek: Çalışanların ve müşterilerin gizlilik haklarına saygı göstermek
- ▶ Rekabet
 - ◆ Örnek: Kendi yetkinliğiniz dışında iş kabul etmemek
- ▶ Özük hakları
 - ◆ Örnek: Patent, mülkiyet, vb. haklarına dikkat etmek
- ▶ Bilgisayarın amaç-dışı kullanımı
 - ◆ Örnek: Başkasına ait bir makinede oyun oynayarak virüs bulastırmamak

Bir Meslek Olarak Yazılım Mühendisliği

- Sektörde çalışanların yaklaşık yarısı “Bilgisayar Mühendisliği” lisans eğitimi sahip
 - ▶ Alana göre farklılık gösteren lisans dereceleri var
 - ◆ Örneğin; Yönetim Bilgi Sistemleri uygulamaları için İşletme lisansı, gömülü uygulamalar için Elektrik ve Elektronik Mühendisliği lisansı
- Yazılım mühendisliği geniş kesimlerce bir meslek olarak algılanmakta
- Üniversitelerin Yazılım Mühendisliği lisans ve yüksek lisans programları var

Yazılım Süreç Modelleri

Yazılım Süreç Modelleri

- Yazılım geliştirmenin bahsedilen zorluklarıyla başedebilmek için, geliştirmeyi sistematik hale getirmeyi hedefleyen çeşitli süreç modelleri ortaya çıkmıştır.
 - ▶ Bu modellerin temel hedefi; proje başarısı için, yazılım geliştirme yaşam döngüsü (“software development life cycle”) boyunca izlenmesi önerilen mühendislik süreçlerini tanımlamaktır.
 - ◆ *Yazılım geliştirme yaşam döngüsü*: Bir yazılım ürününün ihtiyacının ortaya çıkışından kullanımından kalkmasına kadar geçen dönemdir.
 - ▶ Modellerin ortaya çıkışında, ilgili dönemin donanım ve yazılım teknolojileri ile sektör ihtiyaçları önemli rol oynamıştır.
 - ▶ Örnek:
 - ◆ Geleneksel modeller (örneğin; çağlayan (“waterfall”) modeli)
 - ◆ Çevik (“agile”) modeller (örneğin; uçdeğer (“extreme”) programlama modeli -- XP)

Yazılım Süreci ve Süreç Modeli

■ Süreç nedir?

- ▶ Belirli bir hedef için gerçekleştirilen adımlar zinciridir. [IEEE]

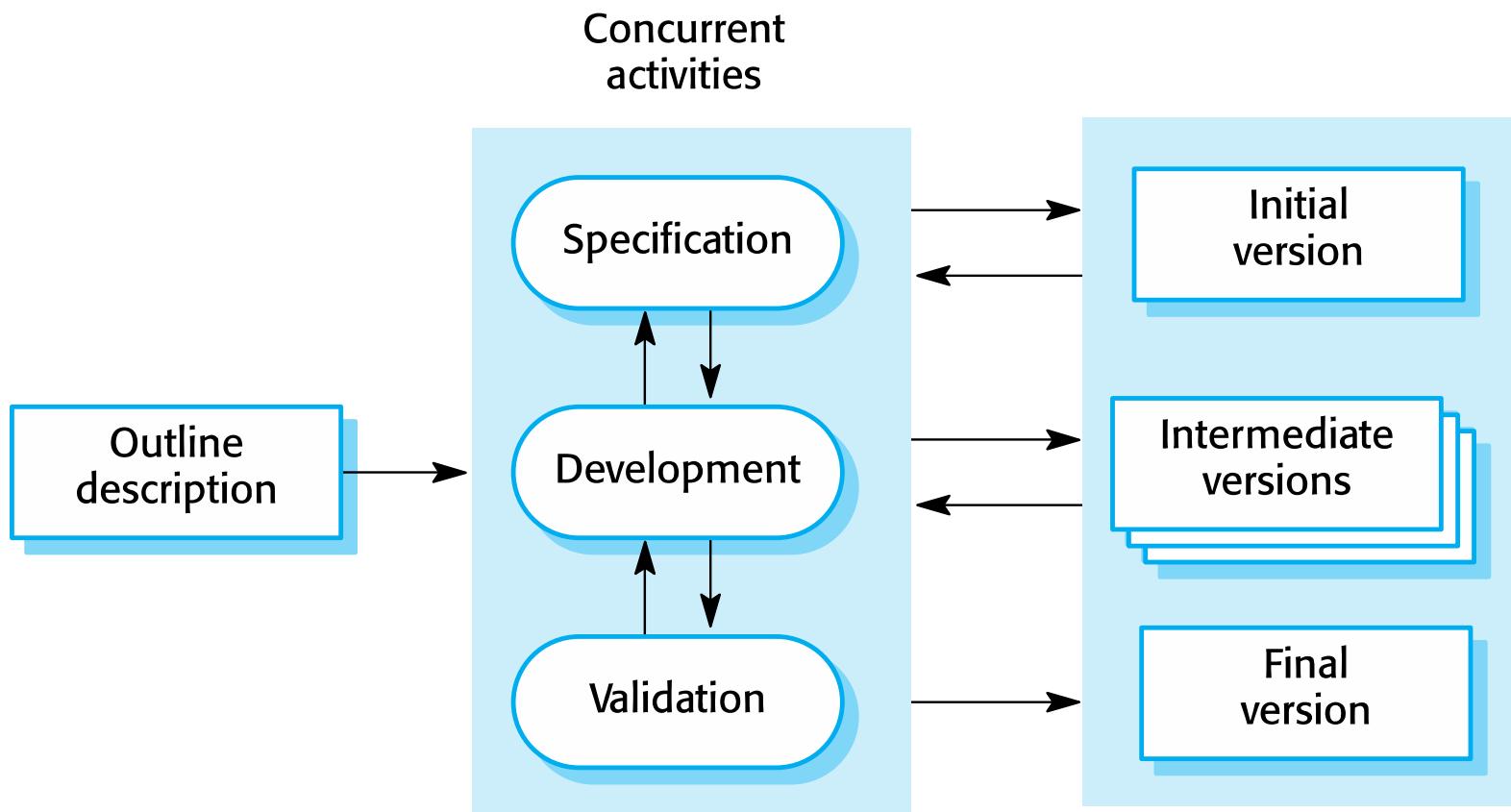
■ Yazılım süreci nedir?

- ▶ Yazılımı ve ilişkili ürünlerini geliştirmek ve idame ettirmek için kullanılan etkinlikler, yöntemler, pratikler ve dönüşümlerdir. [SEI]
- ▶ Yazılım geliştirme ve idame amacı güden etkinlikler setidir.

■ Yazılım süreç modeli nedir?

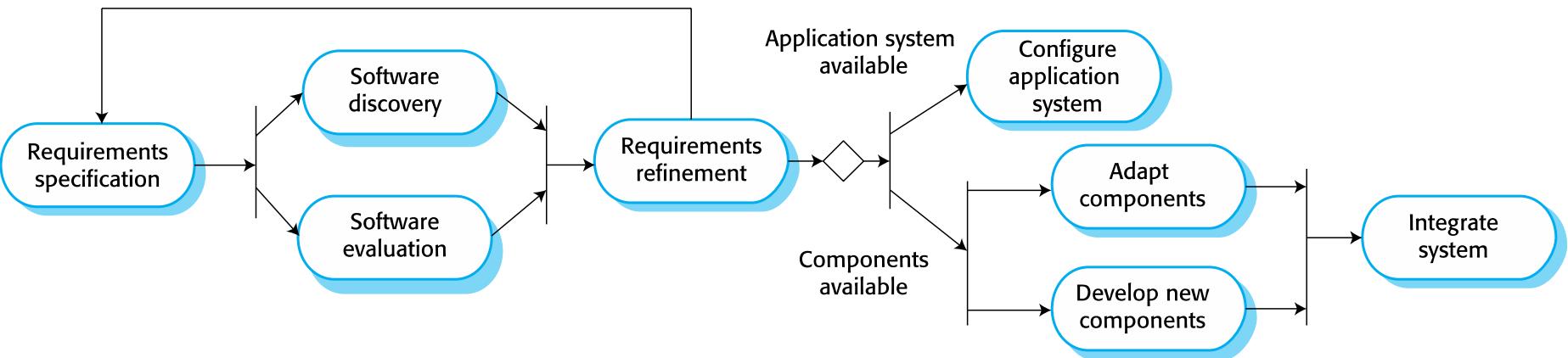
- ▶ Bir yazılım sürecinin belirli bir bakış açısıyla gösterilmiş, basitleştirilmiş temsildir.
- ▶ Örnek bakış açıları:
 - ◆ İş-akışı → etkinlikler nasıl sıralı?
 - ◆ Veri-akış → bilgiler nasıl sıralı?
 - ◆ Rol-hareket → kim ne正在做?

Incremental development



Reuse-oriented software engineering

Yeniden Kullanıma yönelik





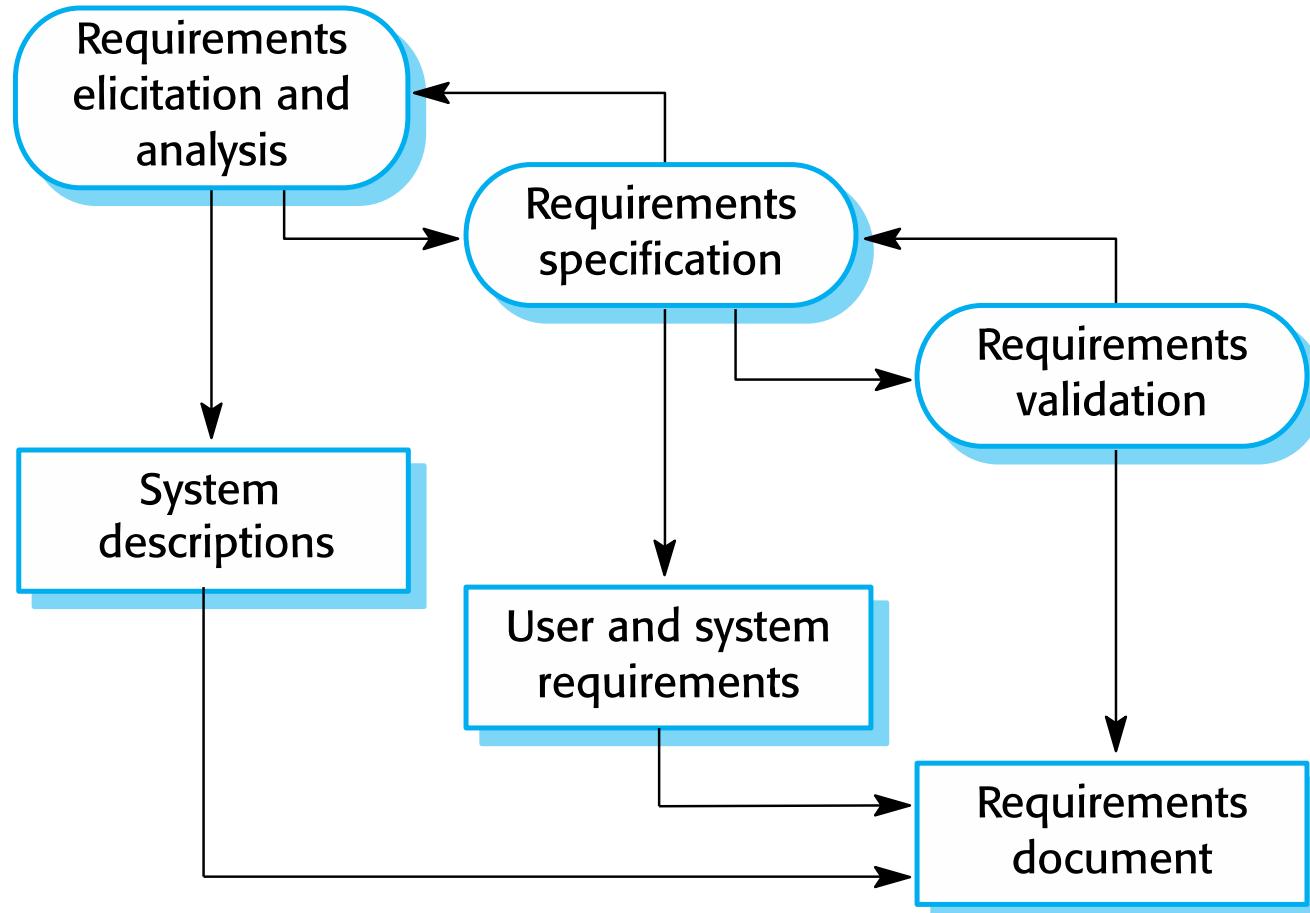
Process activities

Process activities

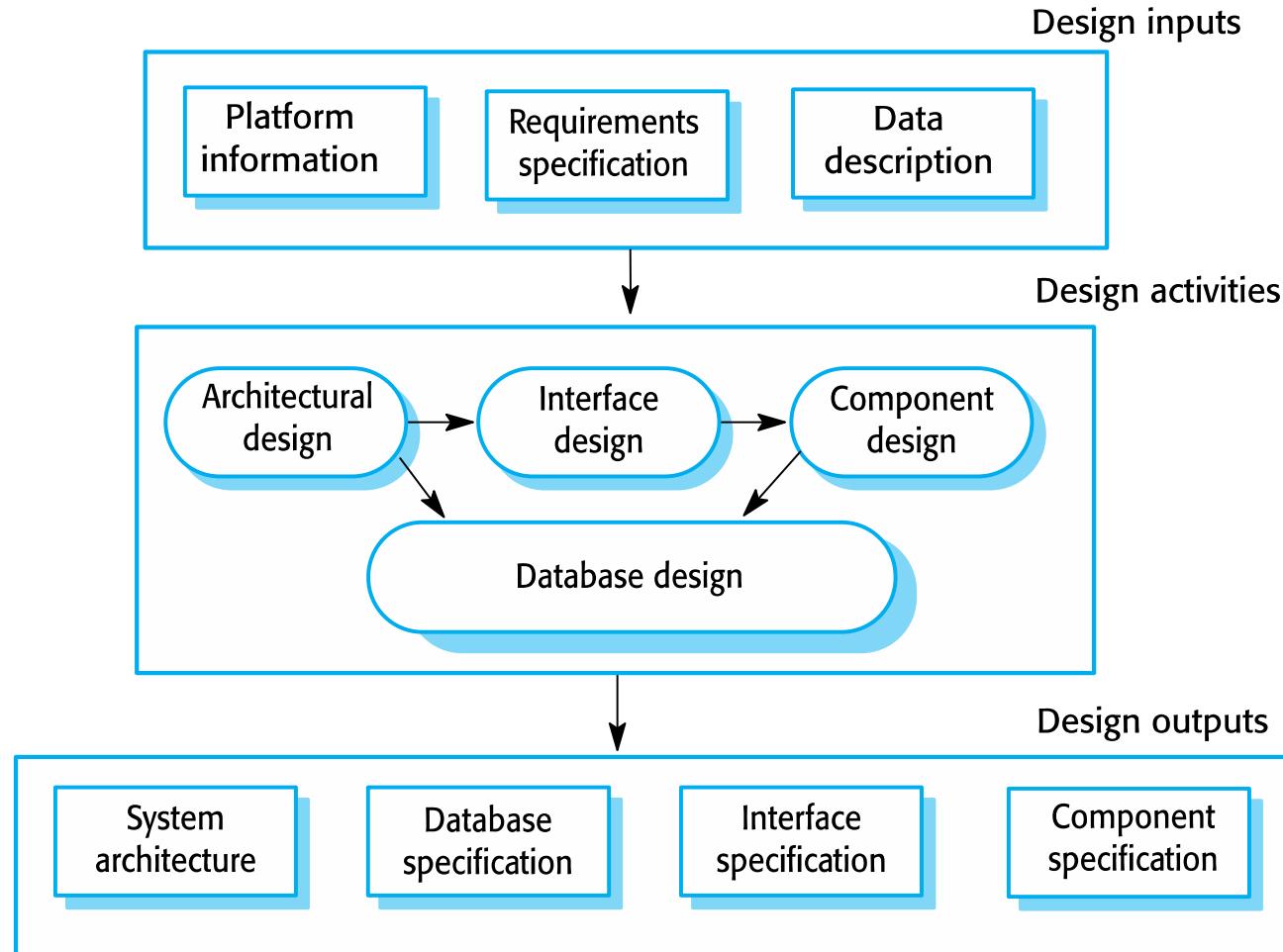


- ✧ Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.
- ✧ The four basic process activities of specification, development, validation and evolution are organized differently in different development processes.
- ✧ For example, in the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.

The requirements engineering process



A general model of the design process



Design activities



- ✧ *Architectural design*, where you identify the overall structure of the system, the principal components (subsystems or modules), their relationships and how they are distributed.
- ✧ *Database design*, where you design the system data structures and how these are to be represented in a database.
- ✧ *Interface design*, where you define the interfaces between system components.
- ✧ *Component selection and design*, where you search for reusable components. If unavailable, you design how it will operate.

System implementation



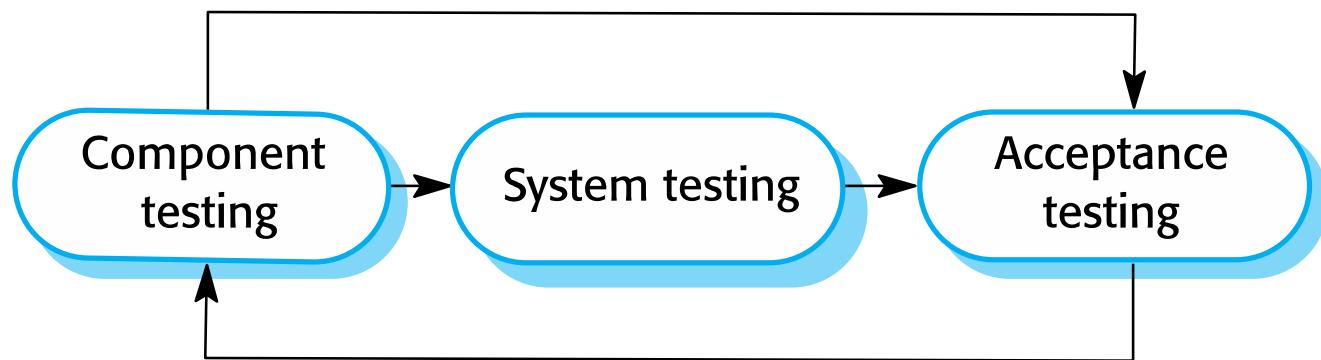
- ✧ The software is implemented either by developing a program or programs or by configuring an application system.
- ✧ Design and implementation are interleaved activities for most types of software system.
- ✧ Programming is an individual activity with no standard process.
- ✧ Debugging is the activity of finding program faults and correcting these faults.

Software validation



- ✧ Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.
- ✧ Involves checking and review processes and system testing.
- ✧ System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.
- ✧ Testing is the most commonly used V & V activity.

Stages of testing



Testing stages



✧ Component testing

- Individual components are tested independently;
- Components may be functions or objects or coherent groupings of these entities.

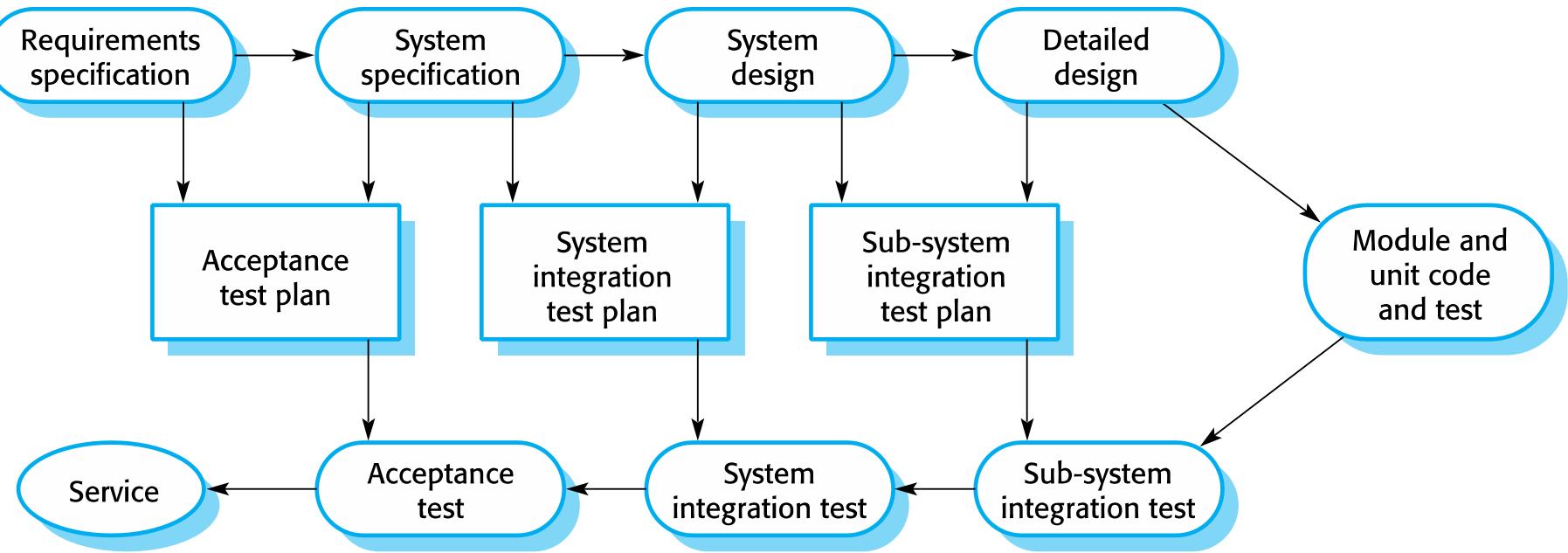
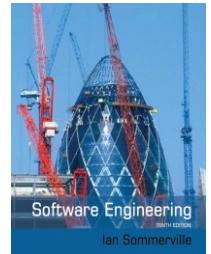
✧ System testing

- Testing of the system as a whole. Testing of emergent properties is particularly important.

✧ Customer testing

- Testing with customer data to check that the system meets the customer's needs.

Testing phases in a plan-driven software process (V-model)

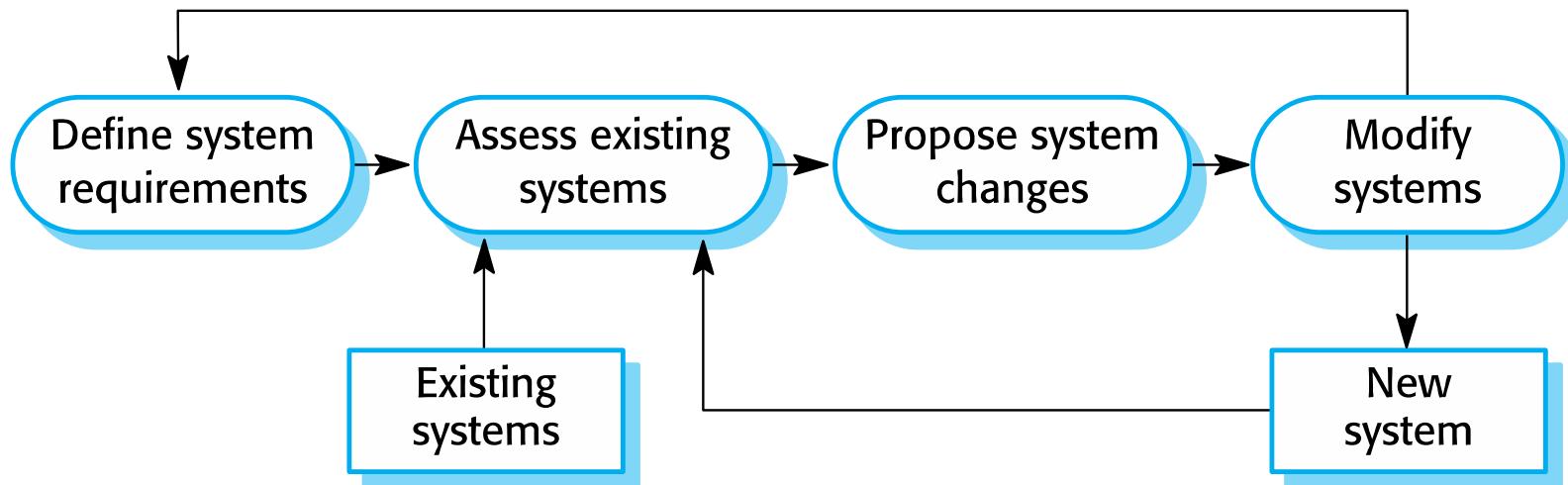


Software evolution



- ✧ Software is inherently flexible and can change.
- ✧ As requirements change through changing business circumstances, the software that supports the business must also evolve and change.
- ✧ Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.

System evolution





Coping with change

Coping with change



- ✧ Change is inevitable in all large software projects.
 - Business changes lead to new and changed system requirements
 - New technologies open up new possibilities for improving implementations
 - Changing platforms require application changes
- ✧ Change leads to rework so the costs of change include both rework (e.g. re-analysing requirements) as well as the costs of implementing new functionality

Reducing the costs of rework



- ✧ Change anticipation, where the software process includes activities that can anticipate possible changes before significant rework is required.
 - For example, a prototype system may be developed to show some key features of the system to customers.
- ✧ Change tolerance, where the process is designed so that changes can be accommodated at relatively low cost.
 - This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have to be altered to incorporate the change.

Coping with changing requirements



- ✧ System prototyping, where a version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of design decisions. This approach supports change anticipation.
- ✧ Incremental delivery, where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance and change tolerance.

Software prototyping



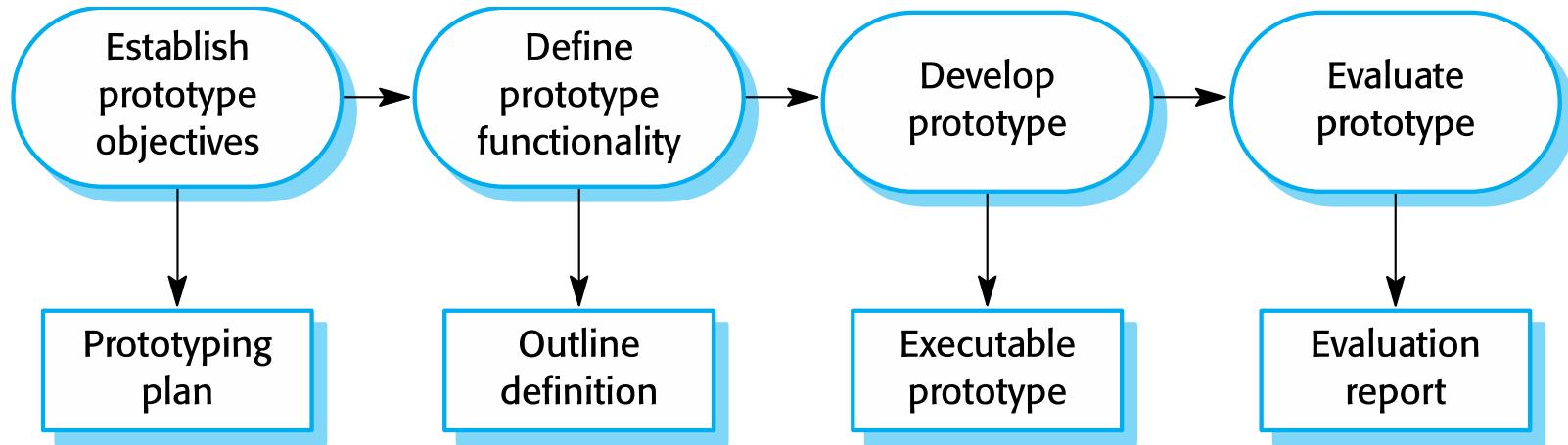
- ✧ A prototype is an initial version of a system used to demonstrate concepts and try out design options.
- ✧ A prototype can be used in:
 - The requirements engineering process to help with requirements elicitation and validation;
 - In design processes to explore options and develop a UI design;
 - In the testing process to run back-to-back tests.

Benefits of prototyping



- ✧ Improved system usability.
- ✧ A closer match to users' real needs.
- ✧ Improved design quality.
- ✧ Improved maintainability.
- ✧ Reduced development effort.

The process of prototype development



Prototype development



- ✧ May be based on rapid prototyping languages or tools
- ✧ May involve leaving out functionality
 - Prototype should focus on areas of the product that are not well-understood;
 - Error checking and recovery may not be included in the prototype;
 - Focus on functional rather than non-functional requirements such as reliability and security

Throw-away prototypes



- ✧ Prototypes should be discarded after development as they are not a good basis for a production system:
 - It may be impossible to tune the system to meet non-functional requirements;
 - Prototypes are normally undocumented;
 - The prototype structure is usually degraded through rapid change;
 - The prototype probably will not meet normal organisational quality standards.

Incremental delivery



- ✧ Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
- ✧ User requirements are prioritised and the highest priority requirements are included in early increments.
- ✧ Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

Incremental development and delivery



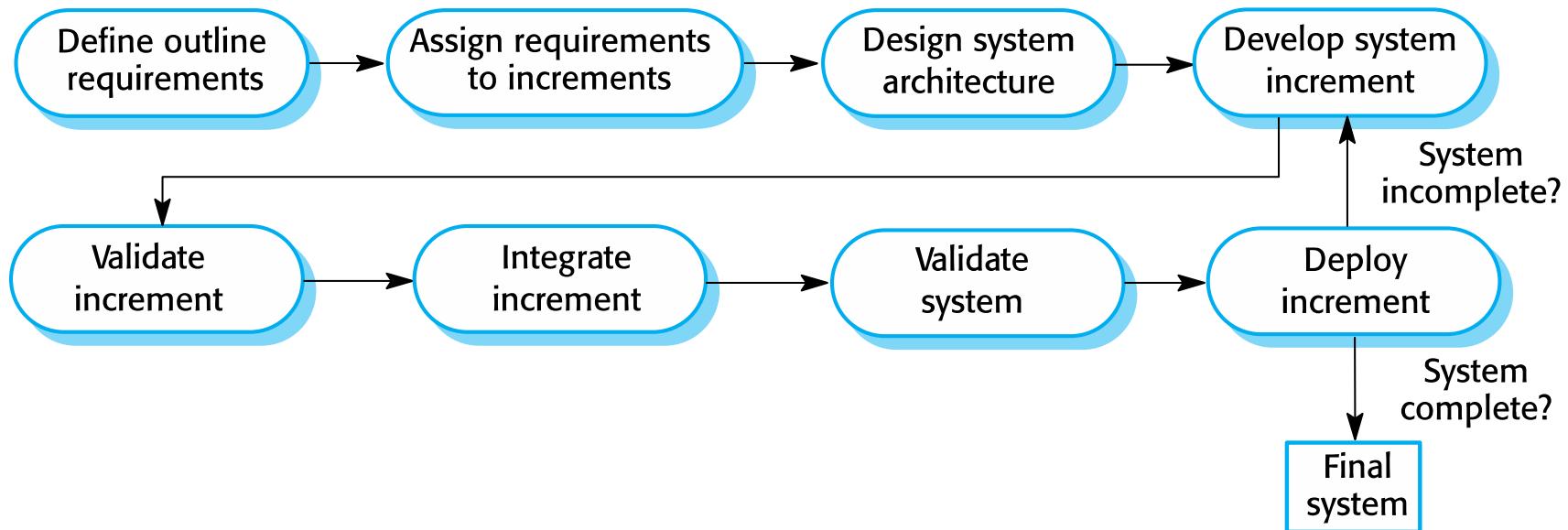
✧ Incremental development

- Develop the system in increments and evaluate each increment before proceeding to the development of the next increment;
- Normal approach used in agile methods;
- Evaluation done by user/customer proxy.

✧ Incremental delivery

- Deploy an increment for use by end-users;
- More realistic evaluation about practical use of software;
- Difficult to implement for replacement systems as increments have less functionality than the system being replaced.

Incremental delivery



Incremental delivery advantages



- ✧ Customer value can be delivered with each increment so system functionality is available earlier.
- ✧ Early increments act as a prototype to help elicit requirements for later increments.
- ✧ Lower risk of overall project failure.
- ✧ The highest priority system services tend to receive the most testing.

Incremental delivery problems



- ✧ Most systems require a set of basic facilities that are used by different parts of the system.
 - As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.
- ✧ The essence of iterative processes is that the specification is developed in conjunction with the software.
 - However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.



Process improvement

Process improvement



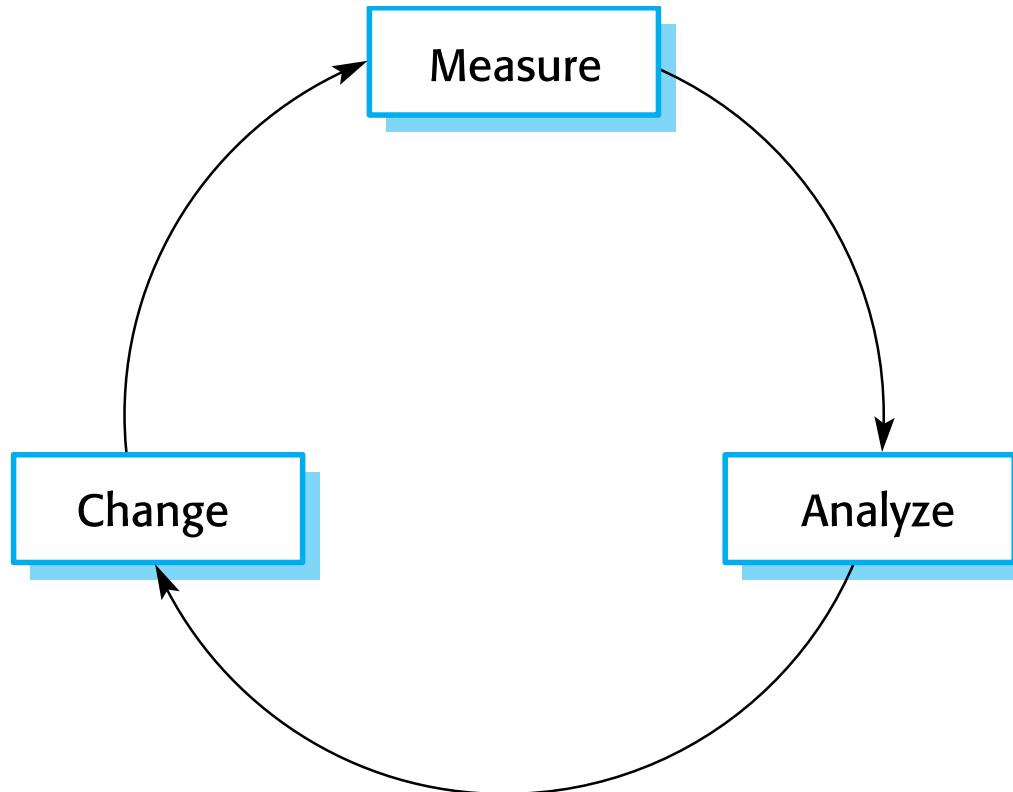
- ✧ Many software companies have turned to software process improvement as a way of enhancing the quality of their software, reducing costs or accelerating their development processes.
- ✧ Process improvement means understanding existing processes and changing these processes to increase product quality and/or reduce costs and development time.

Approaches to improvement



- ✧ The process maturity approach, which focuses on improving process and project management and introducing good software engineering practice.
 - The level of process maturity reflects the extent to which good technical and management practice has been adopted in organizational software development processes.
- ✧ The agile approach, which focuses on iterative development and the reduction of overheads in the software process.
 - The primary characteristics of agile methods are rapid delivery of functionality and responsiveness to changing customer requirements.

The process improvement cycle



Process improvement activities



✧ *Process measurement*

- You measure one or more attributes of the software process or product. These measurements forms a baseline that helps you decide if process improvements have been effective.

✧ *Process analysis*

- The current process is assessed, and process weaknesses and bottlenecks are identified. Process models (sometimes called process maps) that describe the process may be developed.

✧ *Process change*

- Process changes are proposed to address some of the identified process weaknesses. These are introduced and the cycle resumes to collect data about the effectiveness of the changes.

Process measurement



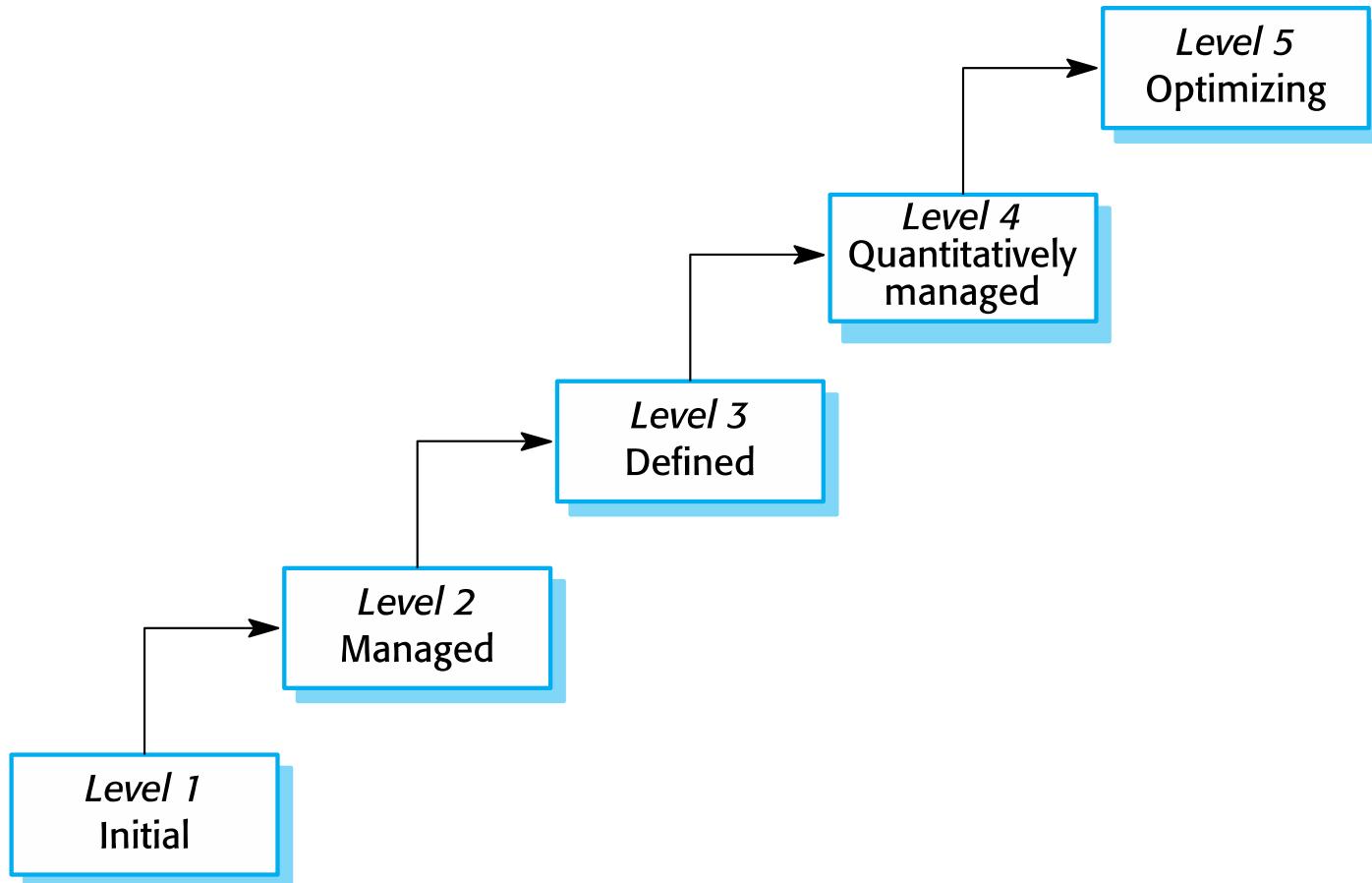
- ✧ Wherever possible, quantitative process data should be collected
 - However, where organisations do not have clearly defined process standards this is very difficult as you don't know what to measure. A process may have to be defined before any measurement is possible.
- ✧ Process measurements should be used to assess process improvements
 - But this does not mean that measurements should drive the improvements. The improvement driver should be the organizational objectives.

Process metrics



- ✧ Time taken for process activities to be completed
 - E.g. Calendar time or effort to complete an activity or process.
- ✧ Resources required for processes or activities
 - E.g. Total effort in person-days.
- ✧ Number of occurrences of a particular event
 - E.g. Number of defects discovered.

Capability maturity levels



The SEI capability maturity model



- ✧ Initial
 - Essentially uncontrolled
- ✧ Repeatable
 - Product management procedures defined and used
- ✧ Defined
 - Process management procedures and strategies defined and used
- ✧ Managed
 - Quality management strategies defined and used
- ✧ Optimising
 - Process improvement strategies defined and used

Key points



- ✧ Software processes are the activities involved in producing a software system. Software process models are abstract representations of these processes.
- ✧ General process models describe the organization of software processes.
 - Examples of these general models include the ‘waterfall’ model, incremental development, and reuse-oriented development.
- ✧ Requirements engineering is the process of developing a software specification.

Key points



- ✧ Design and implementation processes are concerned with transforming a requirements specification into an executable software system.
- ✧ Software validation is the process of checking that the system conforms to its specification and that it meets the real needs of the users of the system.
- ✧ Software evolution takes place when you change existing software systems to meet new requirements. The software must evolve to remain useful.
- ✧ Processes should include activities such as prototyping and incremental delivery to cope with change.

Key points

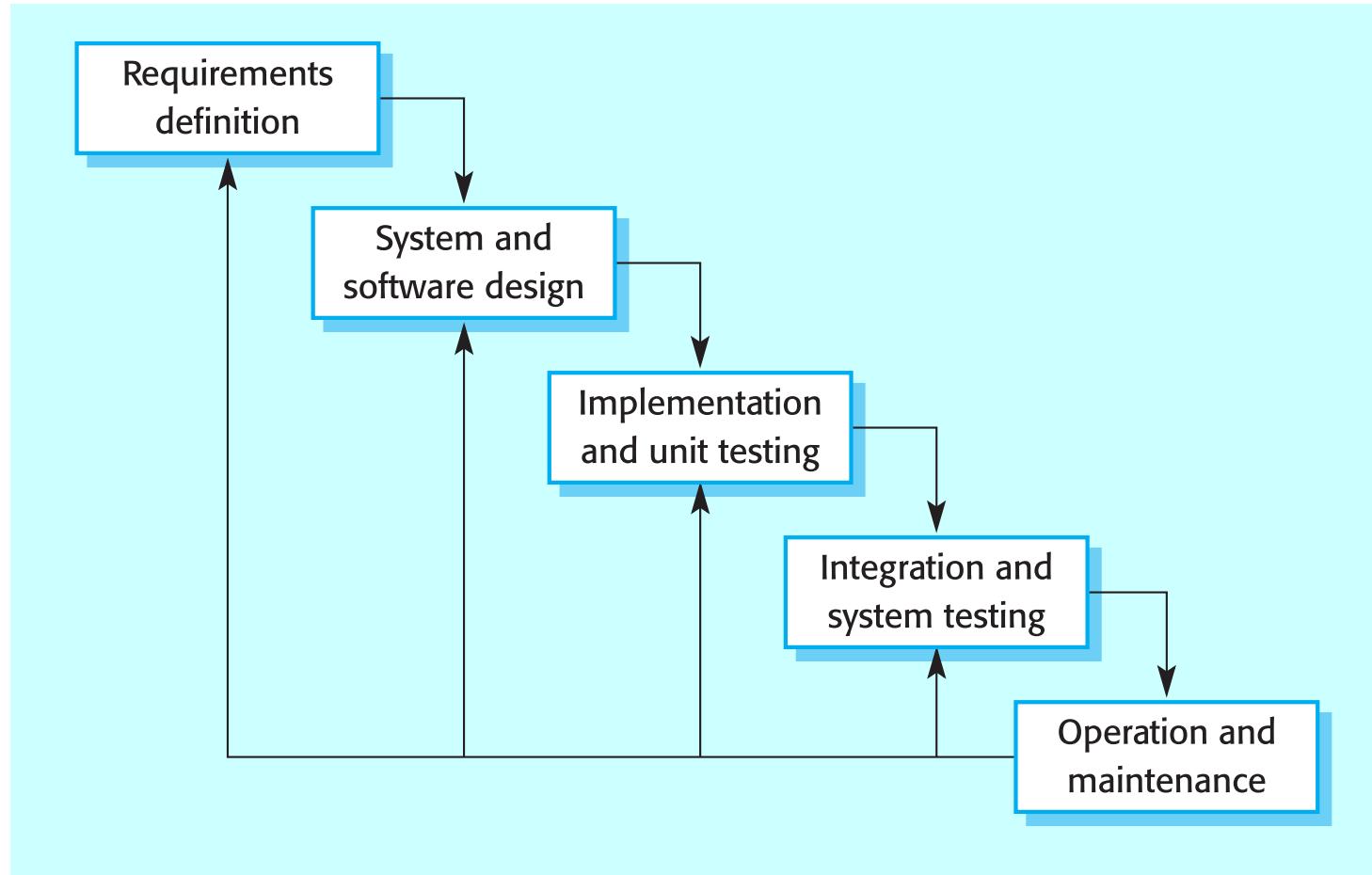


- ✧ Processes may be structured for iterative development and delivery so that changes may be made without disrupting the system as a whole.
- ✧ The principal approaches to process improvement are agile approaches, geared to reducing process overheads, and maturity-based approaches based on better process management and the use of good software engineering practice.
- ✧ The SEI process maturity framework identifies maturity levels that essentially correspond to the use of good software engineering practice.

Geleneksel Yazılım Süreç Modelleri

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

Çağlayan (“Waterfall”) Modeli



Çağlayan Modeli – Aşamalar

- **Gereksinim Tanımlama:** Gerçekleştirilecek sistemin gereksinimlerinin belirlenmesi işidir.
 - ▶ Müşteri ne istiyor? Ürün ne yapacak, ne işlevsellik gösterecek?
- **Tasarım:** Gereksinimleri belirlenmiş bir sistemin yapısal ve detay tasarımını oluşturma işidir.
 - ▶ Ürün, müşterinin beklediği işlevselligi nasıl sağlayacak?
- **Gerçekleştirme ve Birim Test:** Tasarımı yapılmış bir yazılım sisteminin kodlanarak gerçekleştirilmesi işidir.
 - ▶ Yazılım ürünü, tasarımı gerçekleştirecek şekilde kodlandı mı?
- **Tümleştirme ve Test:** Gerçekleştirilmiş sistemin beklenen işlevselligi gösterip göstermediğini sınama işlemidir.
 - ▶ Ürün, müşterinin beklediği işlevselligi sağlıyor mu?
- **İşletme ve Bakım:** Müşteriye teslim edilmiş ürünü, değişen ihtiyaçlara ve ek müşteri taleplerine göre güncelleme işidir.
 - ▶ Ürün müşteri tarafından memnuniyetle kullanılabiliyor mu?

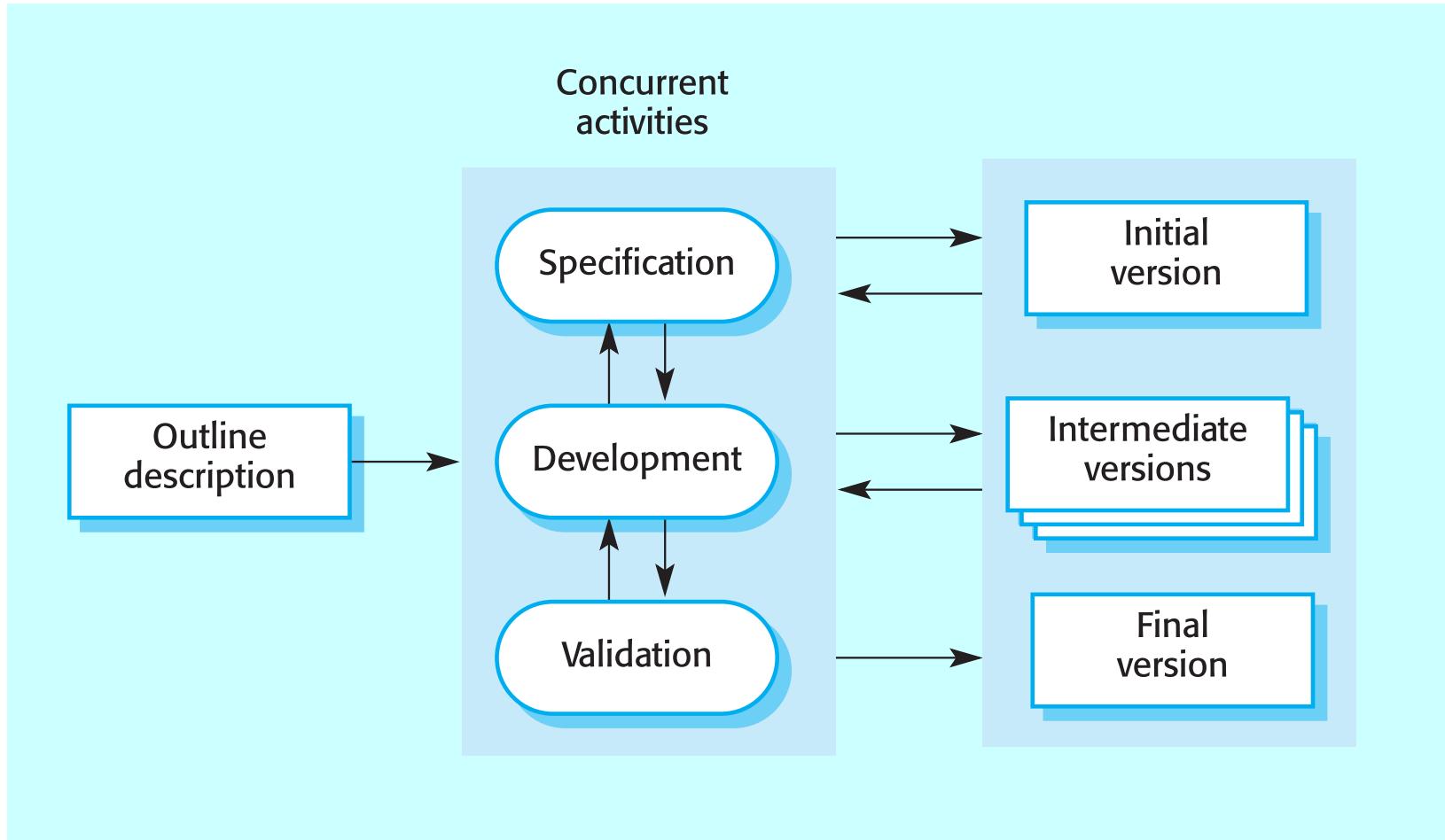
Çağlayan Modeli – Zorluklar

- Bir sonraki aşamaya geçmeden, önceki aşama neredeyse tümüyle tamamlanmış olmalıdır (örneğin, gereksinim tanımlama aşaması bitmeden tasarım aşamasına geçilemez.)
 - ▶ Bu şekilde geliştirme boyunca değişen müşteri isteklerinin sisteme yansıtılması zorlaşır.
 - ▶ Önceki nedenle bu model, gereksinimleri iyi tanımlı ve değişiklik oranı az olacak sistemler için daha uygundur.
 - ▶ Çok az sayıda iş sisteminin gereksinimleri başlangıçta iyi şekilde tanımlanabilir. Bu zorluğu aşmak için; gereksinim tanımlama aşamasından önce iş gereksinimlerinin anlaşılması ve tanımlanması faydalı olabilir.
 - ▶ Daha çok, geniş kapsamlı sistem mühendisliği projeleri için tercih edilir.

Evrimsel (“Evolutionary”) Model

- Sistem, zaman içinde kazanılan anlayışa göre gelişir.
 - ▶ Amaç, müşteriyle birlikte çalışarak taslak bir sistem gereksinimleri tanımından çalışan bir sisteme ulaşmaktır.
 - ▶ En iyi bilinen gereksinimlerle başlanır ve müşteri tarafından talep edildikçe yeni özellikler eklenir.
- Öğrenme amacıyla, sonradan atılabilecek prototipler (“throw-away prototyping”) geliştirilir.
 - ▶ Amaç, sistem gereksinimlerini anlamaktır.
 - ▶ En az bilinen gereksinimlerle başlanır ve gerçek ihtiyaç anlaşılmaya çalışılır.

Adımlar



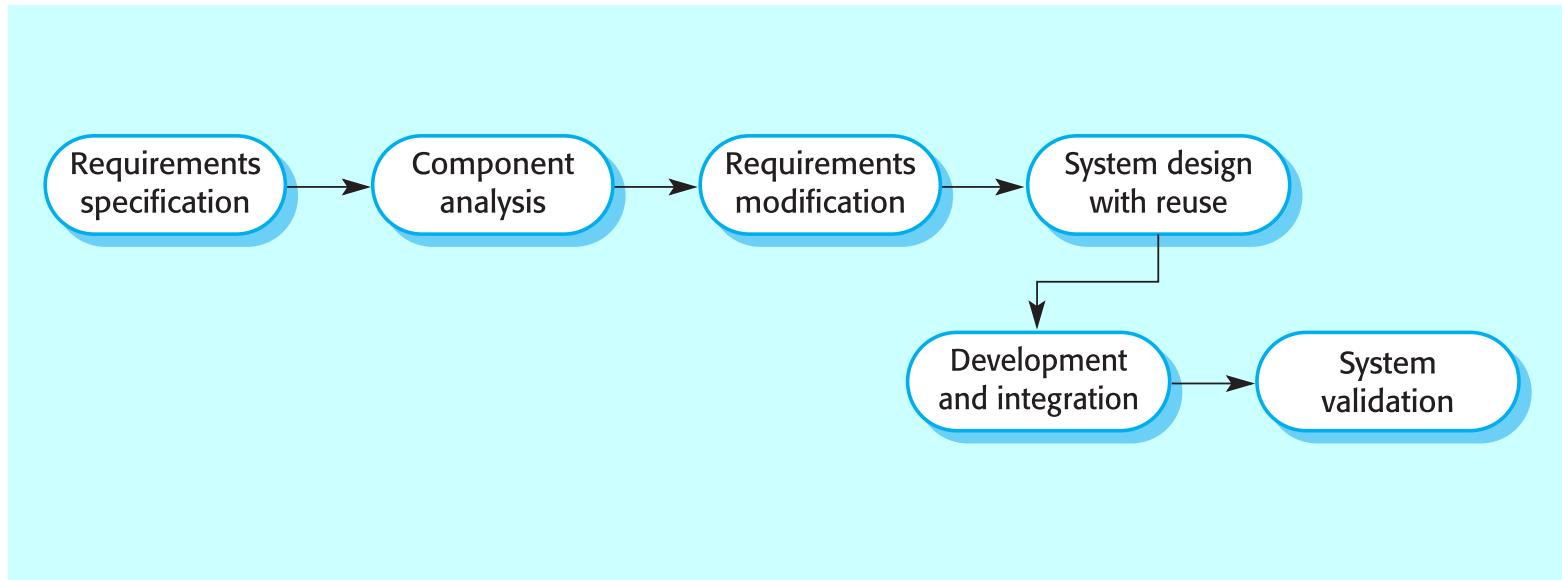
Zorluklar

- Geliştirme süreci izlenebilir değildir. Her seferinde eklemelerle çalışan sistem, müşteriyle gözden geçirilir.
- Zaman içinde kazanılan anlayışa göre geliştirilen sistemler, sıkılıkla kötü tasarlanır.
- Küçük- ve orta-ölçekli, etkileşimli (“interactive”) sistemler için uygulanabilir.
- Daha büyük ölçekli sistemlerin belirli bir bölümü (örneğin, kullanıcı arayüzleri) için uygulanabilir.
- İdamesi nispeten kısa sürecek sistemler için uygulanması önerilir.
 - ▶ Uzun yıllar idame edilecek sistemler, kötü / kötüleşen tasarım sebebiyle etkin çalışmayaçaktır.

Bileşen-Tabanlı (“Component-Based”) Model

- Sistemin COTS (“commercial-off-the-shelf”) adı verilen hazır bileşenler kullanılarak tümleştirilmesi esasına dayanır.
- Süreç adımları:
 - ▶ Bileşen analizi
 - ▶ Gereksinim günleme
 - ▶ Bileşenler kullanarak sistem tasarıımı
 - ▶ Geliştirme ve tümleştirme
- Bu yaklaşım, bileşen standartlarındaki gelişmeler ilerledikçe daha yaygın olarak kullanılmaya başlanmıştır.

Bileşen-Tabanlı (“Component-Based”) Model – Adımlar



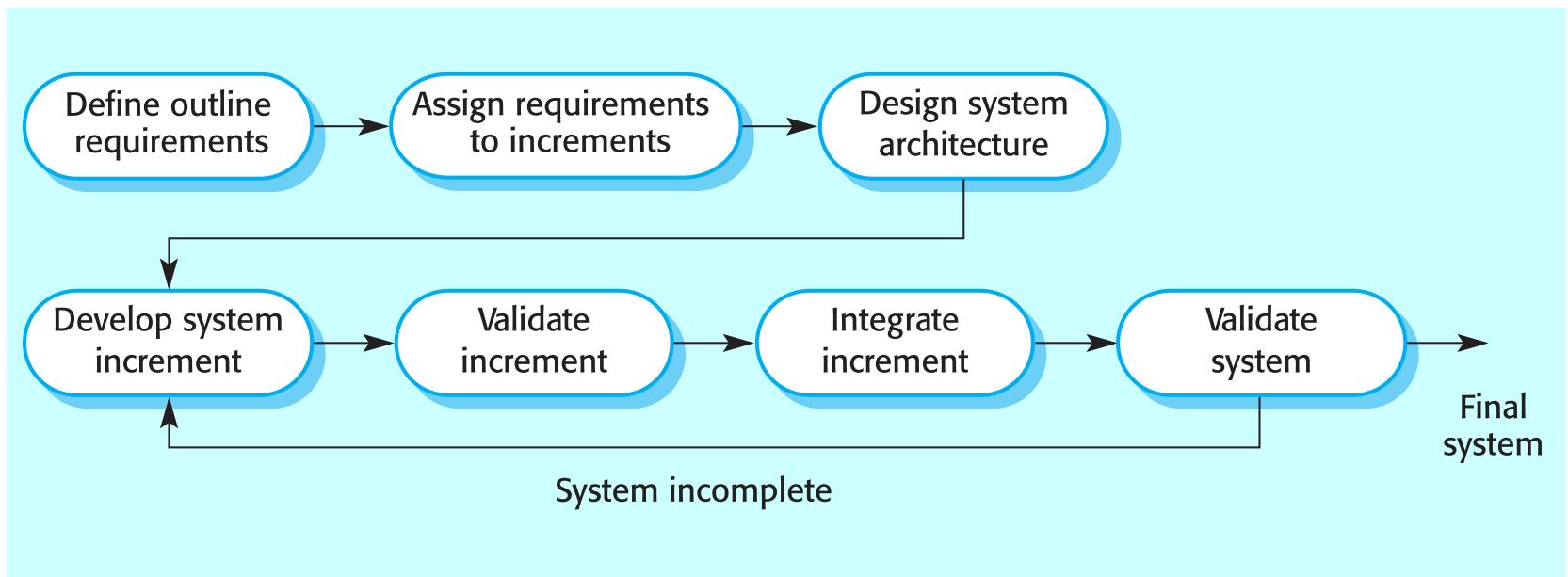
Yazılım Süreç Modellerinde Süreç Tekrarı (“Process Iteration”)

- Yazılım süreç modelleri tek bir defada uygulanmak yerine, birkaç tekrarda uygulanabilir.
 - ▶ Örneğin, geniş kapsamlı 5 alt sistemden oluşan bir sistemin; ilk alt sistemi için çağrıyan modeli uygulandıktan sonra, geri kalanı için çağrıyan modeli tekrar uygulanabilir.
 - ▶ Bu şekilde geliştirme riskleri en aza indirilerek ilk tekrarda kazanılan deneyimden, sistemin geri kalanı geliştirilirken faydalанılabilir.
- Hangi süreç modelinin, sistemin hangi bölümleri için ve kaç tekrarda uygulanacağına proje başında karar verilir.
- Süreç tekrarıyla yakından ilişkili iki geleneksel model vardır:
 - ▶ Artırımlı (“incremental”) model
 - ▶ Döngüsel (“spiral”) model

Artırımlı (“Incremental”) Model

- Sistemi tek seferde teslim etmek yerine, geliştirme ve teslim parçalara bölünür. Her teslim beklenen işlevselligin bir parçasını karşılar.
- Kullanıcı gereksinimleri önceliklendirilir ve öncelikli gereksinimler erken teslimlere dahil edilir.
- Bir parçanın geliştirmesi başladığında, gereksinimleri dondurulur. Olası değişiklikler sonraki teslimlerde ele alınır.

Artırımlı Model – Adımlar



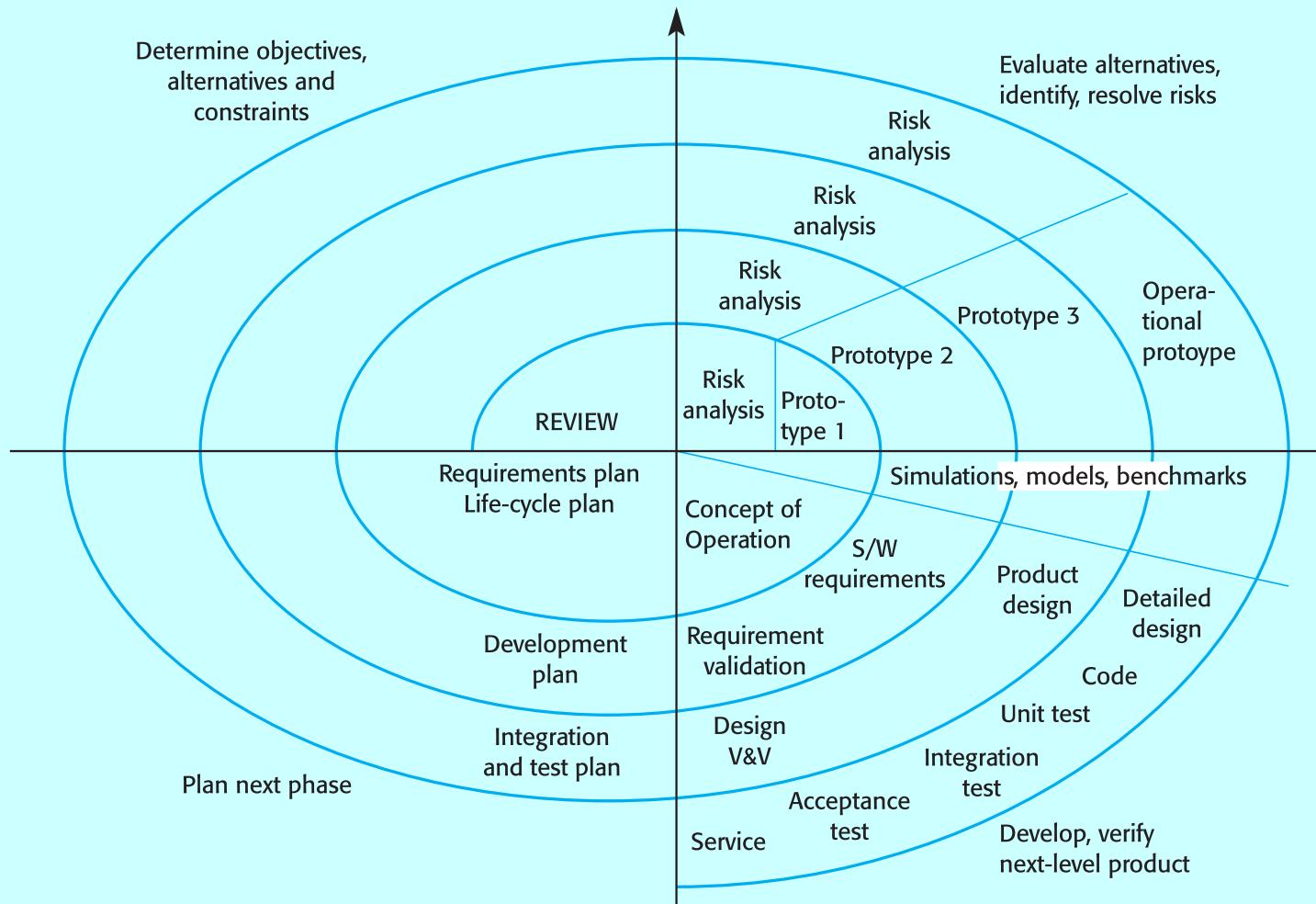
Artırımlı Model – Kazançlar

- Her teslimle birlikte müşteriye görünen bir değer döndüğünden, sistemin işlevselligi erken aşamalarda ortaya çıkar.
- Erken teslimler, sonraki teslimler için gereksinimleri çıkarmada prototip vazifesi görür.
- Projenin tümden batması riskini azaltır.
- Öncelikli gereksinimleri karşılayan sistem işlevleri daha çok test edilir.

Döngüsel (“Spiral”) Model

- Süreç, geri dönüşümlü etkinlikler zinciri yerine döngüsel olarak ifade edilir.
- Her döngü, süreçteki bir aşamayı ifade eder.
- 4 sektörden oluşur:
 - ▶ Hedef belirleme: Aşamanın başarımı için somut hedefler belirlenir.
 - ▶ Risk değerlendirme ve azaltma: Riskler adreslenerek azaltıcı eylemler gerçekleştirilir.
 - ▶ Geliştirme ve doğrulama: Genel modeller içinden geliştirme için bir model seçilir.
 - ▶ Planlama: Proje gözden geçirilir ve bir sonraki aşama planlanır.
- Riskler süreç boyunca özel olarak ele alınır ve çözümlenir.
- Tanımlama ve tasarım gibi sabit aşamalar yoktur; her döngü ihtiyaca göre seçilir.

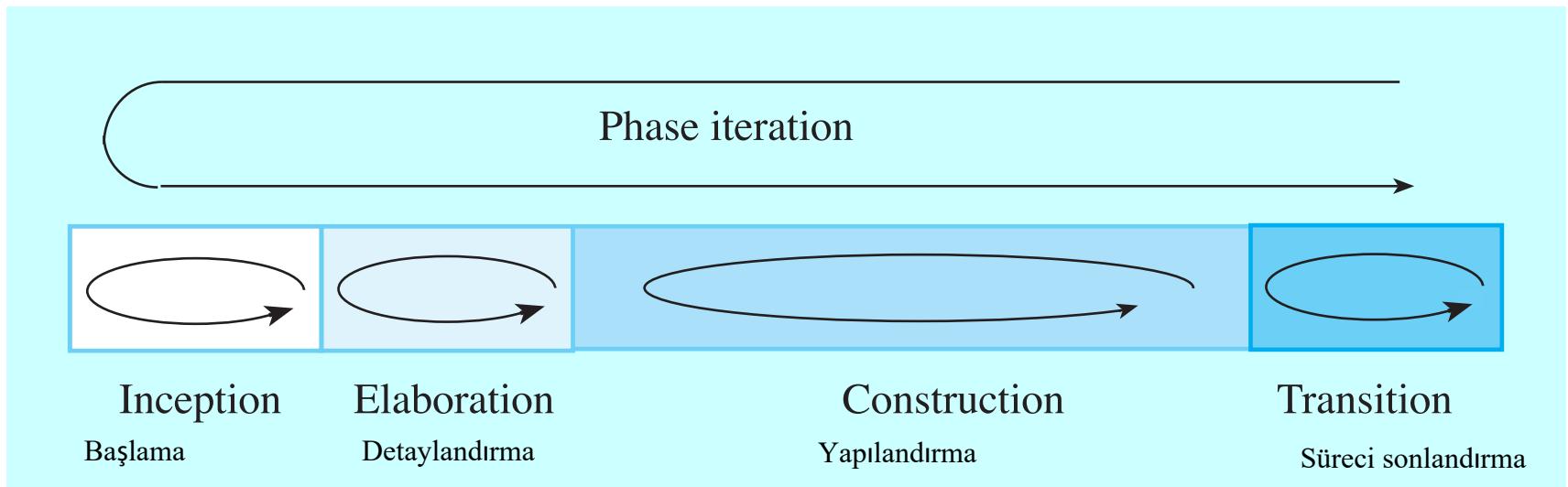
Döngüsel Geliştirme – Sektörler



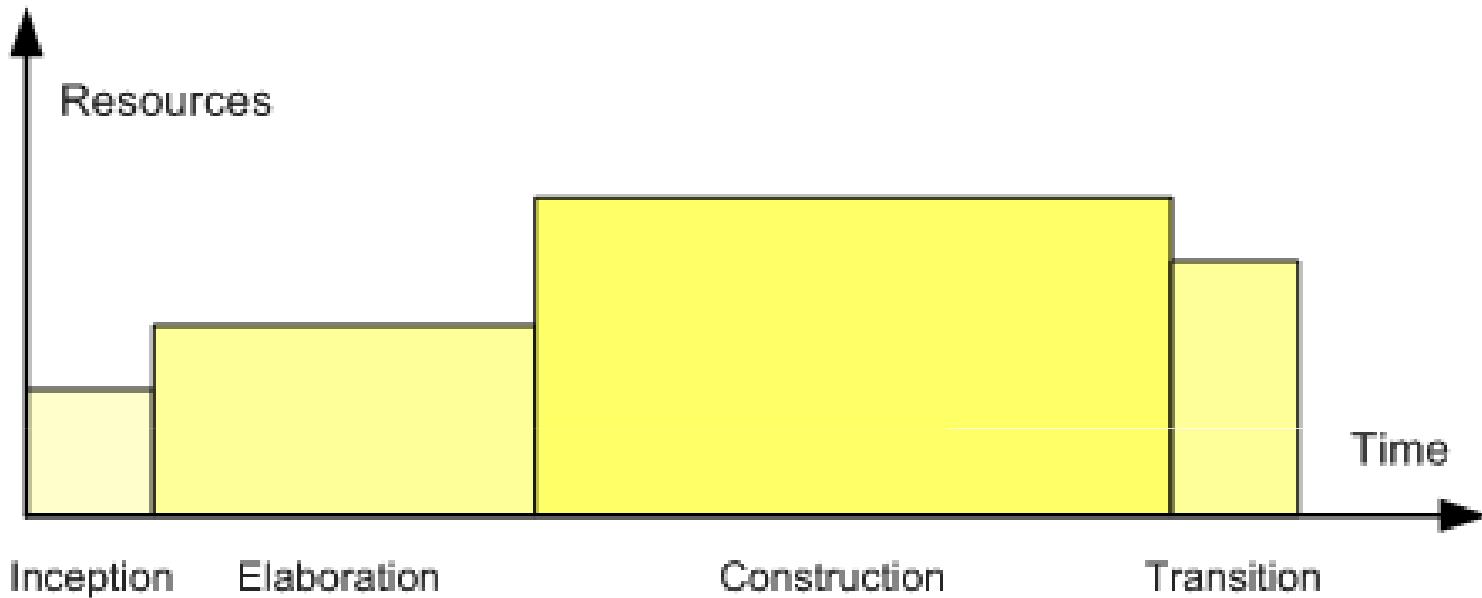
Örnek Bir Yazılım Süreç Modeli: Tümleşik Süreç (“Unified Process”)

- 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 – Aşamalar



Tümleşik Süreç – Yaşam Döngüsü



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

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

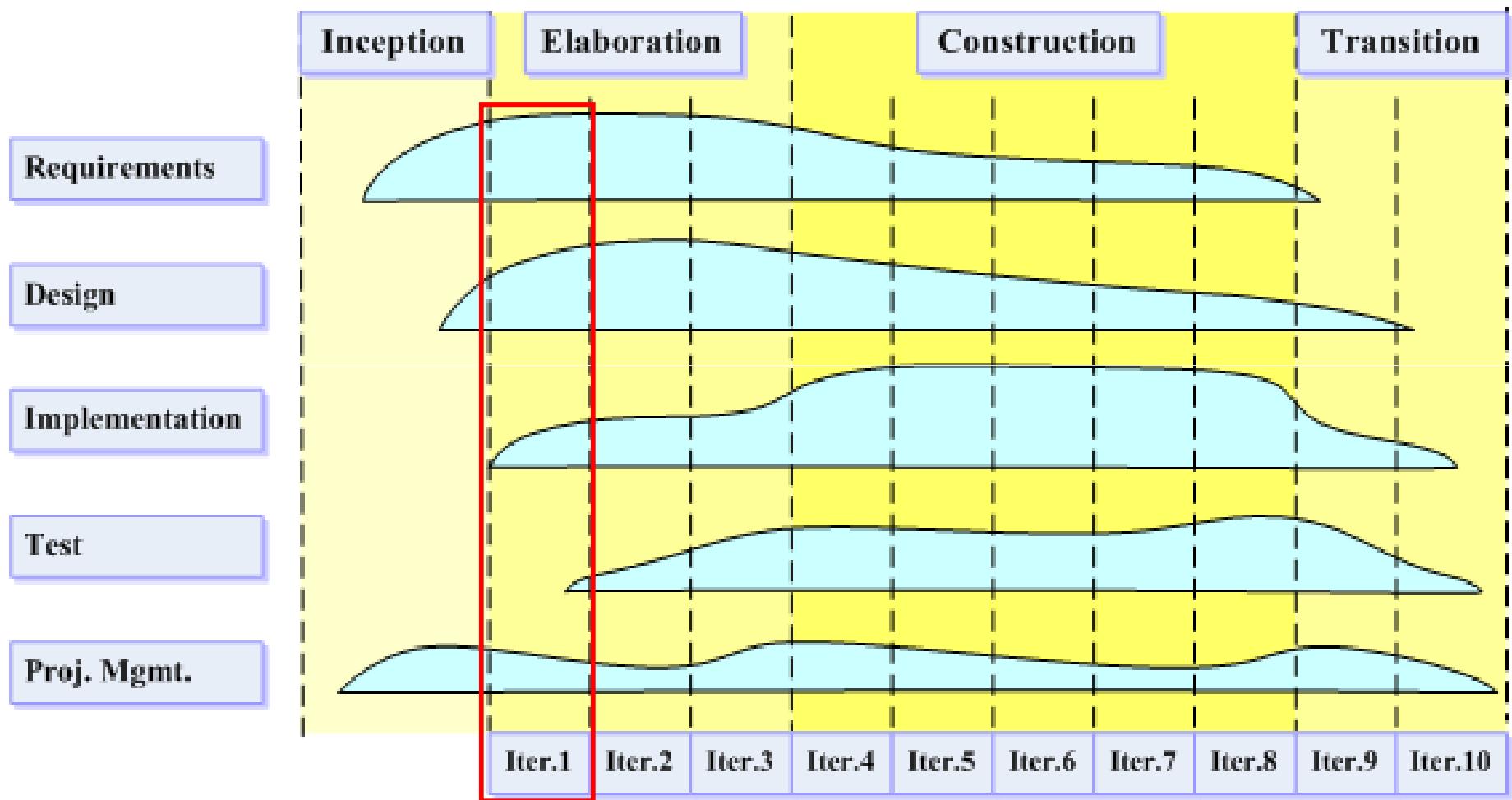
“Construction” – Ürünü gerçekleştir

“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



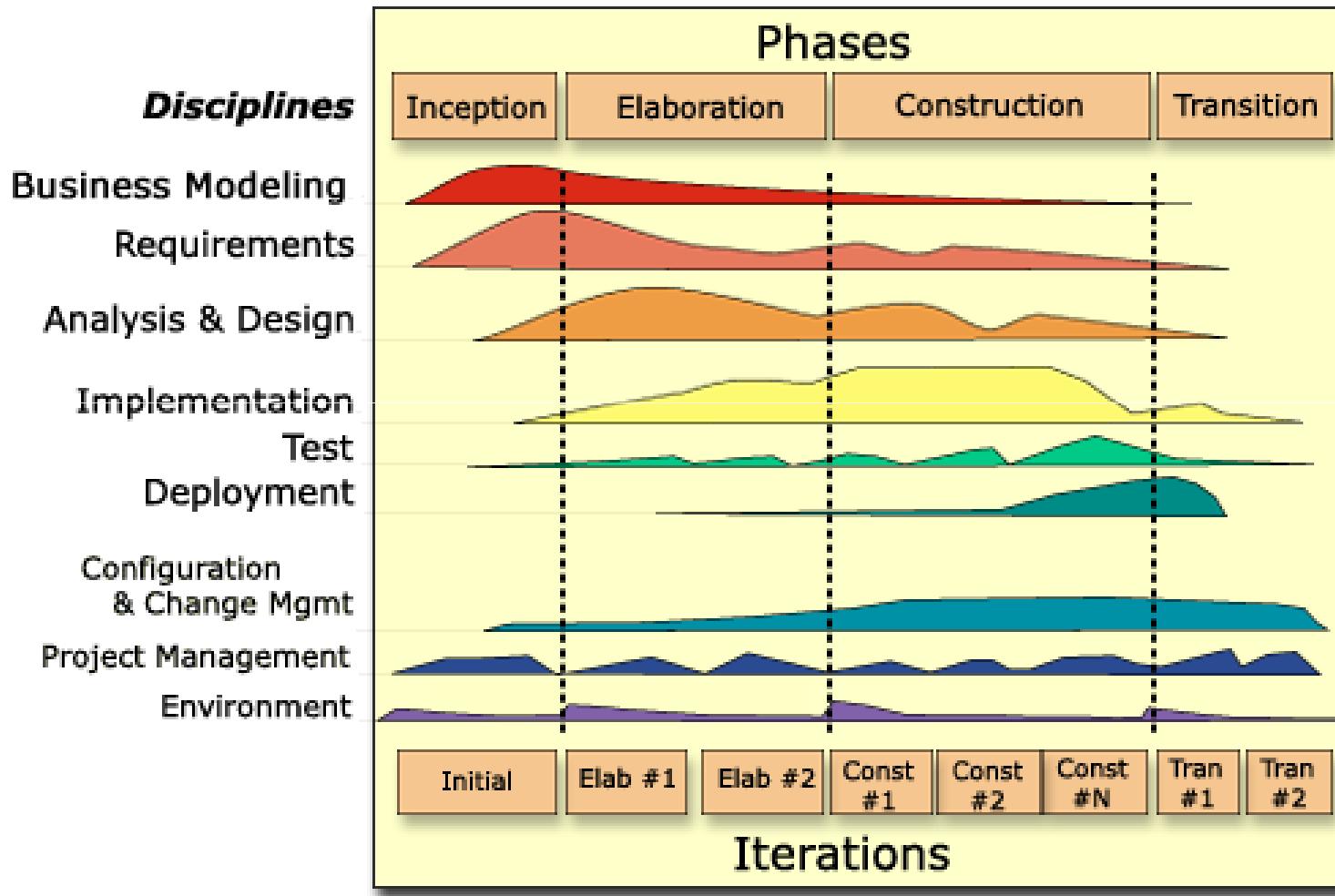
Tekrarlı Geliştirme ("Iterative Development")



Each iteration results in an executable release.

Her tekrar ("iteration"), tanımlı bir plan çerçevesinde gerçekleştirilen ve değerlendirme kriterleri tanımlanmış bir dizi etkinlikten oluşur.

“The Rational Unified Process” (Rational Software'in Tümleşik Süreci)



Kaynaklar ...

■ Kaynaklar:

- ▶ “Software Engineering” (10th. Ed.), Ian Sommerville, 2015.
- ▶ “*Guide to the Software Engineering Body of Knowledge*”, 2004.
- ▶ “The Unified Software Development Process”, ISBN 0-201-57169-2, 1999.
Ivar Jacobson, Grady Booch, James Rumbaugh.
- ▶ “*CMMI for Development*”, v1.2, 2006.
- ▶ “*A Guide to the Project Management Body of Knowledge*”, 1996.



BM306

Yazılım Mühendisliği



DERS 2

Yazılım Süreç Modelleri-devam

Dr. Öğr. Üyesi Hasan BADEM
hbadem@ksu.edu.tr



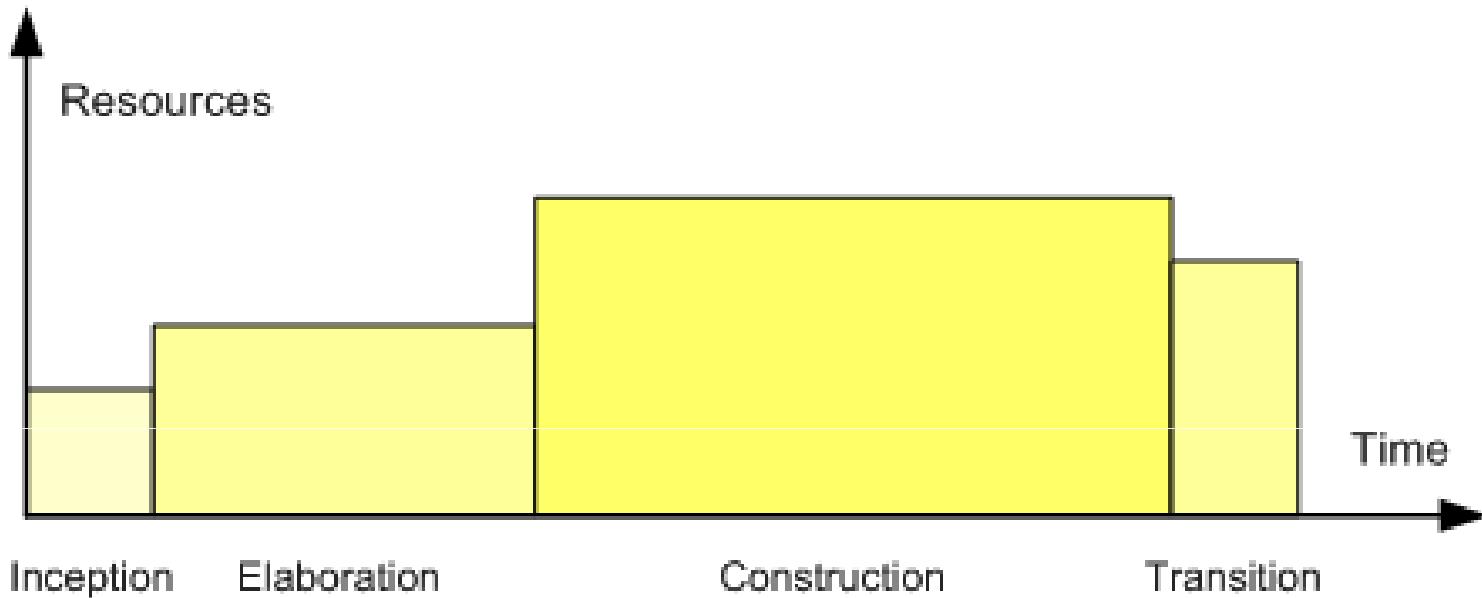
Geleneksel Yazılım Süreç Modelleri

- 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

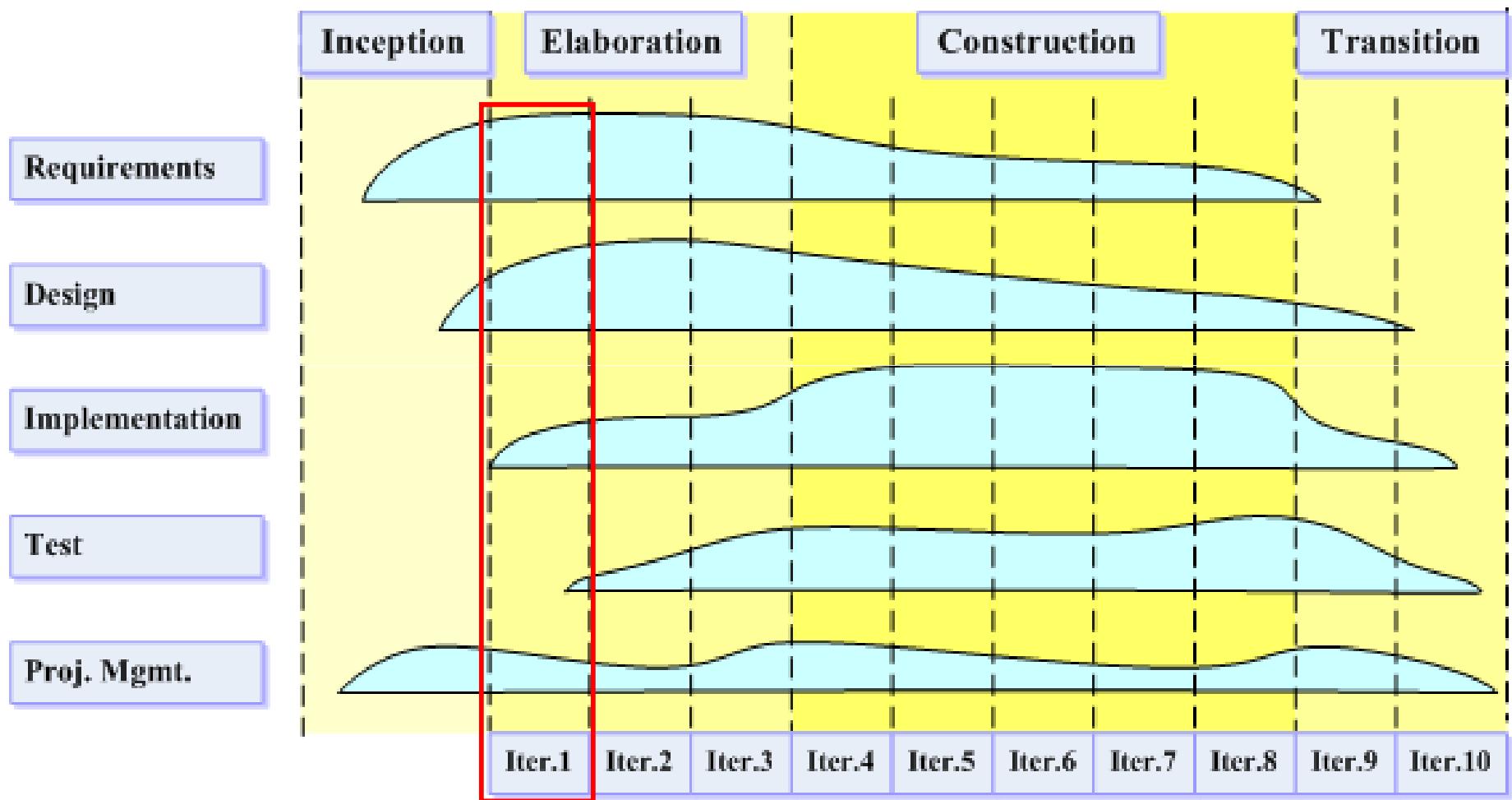
“Construction” – Ürünü gerçekleştir

“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

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 davranışan, tetik, atık.* [TDK]
 - ▶ 1950'lerdeki üretim alanında verimliliğin artırılması için geliştirilen yalın yaklaşımının, 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 metodlarına 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 yayımlamışlardır. 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.
- ▶ Mمmkün olduğunda 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 olduğunda sabit hızlı, sürdürülebilir geliştirmeye önem verir.
- ▶ Sağlam teknik alt yapı ve tasarım çevikliği arttırmır.
- ▶ Basitlik önemlidir.
- ▶ En iyi mimariler, gereksinimler ve tasarımlar kendini organize edebilen ekipler tarafından yaratılır.
- ▶ Düzenli aralıklarla ekip kendi yöntemlerini gözden geçirerek verimliliği artırmak için gerekli iyileştirmeleri yapar.

Çevik Yazılım Süreç Modelleri

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

Uçdeğer Programlama ("Extreme Programming - XP")

- Sınırsız 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 standarı
 - ▶ Sürekli entegrasyon ("continuous integration")
 - ▶ Haftada 40 saat çalışma

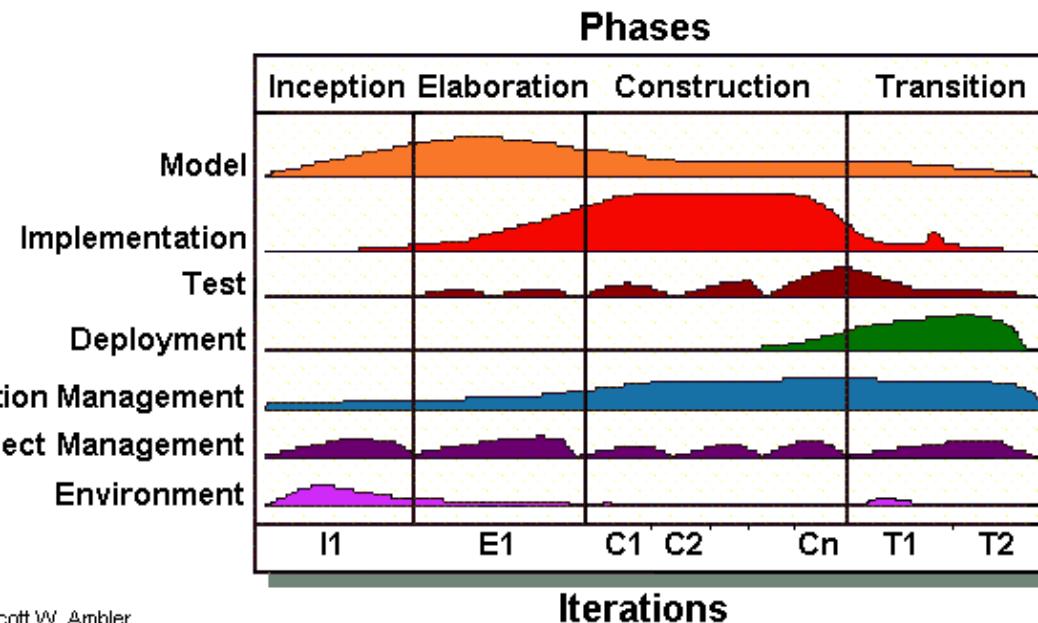
- Jeff Sutjerland ve Ken Schawaber tarafından 1990'ların ortalarında geliştirilen, proje yönetimi ve planlama ile ilgili yöntemlere odaklı olan ve mühendislik detayları içermeyen bir modeldir.
- Yazılımı küçük birimlere (sprint) bölerek, yinelemeli olarak geliştirmeyi öngörür.
- Bir yinelemenin tanımlanması 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 Gündümlü Geliştirme ("Feature Driven Development - FDD")

- Jeff De Luca ve Peter Coad tarafından 1997'de geliştirilen ve özellik tabanlı gerçekleştirmi 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 gündümlü bir planlama yapılması,
 - ▶ Özellik gündümlü tasarımın oluşturulması,
 - ▶ Özellik gündümlü 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 yaklaşımına 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 gelişmiş 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 dönüştürüler
- Ancak bu gelişmelerden hiçbirini, 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.

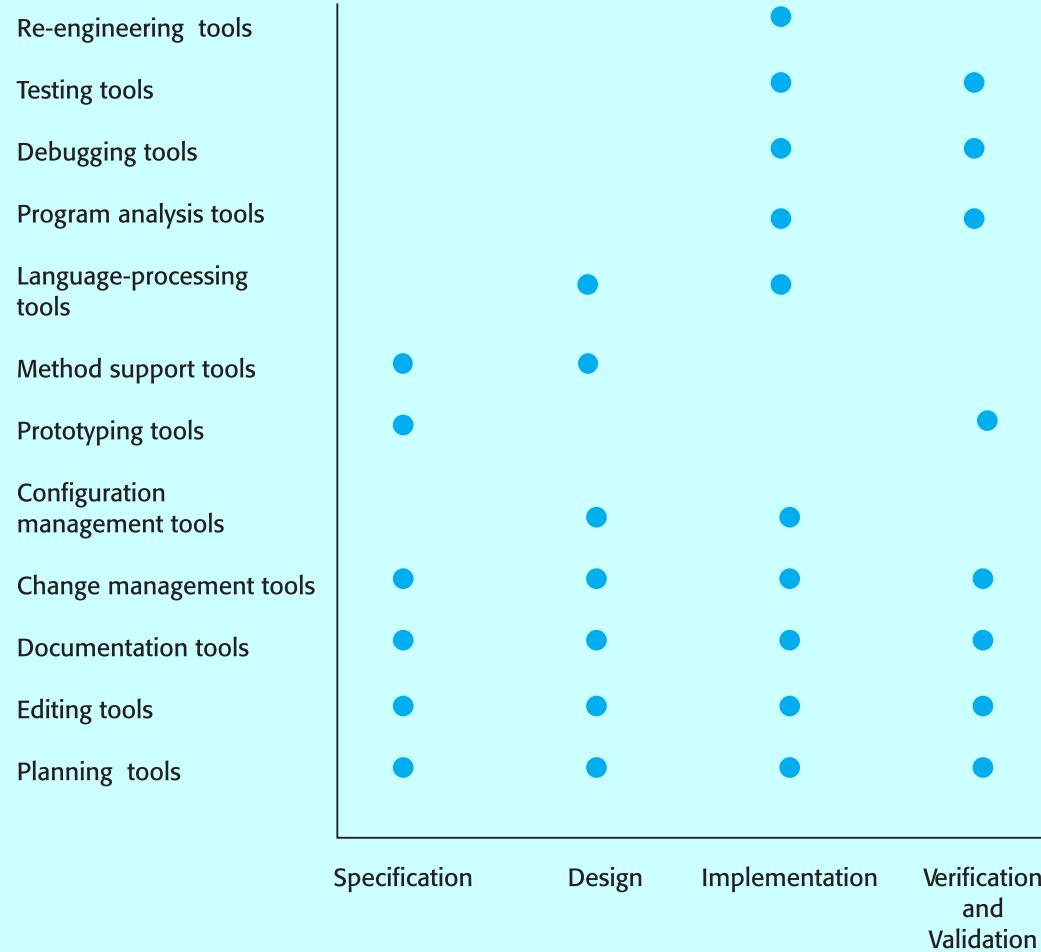
“CASE” Sınıflandırma

- CASE araçlarını sınıflandırmak, farklı tiplerdeki araçları ve yazılım sürecini nasıl desteklediklerini anlamayı kolaylaştırır:
 - ▶ İşlevsel bakış açısı (“functional perspective”)
 - ◆ İşlev'e göre sınıflandırma
 - ▶ 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.

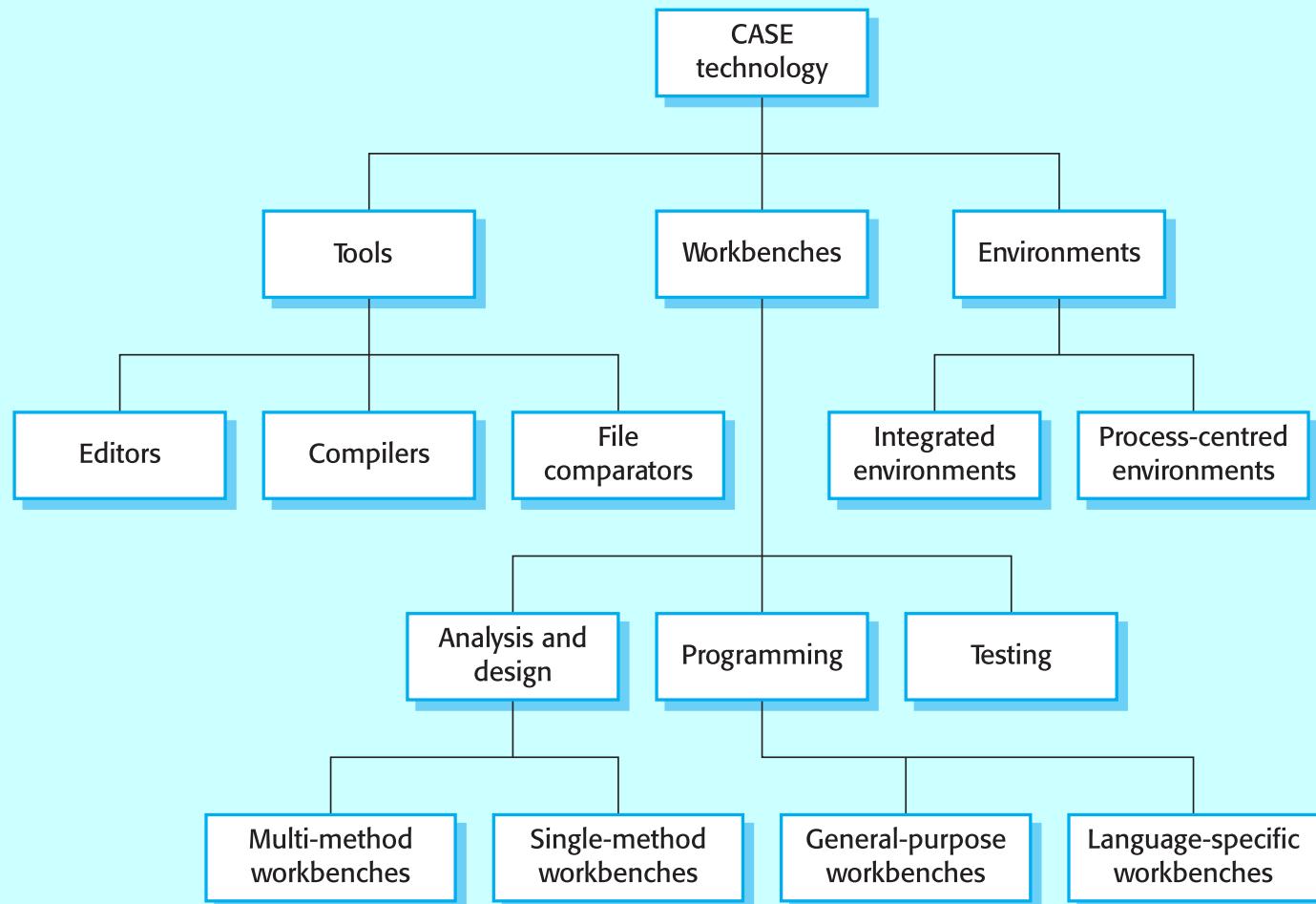
■ “Workbenches”

- ▶ 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)

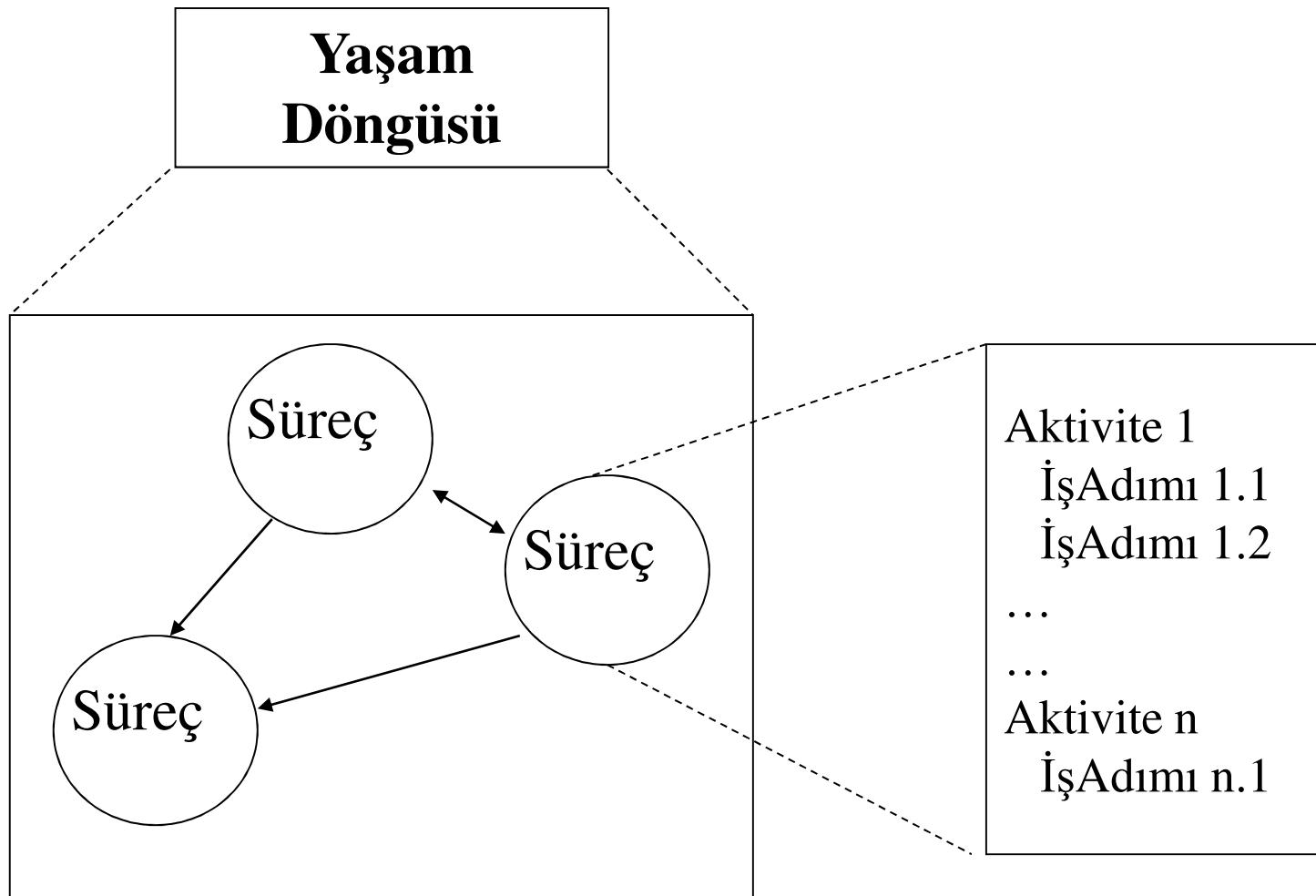


Yaşam Döngüsü Süreçleri

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 uyarılama kuralları ve önerileri dikkate alınmalıdır.
 - ▶ Tam uyumluluk / Seçerek uyarılama

ISO/IEC 12207 Genel Yapısı



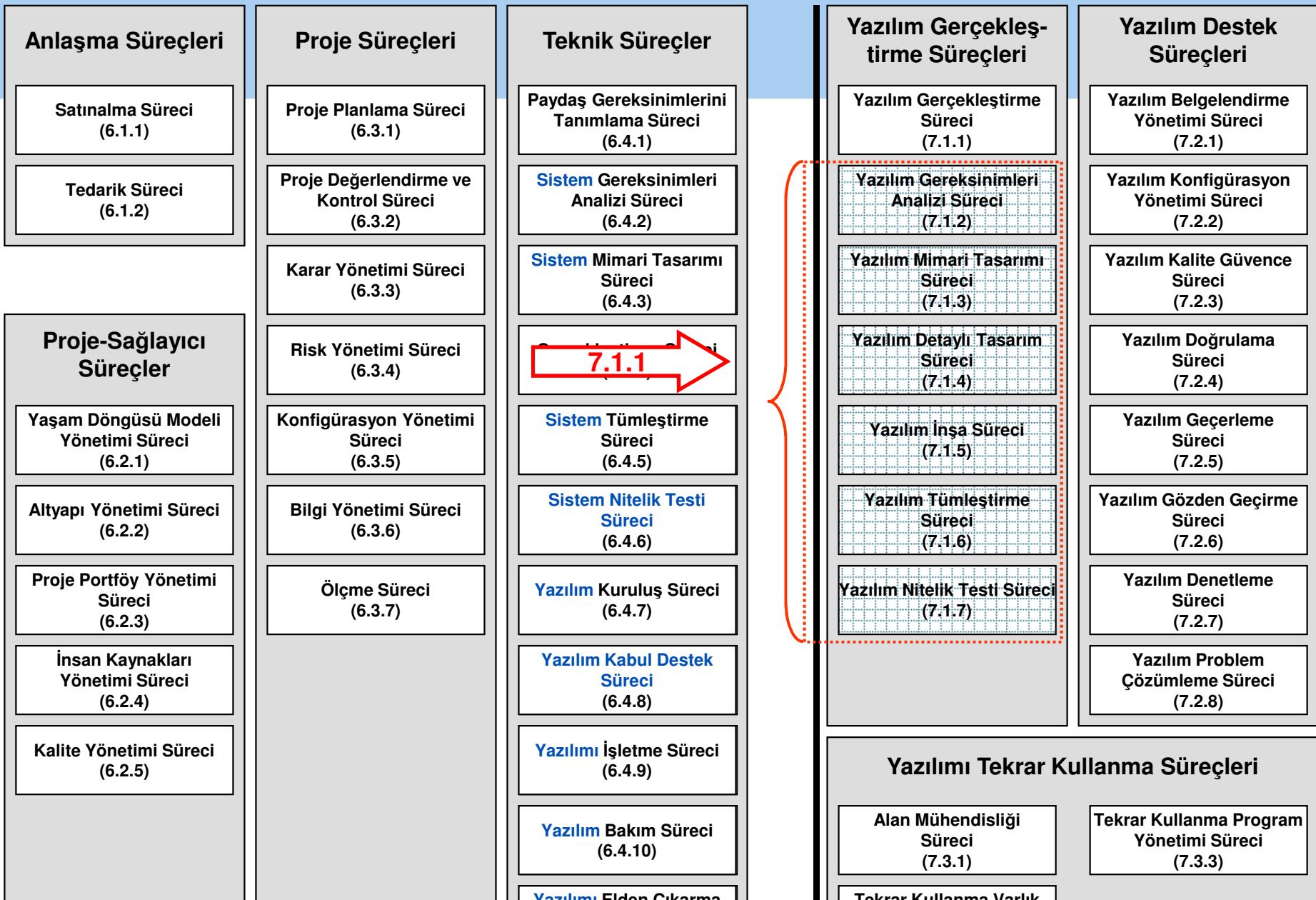
ISO/IEC 12207 Kapsamı

- 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 satın alınması, tedarıği, 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 üretecek 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 (1228)



Sistem Kapsamlı Süreçler

Anlaşma Süreçleri

Satınalma Süreci
(6.1.1)

Tedarik Süreci
(6.1.2)

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)

Yazılıma Özel Süreçler

Yazılım Gerçekleştirme 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 Program
Yönetimi Süreci
(7.3.3)

Tekrar Kullanma Varlık

Örnek: Yazılım Geçerleme Süreci (7.2.5)

■ Amaç:

- ▶ Yazılım iş ürününün, amaca özel kullanımına yönelik gereksinimleri karşıladığı 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 işgürünü garanti edip edemeyeceği ve bu işgürünün ne derecede kurumsal bağımsızlık gerektirdiği belirlenecektir.
2. Proje geçerleme işgürünü garanti ediyorsa, bir geçerleme süreci oluşturulacaktır. Geçerleme görevleri; ilişkili yöntemler, teknikler ve araçlarla birlikte, seçilecektir.
3. Proje bağımsız bir geçerleme işgürünü 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")

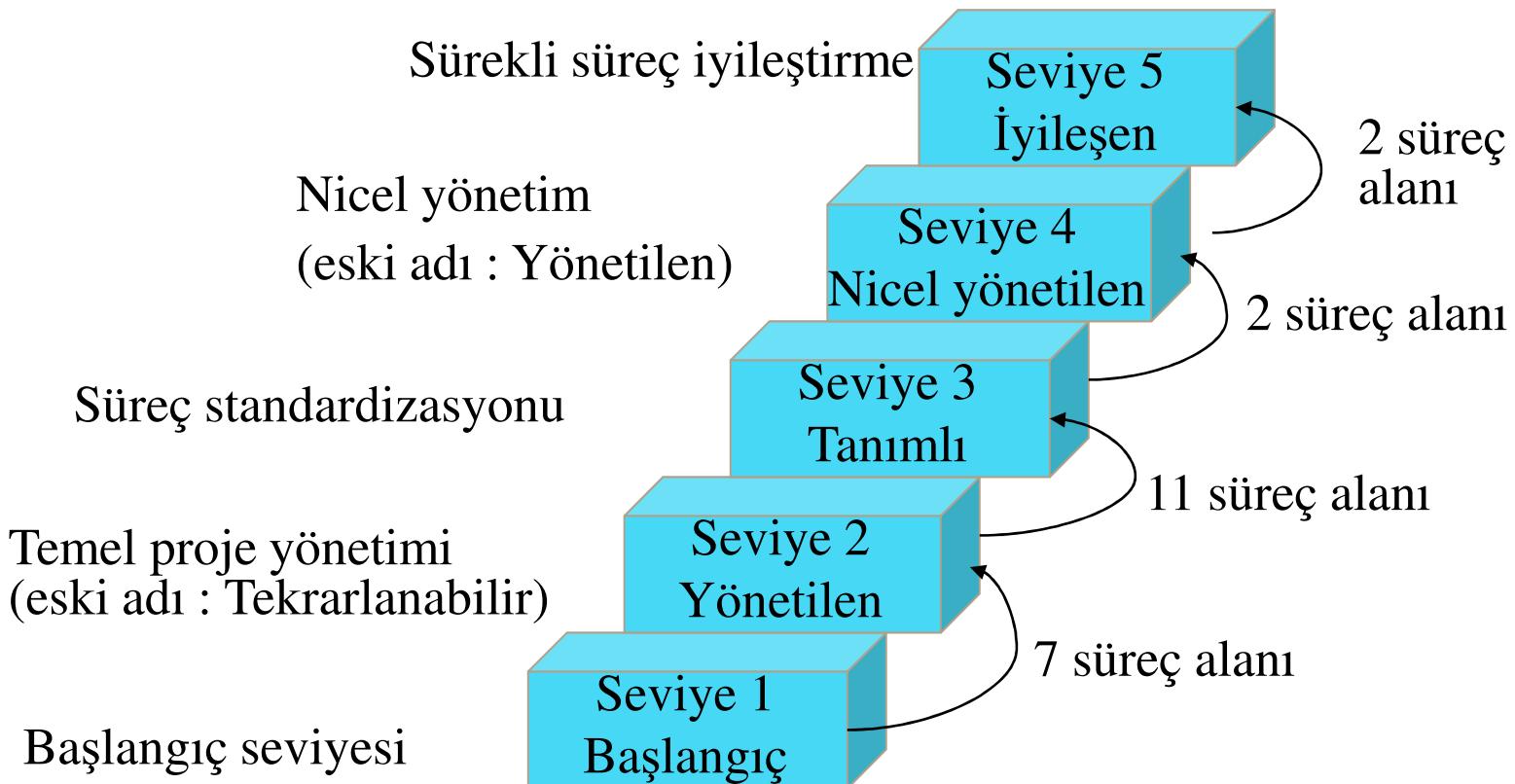
Capability Maturity Model (CMM)

- Toplam kalite yönetimi kavramlarının yazılım geliştirme alanına uyarlanması
 - ▶ “Crosby Maturity Grid” temel alınmıştır
 - ▶ Sisteminin kalitesini, üretimde ve bakımda kullanılan süreçlerin kalitesi belirler
 - ▶ Yazılım sektöründe yaşanan 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

- | | |
|---|---|
| <ul style="list-style-type: none">■ Gereksinim geliştirme■ Teknik çözüm■ Ürün entegrasyonu■ Doğrulama■ Geçerleme■ Kurumsal süreç odağı■ Kurumsal süreç tanımı | <ul style="list-style-type: none">■ 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”)

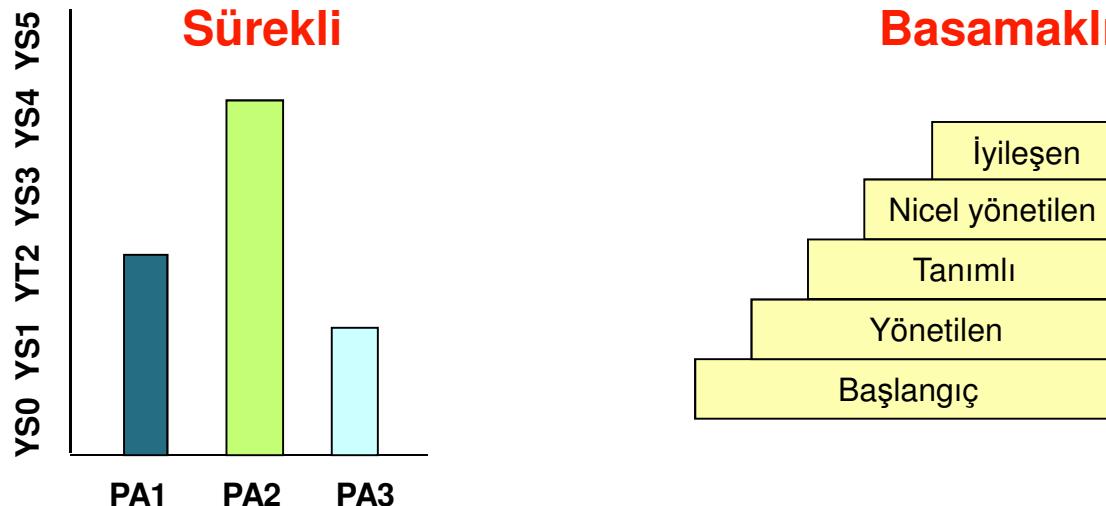
- Birlikte gerçekleştirildiklerinde, belirli bir alanda, süreç iyileştirme için önemli olan hedeflere ulaşılmasını sağlayacak bir grup uygulama
 - ▶ Her süreç alanının amacı “beklenen davranışı” tanımlar
 - ◆ Beklenen davranış
 - ▶ Uygulamalar süreç alanı hedeflerine ulaşılması için atılması gereken 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 Risk yönetimi Nicel proje yönetimi Bütünleşik proje yönetimi	Proje izleme ve kontrol Entegre proje yönetimi Tedarikçi anlaşma yönetimi Bütünleşik tedarikçi yönetimi
Mühendislik	Gereksinim yönetimi Teknik çözümleme Doğrulama	Gereksinim geliştirme Ürün entegrasyonu Geçerleme
Destek	Konfigürasyon yönetimi Süreç ve ürün kalite güvencesi Sebep analizi ve çözümleme Karar analiz ve çözümleme	Ölçme ve analiz Bütünleştirme için kurumsal ortam
Süreç yönetimi	Kurumsal süreç odağı Kurumsal eğitim Kurumsal yenilik ve yayılma	Kurumsal süreç tanımı Kurumsal süreç performansı

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 gereği 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 gerekliliğini kılar
 - ▶ Kullanılan süreçlerde nelerin iyileştirilmesi gerektiğini bilmek
 - ▶ CMMI'da tanımlanan süreçler arasındaki bağımlılıkları bilmek

Örnek Süreç Alanı: Geçerleme

■ Amaç:

- ▶ Ürünün hedeflenen çalışma ortamında hedeflenen kullanımını 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 değerlendirir	Geçerleme stratejisi, prosedürleri
GP2.10. Durumu üst yönetimle gözden geçir	-



BM306

Yazılım Mühendisliği



DERS 3

Gereksinim Mühendisliği

Dr. Öğr. Üyesi Hasan BADEM
hbadem@ksu.edu.tr



■ Gereksinimler

- ▶ Gereksinim nedir, türleri nelerdir?
- ▶ Gereksinim tanımlama

■ Gereksinim Mühendisliği Süreci

- ▶ Gereksinim çıkarma, analiz, geçerleme, yönetim

■ İş Gereksinimleri Analizi ve Gereksinim Analizi İlişkisi

- ▶ İş süreçlerini modelleme, gereksinim analizi

■ Sistem Modelleri

- ▶ Bağlam (“context”), süreç (“process”), davranış (“behavior”), veri (“data”), nesne (“object”) modelleri

Gereksinim Türleri ve Gereksinim Tanımlama

Gereksinim Mühendisliği

- **Gereksinim mühendisliği**; müşterinin sistemden istediği servisleri ve sistemin altında çalışacağı kısıtları ortaya çıkarma ve tanımlama sürecidir.
- **Gereksinimler**; gereksinim mühendisliği süreci boyunca ortaya çıkan sistem servislerinin ve kısıtlarının bir tanımıdır.

Gereksinim Nedir?

- Bir sistem servisinin / kısıtının üst seviyeli ifadesinden detaylı matematiksel tanımına kadar farklılık gösterebilir.
 - ▶ Sözleşme için teklife çağrı belgesine esas olabilir – farklı çözümlerin önerilmesine açık olmalıdır.
 - ▶ Sözleşmeye esas olabilir – detaylı olarak tanımlanmalıdır.
- Geliştirilecek sistemin sağlama gereken bir durum veya yetenek
- Sistemin bir iş ihtiyacını karşılaması için göstermesi gereken özellik
- ...

Kullanıcı ve Sistem Gereksinimleri

■ Kullanıcı gereksinimleri

- ▶ Doğal dilde ifade edilir.
- ▶ Sistemin sağlayacağı servislerin ve uyacağı kısıtların ifadesidir.
- ▶ Müşteriler için yazılır.

■ Sistem gereksinimleri

- ▶ Sistem işlevlerinin, servislerinin ve kısıtlarının detaylı tanımlarını içeren yapısal bir belge ile ifade edilir.
- ▶ Müşteri ve sağlayıcı arasındaki sözleşmenin bir bölümü olarak, sistemin neleri gerçekleştireceğini tanımlar.

Kullanıcı ve Sistem Gereksinimleri: Örnek

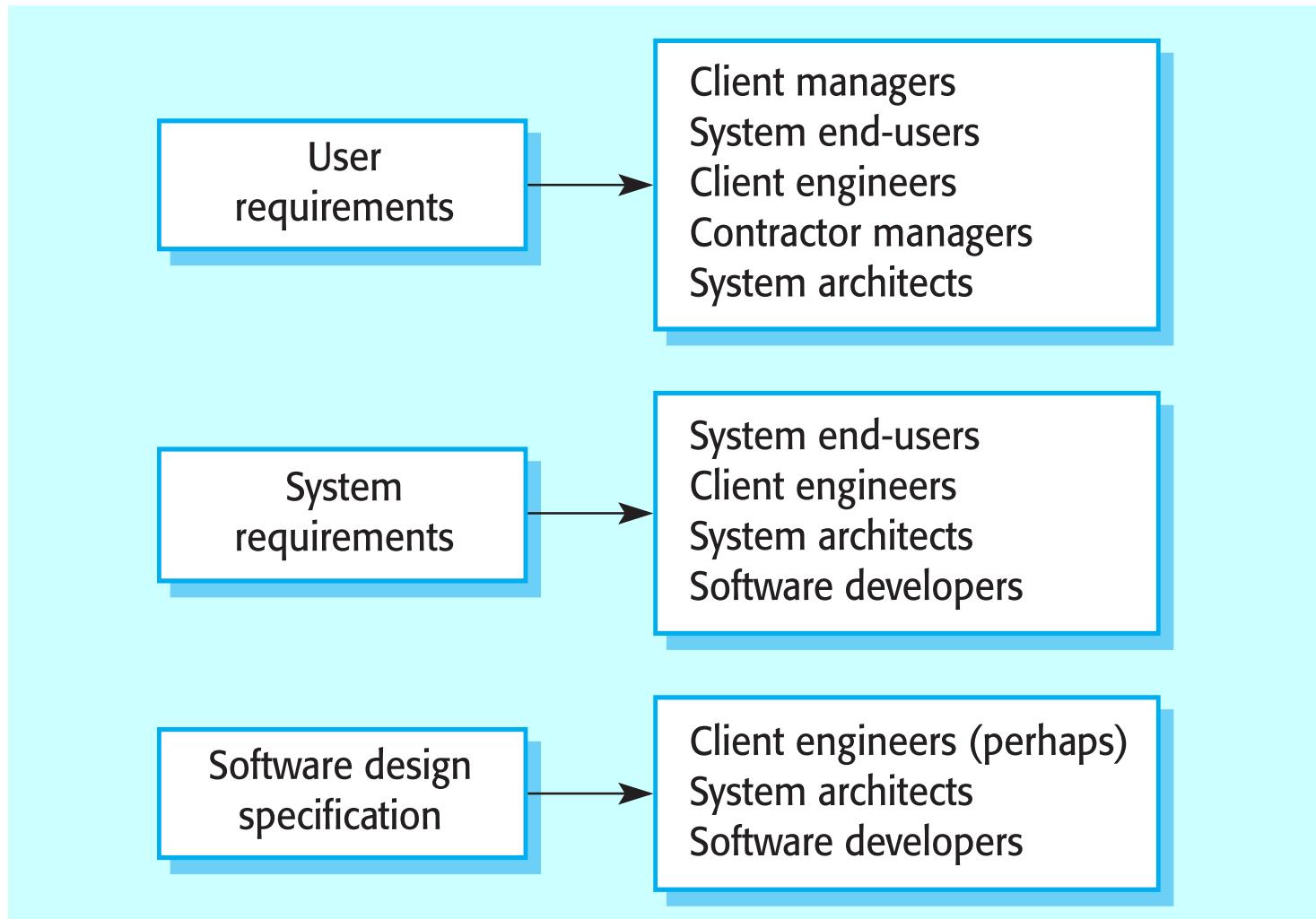
User requirement definition

1. The software must provide a means of representing and accessing external files created by other tools.

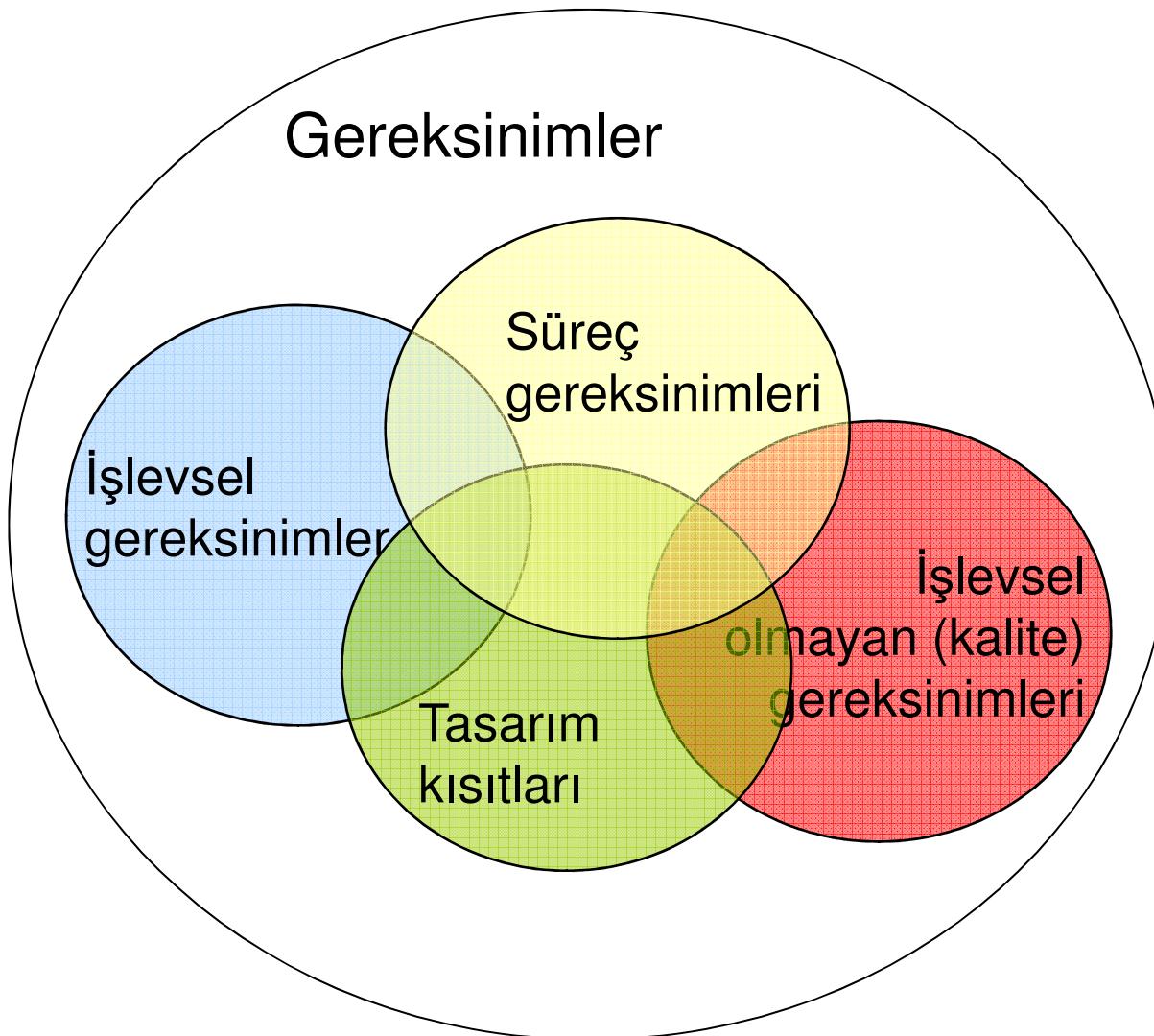
System requirements specification

- 1.1 The user should be provided with facilities to define the type of external files.
- 1.2 Each external file type may have an associated tool which may be applied to the file.
- 1.3 Each external file type may be represented as a specific icon on the user's display.
- 1.4 Facilities should be provided for the icon representing an external file type to be defined by the user.
- 1.5 When a user selects an icon representing an external file, the effect of that selection is to apply the tool associated with the type of the external file to the file represented by the selected icon.

Gereksinimler ve İlişkili Kişiler



Gereksinim Uzayı



Gereksinim Türleri (1)

■ İşlevsel gereksinimler

- ▶ Kullanıcıların sistem ile gerçekleştirmeyi istediği işlemler ve bu işlemlerin özellikleri
- ▶ Sistem “ne” yapacak ?
 - ◆ Girdilerin ve çıktıların tanımı
 - ◆ Girdileri çıktılara dönüştüren işlemlerin tanımı

Gereksinim Türleri (2)

■ İşlevsel olmayan gereksinimler

► Ürün kalite kriterleri

- ◆ Performans, güvenilirlik, kullanılabilirlik vb.
- ◆ Örnek: Hatalar arası ortalama zaman
 - “Bir sistem ya da sistem ögesi için hata oluşumları arasındaki ortalama zamandır.”
 - “Tanımlı bir işletim süresince oluşan hatalar sayılarak ve işletim süresi bu hata sayısına bölünerek hesaplanır.”
 - » $H.A.O.Z. = \text{işletim süresi} / \text{işletim süresince oluşan hata sayısı}$
 - “Sistemin kesin kabul testi süresince ölçülen tüm sistem unsurları için hatalar arası ortalama zamanı en az 60 (altmış) saat olacaktır.”

Gereksinim Türleri (3)

■ Süreç gereksinimleri

- ▶ Geliştirme adımları ile ilgili istekler
- ▶ Tanımlı yaşam döngüsü, kalite güvence etkinlikleri, vb.

■ Tasarım kısıtları

- ▶ Tasarımı etkileyebilecek istekler
- ▶ Geliştirme ortamı (örnek: J2EE), ilişkisel veri tabanı, vb.

Gereksinimlerin Bulanıklığı

- Gereksinimler net olarak ifade edilmediği zaman problemler yaşanır.
- Bulanık gereksinimler geliştiriciler ve kullanıcılar tarafından farklı algılanabilir.
 - ▶ Örnek: “Sistem, kullanıcının doküman ambarındaki dokümanları okuması için, **uygun görüntüleyiciler** sağlamalıdır.”
 - ▶ “uygun görüntüleyiciler”:
 - ◆ Kullanıcı yorumu : her farklı doküman tipi için farklı kullanıcı
 - ◆ Geliştirici yorumu : her dokümanın içeriğini gösteren bir metin görüntüleyici

Gereksinimlerin Tamlığı ve Tutarlılığı

- Gereksinimler tam ve birbiriyle tutarlı olarak ifade edilmelidir.
 - ▶ Tamlik : Sistemin beklenen tüm özellikleri tanımlanmalıdır.
 - ▶ Tutarlılık: Sistemin tanımlanan özellikleri arasında çelişkiler bulunmamalıdır.
- Pratikte, doğal dilden kaynaklanan zorluklar sebebiyle, gereksinimleri tam ve tutarlı olarak ifade etmek çok kolay değildir.
- Tanımlanan gereksinimlerin ilgili tüm kişilerce gözden geçirilmesi, tamlığı ve tutarlılığı büyük ölçüde sağlamamanın en basit yoludur.

Gereksinimleri Yazmak İçin Öneriler

- Standart bir biçim belirleyerek gereksinimleri tanımlarken kullanın.
- Doğal dili tutarlı olarak kullanın. Zorunlu ve seçimli gereksinimleri farklı kalıplarla ifade edin.
- Gereksinimlerin önemli kısımlarını ayırt etmek için farklı yazı tipi (büyük harf, alt çizme, farklı renk, vb.) kullanın.
- Bilgisayar terimlerini kullanmaktan kaçının.

Gereksinimler ve Tasarım

- Gereksinimler, sistemin “ne” yapacağını tanımlar.
- Tasarım, sistemin tanımlanan gereksinimlerinin “nasıl” gerçekleştirileceğini belirtir.
- Pratikte, gereksinimler ve tasarım her zaman net olarak ayrılamayabilir.
 - ▶ Sistem mimarisi, gereksinimleri yapısallaştırmak için tasarlanır.
 - ▶ Sistem işlevleri, tasarım kısıtlarını belirleyen diğer sistemlerle ilişki içinde gerçekleştiriliyor olabilir.
 - ▶ Müşteri tarafından, sistemin özel bir tasarıma uyması isteniyor olabilir.

Gereksinimlerin türlerine göre ayrı başlıklar altında tanımlanması, bu karışıklığı azaltacaktır.

Doğal Dile İle İlgili Problemler

■ Muğlaklık (“Ambiguity”)

- ▶ Gereksinimler, okuyan herkes tarafından aynı yorumlanacak şekilde yazılmalıdır. Doğal dil muğlak ifadelere açıktır.

■ Aşırı esneklik (“Over-flexibility”)

- ▶ Bir gereksinim, doğal dil ile çok farklı şekillerde ifade edilebilir.

■ Modülerliğin olmayışı (“Lack of modularisation”)

- ▶ Doğal dilin öğeleri, sistem gereksinimlerini yapısallaştırmak için yetersiz kalmaktadır.

Bu problemlere rağmen doğal dilin kullanılması, müşteri ve geliştirici arasındaki iletişim açısından önem taşımaktadır.

Doğal Dile Alternatifler

Notation	Description
Structured natural language	This approach depends on defining standard forms or templates to express the requirements specification.
Design description languages	This approach uses a language like a programming language but with more abstract features to specify the requirements by defining an operational model of the system. This approach is not now widely used although it can be useful for interface specifications.
Graphical notations	A graphical language, supplemented by text annotations is used to define the functional requirements for the system. An early example of such a graphical language was SADT. Now, use-case descriptions and sequence diagrams are commonly used .
Mathematical specifications	These are notations based on mathematical concepts such as finite-state machines or sets. These unambiguous specifications reduce the arguments between customer and contractor about system functionality. However, most customers don't understand formal specifications and are reluctant to accept it as a system contract.

Yapısal Doğal Dil – Örnek: Form Esaslı Tanımlama

Insulin Pump/Control Software/SRS/3.3.2

Function Compute insulin dose: Safe sugar level

Description Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.

Inputs Current sugar reading (r2), the previous two readings (r0 and r1)

Source Current sugar reading from sensor. Other readings from memory.

Outputs CompDose – the dose in insulin to be delivered

Destination Main control loop

Action: CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.

Requires Two previous readings so that the rate of change of sugar level can be computed.

Pre-condition The insulin reservoir contains at least the maximum allowed single dose of insulin..

Post-condition r0 is replaced by r1 then r1 is replaced by r2

Side-effects None

Arayüz (“Interface”) Tanımlama

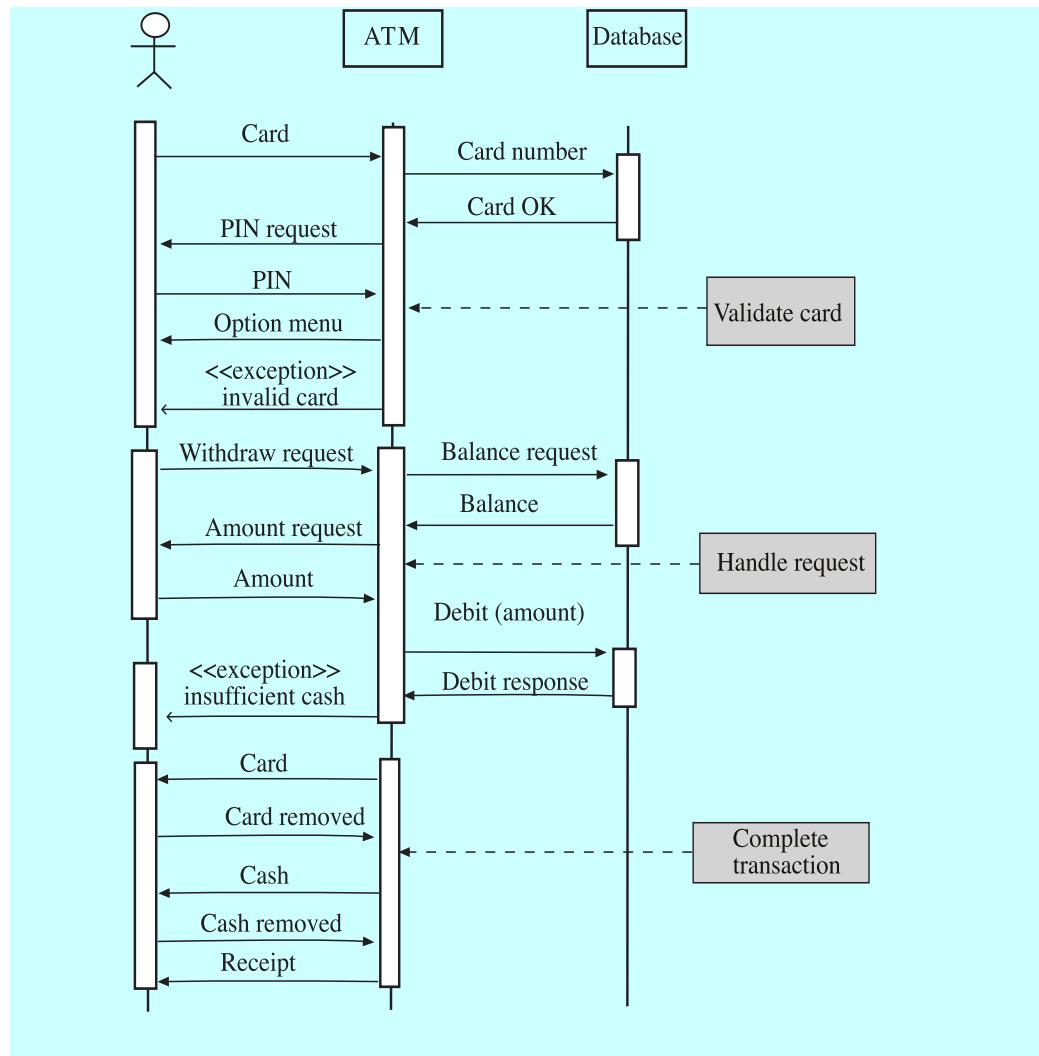
- Geliştirilen birçok sistem, diğer sistemlerle birlikte çalışmak zorundadır. Birlikte çalışmayı sağlayacak arayüzler de gereksinimlerin bir parçası olarak tanımlanmalıdır.
- Tanımlanabilecek arayüz türleri:
 - ▶ Yordam arayüzleri
 - ▶ Değiş-tokuş edilecek veri yapılarının arayüzleri
 - ▶ Veri göstergelerine ilişkin arayüzler
- Formal göstergeler, arayüz tanımlamları için daha uygundur.

Arayüz Tanımlama – Örnek: Java PDL Arayüz Tanımı

```
interface PrintServer {  
  
    // defines an abstract printer server  
    // requires:      interface Printer, interface PrintDoc  
    // provides: initialize, print, displayPrintQueue, cancelPrintJob, switchPrinter  
  
        void initialize ( Printer p ) ;  
        void print ( Printer p, PrintDoc d ) ;  
        void displayPrintQueue ( Printer p ) ;  
        void cancelPrintJob (Printer p, PrintDoc d) ;  
        void switchPrinter (Printer p1, Printer p2, PrintDoc d) ;  
}  
//PrintServer
```

Grafik Gösterim –

Örnek: UML Ardıl İşlem (“Sequence”) Diyagramı



Matematiksel Tanımlama – Örnek: Z Gösterimi

- ▶ The Z notation is a formal specification notation based on set theory and predicate calculus.

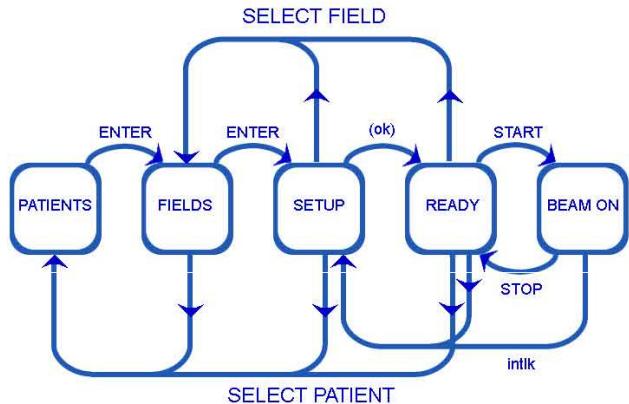


Figure 6.6: Therapy control cascade: state transition diagram

Graphic courtesy of kimberlybatteau.com

STATE ::= patients | fields | setup | ready | beam_on
EVENT ::= select_patient | select_field | enter | start | stop | ok | intlk
FSM == (STATE × EVENT) → STATE

no_change, transitions, control: FSM

control = no_change ⊕ transitions

no_change = { s: STATE; e: EVENT • (s, e) ↦ s }

transitions = { (patients, enter) ↦ fields,

(fields, select_patient) ↦ patients, (fields, enter) ↦ setup,

(setup, select_patient) ↦ patients, (setup, select_field) ↦ fields, (setup, ok) ↦ ready,

(ready, select_patient) ↦ patients, (ready, select_field) ↦ fields, (ready, start) ↦ beam_on, (ready, intlk) ↦ setup,

(beam_on, stop) ↦ ready, (beam_on, intlk) ↦ setup }

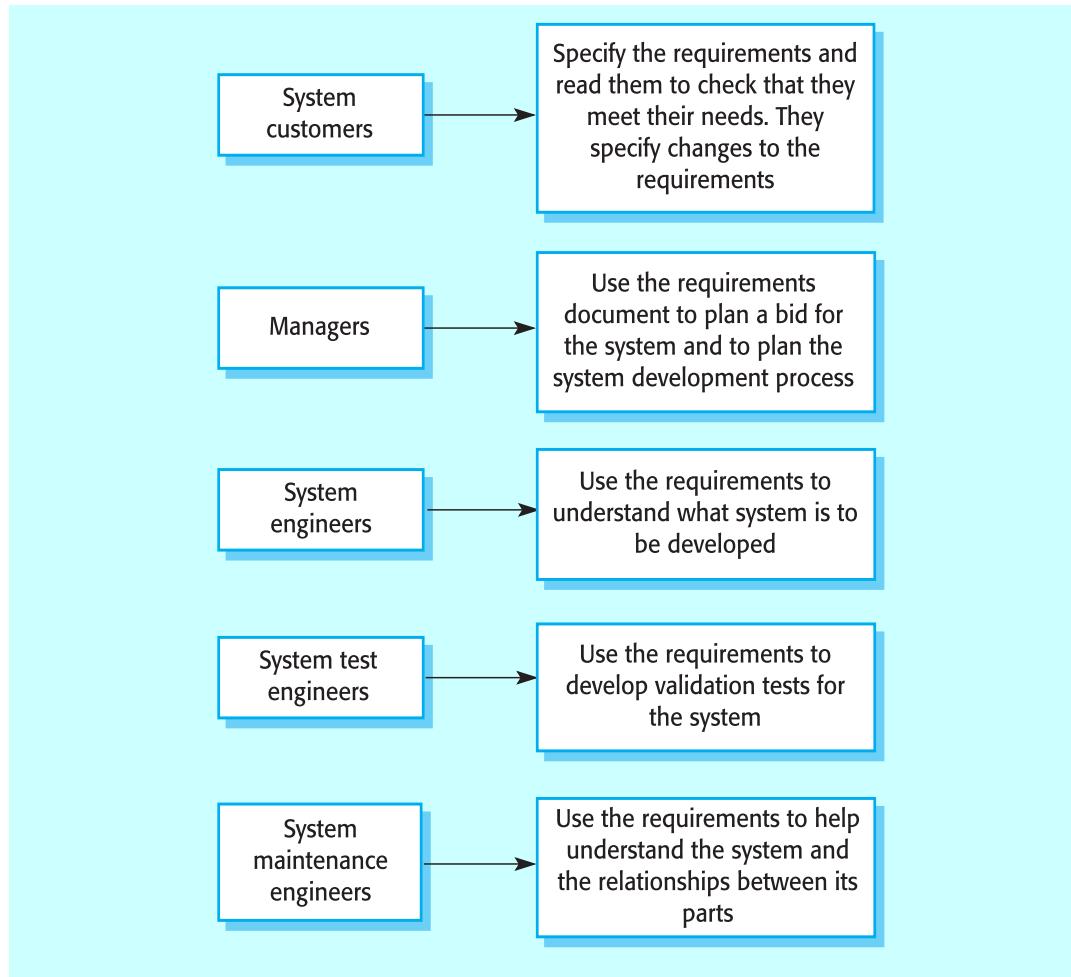
Gereksinim Belgesi

- Sistemin sağlanması beklenen özelliklerin resmi tanımıdır.
- Hem kullanıcı gereksinimlerini, hem de sistem gereksinimlerini içermesi beklenir.
 - ▶ Bazı modeller bu ikisi için ayrı belgelerin oluşturulmasını önermektedir.
- Mمungkin olduğunda sistemin “ne yapacağını” tanımlamalı, “nasıl” yapacağı detayına girmemelidir.

Gereksinim Belgesi Tipik İçeriği

- Önsöz
- Giriş
- Tanımlar
- Kullanıcı gereksinimleri
- Sistem mimarisi
- Sistem gereksinimleri
- Sistem modelleri
- Sistem gelişimi
- Ekler
- Endeks

Gereksinim Belgesinin Kullanıcıları



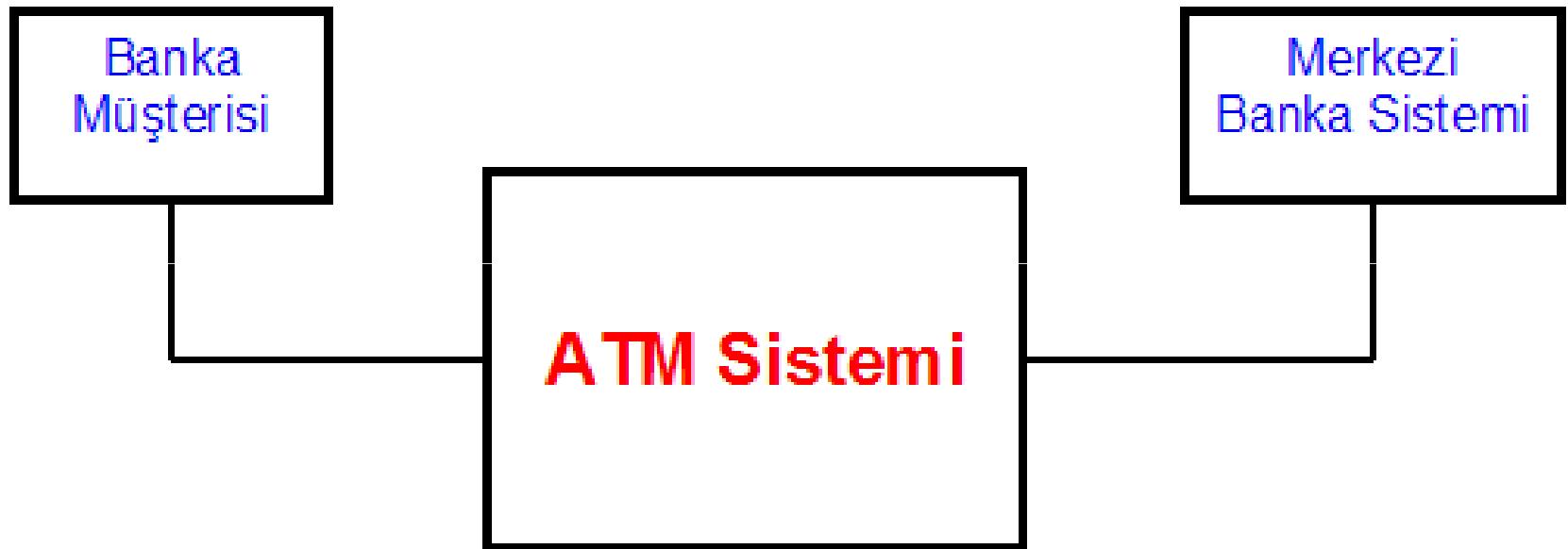
IEEE Gereksinim Belgesi Standardı

- IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications
- Gereksinim belgesi için, her sistem için özelleştirilecek genel bir yapı sunar.
 - ▶ “Introduction”
 - ▶ “General description”
 - ▶ “Specific requirements”
 - ▶ “Appendices”
 - ▶ “Index”

Örnek: ATM Uygulaması

- Bir bankanın ATM cihazı için yazılım geliştirilecektir. ATM, banka kartı olan müşterilerin hesaplarından para çekmelerine, hesaplarına para yatırmalarına ve hesapları arasında para transferi yapmalarına olanak sağlayacaktır. ATM, banka müşterisi ve hesapları ile ilgili bilgileri, gerekiğinde merkezi banka sisteminden alacaktır. Banka sistemi ayrıca her günün sonunda, ATM'den günlük işlemlerin bir özeti isteyecektir.

ATM Uygulaması – Kapsam



Bağlam ("context") diyagramı

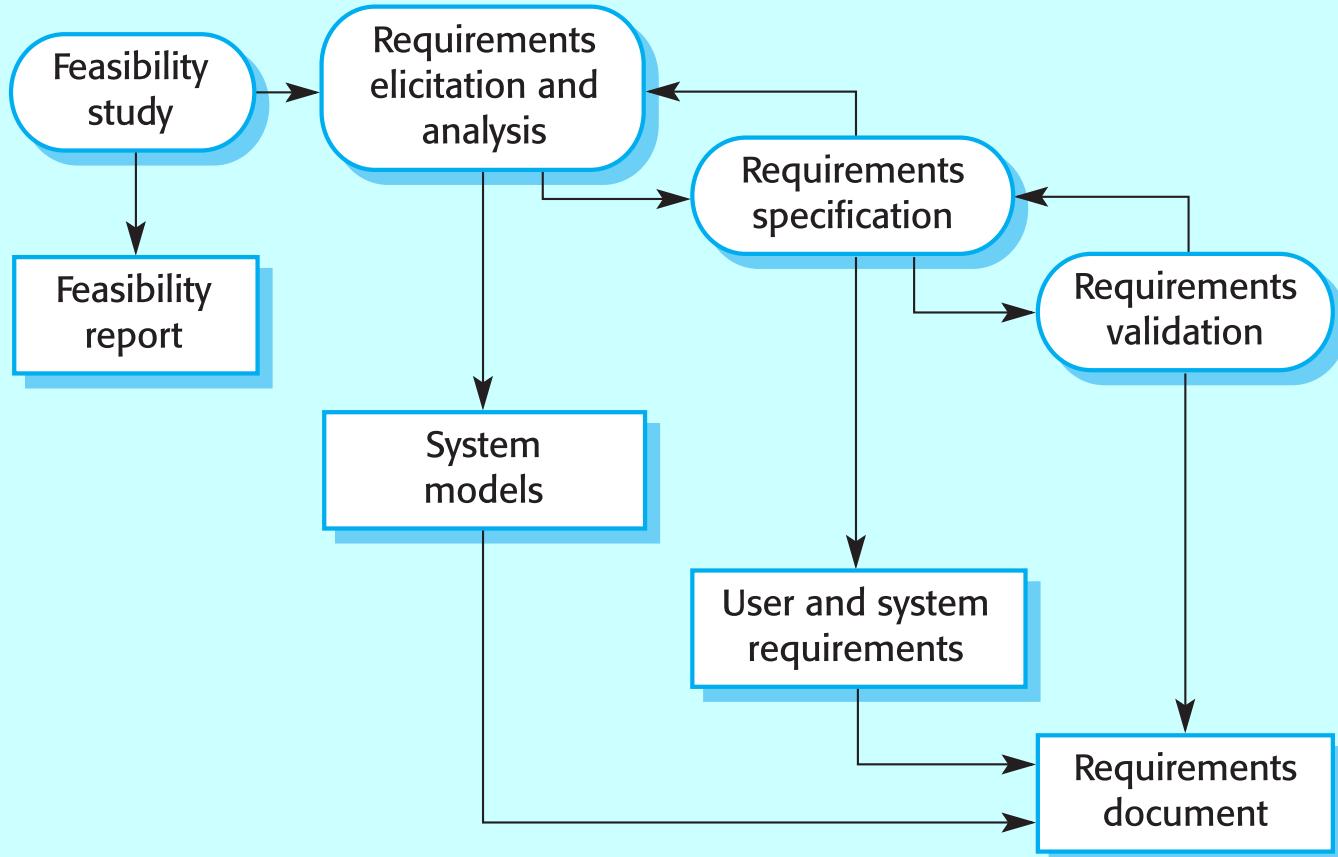
Gereksinim Mühendisliği Süreci

Gereksinim Mühendisliği Süreci (1)

■ Temel olarak aşağıdaki etkinlikleri içerir:

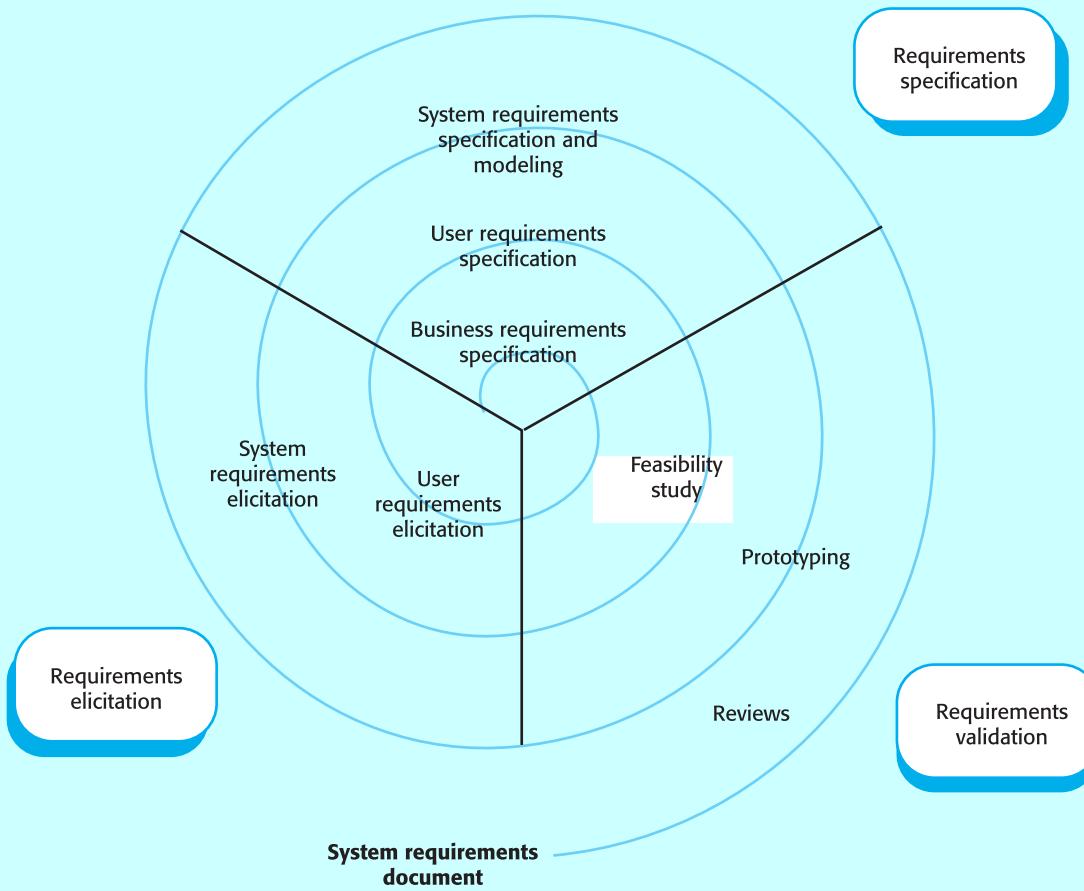
- ▶ Gereksinim çıkarma (“Requirements elicitation”)
- ▶ Gereksinim analizi (“Requirements analysis”)
- ▶ Gereksinim geçerleme (“Requirements validation”)
- ▶ Gereksinim yönetimi (“Requirements management”)

Gereksinim Mühendisliği Süreci (2)



Gereksinim Mühendisliği Süreci (3)

– Etkinliklerin Gelişimi



Olurluk Çalışmaları (“Feasibility Studies”)

- Olurluk çalışmasının amacı, sistemin gerçekleştirilebilirlik ve uygulanabilirliğine karar vermektedir.
- Bilgi toplamaya, değerlendirmeye ve raporlamaya dayanır.
 - ▶ Sistem kurumsal hedeflere hizmet edecek mi?
 - ▶ Sistem mevcut teknoloji ve bütçe sınırları içinde gerçekleştirilebilir mi?
 - ▶ Sistem mevcut diğer sistemlere entegre edilebilir mi?

Gereksinim Çıkarma ve Analizi (1)

- Uygulama (iş) alanını anlamayı ve sistemin bu alanı ne kapsamda destekleyeceğini bulmayı gerektirir.
 - ▶ İş alanında yürütülen işlevler nelerdir ve nasıl ilişkilidir?
 - ▶ Sistem bu işlevleri ne kapsamında ve hangi noktalarda destekleyebilir?
 - ▶ Sistemin destekleyeceği iş kapsamı için gereksinimler ve kısıtlar nelerdir?
- Son kullanıcılar, yöneticiler, alan uzmanları ve geliştiriciler; gereksinim çıkarma ve analiz etkinliklerine dahil olur. Tüm bu kişiler *sistemin paydaşları* (“stakeholders”) olarak isimlendirilir.

Gereksinim Çıkarma ve Analizi (2)

■ Yapılan hatanın düzeltilme maliyeti çok yüksek

- ▶ Bakım aşamasında 20 kat fazla
- ▶ Bulunan hataların %40'ı gereksinimlerden kaynaklanıyor

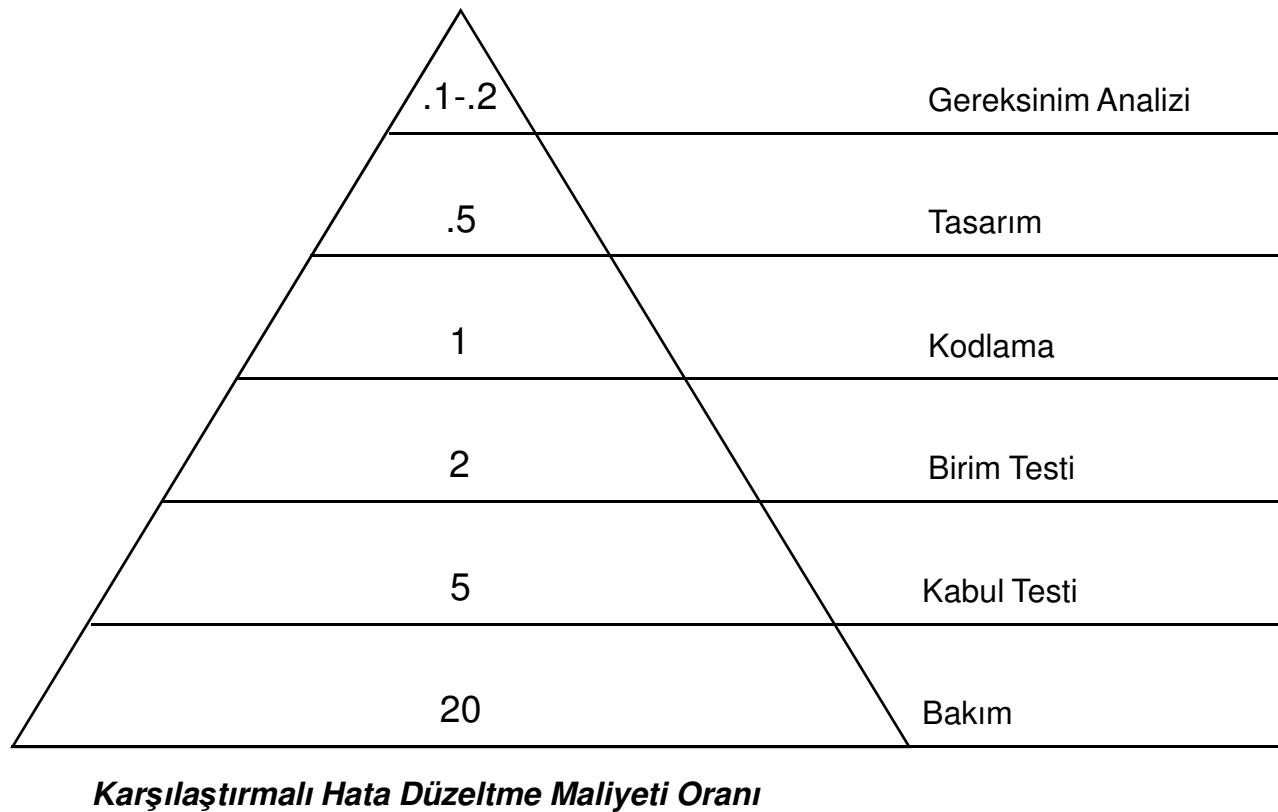
■ “Doğru” sistemi oluşturmak için

- ▶ Kurumsal ortam nedir ?
- ▶ Müşteri ne istiyor ?
- ▶ Bağlı bulunduğu sistemler var mı ?

■ Gereksinim tanımı iletişime esas

- ▶ Müşteri – teknik ekip
- ▶ Analiz uzmanları – tasarım uzmanları

Gereksinim Analizi ve Hata Düzeltme Maliyeti



Referans: *Managing Software Requirements – A Unified Approach*, Leffingwell & Widrig, 2000

Gereksinim Çıkarma ve Analizi – Zorluklar

- Yazılım geliştirme çalışmalarının ön aşamaları
 - ▶ Kimse birbirini tanımıyor, sistemi bilmiyor
 - ▶ Yöntem, teknoloji, vb. yeni olabilir
- Kullanıcı odaklı problemler yaşanabilir.
 - ▶ Kullanıcılar ne istediklerini bilmeyebilir.
 - ▶ Kullanıcılar isteklerini kendi terimleriyle ifade edebilir.
 - ▶ Farklı kullanıcıların çelişen istekleri olabilir.
- Farklı grupların beraber çalışmasını gerektiriyor.
 - ▶ Kullanıcı grupları, analiz uzmanları, mimari/tasarım uzmanları
- Kurumsal ve politik etkenler sistem gereksinimlerini etkileyebilir.
- Analiz boyunca gereksinimler değişimelidir; yeni kullanıcılar çıkabilir ve iş ortamı değişimelidir.

Gereksinim Geçerleme

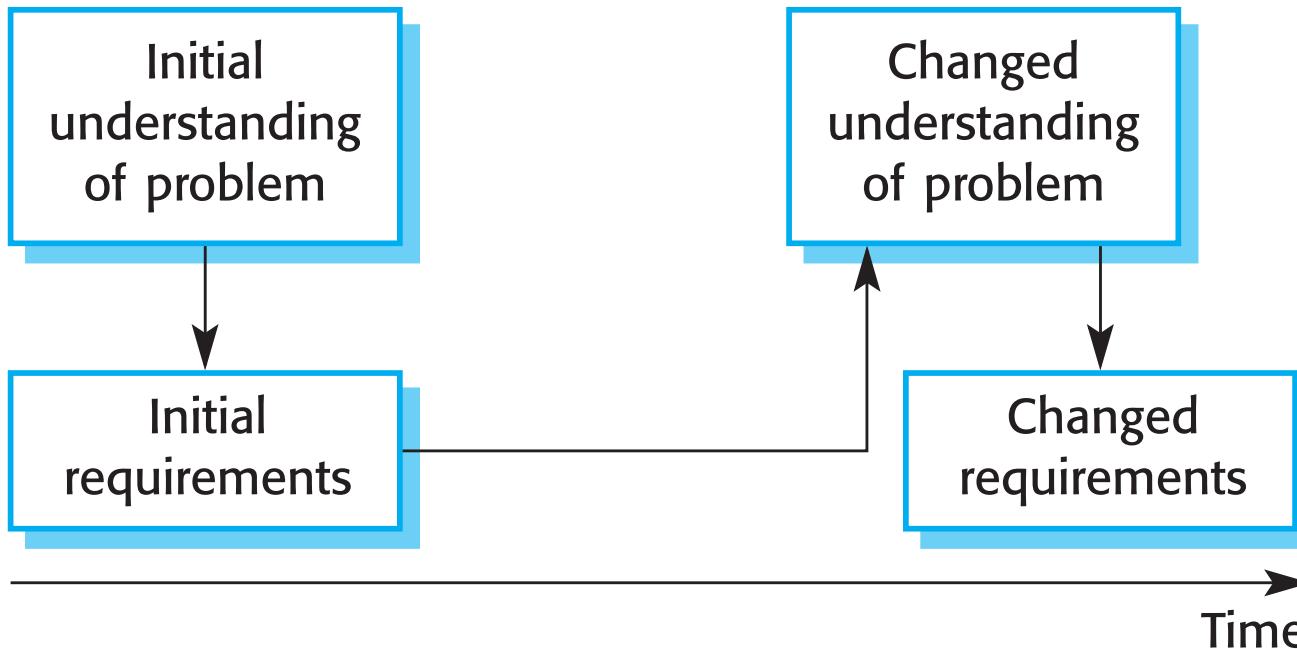
- Gereksinimlerin müşterinin istediği sistemi tanımladığını güvence altına almayı hedefler.
- Gereksinimlerden kaynaklanan bir hatayı sonradan düzeltmenin maliyeti yüksek olduğundan, geçerleme büyük önem taşır.
- Gereksinim geçerleme teknikleri:
 - ▶ Gözden geçirme – gereksinimlerin sistematik ve paydaşlara dayalı analizi
 - ▶ Prototip oluşturma – sistemin çalışan bir taslağı üzerinden kontrol
 - ▶ Test durumu geliştirme – sistemin test edilebilirliğini kontrol amacıyla gereksinimler için test durumu geliştirme

Gözden Geçirme

- Gereksinimlerinin gözden geçirilmesi, hataların en aza indirilmesi, kalitenin güvencesi ve projenin başarısı açısından son derece önemlidir.
 - ▶ Gereksinim aşaması yazılım geliştirme sürecindeki en zayıf noktadır.
 - ▶ Yapılan hatalar ancak sürecin çok ilerleyen aşamalarında (büyük olasılıkla kabul testlerinde) ortaya çıkacaktır.
 - ▶ Analiz ekibi gerek kendi içinde gerekse sistemin kullanıcıları ile gereksinimleri gözden geçirmelidir.
- Gözden geçirme çalışmalarında hedefimiz, yazılım gereksinimlerinin aşağıdaki özellikleri taşıdığını emin olmaktır:
 - ▶ Tam ve doğru
 - ▶ Anlaşılabilir
 - ▶ Tutarlı
 - ▶ Test edilebilir
 - ▶ Diğer belgelerde tanımlanan iş/kullanıcı/sistem gereksinimlerine izlenebilir

Gereksinim Yönetimi

- Geliştirme boyunca değişen gereksinimlerin yönetilmesi etkinliğidir.
 - ▶ Geliştirme ilerledikçe (ve sisteme ilişkin anlayış iyileşikçe) yeni gereksinimler ortaya çıkabilir.
 - ▶ Farklı bakış açılarının gereksinimleri zaman içinde çelişebilir.



İzlenebilirlik

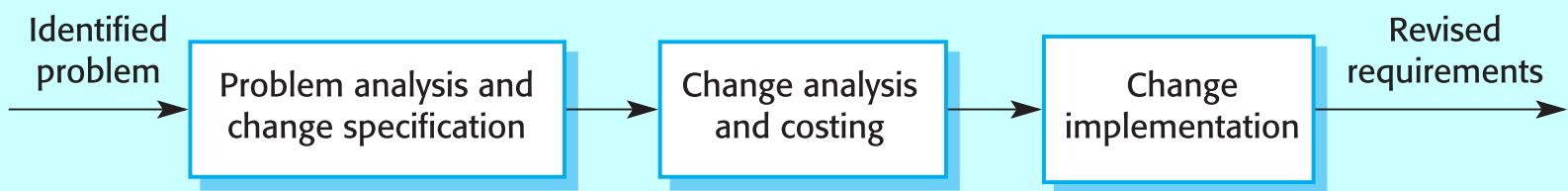
- Gereksinim Belgesinde yer alan her gereksinim, analizde kaynak olarak kullanılan belgelere ve sistem tasarıımına izlenebilir olmalıdır.
- İzlenebilirlik aşağıdaki gibi sınıflandırılabilir:
 - ▶ Kaynağa olan izlenebilirlik
 - ◆ Gereksinimlerden, onu oluşturan paydaşlara ve kaynak belgelere
 - ▶ Gereksinimlere olan izlenebilirlik
 - ◆ Gereksinimlerden, bağımlı diğer gereksinimlere
 - ▶ Tasarıma olan izlenebilirlik
 - ◆ Gereksinimlerden tasarıma
- Gereksinimler ve diğer belgeler arasındaki ilişki bir matrisle tanımlanır.
 - ▶ Gözden geçirme sırasında bu matristeki bilginin doğruluğu, tamlığı ve tutarlılığı kontrol edilir.
 - ▶ İzlenebilirlik çift yönlü olarak sağlanmalıdır (geliştirme boyunca ileri ve geri).

İzlenebilirlik Matrisi – Örnek: Gereksinimlere İzlenebilirlik

Req. id	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		D	R					
1.2			D			D		D
1.3	R			R				
2.1			R		D			D
2.2								D
2.3		R		D				
3.1								R
3.2							R	

D: bağımlı (“dependent”)
R: ilişkili (“relational”)

Gereksinim Yönetimi Adımları



Genellikle bir CASE aracı ile desteklenir.

CMMI – Gereksinim Geliştirme ("Requirements Development") Süreç Alanı (ML3)

- SG 1 Müşteri gereksinimlerini geliştir
 - ▶ SP 1.1 İhtiyacı çıkar
 - ▶ SP 1.2 Müşteri gereksinimlerini geliştir
- SG 2 Ürün gereksinimlerini geliştir
 - ▶ SP 2.1 Ürün ve ürün bileşen gereksinimlerini oluştur
 - ▶ SP 2.2 Ürün bileşen gereksinimlerini ata
 - ▶ SP 2.3 Arayüz gereksinimlerini belirle
- SG 3 Gereksinimleri analiz et ve onayla
 - ▶ SP 3.1 İşlevsel kavramları ve senaryoları oluştur
 - ▶ SP 3.2 Beklenen işlevselliğin tanımını oluştur
 - ▶ SP 3.3 Gereksinimleri analiz et
 - ▶ SP 3.4 Gereksinimleri dengeyi sağlamak için analiz et
 - ▶ SP 3.5 Gereksinimleri onayla

CMMI – Gereksinim Yönetimi ("Requirements Management") Süreç Alanı (ML2)

■ SG 1 Gereksinimleri yönet

- ▶ SP 1.1 Gereksinimlerin anlaşılmasını sağla
- ▶ SP 1.2 Gereksinimlere taahhütü sağla
- ▶ SP 1.3 Gereksinim değişikliklerini yönet
- ▶ SP 1.4 Gereksinimlerin çift-yönlü izlenebilirliğini sürdür
- ▶ SP 1.5 Proje adımları ve gereksinimler arasındaki tutarsızlıkları belirle

İş Gereksinimleri Analizi – Gereksinim Analizi İlişkisi

Yazılım Yaşam Döngüsü

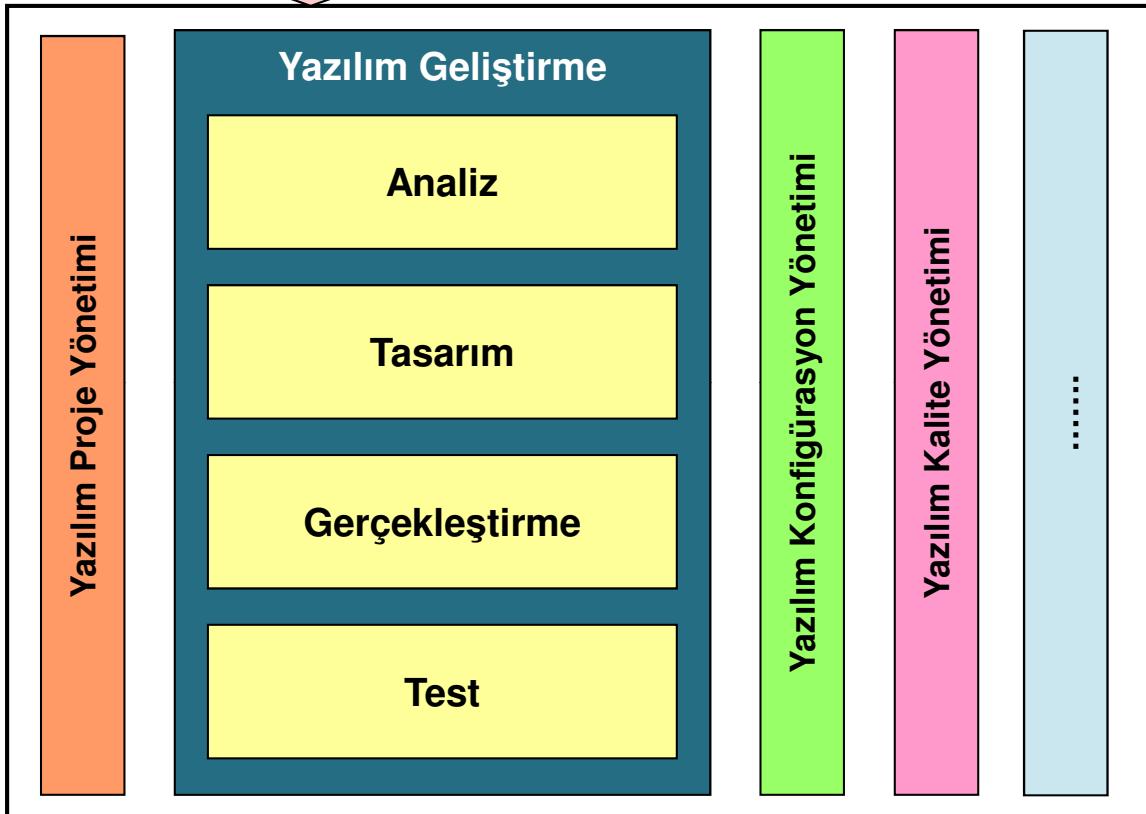
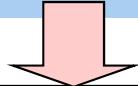
- Girdi: İş gereksinimleri (iş alanı bilgisini gerektirir)

- Yazılım ürününün geliştirilmesi ve yönetilmesi için izlenmesi gereken adımların bütünüdür
 - ▶ Analiz, tasarım, kodlama, test, ...
 - ▶ Proje yönetimi, kalite yönetimi, yapılandırma yönetimi, ...
- Yazılım ürününe mühendislik etkinliklerinin uygulanmasına olanak sağlar
 - ▶ Örnek: ISO/IEC 12207 Yazılım Yaşam Döngüsü Sürecleri

- Çıktı: Yazılım sistemi (çözüm alanı bilgisini gerektirir)

Yazılım Yaşam Döngüsü

Müşteri İhtiyaçları



Yazılım Ürünü



İş Gereksinimleri Analizi



*Yazılım sisteminin başarısında;
iş alanına ilişkin bilgi, en az çözüm alanına
ilişkin bilgi kadar önemlidir*

İş Gereksinimleri Analizi

- Amaç: İş alanına (“business domain”) ilişkin kavramları, süreçleri, ortamı ve kullanıcı gereksinimlerini anlamak
 - ▶ Yazılım sisteminin içinde çalışacağı iş ortamı belirlenir
 - ▶ İş ihtiyaçları tanımlanır
 - ▶ İhtiyaç tanımında fikir birliğine varılır
 - ▶ Yeni sistemden etkilenecek kişiler ve kullanıcılar tespit edilir
 - ▶ Çözüm sisteminin çerçevesi çizilir
 - ▶ Çözümü etkileyebilecek kısıtlar ortaya koyulur

Süreç (“Process”) ve İş Süreci (“Business Process”)

■ Süreç (“process”) [Davenport 1993]:

- ▶ “a specific ordering of work activities across time and place, with a beginning, an end, and clearly defined inputs and outputs -- structure for action”

■ İş süreci (“business process”)

- ▶ “a process that defines how an organization achieves its purpose including vision and goals”

İş Süreçlerinin Modellenmesi

- Bilgi sistemi/bilgi teknolojisi ortamları için yaygın olarak kullanılan bir iş analizi yöntemidir
- Amaç:
 - ▶ Kuruluşa ilişkin dinamikleri anlamak
 - ▶ Müşteri, son kullanıcı ve geliştiricilerin, kuruluş süreçlerinden aynı şeyleri anladıklarını garanti etmek
- Uygulama ortamının karmaşık, çok boyutlu ve çok kullanıcılı olduğu durumlarda, getirileri maliyetini daha çok karşılar [Yourdon 2000]

İş Süreçlerinin Modellenmesi - Örnek

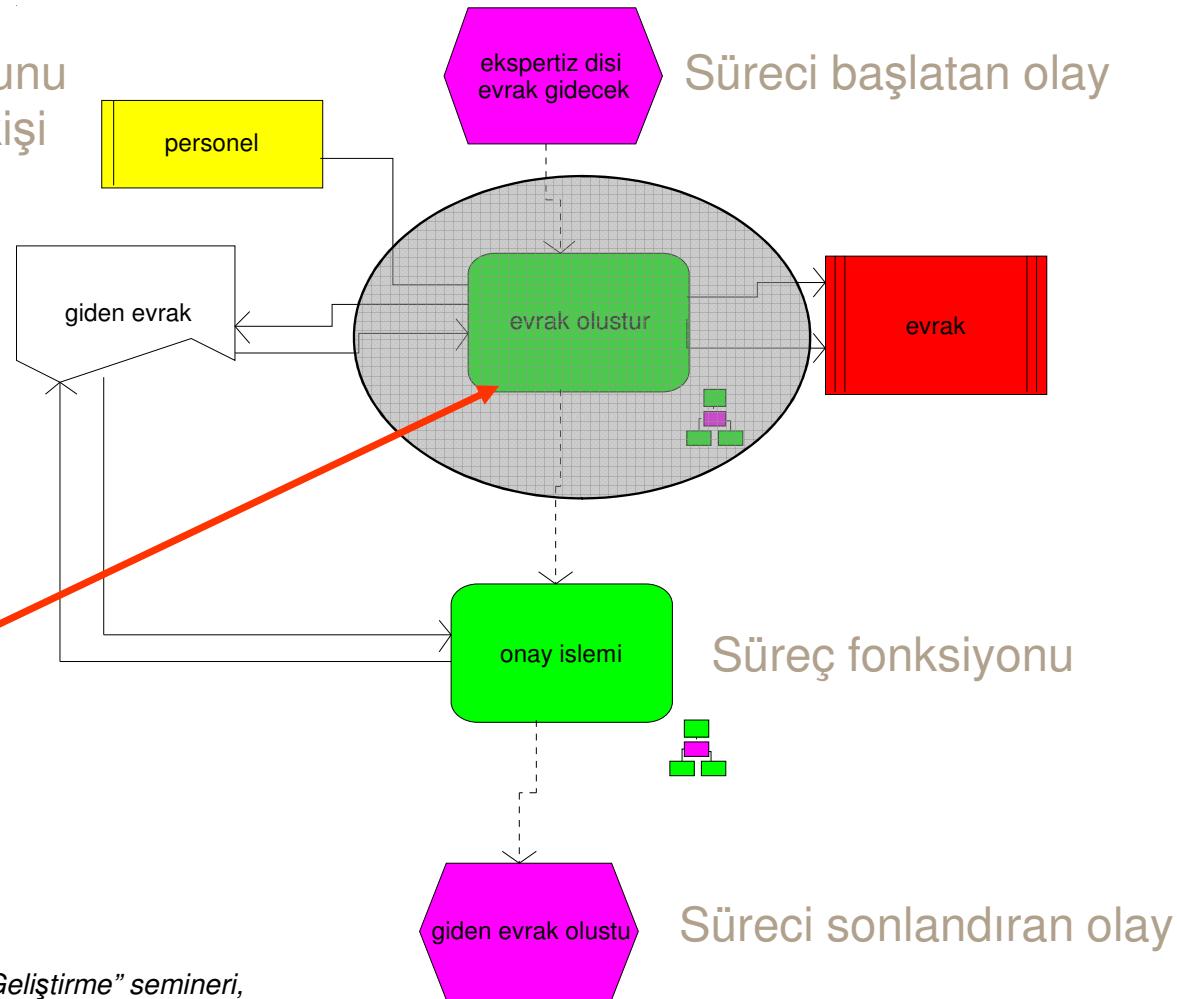
Süreç fonksiyonunu
gerçekleştiren kişi

Süreç varlıkları
(girdi ve çıktılar)

*Yazılım sisteminin
kapsamı*

Referans:

"UML ile Nesneye Yönelik Yazılım Geliştirme" semineri,
Bilgi Grubu Ltd., 2006



İş Süreçlerinin Modellenmesi – Kazançlar

- İş alanına daha geniş bir bakış açısı getirir.
- Mevcut süreçte aksayan yönleri görmek ve iyileştirmek fırsatını doğurur.
- İş alanındaki kişilerle çözüm alanındaki geliştiriciler arasında ortak bir dil oluşturur.
 - ▶ Müşterinin gereksinim çıkarma sürecine katılımı artar.
 - ▶ Kapsama ilişkin daha sonra yaşanabilecek güçlükler azalır.

İş Süreçleri ve Gereksinim Analizi (1)

- İş süreçlerinin tanımlı olduğu durumlarda aşağıdakiler de tanımlıdır:
 - ▶ Sistemin hangi iş süreci adımlarında kullanılacağı
 - ▶ Sistemin girdileri
 - ▶ Systemden beklenen çıktılar/sonuçlar
 - ▶ Sistemi kimlerin, hangi amaçla kullanacakları
- Gereksinim analizinin hedefi:
 - ▶ İş süreçlerinin tanımladığı bağlamda detaylı yazılım özelliklerini belirlemek

İş Süreçleri ve Gereksinim Analizi (2)

- İş süreçlerinin tanımsız olduğu veya sadece kişilerin uzmanlıklarında gizli olduğu durumlarda:
 - ▶ Gereksinim analizi süreç modellemesi etkinliklerini de içermeye başlar
 - ▶ Yazılım geliştiriciler bu konuda en yetkin kişiler değildir
 - ▶ Bütçe ve zaman buna göre ayarlanmamıştır

İş Süreçleri ve Gereksinim Analizi (3)

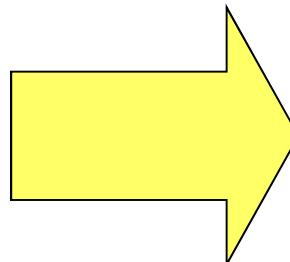
- Geliştirilecek yazılım ürününden beklenen özellikleri *anlamak* ve *tanımlamak*

- İş süreçleri/kuralları

Yapılan iş ne?

İşe ilişkin kısıtlar ne?

Nasıl bir ürün?



Gereksinim tanımı

Anlaşıılır, tam, doğru

İletişime esas

Geliştirme sürecine esas

Gereksinim Analizi – Kazançlar

■ Artan müşteri memnuniyeti

- ▶ Müşteri/kullanıcılar çalışmalara katılıyor
- ▶ İstekler tam ve doğru olarak tanımlanıyor

■ Artan yazılım kalitesi

- ▶ Gereksinimler doğru ve tam olarak tanımlanıyor
- ▶ Gereksinimleri tüm yazılım ekibi biliyor
- ▶ Gereksinim değişiklikleri en aza indirilebilecek

■ Etkin proje yönetimi

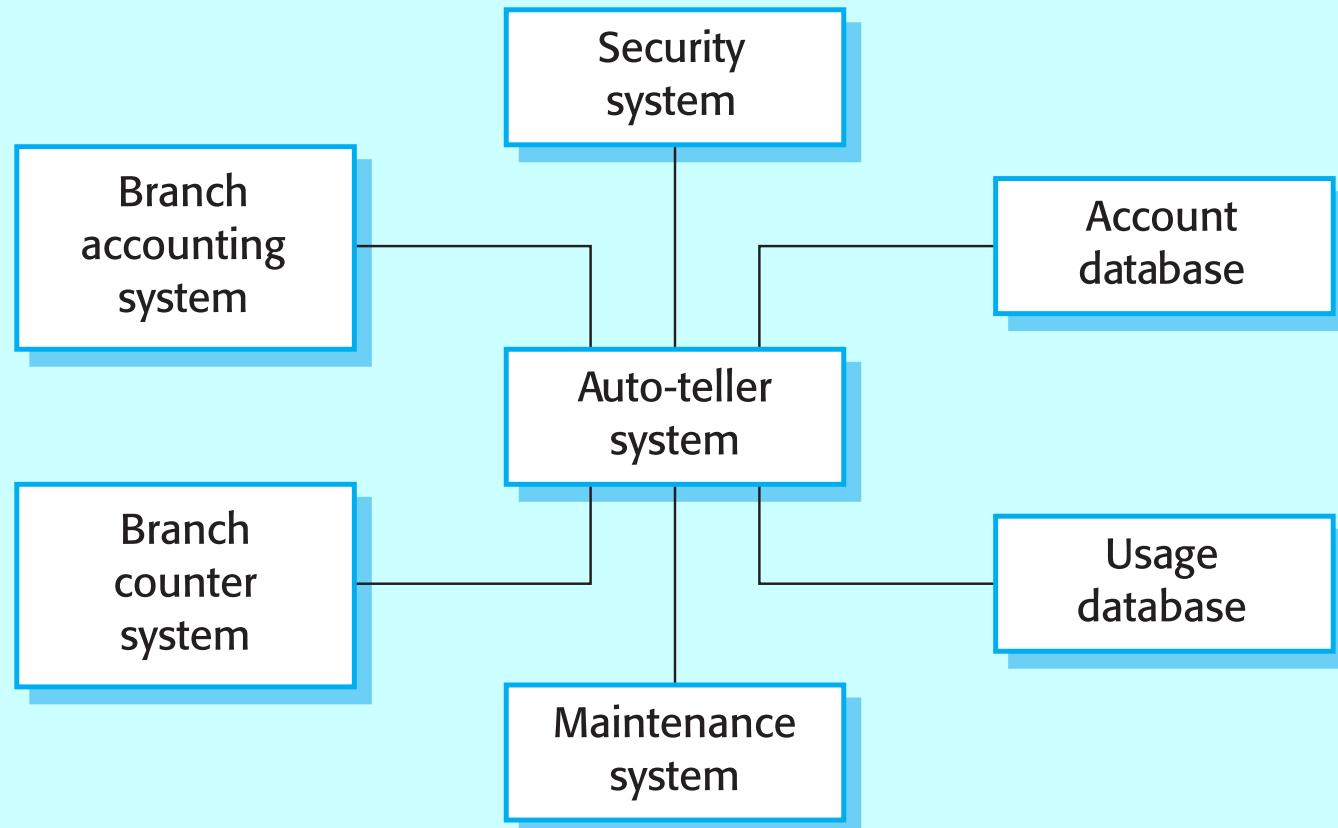
- ▶ Daha doğru tahminleme (takvim ve bütçe)
- ▶ Daha kolay izleme (gereksinimler esaslı)
- ▶ Daha düzgün iş atamaları (gereksinimler esaslı)

Sistem Modelleri

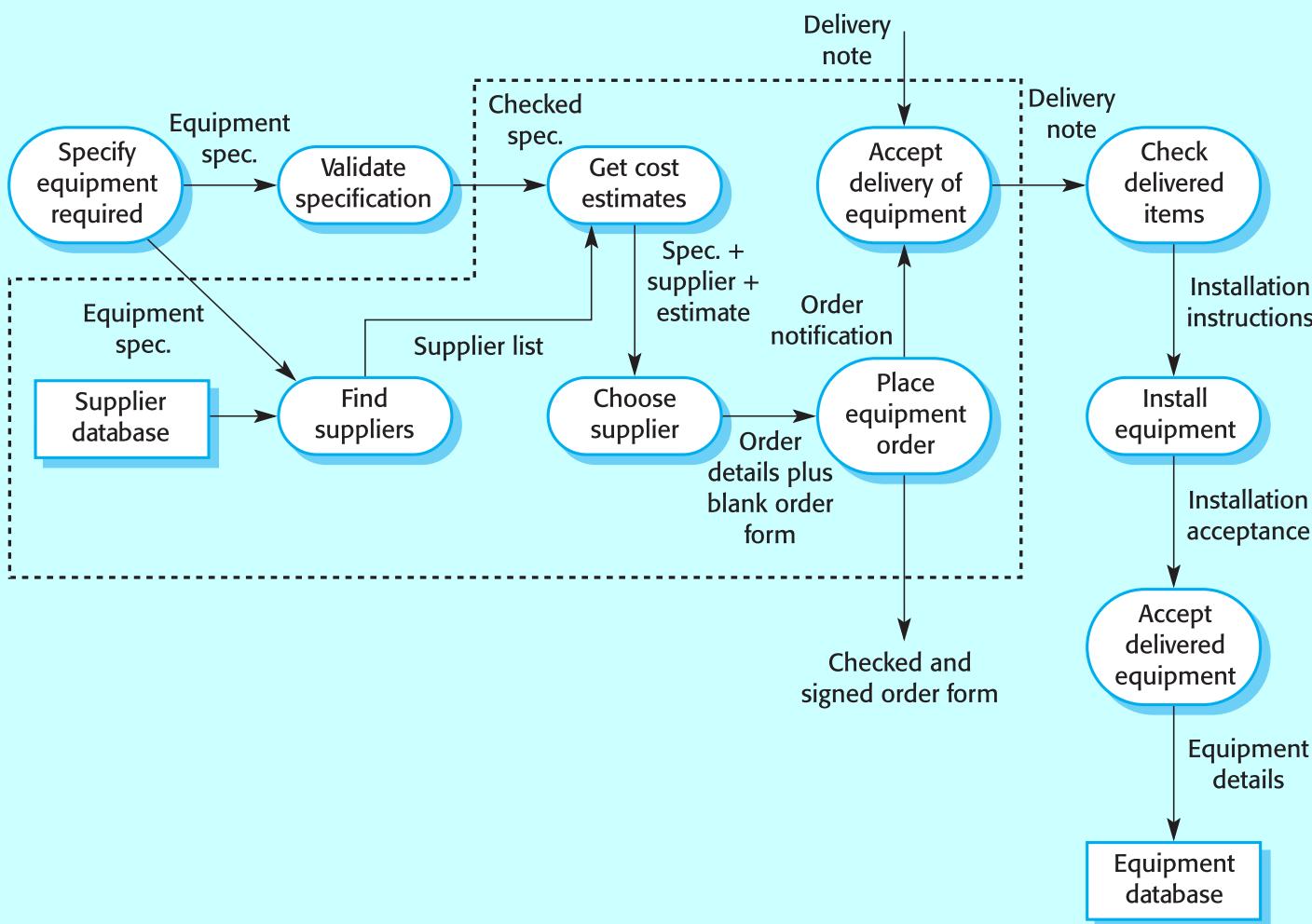
Sistem Modelleme

- Sistem modelleme; gereksinim çıkarma ve analizi aşamaları boyunca sistemin işlevselliğini anlamak için, müşteri ve teknik ekip ile iletişime esas olarak gerçekleştirilir.
- Model, sistemin soyut bir gösterimidir. Tamamlayıcı modeller, sistem hakkında farklı bilgileri ortaya çıkarır.
 - ▶ **Bağlam (“context”) modeli**, sistemin kendi ortamındaki konumunu ve diğer sistem ve süreçlerle olan etkileşimini tanımlar.
 - ▶ **Süreç (“process”) modeli**, sistemin destekleyeceği işleri ve iş akışlarını tanımlar.
 - ▶ **Davranış (“behavioral”) modeli**, sistemin davranışını tanımlar.
 - ◆ Veri-akış (“data flow”) modeli, sistem içindeki verinin akışını tanımlar.
 - ◆ Durum-makinesi (“state-machine”) modeli, sistemin davranışını iç ve dış olaylara gösterdiği tepkilerle tanımlar.
 - ▶ **Veri (“data”) modeli**, sistemin işlediği verinin anlamsal biçimini tanımlar.
 - ▶ **Nesne (“object”) modeli**, sistemdeki veriyi ve nasıl işlemlediğini tanımlar.
 - ◆ Nesne kalıtım (“object inheritance”) modeli
 - ◆ Nesne bileşen (“object aggregation”) modeli
 - ◆ Nesne davranış (“object behavior”) modeli

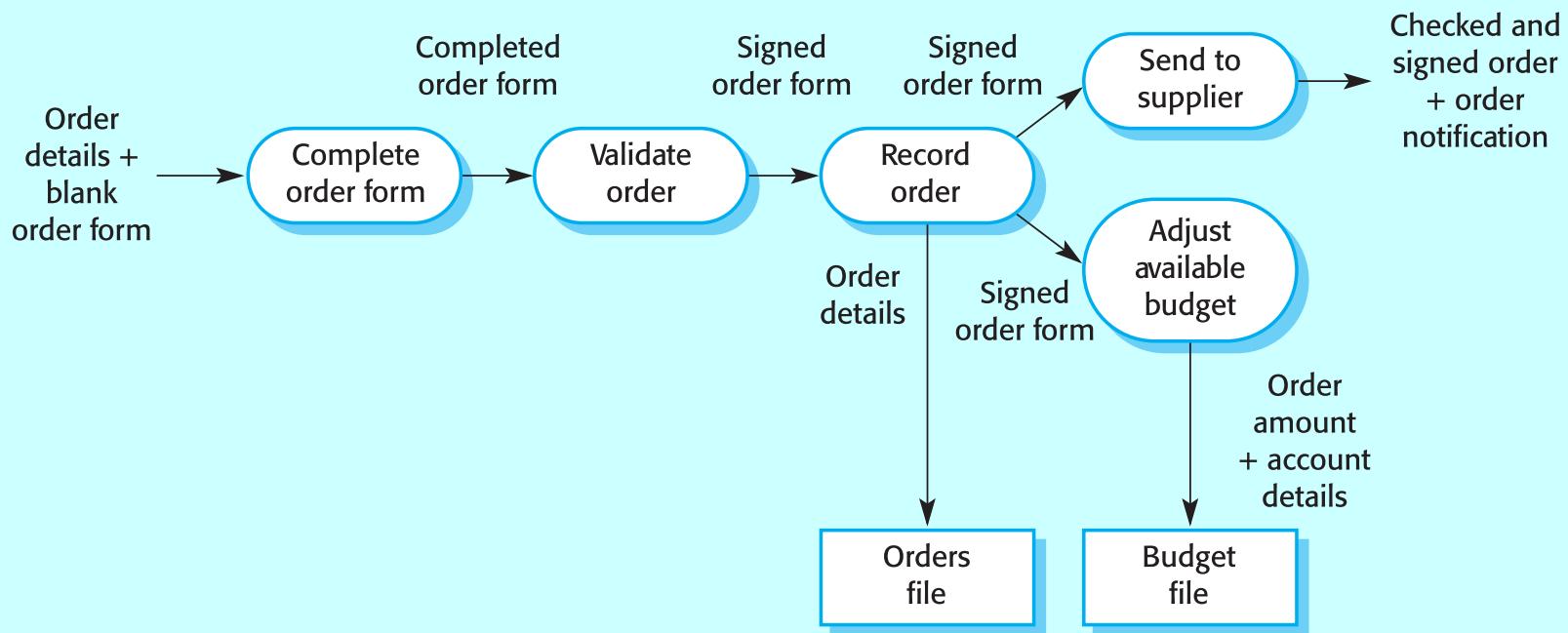
Bağlam (“Context”) Modeli – Örnek: ATM Sistemi



Süreç (“Process”) Modeli – Örnek: Donanım Satınalma Süreci Modeli



Veri-Akış (“Data-Flow”) Modeli – Örnek: Sipariş İşleme Veri-Akış Modeli



Veri-Akış (“Data-Flow”) Modeli – Bir Başka Örnek: “Data Flow Diagram (DFD)”

Figure 9–1: Data Flow Diagram (Gane-Sarson Notation)

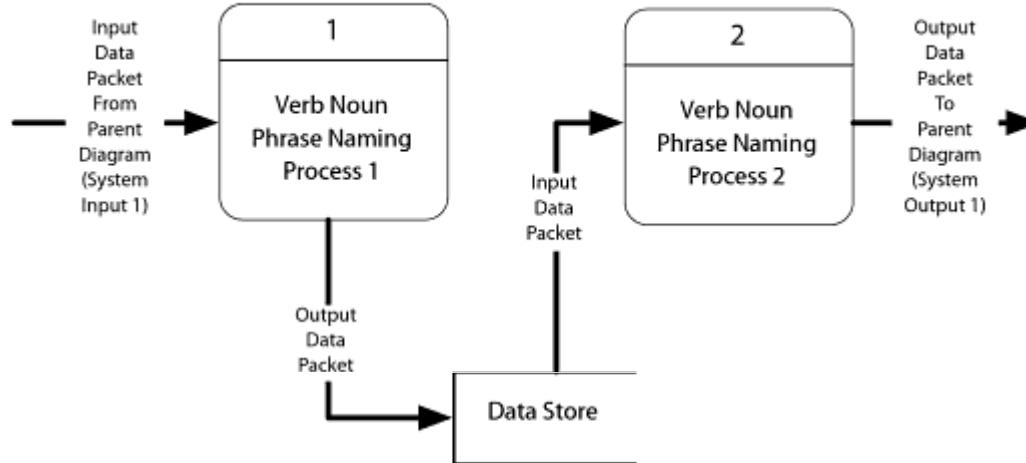
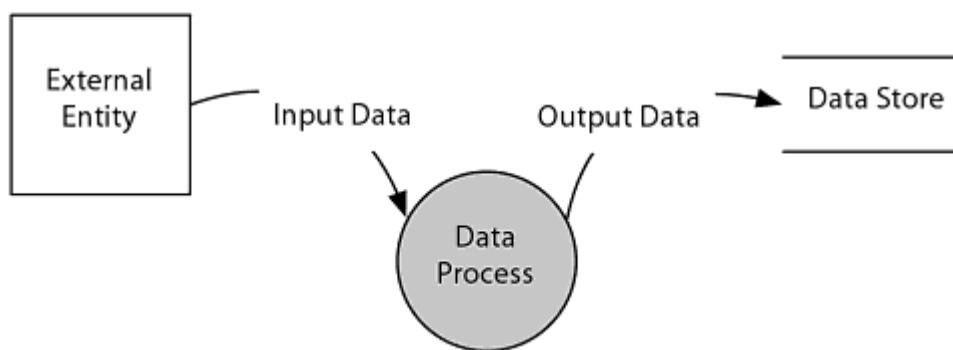
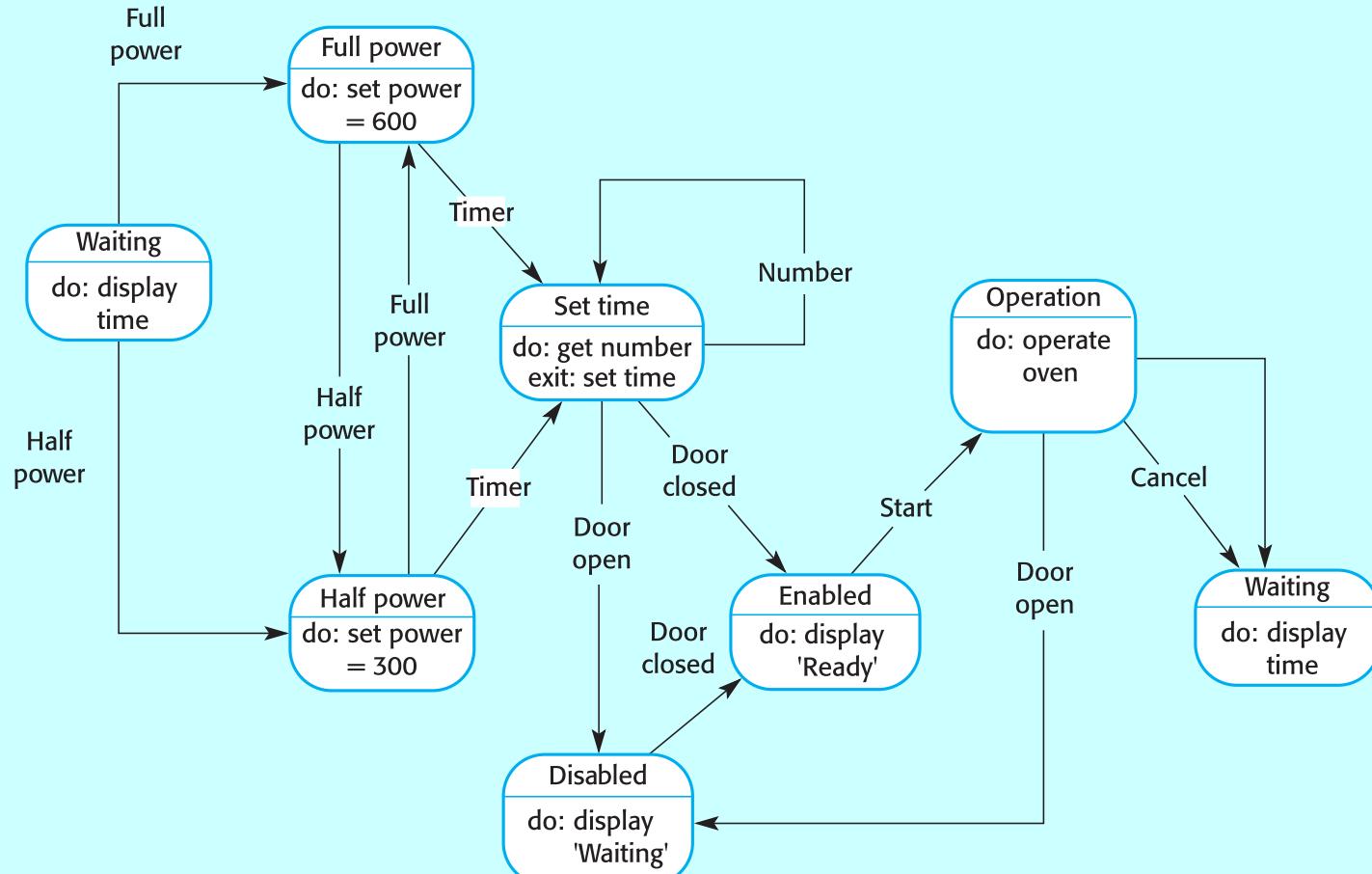


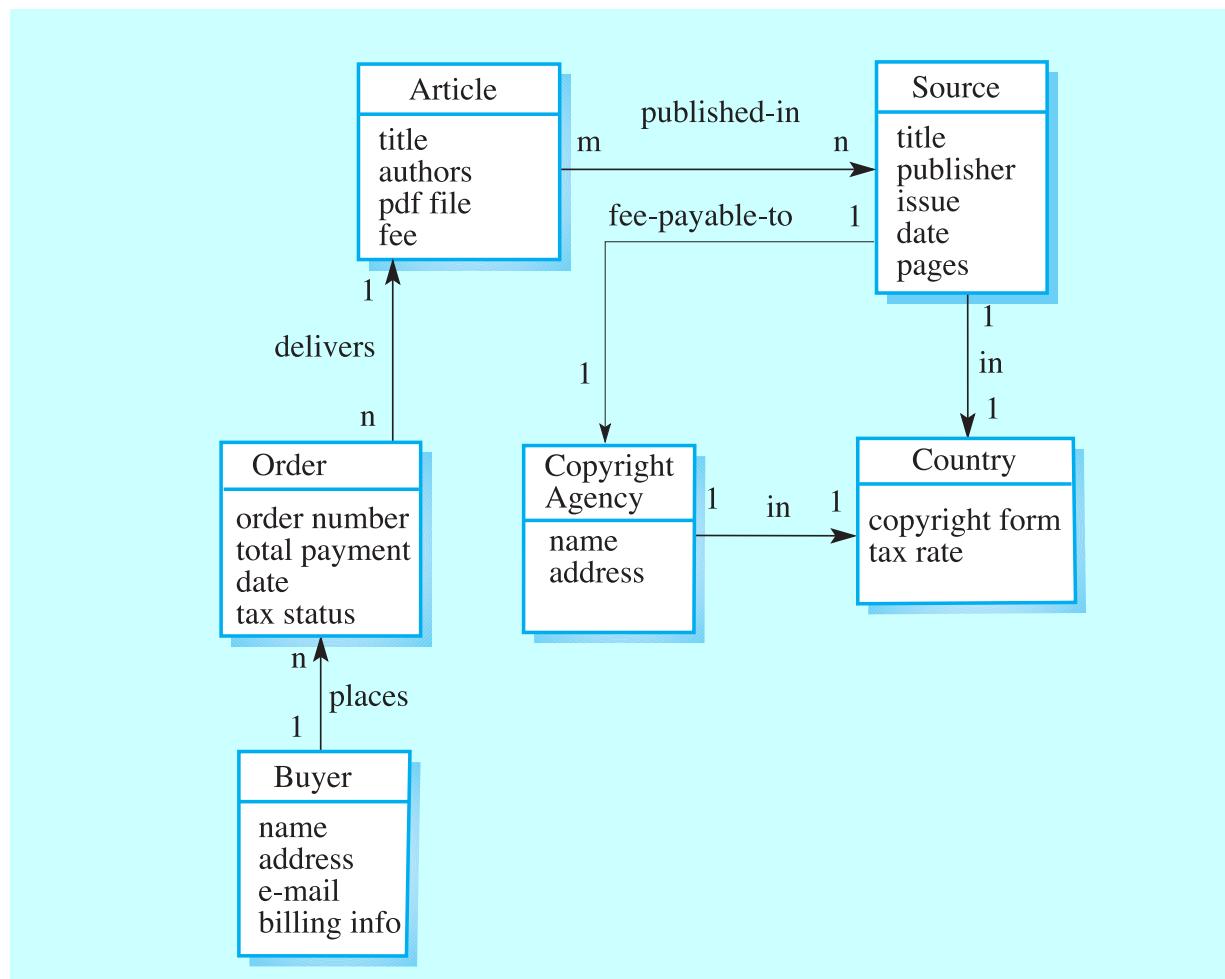
Figure 9–2: Data Flow Diagram (Yourdon Notation)



Durum-Makinesi (“State-Machine”) Modeli – Örnek: Mikrodalga Fırın Durum-Makinesi Modeli



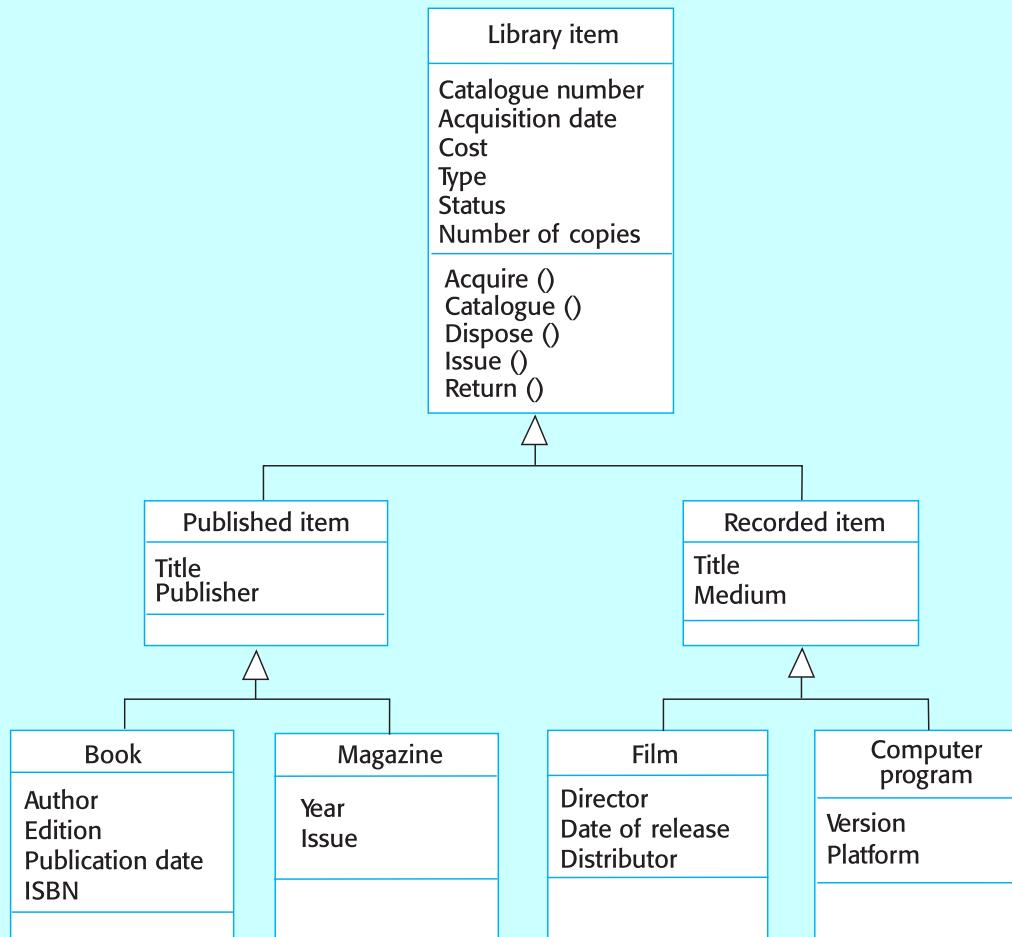
Veri (“Data”) Modeli – Örnek: Kütüphane Anlamsal Modeli



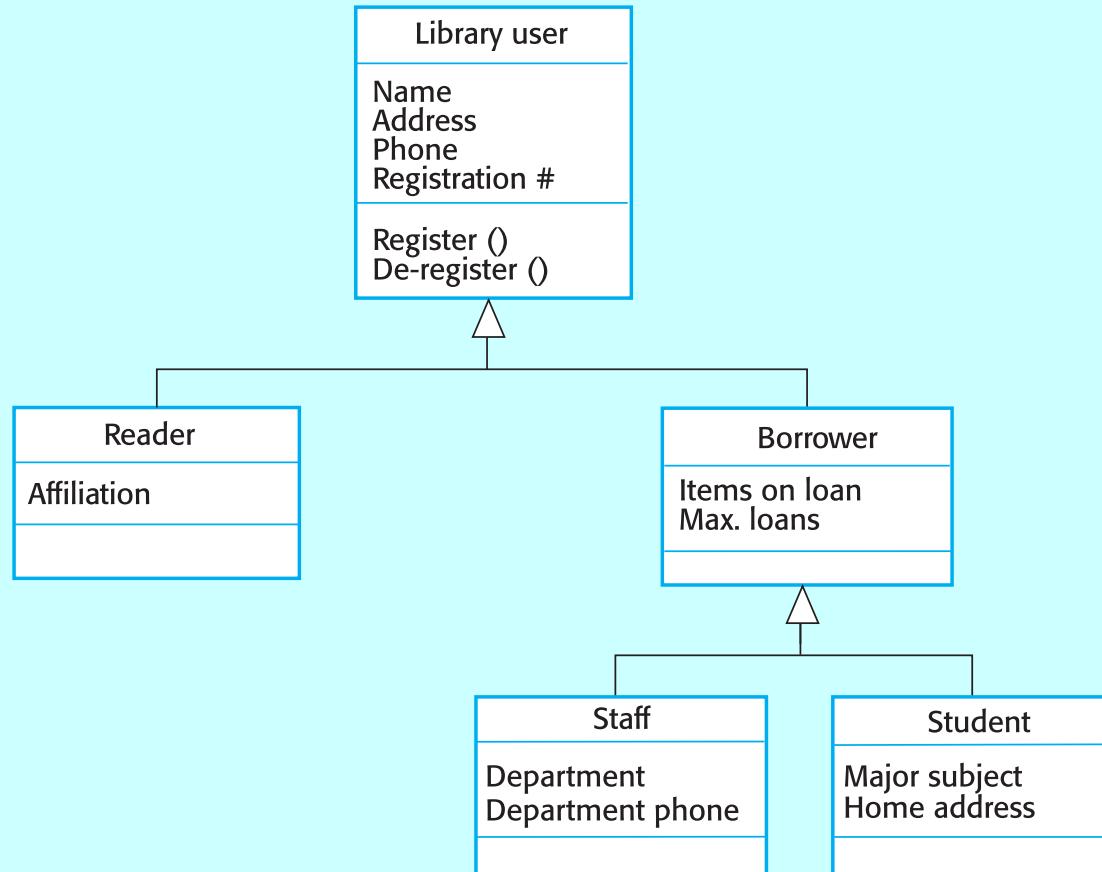
Veri (“Data”) Modeli – Örnek: Kütüphane Veri Sözlüğü (“Data Dictionary”)

Name	Description	Type	Date
Article	Details of the published article that may be ordered by people using LIBSYS.	Entity	30.12.2002
authors	The names of the authors of the article who may be due a share of the fee.	Attribute	30.12.2002
Buyer	The person or organisation that orders a copy of the article.	Entity	30.12.2002
fee-payable-to	A 1:1 relationship between Article and the Copyright Agency who should be paid the copyright fee.	Relation	29.12.2002
Address (Buyer)	The address of the buyer. This is used to any paper billing information that is required.	Attribute	31.12.2002

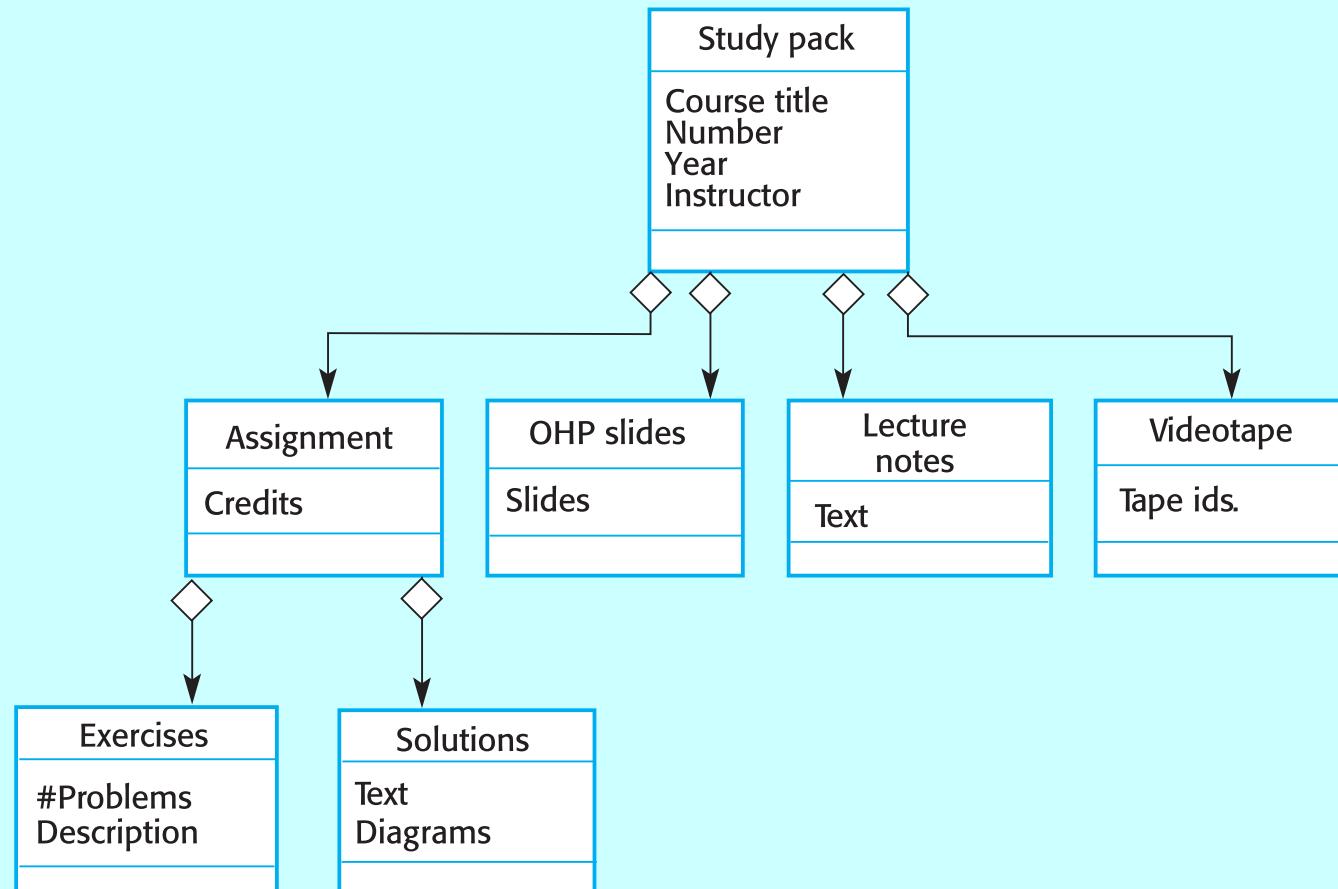
Nesne Kalıtım (“Object Inheritance”) Modeli – Örnek: Kütüphane Sınıf Hiyerarşisi



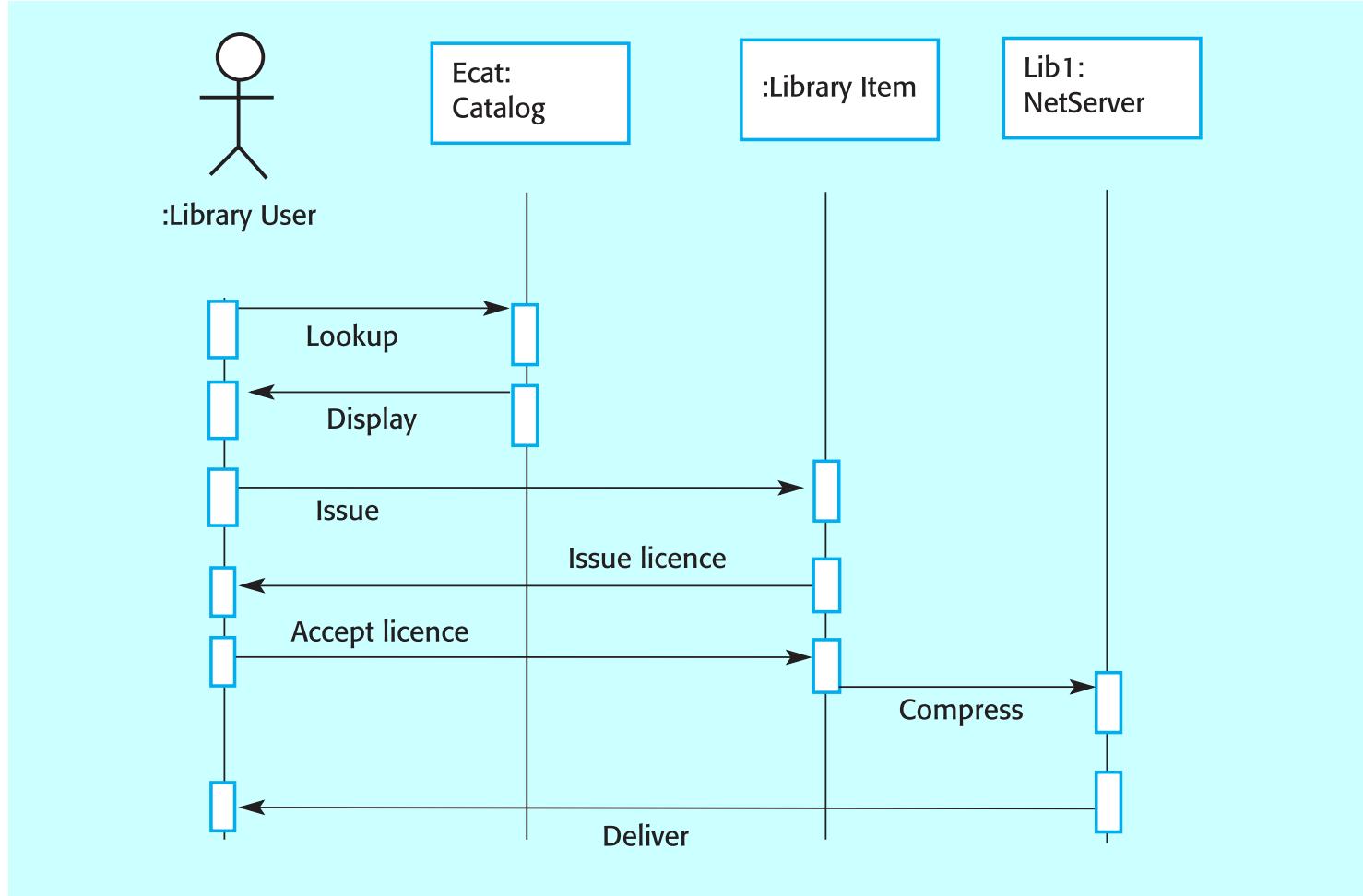
Nesne Kalıtım (“Object Inheritance”) Modeli – Örnek: Kütüphane Kullanıcısı Sınıf Hiyerarşisi Modeli



Nesne Bileşen (“Object Aggregation”) Modeli – Örnek: Ders Nesnesi Bileşen Modeli



Nesne Davranış (“Object Behavior”) Modeli – Örnek: Kütüphaneden Elektronik Öğe İndirme Davranış Modeli





BM306

Yazılım Mühendisliği



DERS 4

«*USE CASE*»

Gereksinim Analizi

Dr. Öğr. Üyesi Hasan BADEM
hbadem@ksu.edu.tr



- UML Giriş
- Gereksinim Analizi İçin Kullanılan Başlıca UML Elemanları
- “Use-Case” Esaslı Gereksinim Analizi
- Örnek Çözümleme: Kütüphane Destek Sistemi

“Unified Modeling Language” - 1

- Yazılım sistemlerinin modellemesi için geliştirilmiş standart bir dildir
 - ▶ Yazılım iş ürünlerinin; tanımlanması, görsel hale getirilmesi, belgelendirilmesi
 - ▶ Açık standarttır; birçok araç tarafından desteklenir
 - ▶ Tüm yazılım geliştirme sürecini destekler
- Çıkış hedefleri:
 - ▶ Kullanımı kolay, görsel bir modelleme dili sunmak
 - ▶ Programlama dillerinden ve geliştirme sürecinden bağımsız olmak
 - ▶ En iyi yöntemleri bütünlüğe getirmek
- 1995 yılından beri sektörün katkıları ile gelişmektedir
 - ▶ Son sürüm: UML 2.3 (Mart 2010), Object Management Group (OMG)
 - ▶ İlgili web sitesi: www.uml.org

“Unified Modeling Language” - 2

■ Sundukları:

- ▶ Yazılım ürünlerinin gösterimi için yapı taşları ve ilişkiler
- ▶ Yapı taşlarını ve ilişkileri kullanarak farklı bakış açılarını destekleyen diyagramlar
 - ◆ Tanımlanan yapı taşlarının ve diyagramların bütünü, geliştirilen yazılım sistemine karşılık gelir

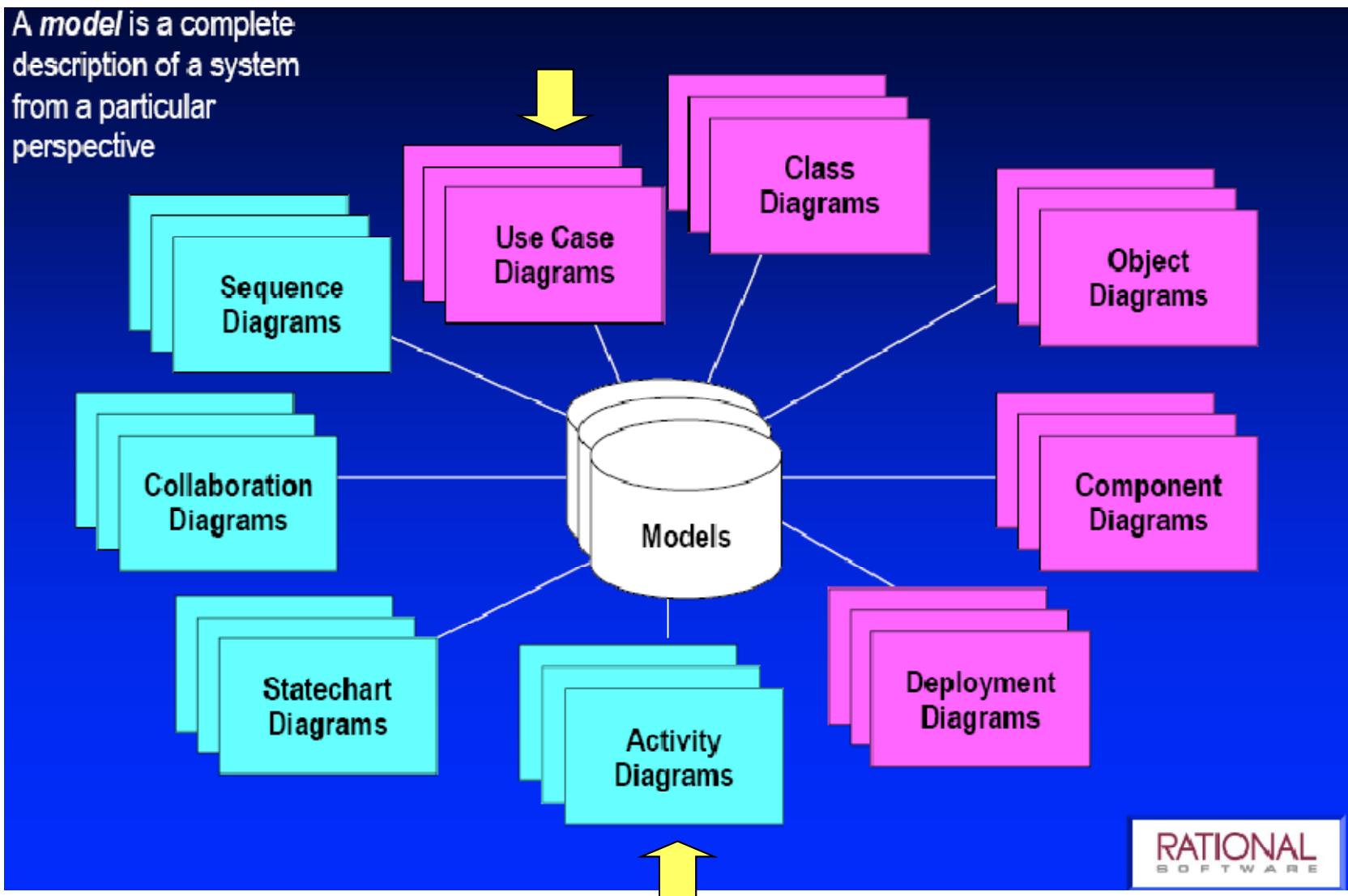
■ Sunmadıkları:

- ▶ Sisteminin nasıl geliştirilmesi gerektiğini tanımlamaz
- ▶ Nesne yönelimli yazılım modellemesi için yapılar sunar, ancak;
 - ◆ Bu yapıların hangi sıra ile kullanılması gerektiğini tanımlamaz
 - ◆ Yapıların geliştirme sürecinin hangi aşamalarında kullanılması gerektiğini tanımlamaz

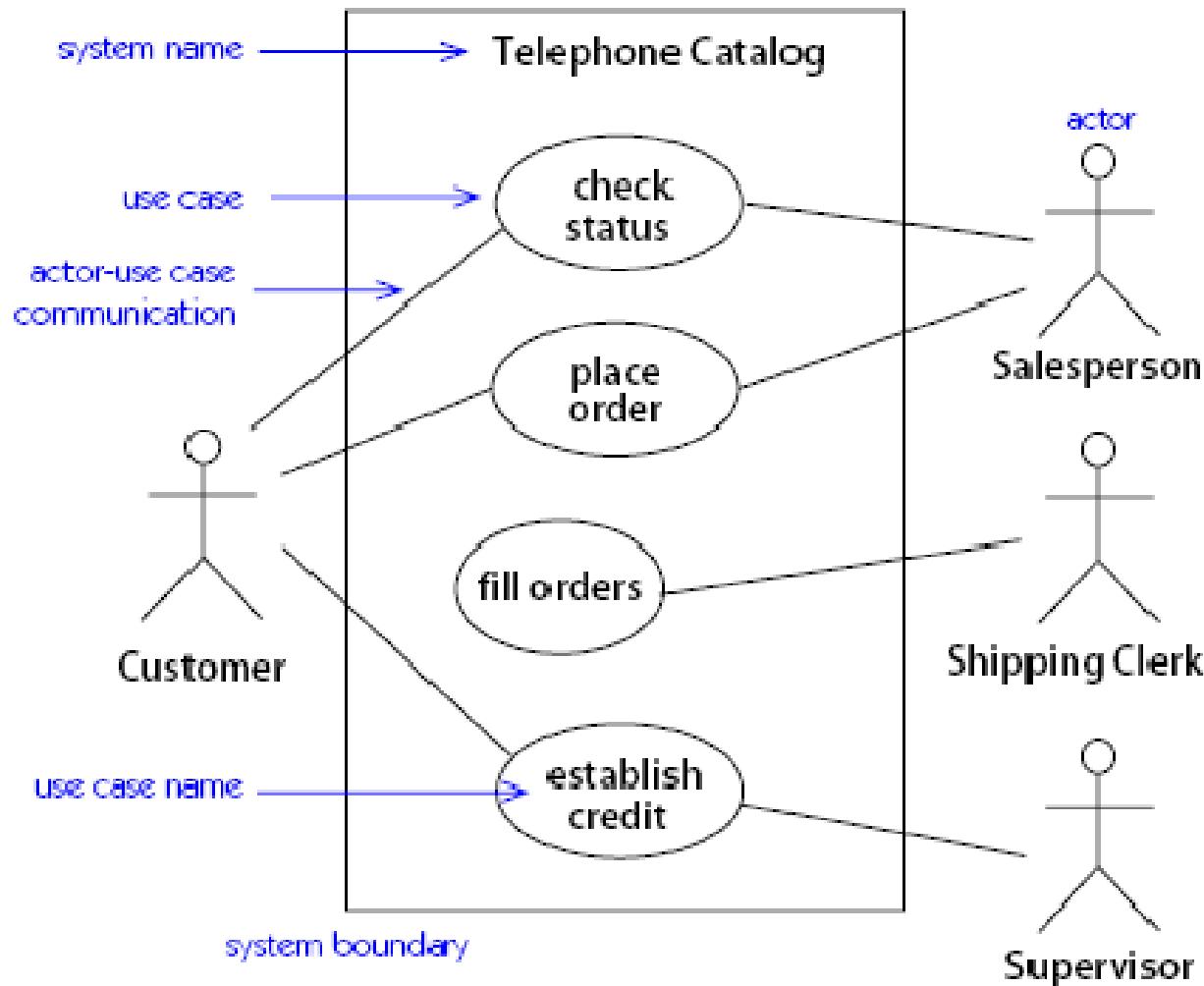
Gereksinim Analizi İçin Kullanılan Başlıca UML Elemanları

UML Diyagramları

A *model* is a complete description of a system from a particular perspective



“Use-Case Diagram”: Örnek



“Use-Case Diagram” Modelleme Öğeleri

■ Aktör

- ▶ Sistemin kullanıcıları
- ▶ “An outside user of a system”



■ “Use-case”

- ▶ Sistemin destekleyeceği işler
- ▶ A specification of the behavior of an entity in its interaction with outside agents



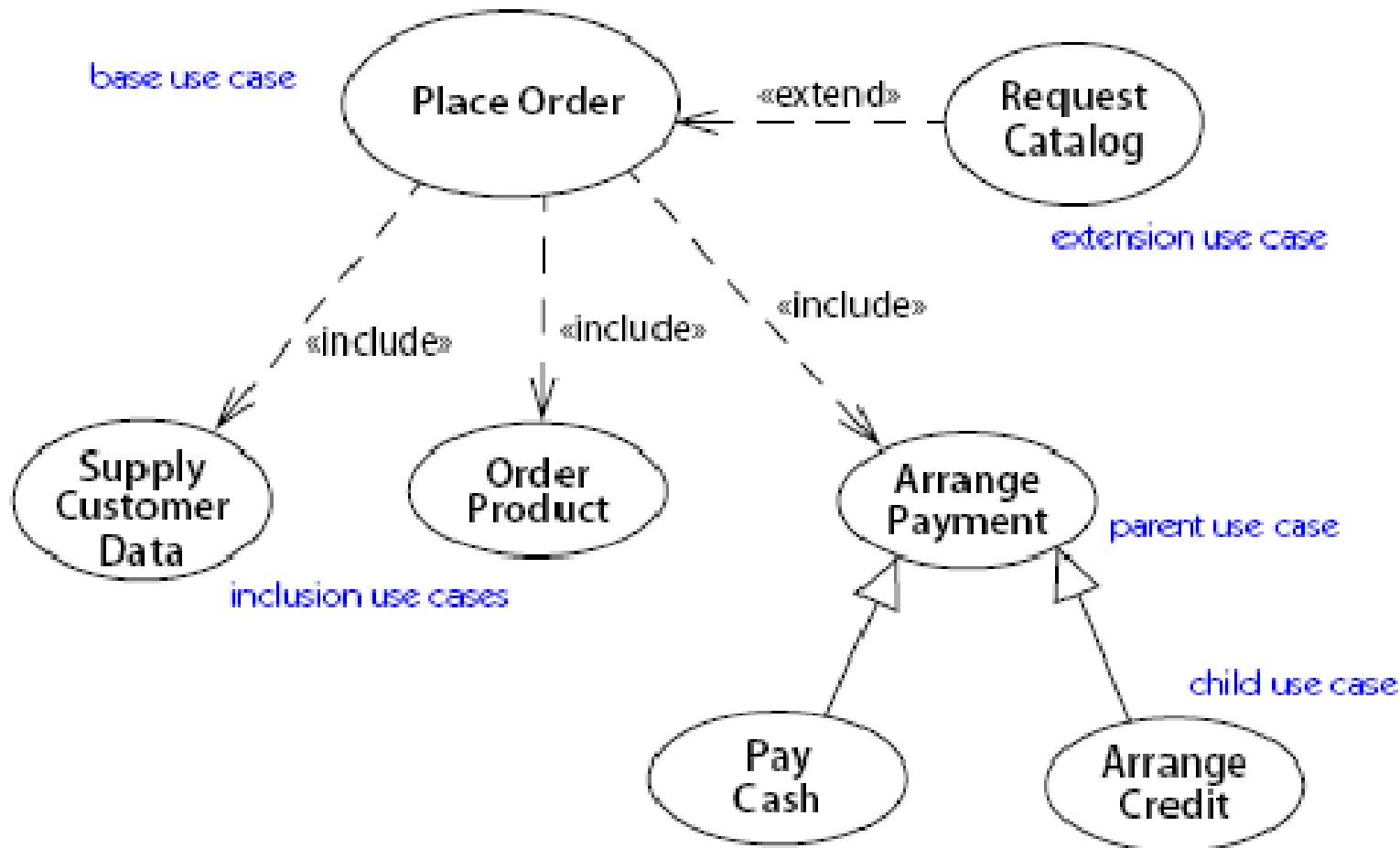
■ İlişki (“relationship”)

- ▶ “Association”: Aktör ve “use-case” arasındaki bağlantı
- ▶ “Generalization”: İki “use-case” veya iki aktör arasındaki kalıtım ilişkisi
- ▶ “Extend”: Bir “use-case”den diğerine geçiş (kontrol dışı)
- ▶ “Include”: Bir “use-case”的in diğerinin davranışını içermesi

“Use-Case” İlişki Türleri

<i>Relationship</i>	<i>Function</i>	<i>Notation</i>
association	The communication path between an actor and a use case that it participates in	_____
extend	The insertion of additional behavior into a base use case that does not know about it	«extend» - - - - >
use case generalization	A relationship between a general use case and a more specific use case that inherits and adds features to it	→
include	The insertion of additional behavior into a base use case that explicitly describes the insertion	«include» - - - - >

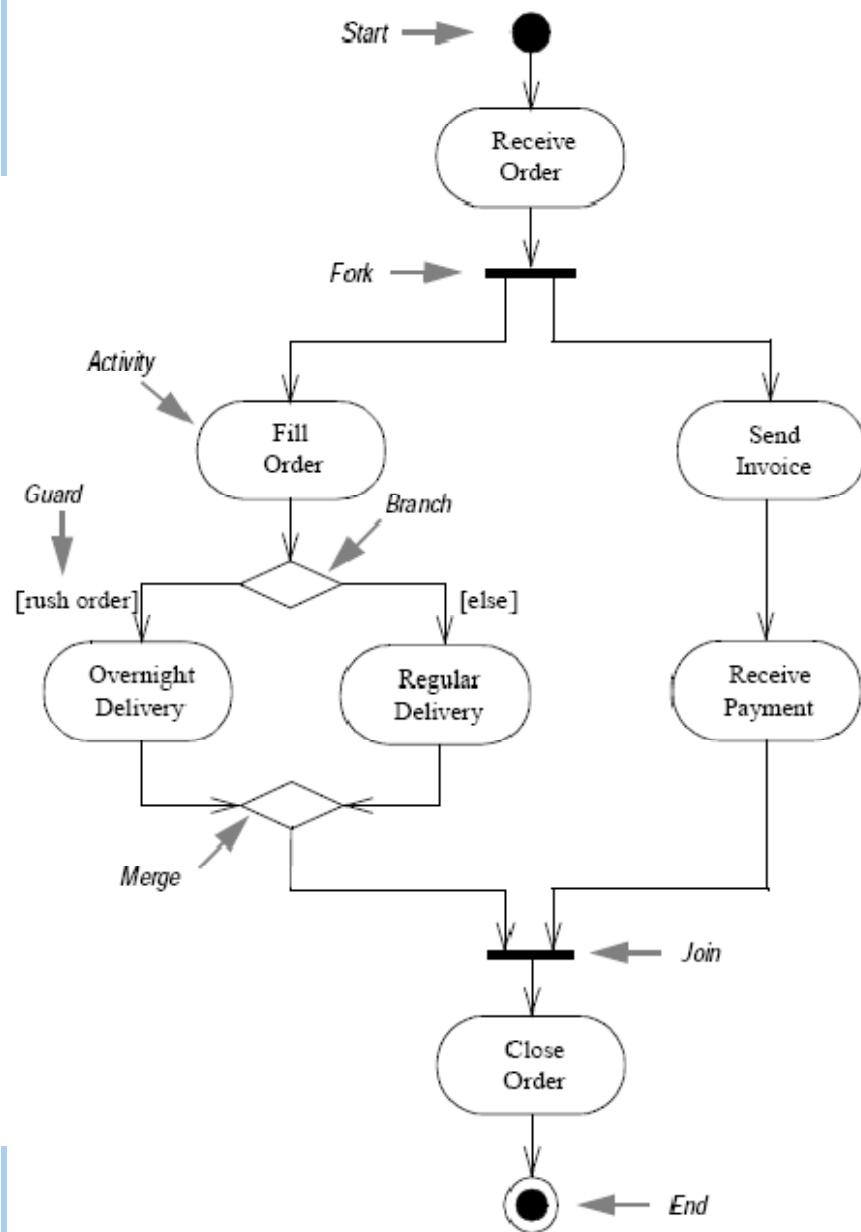
“Use-Case” İlişki Türleri: Örnek



“Use-Case” Modeli

- Sistemin tüm “use-case” diyagramları, “use-case” modelini tanımlar
 - ▶ Sistem belirli büyüklüğün üzerinde olduğunda, gereksinimler birden fazla “use-case” diyagramı kullanılarak tanımlanır
 - ▶ Sistem sınırını gösteren dikdörtgen kutu sistemin içinde ve dışında kalan öğeleri belirtmek için kullanılır

“Activity Diagram”: Örnek



“Activity Diagram” Modelleme Öğeleri

■ Etkinlik (“activity”)

- ▶ Sistem ve aktörler tarafından yapılan işleri ifade etmek için kullanılır
- ▶ “An activity is a state of doing something”

■ Geçiş (“transition”)

- ▶ Etkinlikler arasındaki geçişleri ifade etmek için kullanılır
- ▶ Geçişin koşulu varsa geçişin üzerine “guard” eklenerek ifade edilir

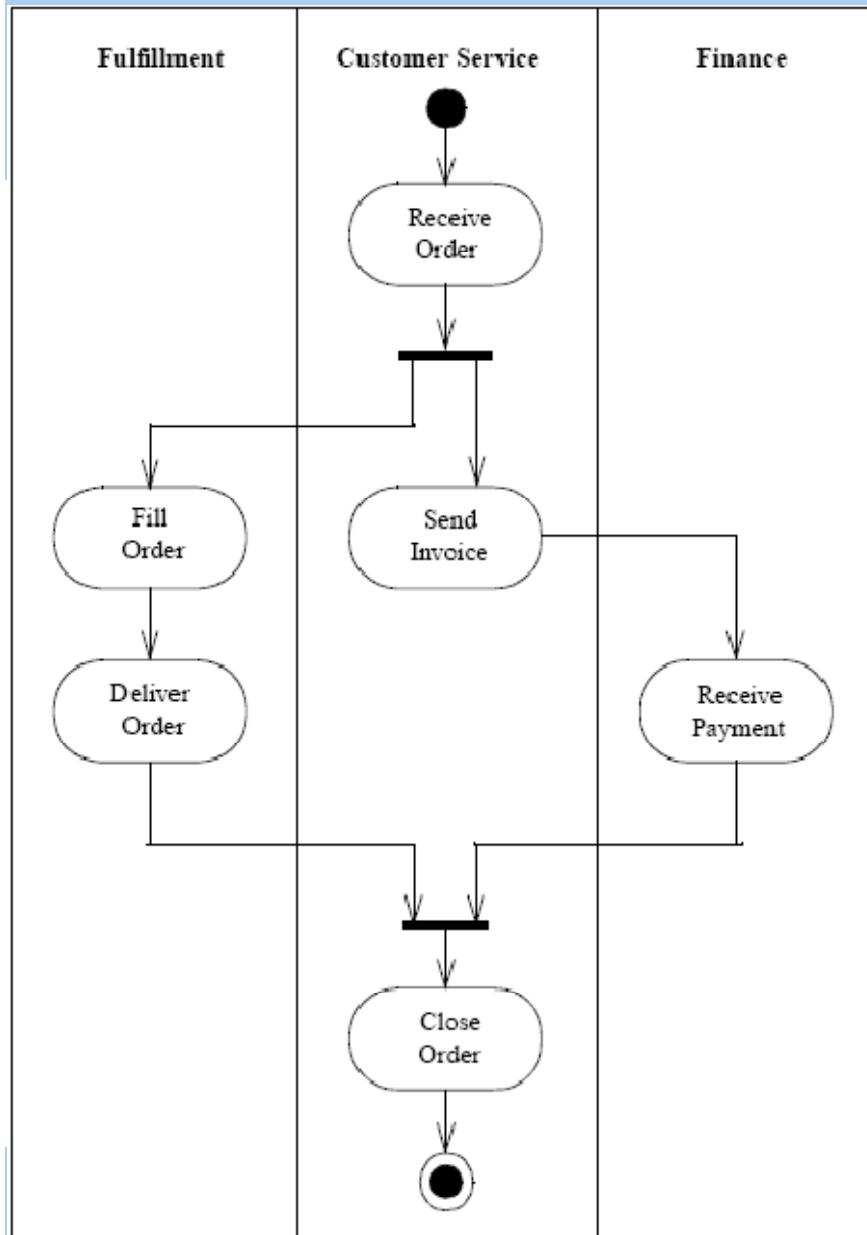
■ “Branch” / “merge”

- ▶ Koşullu davranışı ifade etmek için kullanılır
 - ◆ “A branch has a single incoming transitions and several guarded outgoing transitions ... only one of the outgoing transitions can be taken, so the guards should be mutually exclusive”
 - ◆ “A merge has multiple input transitions and a single output – the end of conditional behavior”

■ “Fork” / “join”

- ▶ Paralel davranışını ifade etmek için kullanılır
 - ◆ “A fork has one incoming transition and several outgoing transitions ... when the incoming transition is triggered, all of the outgoing transitions are taken in parallel”
 - ◆ “With a join, the outgoing transition is only taken when all the states on the incoming transitions have completed their activity”

“Activity Diagram with Swimlanes”: Örnek



“Use-Case” Esaslı Gereksinim Analizi

Gereksinim Analizinde “Use-Case” Yaklaşımı

■ Bakış açısı: Sistem, kullanıcısı için “ne” yapacak ?

- ▶ Sistem kapalı bir kutu (“black-box”)
- ▶ Sistem-kullanıcı etkileşimi
- ▶ Sistemin dışarıdan görünen davranışı

■ İlgiilenmediklerimiz:

- ▶ Sistemin iç yapısı
- ▶ Sistem belirlenen davranışı “nasıl” yapacak ?
- ▶ Belirlenen davranış “nasıl” kodlanacak ?

Bu bakış açısı, sistemdeki tüm işlevselligi değil, kullanıcılar için artı değer oluşturacak işlemleri düşünmemizi sağlar

“Use-Case” Nedir? (1)

■ Yazılım sisteminin kullanıcısına değer döndüren, dışarıdan gözlemlenebilen davranışının bütünüdür

▶ Değer

- ◆ Sistem “use case” kapsamındaki işleri gerçekleştirdiğinde oluşacak çıktı, sonuç veya durumdur

▶ Gözlemlenebilirlik

- ◆ Kullanıcı ve sistem arasındaki etkileşimi tanımlar

▶ Bütünlük:

- ◆ “Use-case” kullanıcının istediği değerin üretilebilmesi için gereken tüm ilişkili adımları içerir

“Use-Case” Nedir? (2)

■ Her “use-case”in bir amacı vardır

- ▶ Niye belirlendi ? Kullanıcının hangi işlemini gerçekleştirecek ?
- ▶ Hangi davranışı modelliyor ? Ne yapacak ?
- ▶ Ne zaman sonlanacak ? Hangi değeri üretecek ?

■ Örnekler:

- ▶ Öğrencinin bir derse kaydolması
- ▶ Muhasebe görevlisinin aylık bordroları hazırlaması
- ▶ Banka müşterisinin ATM’den para çekmesi
- ▶ Bir kişinin asansörü çağırması

Gereksinim Analizinde Yapısal Yöntem ya da “Use-Case” Yaklaşımı: Örnek

■ Yapısal yöntem:

- ▶ Sistem müşteri kartını kabul eder.
- ▶ Sistemde işlemlerin yapılabilmesi için şifre geçerli olmalıdır.
- ▶ Sistemde para çekme ve para yatırma seçenekleri mevcuttur.
- ▶ Sistem istendiğinde işlemlerin makbuzunu verir.
- ▶

■ Use case yaklaşımı:

- ▶ Müşteri kartını yerleştirir
- ▶ Sistem şifreyi sorar
- ▶ Müşteri şifresini girer
- ▶ Şifre doğruysa, sistem para çekme ve para yatırma seçeneklerini sunar
- ▶ Müşteri para çekme işlemini seçer
- ▶

“Use-Case” Esaslı Gereksinim Analizi – Kazançlar

- Kullanıcının gereksinimi olmayan özellikleri tanımlamamızı engeller
- Kullanıcının da anlayabileceği şekilde sistemin davranışlarını ve sorumluluklarını tanımlar
 - ▶ Kullanıcı ile iletişimini kolaylaştırır
- Kullanıcı arayüzlerinin tasarılanmasını kolaylaştırır
- Kullanıcı kılavuzlarını yazarken başlangıç noktasını oluşturur
- Geliştirme sürecini başlatır ve tüm temel iş adımlarını birbirine bağlar
- Tasarlanacak test durumlarına esas oluşturur

“Use-Case” Esaslı Gereksinim Analizi – Dikkat Edilecek Noktalar

- “Use-case”ler sistemin iç yapısını tanımlamaz (sistem kapalı kutu)
 - ▶ Tasarım öğeleri belirsizdir
 - ◆ Tipik olarak birden çok tasarım ögesi, bir use case'in işletilmesi için kullanılır
 - ▶ Yapısal veya nesne yönelimli yaklaşılarda kullanılabilmesinin temel nedeni budur
- “Use case”ler sadece işlevsel gereksinimleri, kullanıcı bakış açısıyla tanımlar
 - ▶ Sistemin iç davranışına ilişkin gereksinimler, kullanıcı gereksinimleri gerçekleştirilirken daha sonraki adımlarda karşılanır
 - ◆ Bu tür gereksinimler sıkılıkla, iş kuralı veya tasarım kısıtı olarak “use-case”lerle ilişkilendirilir

“Use-Case” Esaslı Gereksinim Analizi: Yöntem

1. Aktörleri ve “use-case”leri belirle

- ▶ Amaç: Sisteminin aktörlerini ve “use-case”lerini belirlemek ve üst seviye “use-case” modelini oluşturmak
 - ◆ Aktörler belirlenir
 - ◆ “Use-case”ler belirlenir
 - ◆ Her aktör ve “use case” kısaca tanımlanır
 - ◆ Üst seviye “use-case” modeli tanımlanır

2. “Use-case”leri detaylandırır

- ▶ Amaç: Belirlenen tüm “use-case”lerin iş akışlarını detaylı olarak tanımlamak
 - ◆ Ana akış tanımlanır
 - ◆ Alternatif akışlar tanımlanır

“Use-Case” Esaslı Gereksinim Analizi: Yöntem (devamı)

3. “Use-case” modelini yapılandırır

- ▶ Amaç: Oluşturulan use case modelini ortak noktaları en aza indirecek şekilde yapılandırmak
 - ◆ Gereken yerlerde “extend” ve “include” ilişkileri kullanılabilir
 - ◆ Yapılandırılan “use-case” modeli, iş süreçlerini referans alınarak değerlendirilir

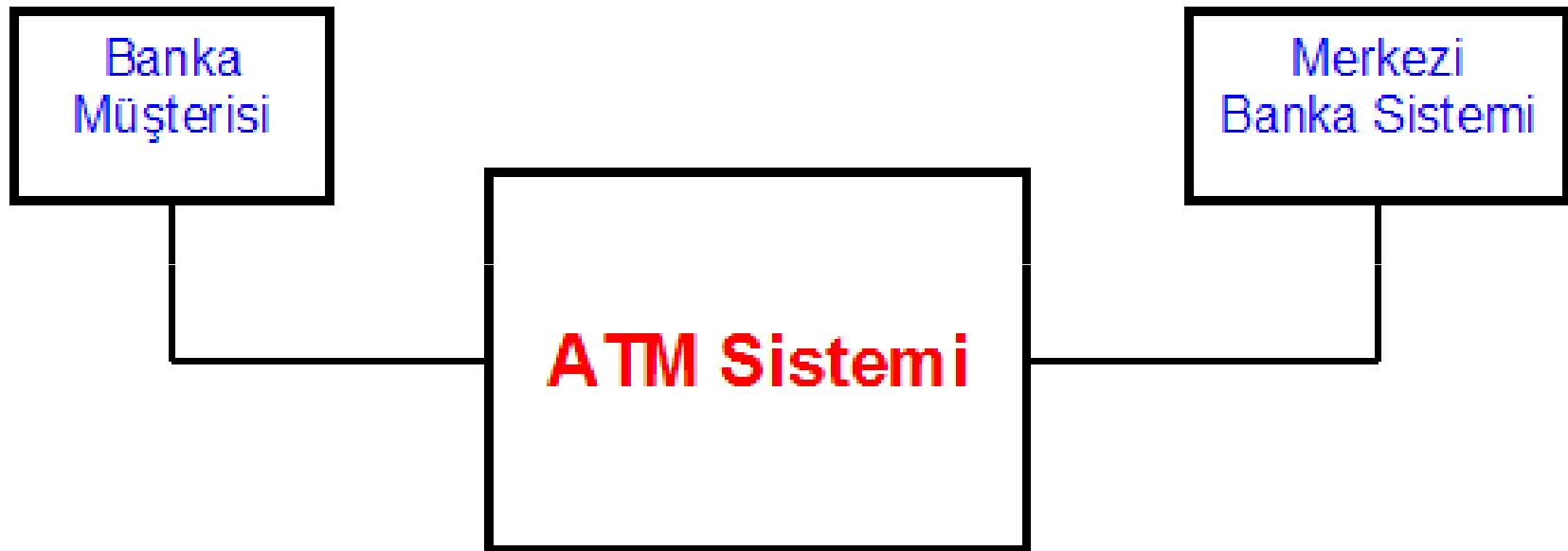
4. Kullanıcı arayüzlerini tanımla

- ▶ Amaç: Use case tanımları esas alınarak kullanıcı arayüzlerini üst seviyeli olarak tanımlamak
 - ◆ Kağıt üzerinde çizim yapılabilir
 - ◆ Arayüz prototipleme aracı kullanılabilir

Örnek: ATM Uygulaması

- Bir bankanın ATM cihazı için yazılım geliştirilecektir. ATM, banka kartı olan müşterilerin hesaplarından para çekmelerine, hesaplarına para yatırmalarına ve hesapları arasında para transferi yapmalarına olanak sağlayacaktır. ATM, banka müşterisi ve hesapları ile ilgili bilgileri, gerekiğinde merkezi banka sisteminden alacaktır. Banka sistemi ayrıca her günün sonunda, ATM'den günlük işlemlerin bir özeti isteyecektir.

ATM Uygulaması – Kapsam



Bağlam ("context") diyagramı

ATM Uygulaması (Adım 1. Aktörleri ve “Use Case”leri Belirle) – Aktörler

■ Soru: ATM uygulama yazılımının kullanıcıları kimlerdir?

- ▶ Banka müşterisi
- ▶ Merkezi Banka Sistemi

ATM Uygulaması (Adım 1. Aktörleri ve “Use Case”leri Belirle) – “Use Case”ler

■ Soru: Belirlenen aktörler ATM'den ne istiyorlar ?

- ▶ Aktör: Banka müsterisi
 - ◆ Para çekme
 - ◆ Para yatırma
 - ◆ Para transferi
- ▶ Aktör: Merkezi Banka Sistemi
 - ◆ Günlük özet alma

ATM Uygulaması (Adım 1. Aktörleri ve “Use Case”ları Belirle) – Kısa Tanımlar

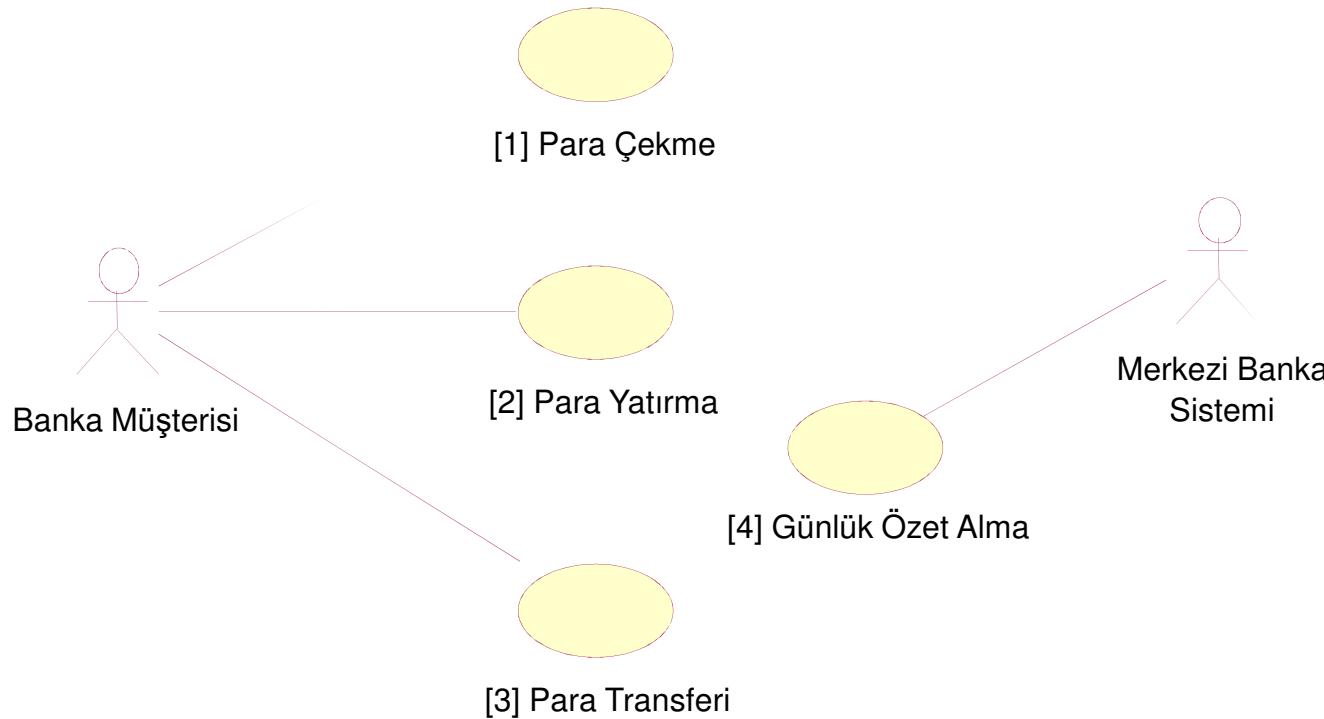
■ Aktör: Banka müsterisi

- ▶ Bankada hesabı ve banka kartı olan, ATM'den işlem yapma hakkı olan kişi

■ Use case: Para çekme

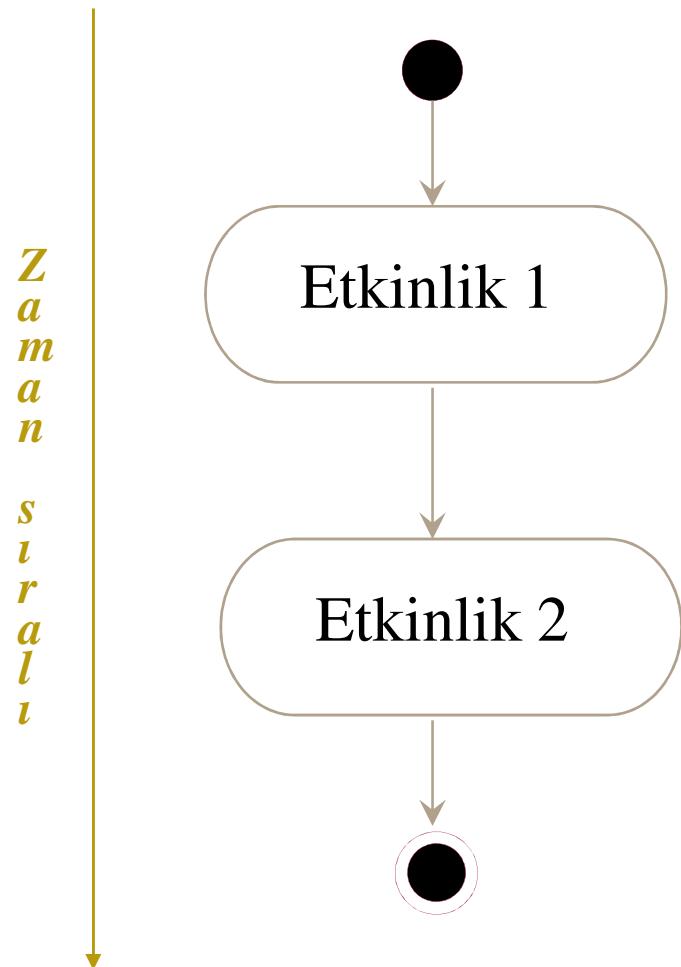
- ▶ Banka müsterisinin nasıl para çekeceğini tanımlar. Para çekme işlemi sırasında banka müsterisinin istediği tutarı belirtmesi ve hesabında bu tutarın mevcut olması gereklidir.

ATM Uygulaması (Adım 1. Aktörleri ve “Use Case”leri Belirle) – “Use-Case” Diyagramı



Her use case biricik (“unique”) olarak numaralandırılmış

“Use Case” Detayı: Etkinlik Zinciri



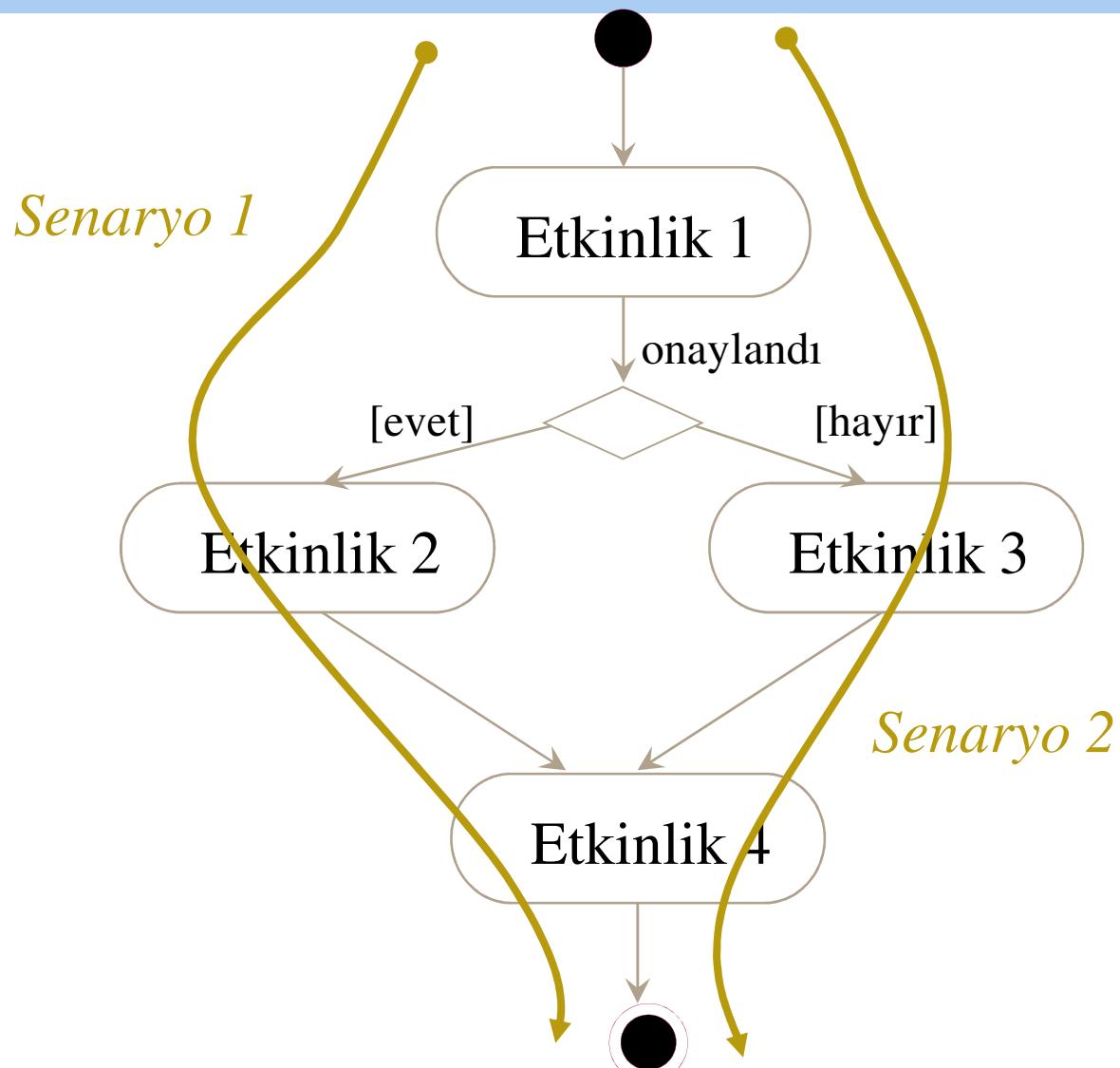
ATM Uygulaması (Adım 2. “Use Case”leri Detaylandırır) – Para Çekme (Ana Akış)

1. Banka Müşterisi kartını yerleştirdir
2. ATM kartı okur
3. Banka Müşterisi şifreyi girer
4. ATM işlem seçeneklerini gösterir
5. Banka Müşterisi “para çekme” işlemini seçer
6. ATM olası para tutarlarını gösterir
7. Banka Müşterisi para tutarını girer
8. ATM para çekme talebini Merkezi Banka Sistemi’ne ileter
9. Merkezi Banka Sistemi, Banka Müşterisi’nin hesabını kontrol eder
10. Merkezi Banka Sistemi, Banka Müşterisi’nin hesabından tutarı düşü ve işlem sonucunu ATM’ye ileter
11. ATM nakit parayı verir
12. ATM kartı iade eder

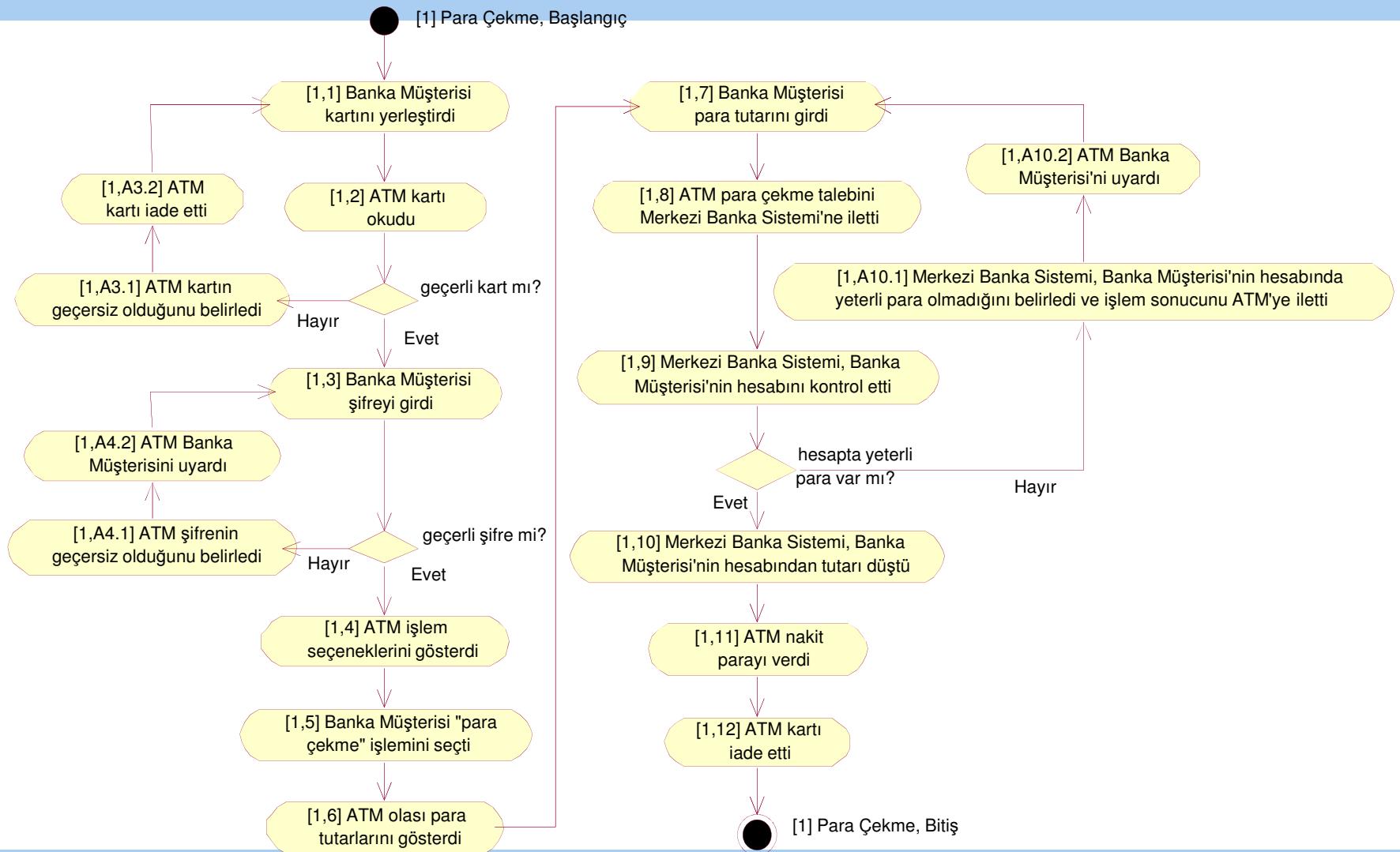
*ana
akış*

Olası sapma noktaları

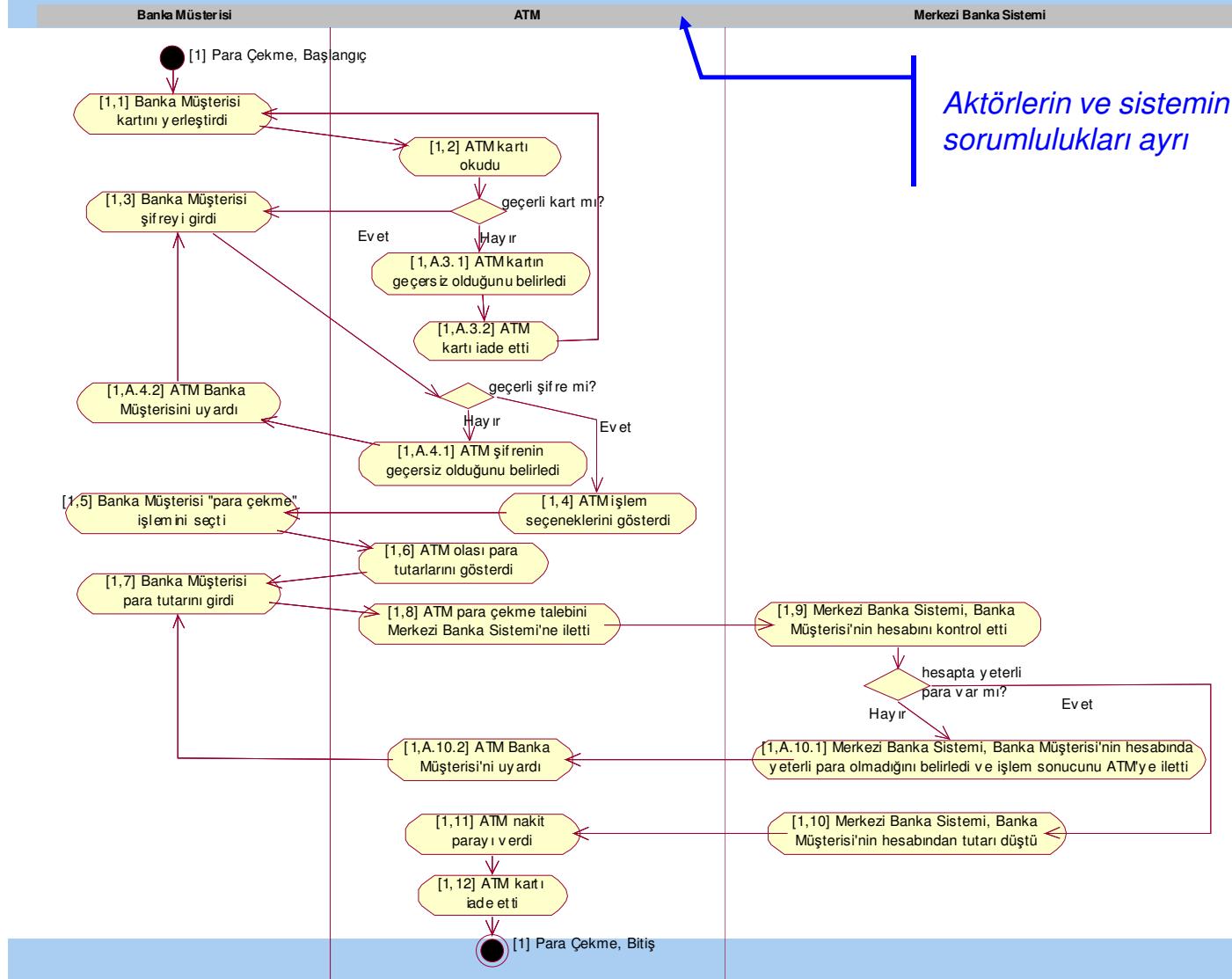
“Use Case” Detayı: Alternatif Akışlar



ATM Uygulaması (Adım 2. "Use Case"leri Detaylandı) – Para Çekme ("Activity Diagram")



Para Çekme – “Activity Diagram with Swimlanes”

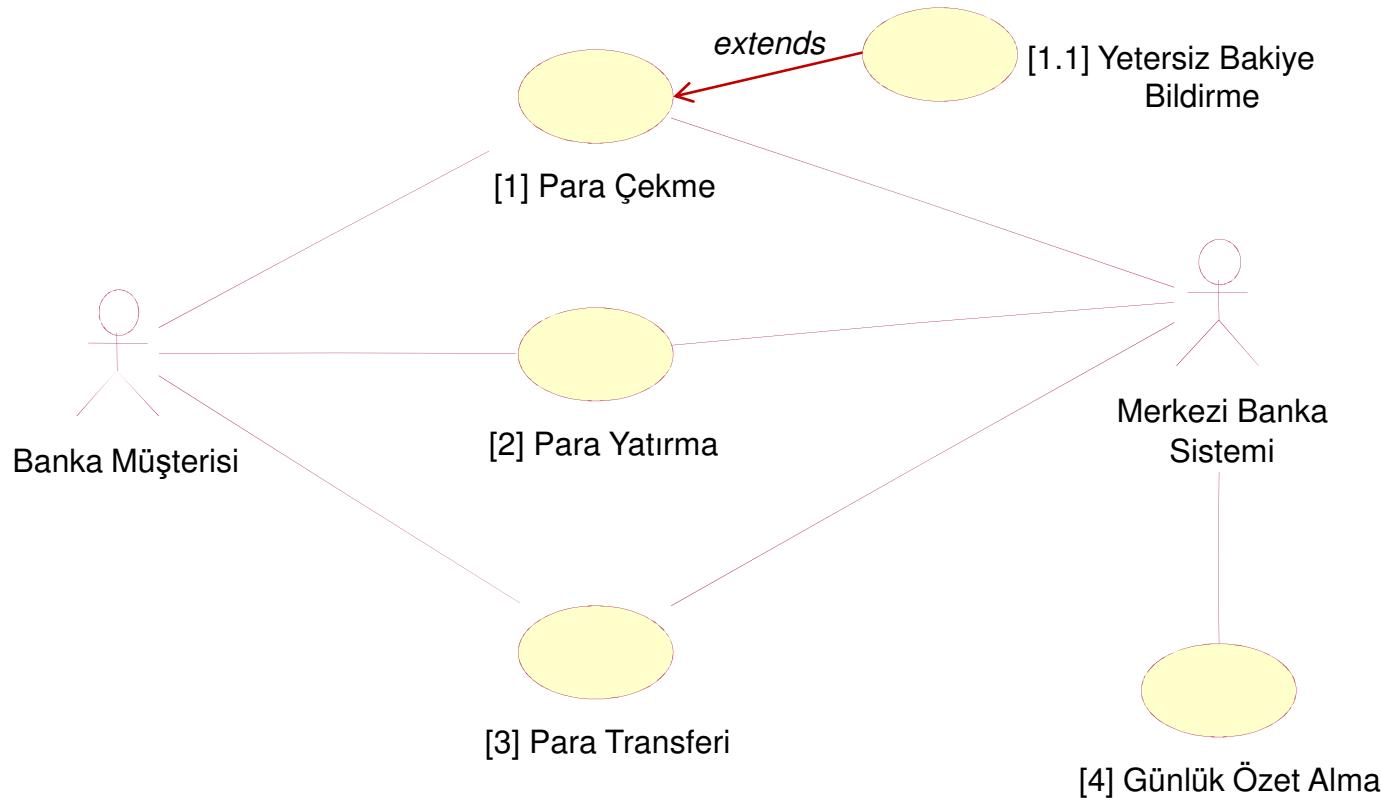


Para Çekme: Alternatif Tanım

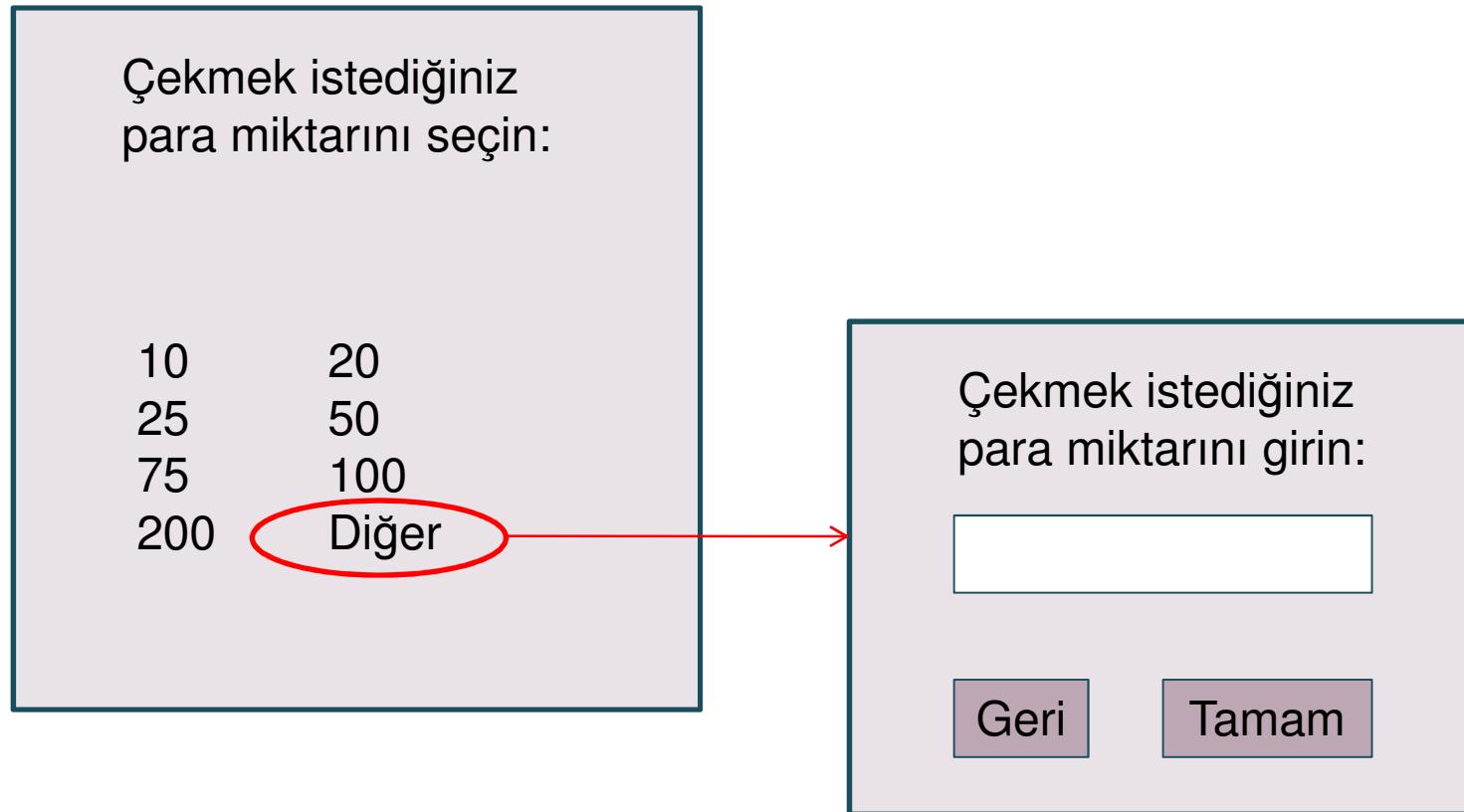
Use Case No:	12		
Use Case Adı:	Para Çekme		
Tanımlayan:		Son Değiştiren:	
Tanımlama Tarihi:		Son Değişiklik Tarihi:	

Aktör:	Banka müsterisi, Merkezi Banka Sistemi
Kısa Tanımı:	Banka müsterisinin nasıl para çekeceğini tanımlar.
Önkoşul:	Merkezi Banka Sistemi erişilebilir durumdadır.
Sonkoşul:	Banka müsterisi nakit parayı ve banka kartını alır.
Önceliği:	1
Kullanım sıklığı:	Çok sık
Ana akış:	<ol style="list-style-type: none"> 1. Banka Müsterisi kartını yerleştirdi 2. ATM kartı okudu 3. Banka Müsterisi şifreyi girdi 4. ATM işlem seçeneklerini gösterdi 5. Banka Müsterisi “para çekme” işlemini seçti 6. ATM olası para tutarlarını gösterdi 7. Banka Müsterisi para tutarını girdi 8. ATM para çekme talebini Merkezi Banka Sistemi’ne iletti 9. Merkezi Banka Sistemi, Banka Müsterisi’nin hesabını kontrol etti 10. Merkezi Banka Sistemi, Banka Müsterisi’nin hesabından tutarı düşüdü ve işlem sonucunu ATM’ye iletti 11. ATM nakit parayı verdi 12. ATM kartı iade etti
Alternatif Akış:	<p>A3: ATM kartın geçersiz olduğunu belirledi</p> <p>A3.1. ATM kartı iade etti</p> <p>A3.2. <i>Adım-1'den devam et</i></p> <p>A4: ATM şifrenin geçersiz olduğunu belirledi</p> <p>A4.1. ATM Banka Müsterisi’ni uyardı</p> <p>A4.2. <i>Adım-3'ten devam et</i></p> <p>A10: Merkezi Banka Sistemi, Banka Müsterisi’nin hesabında yeterli para olmadığını belirledi ve işlem sonucunu ATM’ye iletti</p> <p>A10.1. ATM Banka Müsterisi’ni uyardı</p> <p>A10.2. <i>Adım-7'den devam et</i></p>
İçerdiği use case’ler:	
Özel gereksinimler:	(Adım-2) ATM kartı 3 saniye içinde okulmalıdır.

ATM Uygulaması (Adım 3. “Use-Case” Modelini Yapılandır) – “Use-Case” Diyagramı



ATM Uygulaması (Adım 4. Kullanıcı Arayüzlerini Tanımla) – Para Çekme “Use Case”i için Kullanıcı Arayüzü



(Para Çekme ana akışı 6 ve 7 no'lu adımlarıyla ilişkili kullanıcı arayüzleridir.)

“Use-Case” Esaslı Gereksinim Analizi: Başarı İçin Anahtar Noktalar

- “Use-case” modelini iteratif olarak geliştirin
- Kullanıcıları analize dahil edin
- “Use-case”leri görsel olarak modelleyin
- “Use-case”leri işlevsel olmayan gereksinimleri çıkarmak için kullanın
- “Use-case”leri ve “use-case” senaryolarını önceliklendirin
- “Use-case”leri doğrulayın ve diğer geliştirme öğelerine izlenirliğini kurun

Örnek Çözümleme: Kütüphane Destek Sistemi

- Kütüphane işlemlerini desteklemek amacıyla bir yazılım sistemi oluşturulacaktır.
- Sistem; kayıtlı müşterilere, yine kayıtlı kitap ve dergileri ödünç verecektir.
- Kütüphane, yeni başlıklı kitap ve dergilerin satın alınmasını yapacaktır. Popüler başlıklar, birden çok kopya satın alınacaktır. Eski kitap ve dergiler, zaman aşımına uğradıklarında veya çok yıprandıklarında yok edilecektir.
- Kütüphanede, müşterilerle iletişimini sağlayacak ve yaptığı işler sistem tarafından desteklenecek bir kütüphane görevlisi bulunacaktır.
- Müşteri, kütüphanede o anda bulunmayan bir kitap veya dergiyi rezerve edebilecektir. Kitap veya dergi kütüphaneye geri döndürüldüğünde, rezervasyonu yapan müşteri haberdar edilecektir. Rezervasyon, müşteri kitap veya dergiyi ödünç aldığında veya müşterinin özel isteği üzerine iptal edilecektir.
- Sistem; kitap ve dergi başlıklarının, kitap ve dergi kopyalarının, müşterilerin, ödünç işlemlerinin ve rezervasyonların kaydedilmesine, güncellenmesine ve silinmesine olanak sağlayacaktır.
- Sistem tüm popüler bilgisayar ortamlarında (UNIX, Windows, OS/2, vb.) çalışacak ve modern bir kullanıcı ara yüzüne sahip olacaktır.
- Sistem yeni işlevler eklemek suretiyle genişletilebilir olacaktır.



BM306

Yazılım Mühendisliği



DERS 5

Nesneye Yönerek Tasarım

Dr. Öğr. Üyesi Hasan BADEM
hbadem@ksu.edu.tr



■ Nesneye yönelik tasarım

- ▶ Nesne ve sınıf kavramları
- ▶ Tasarım için kullanılan başlıca UML elemanları
- ▶ Üst düzey tasarım adımları
- ▶ Detay tasarım adımları

■ Örnek çözümleme

- ▶ Kütüphane destek sistemi

■ Ödev 2

- ▶ KDS için örnek UML diyagramlarının hazırlanması

“Nesne” ve “Sınıf” Kavramları

“Nesne” Nedir?

- İyi tanımlı bir kapsamı ve kimliği olan, belirli bir durum ve davranışı içeren, soyut veya somut varlıktır.
 - ▶ “A discrete entity with a well-defined boundary and identity that encapsulates state and behavior”
 - ▶ Nesne gerçek dünyadaki somut bir varlığı temsil edebilir.
 - ◆ Televizyon, motor, vb.
 - ▶ Nesne tamamen kavramsal bir varlığı temsil edebilir.
 - ◆ Banka hesabı, vb.

“Nesne”: Örnek

Müşteri Hesabı

miktar

paraÇek
ParaYatır
miktarSorgula

- Her nesne aşağıdakilere sahiptir:

- ▶ Kişilik (“identity”)
- ▶ Özellik (“attribute”)
- ▶ Durum (“state”)
- ▶ Davranış (“behavior”)
- ▶ Sorumluluk (“responsibility”)

Özellik

- Her nesnenin kendine ait bir dizi özelliği vardır.
 - ▶ Özellikler nesneye ait verileri taşır.

:Arabam

model
marka
renk

:BirMusteri

isim
musteriNo
bakiye

:BirPencere

boyut
pozisyon
renk

Durum (1)

- Nesnenin tüm özellikleri ve bu özelliklerin o anki değerleri, nesnenin durumunu oluşturur.

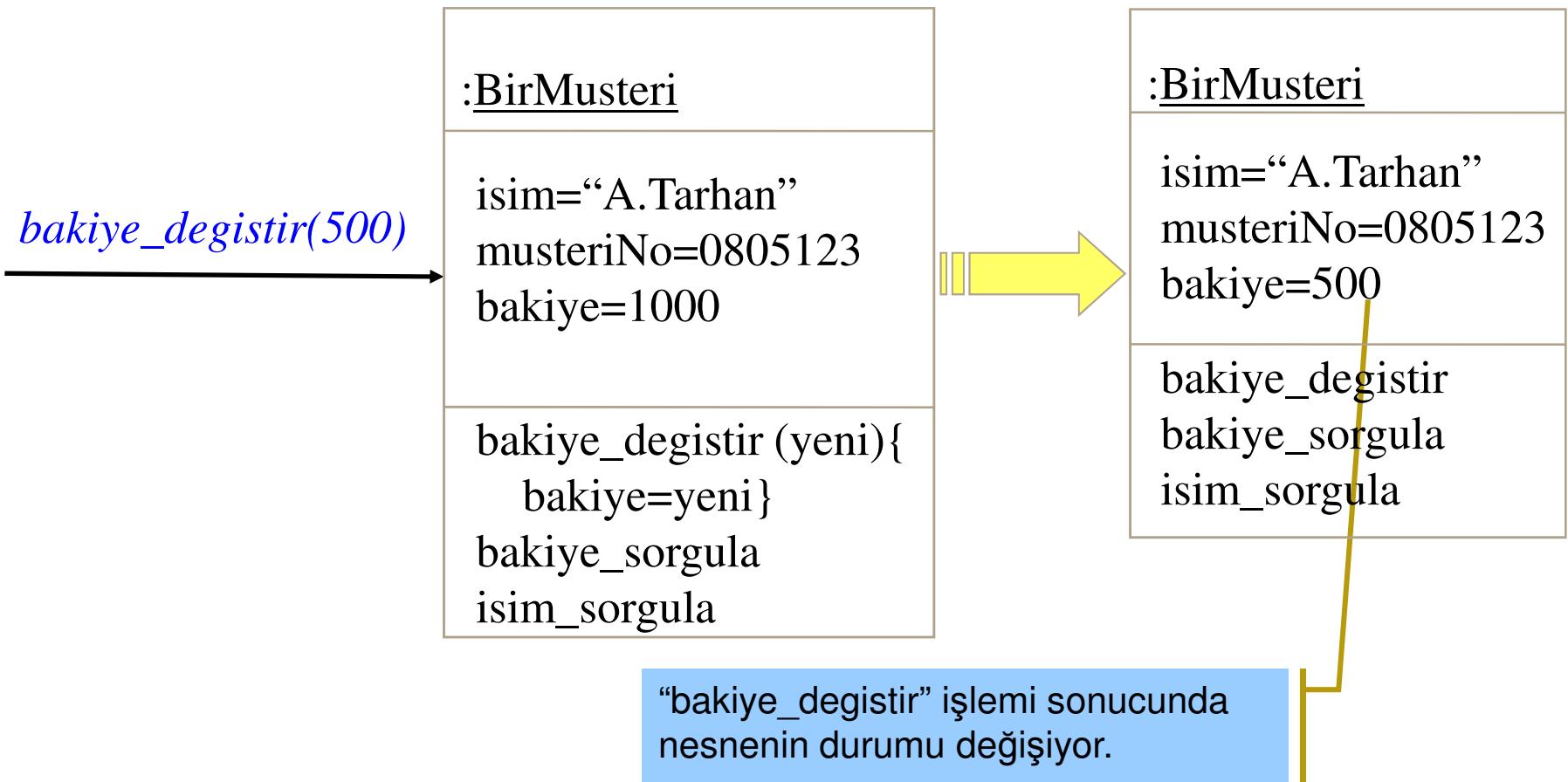


Davranış

- Her nesnenin iş yapabilmeyi sağlayan tanımlı bir davranış şekli vardır.
 - ▶ Nasıl davranışır, nasıl tepki verir ? (“act”/“react”)
 - ◆ Nesnenin operasyonları
 - ◆ Görülebilir aktivite
 - ◆ Diğer nesneler tarafından kullanılan arayüz (“public interface-operations”)
 - ▶ Davranışın gerçekleştirilmesi bilgisayar kodu ile yapılır.
 - ◆ Kodlanan nesne yordamları, davranışını gerçekleştirir.
 - ◆ Gizli gerçekleştirmeye (“encapsulated implementation”)

Durum (2)

- İşlemler sonucunda nesnenin durumu değişir.



Sorumluluk

- Nesnenin sorumluluğu tüm sistemin işlevselligine nasıl katkıda bulunacağını tanımlar.
 - ▶ Nesnenin sistemde oynayacağı rol
 - ▶ Neden böyle bir nesneye gereksinim var ?
 - ▶ Özellikler ve davranışlar, birlikte nesnenin sorumluluklarını yerine getirmesini sağlar.

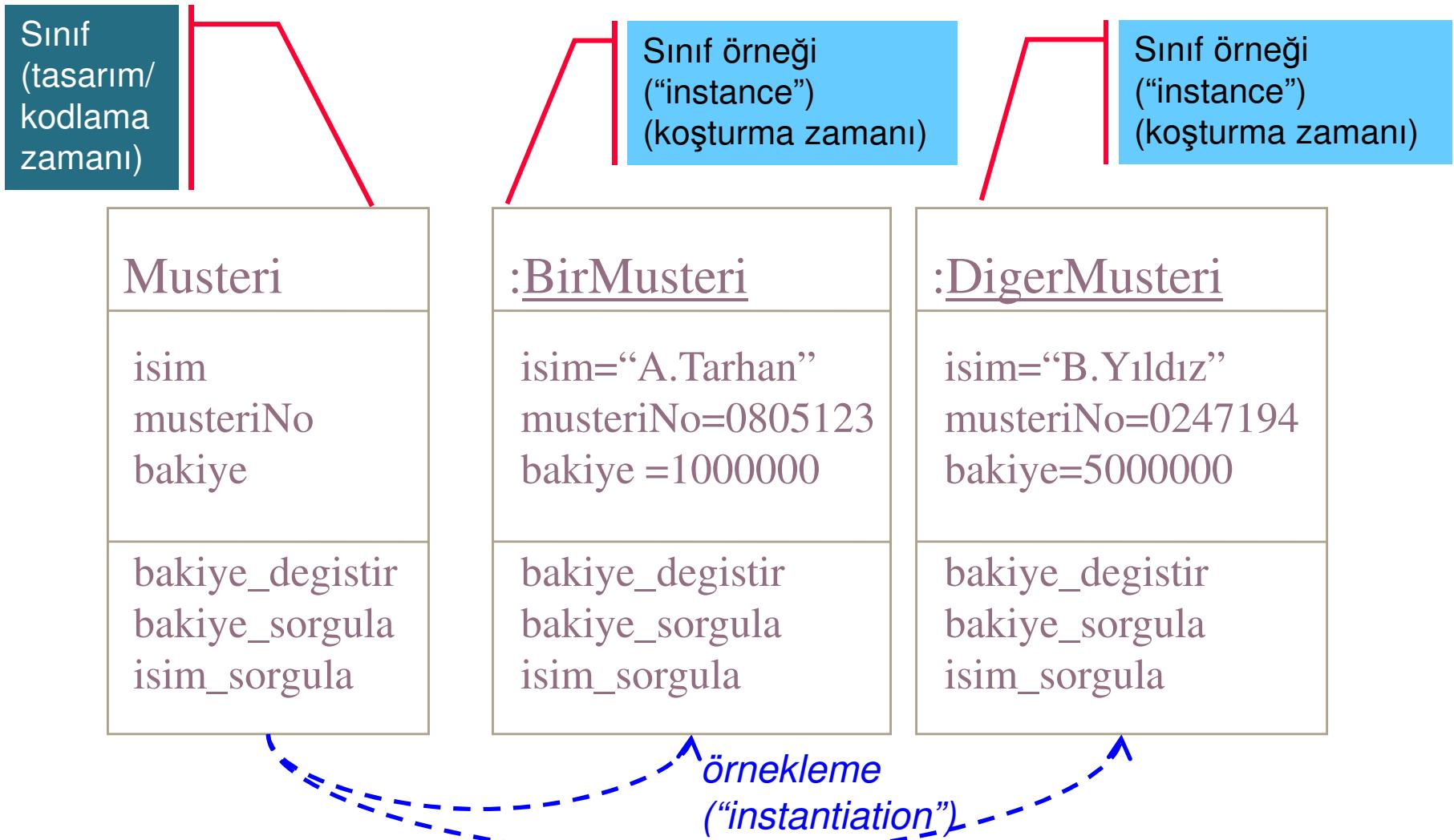
İlişki (“Relationship”)

- Nesneler arasındaki fiziksel veya kavramsal bağlantı
- En yaygın ilişki: Bir nesne diğer nesnenin sunduğu servislerden yararlanır.
 - ▶ Mesaj iletimi
 - ▶ Müşteri/tedarikçi ilişkisi
- Sistem işlevsellliğini sağlamak amacıyla, nesneler birbirleriyle ilişkiler aracılığı ile işbirliği yaparlar.
 - ▶ Nesneye yönelik programlama modeli

“Sınıf” Nedir?

- Yapısal ve/veya davranışsal olarak aynı özelliklere sahip nesneler SINIF altında gruplanır.
 - ▶ Her nesne bir sınıfın örneğidir (“instance”).
 - ▶ Sınıfları tanımlar ve nesneleri sınıf tanımından örnekleriz (“instantiation”).
 - ▶ Her nesne ait olduğu sınıfı bilir.
- Sınıf, nesneler için şablon tanımıdır.
 - ▶ Özellikler ve yordamlar sınıf için yalnızca bir kez tanımlanır.
- Sınıfların birbirleri arasındaki ilişkiler sistemin sınıf yapısını (“class structure”) oluşturur.
 - ▶ İki sınıf arasında ilişki varsa karşılık gelen nesneler arasında da vardır.

“Sınıf”: Örnek



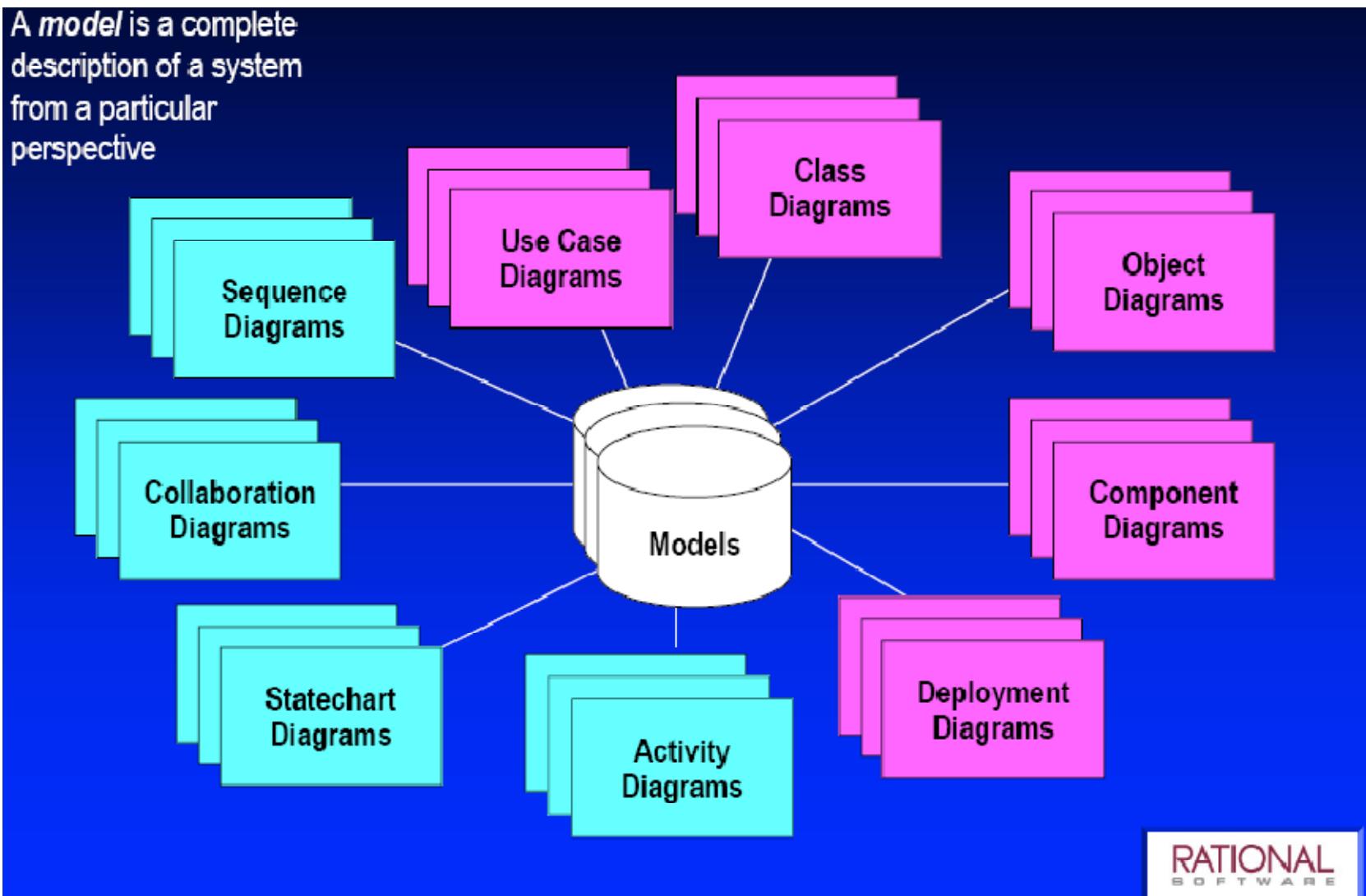
“Sınıf” ve “Nesne”

- Her sınıfın sıfır veya daha fazla örneği vardır.
- Sınıf statik, nesne dinamiktir.
 - ▶ Sınıfın varlığı, semantiği ve ilişkileri program koşturulmadan önce sabit olarak belirlenmiştir.
 - ▶ Nesneler program koşturulduğunda sınıf tanımından dinamik olarak yaratılırlar (“construction”).
 - ▶ Nesneler sorumluluklarını tamamladıklarında ortadan kaldırılırlar (“destruction”).
- Nesnenin sınıfı sabittir ve nesne bir kez yaratıldıkten sonra değiştirilemez.

Nesneye Yönelik Tasarım İçin Kullanılan Başlıca UML Elemanları

UML Diyagramları

A *model* is a complete description of a system from a particular perspective



Tasarım İçin Kullanılan Başlıca UML Elemanları (1)

- Sınıf diyagramı (“class diagram”)
 - ▶ Sistemi oluşturan sınıflar ve bunlar arasındaki ilişkiler
- Ardıl-işlem diyagramı (“sequence diagram”)
 - ▶ Nesneler arası etkileşim (davranış)
 - ▶ Nesneler arası kontrolün akışı (zamana göre sıralı)
- İşbirliği diyagramı (“collaboration diagram”)
 - ▶ Nesneler arası etkileşim (davranış)
 - ▶ Nesneler arası kontrolün akışı (mesaj sıralı)

Tasarım İçin Kullanılan Başlıca UML Elemanları (2)

- Durum diyagramı (“statechart diagram”)
 - ▶ Dinamik davranış, olay (“event”) esaslı
 - ▶ Bir nesnenin iç davranışları (nesne yaşam döngüsü)
- Etkinlik diyagramı (“activity diagram”):
 - ▶ Operasyon akışları
- Bileşen diyagramı (“component diagram”):
 - ▶ Uygulamanın fiziksel yapısı
 - ▶ *Gerçekleştirme aşamasında*
- Yayılma diyagramı (“deployment diagram”):
 - ▶ Donanım topolojisi
 - ▶ *Tasarım ve gerçekleştirme aşamalarında*

Üst Düzey Tasarım

Üst Düzey Tasarım

■ Amaç: Sistemi oluşturacak temel varlıkları ve ilişkilerini tanımlamak

- ▶ Sistemin sınıf yapısı
 - ◆ Uygulama alanına özgü sınıflar (“domain classes”)
 - ◆ Sınıflar arası ilişkiler
- ▶ Davranış modeli
 - ◆ Sınıfların işlevsel gereksinimleri karşılamak için birbirleri ile nasıl işbirliği yapacakları
- ▶ Literatürde “OO Analysis” olarak da geçiyor

“Use Case” ve Tasarım Modelleri

USE-CASE MODELİ :

- Doğal dil
- Dış bakış açısı (“black box”)
- “Use case” ile yapılandırılır
- Müşteriler esaslı
- Tekrar ve tutarsızlıklar olabilir
- İşlevsellinin “ne” olduğunu tanımlar

TASARIM MODELİ :

- Yazılım modelleme dili
- İç bakış açısı
- Sınıflar ile yapılandırılır
- Geliştiriciler esaslı
- Tekrarlar ve tutarsızlıklar olamaz
- İşlevsellinin sistemde “nasıl” gerçekleştirileceğini tanımlar

Üst Düzey Tasarım Adımları

1. Sınıfları belirle

- ▶ Amaç: “Use case” tanımlarından analiz sınıflarını (“domain classes”) ve sorumluluklarını belirlemek
 - ◆ Odak noktası sınıfları belirlemektir (detaylı tanımların sonra yapılması önerilir)

2. Sınıf yapısını belirle

- ▶ Amaç: Belirlenen sınıfların “use case”lerde tanımlanan iş adımlarını gerçekleştirmek için nasıl etkileşimde bulunacağını saptamak
 - ◆ “Use case”ler gözden geçirilerek belirlenen sınıfların arasındaki ilişkiler tanımlanır

3. Davranışı modelle

- ▶ Amaç: “Use case”lerin içeriği işlevselligin, belirlenen sınıfların davranışları aracılığıyla nasıl gerçekleştirileceğini tanımlamak
 - ◆ Nesnelerin belirlenen işi yerine getirmek için nasıl mesajlaşacakları belirlenir
 - ◆ Nesneler arası kontrol akışı dinamik olarak gösterilir

Adım-1. Sınıfları Belirle (1)

- Use case modelinde yer alan her use case tanımı gözden geçirilir
 - ▶ Analiz sınıfı (“domain class”) özelliklerini taşıyan varlıklar belirlenir
 - ▶ Belirlenen sınıflara sorumluluk atanır
 - ◆ Use case tanımlarındaki işler esas alınır
 - ▶ Önceki use case’lerde belirlenmiş sınıfları kullanmaya çalışızız
 - ◆ Analiz sınıfları büyük olasılıkla birçok use case’de karşımıza çıkacaktır

Adım-1. Sınıfları Belirle (2)

■ Use case tanımlarının üzerinden geçerken:

- ▶ Kullanılan “isimler” aday sınıf olarak ele alınır
- ▶ Aday sınıflar belirli kurallar ve gereksinimler doğrultusunda elenir
- ▶ Elemler sonucunda oluşan “isim” kümesi sistemin sınıfları olarak ele alınır
- ▶ Elemlerden geçemeyen “isim”ler belirlenen sınıfların özellikleri olabilirler
- ▶ İşi tanımlayan “fiil”ler sınıfların sorumluluklarını veya sınıflar arası ilişkileri tanımlar

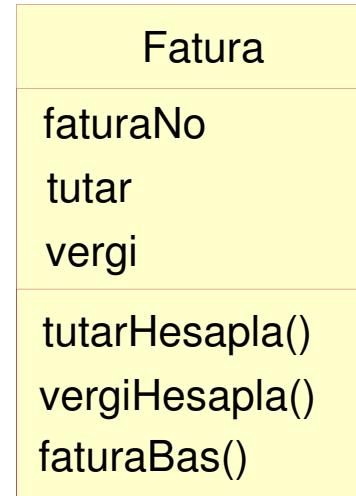
Adım-1. Sınıfları Belirle (3)

- Belirlenen her sınıf için, sınıfın özellikleri tanımlanır:
 - ▶ Özellikler “use case” tanımlarında doğrudan yer alabilir
 - ▶ Özellikler sınıfın sorumluluklarından çıkarılabilir
 - ▶ Bu aşamada özellikler ile ilgili detaylı bir tanımlamaya gerek yoktur
 - ◆ Veri yapılarına girilmeden sadece isimlendirilir
- Belirlenen her sınıf için sınıfın davranışı, sorumluluklar esas alınarak operasyon halinde tanımlanır
 - ▶ “Use case”de tanımlanan işlevsellik, sınıflar arasında paylaştırılır
 - ▶ Sorumluluk, sınıfın farklı “use case”lerde aldığı rollerin birleşimidir
 - ◆ Her rol bir operasyon olabilir
 - ▶ Bu aşamada “class methods”, “signatures” detaylarına girilmez

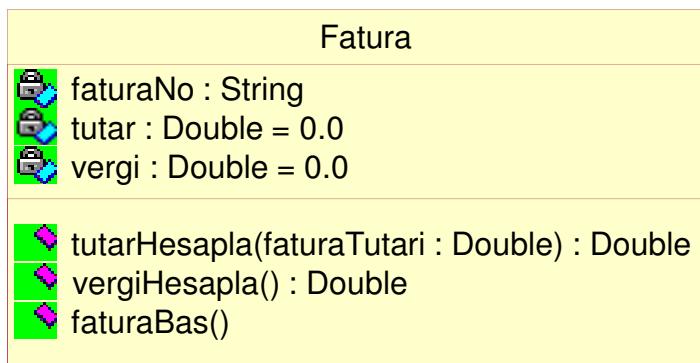
Tasarım Boyunca Sınıf Tanımının Gelişimi: Örnek



Üst Düzey Tasarım
(ön aşamalar)



Üst Düzey Tasarım
(ileri aşamalar)

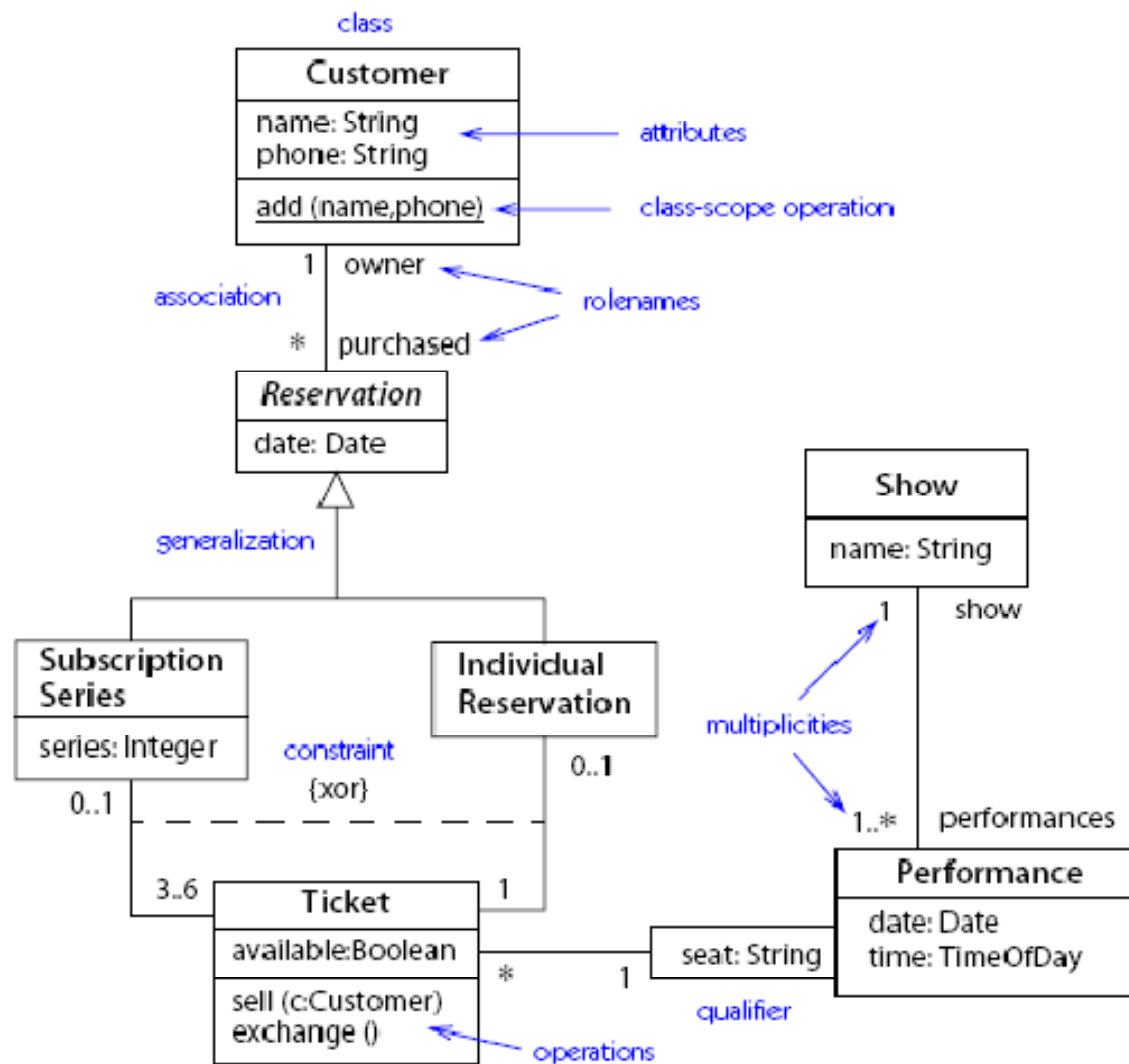


Detay Tasarım

Adım-2. Sınıf Yapısını Belirle

- Belirlenen ilişkiler davranış modellemesine esas oluşturur
 - ▶ “Fiil”ler ilişkilerin göstergesidir
- Sınıflar arası ilişkiler kavramsal olarak tanımlanır
 - ▶ İlişkinin türlerine ve detay özelliklerine girilmez
- Sınıflar arası ilişkiler en az sayıda tutulmaya çalışılır
- Tasarım süreci devam ettikçe basit ilişkiler özel formlara dönüşebilir
 - ▶ “association”, “aggregation”, “composition”, “generalization”, vb.

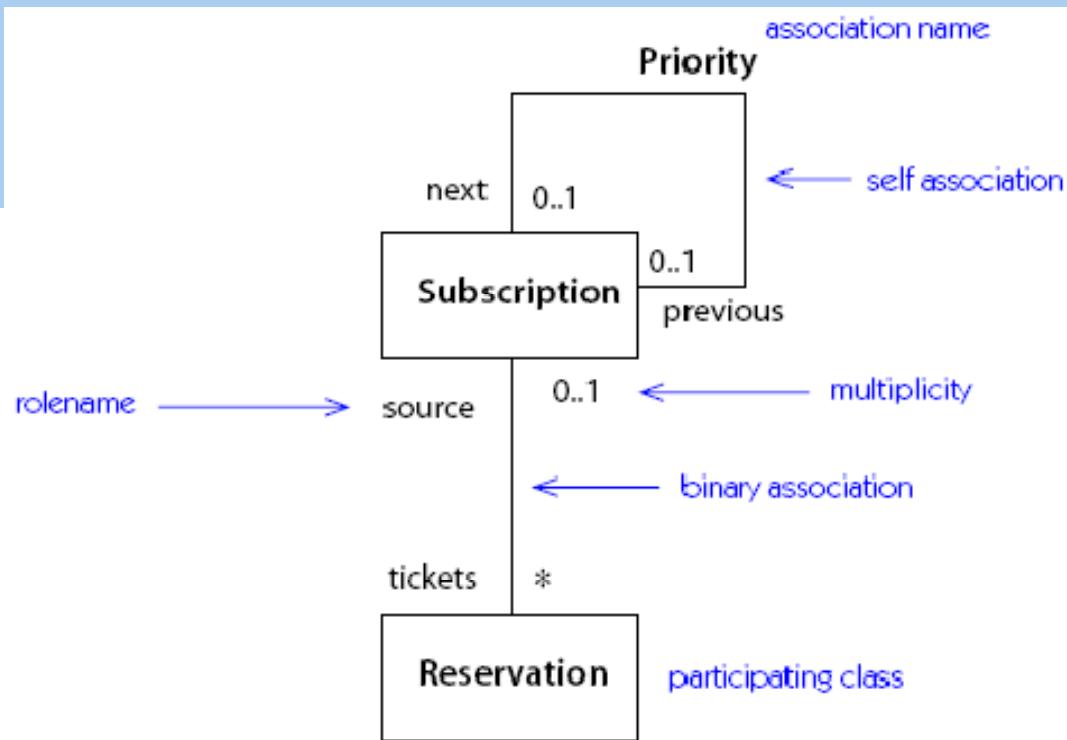
“Static View” “Class Diagram”: Örnek



Sınıf Diyagramı

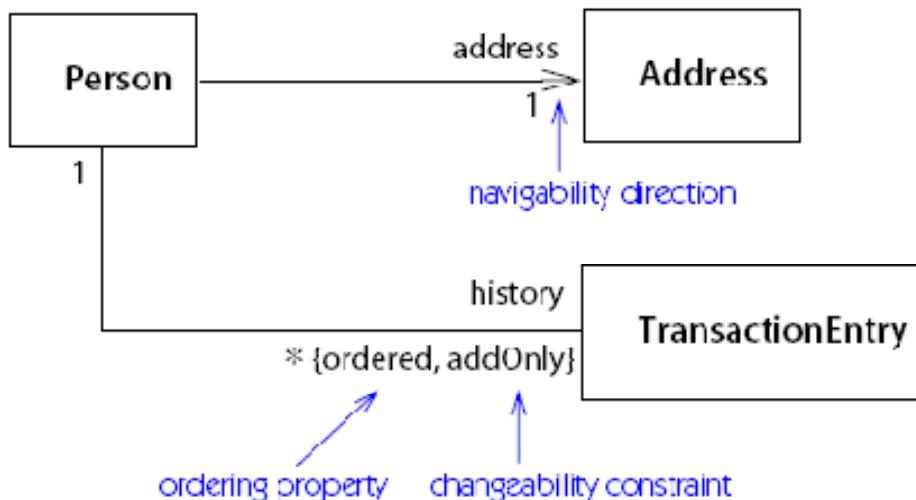
- Sınıf diyagramı “statik” bakış açısı sunar
 - ▶ Yol haritası gibi
 - ◆ Nesneler şehirleri, ilişkiler şehirler arasındaki yolları gösterir
 - ◆ Hedefe ulaşmak için hangi yolun takip edilmesi gerektiğini söylemez
- Sınıf diyagramında hangi nesnelerin işbirliği yaptığı belli, ancak nesnelerin nasıl işbirliği yapacakları belli değildir
 - ▶ Nasıl sorusunun cevabı “sequence” veya “collaboration” diyagramlarında tanımlanır
- Sınıf diyagramları geliştirme boyunca kullanılır
- Sınıf diyagramı, sistem için tanımlanan tüm sınıfları içermeyebilir
 - ▶ Sistem modeli farklı sınıf diyagramlarından oluşabilir
 - ▶ Her sınıf diyagramının bir görevi vardır
 - ◆ “Collaboration”, “generalization”, vb.

“Association” (1)



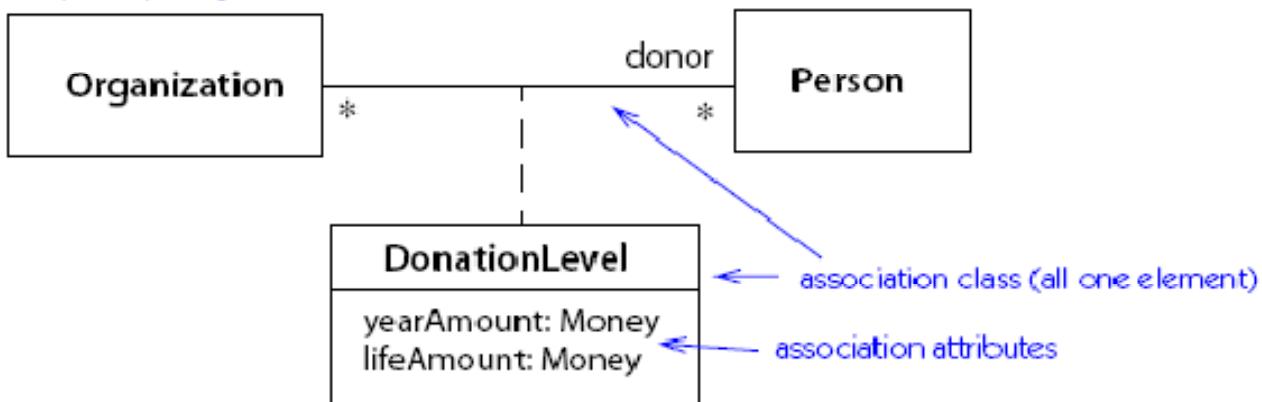
“design properties of an association”

“association”



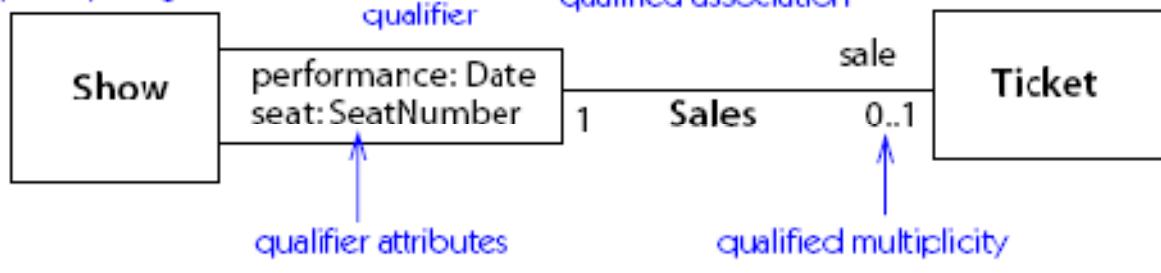
“Association” (2)

participating class



“association class”

participating class

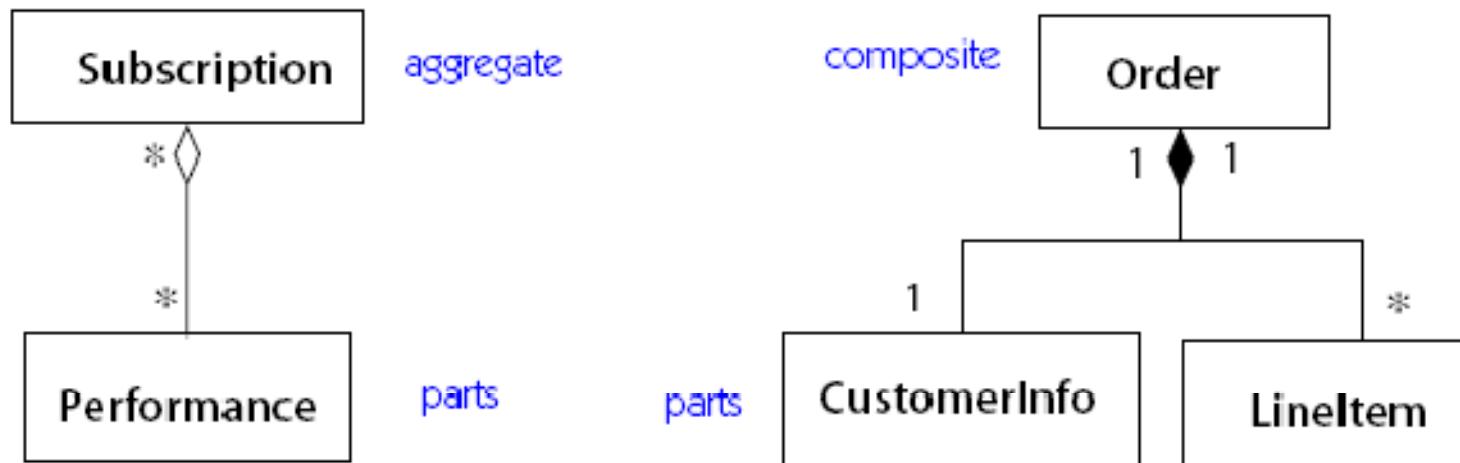


“qualified association”

A **qualifier** is a value that selects a unique object from the set of related objects across an association. (e.g., lookup tables and arrays)

“Aggregation” ve “Composition”

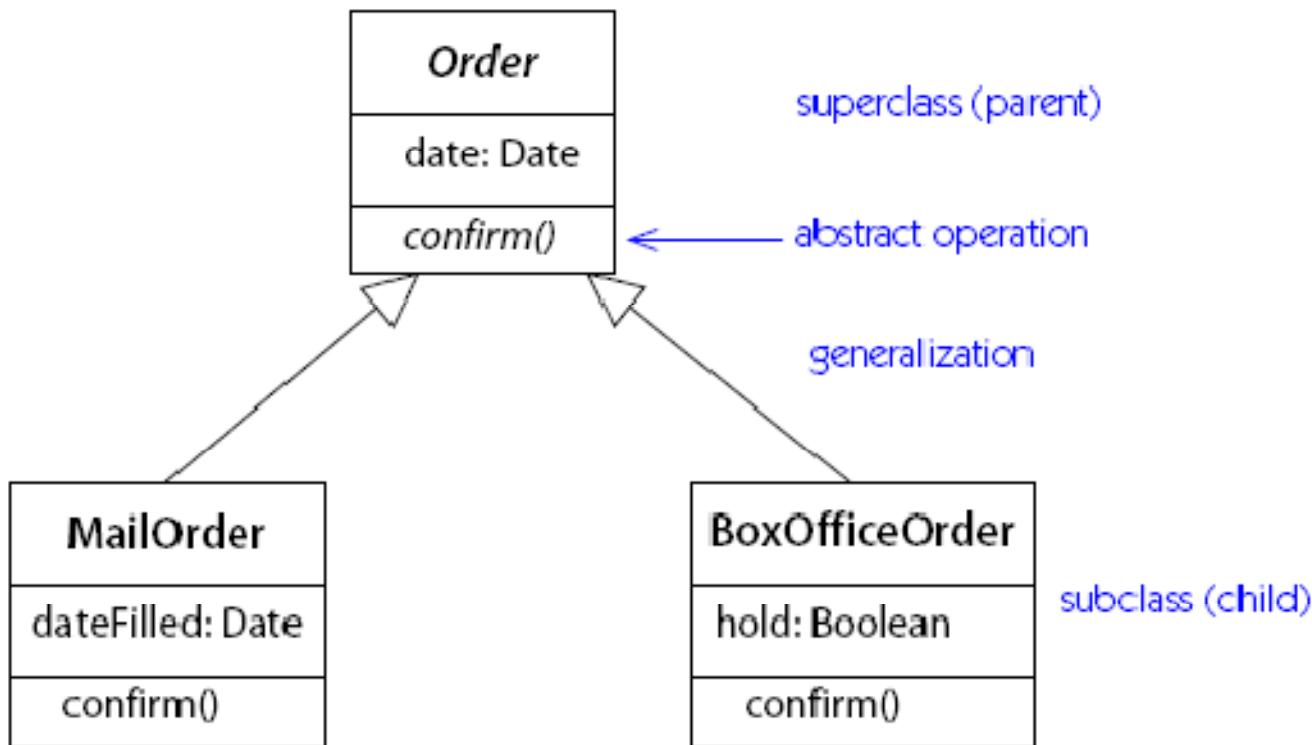
■ Bütün-parça ilişkisi



“An **aggregation** is an association that represents a part-whole relationship.”

“A **composition** is a stronger form of association in which the composite has sole responsibility for managing its parts—such as their allocation and deallocation.”

“Generalization”



Adım-3. Davranışı Modelle

■ Davranışları iki şekilde tanımlanabilir:

- ▶ Ardıl-islem (“sequence”) diyagramları ile
 - ◆ Nesneler arası iletişimin zamana göre sıralaması
- ▶ İşbirliği (“collaboration”) diyagramları ile
 - ◆ Nesnelerin etkileşimde aldığı roller ve ilişkiler
 - ◆ Nesneler arası iletişimin etkileşime göre sıralaması

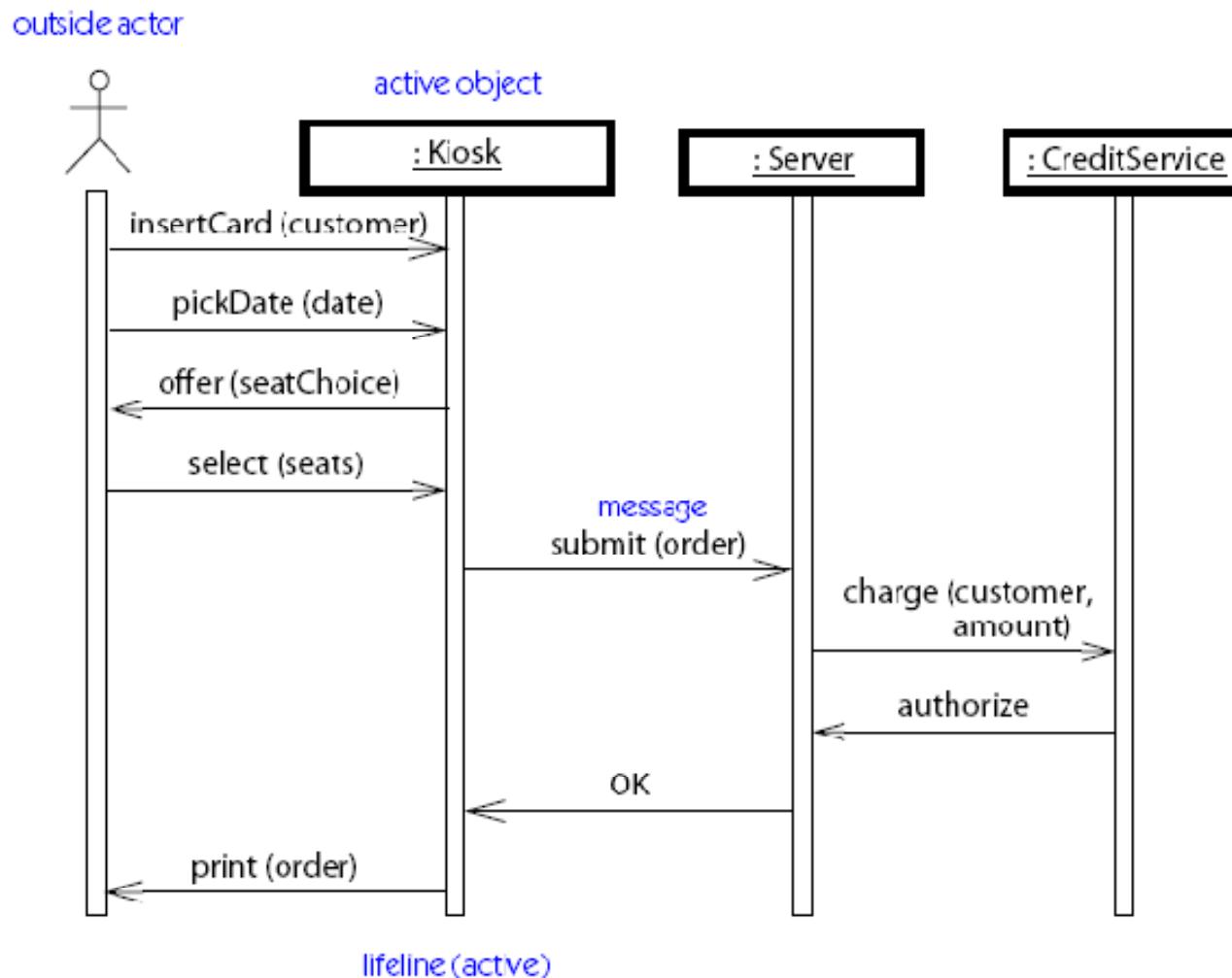
■ Önerilen yöntem:

- ▶ Bir “use case” veya senaryo seçilir
- ▶ Hangi nesnelerin rol alacağı belirlenir
- ▶ Bir nesneden diğerine gönderilecek mesajlar zamana veya etkileşime göre sıralanarak gösterilir

Ardıl-İşlem (“Sequence”) Diyagramı

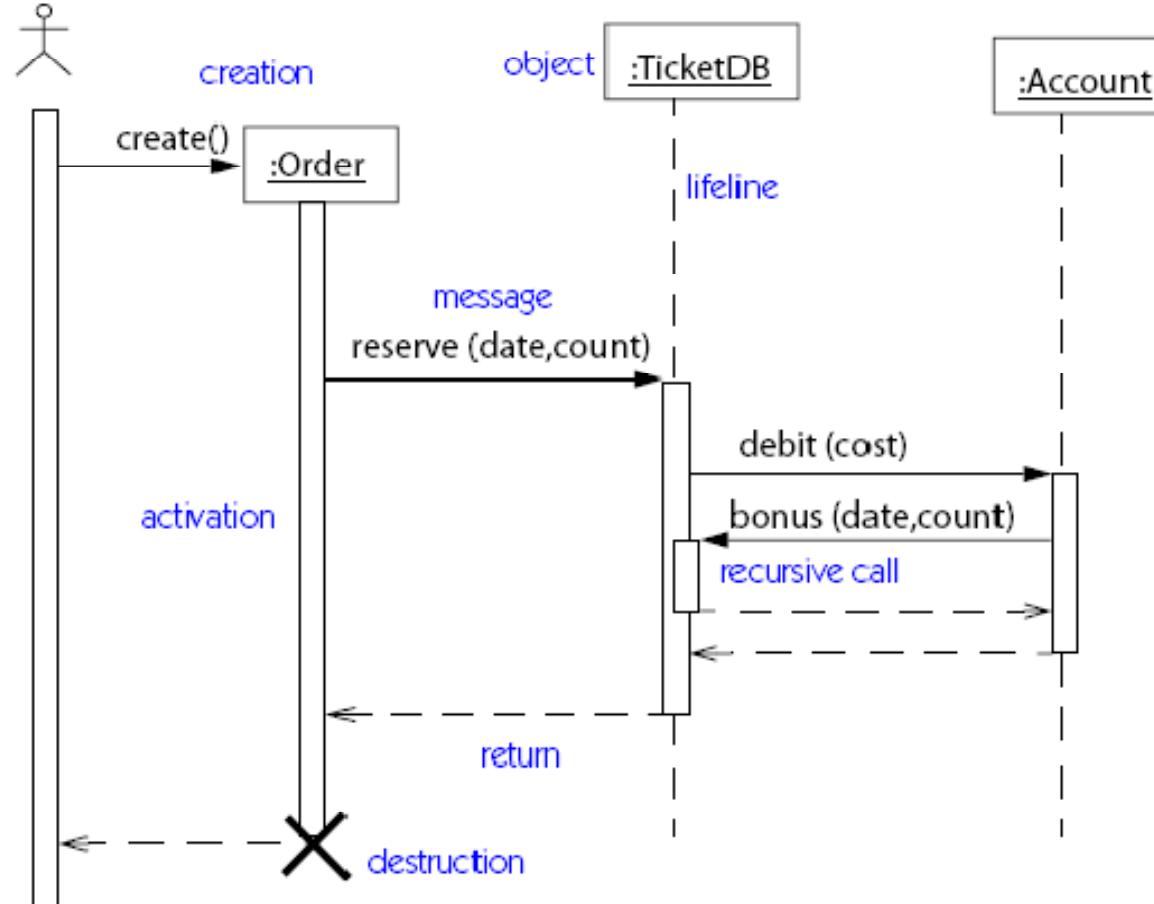
- Nesneler arasındaki ilişkileri zaman sırasına göre düzenlenmiş olarak göstermeye yarar
- Bir etkileşimde yer alan nesneleri ve çağrılan mesajların sırasını gösterir
- İşbirliği diyagramı ile benzer bilgileri farklı biçimde gösterir
- İşbirliği diyagramında bulunan nesne ilişkileri ardıl-işlem diyagramında açıkça görülmez

“Interaction View” “Sequence Diagram”: Örnek



“Activation”

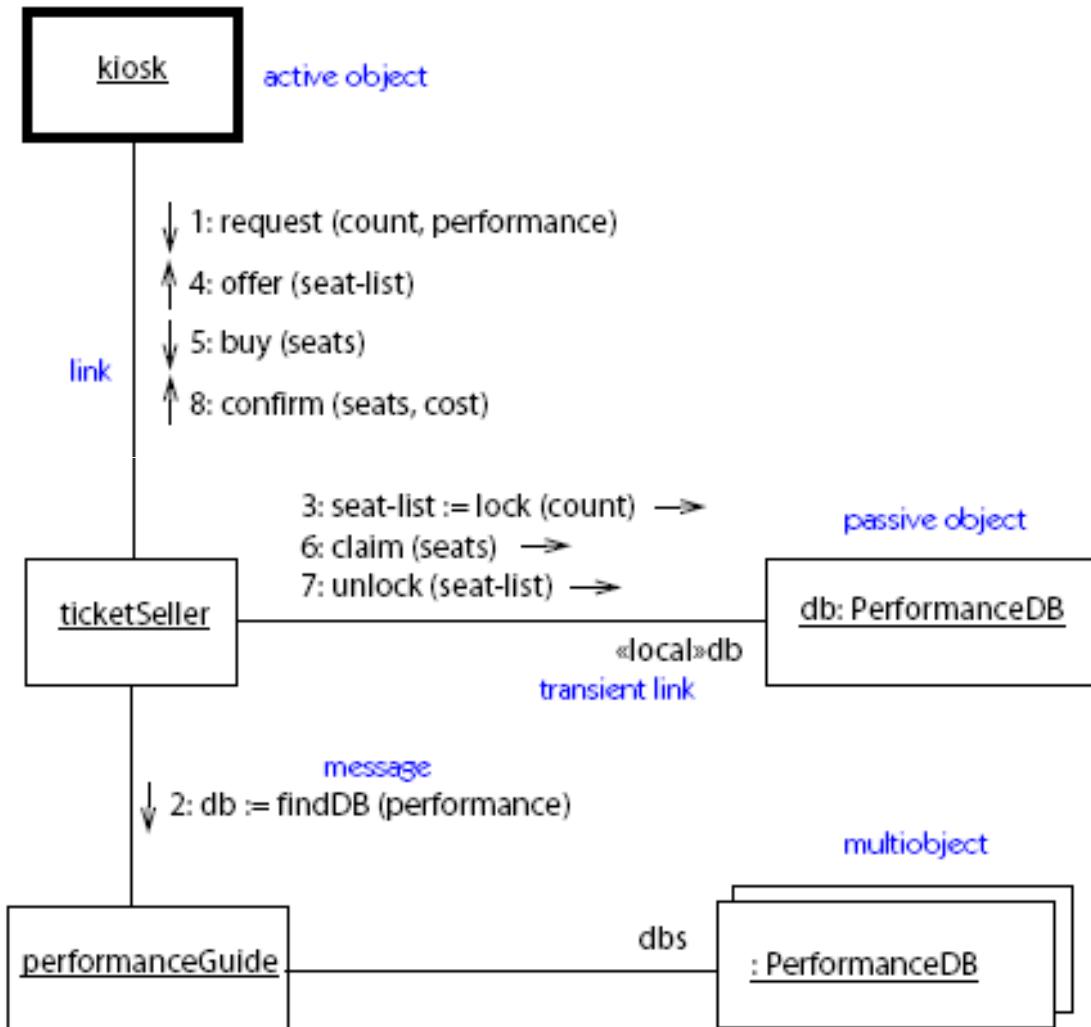
an anonymous caller



İşbirliği (“Collaboration”) Diyagramı

- İşbirliği ve ardıl-işlem diyagramları nesneleri ve aralarındaki mesajlaşmayı dinamik olarak modellemeye yarar
- İşbirliği diyagramı, ardıl-işlem diyagramından farklı olarak, sınıflar arasındaki ilişkileri doğrulamada yardımcı olur
 - ▶ Fazla ya da eksik ilişkileri bulabilirsiniz

“Interaction View” “Collaboration Diagram”: Örnek



Örnek Çözümleme: Kütüphane Destek Sistemi

- Kütüphane işlemlerini desteklemek amacıyla bir yazılım sistemi oluşturulacaktır.
- Sistem; kayıtlı müşterilere, yine kayıtlı kitap ve dergileri ödünç verecektir.
- Kütüphane, yeni başlıklı kitap ve dergilerin satın almasını yapacaktır. Popüler başlıklar, birden çok kopya satın alınacaktır. Eski kitap ve dergiler, zaman aşımına uğradıklarında veya çok yıprandıklarında yok edilecektir.
- Kütüphanede, müşterilerle iletişimini sağlayacak ve yaptığı işler sistem tarafından desteklenecek bir kütüphane görevlisi bulunacaktır.
- Müşteri, kütüphanede o anda bulunmayan bir kitap veya dergiyi rezerve edebilecektir. Kitap veya dergi kütüphaneye geri döndürüldüğünde, rezervasyonu yapan müşteri haberdar edilecektir. Rezervasyon, müşteri kitap veya dergiyi ödünç aldığında veya müşterinin özel isteği üzerine iptal edilecektir.
- Sistem; kitap ve dergi başlıklarının, kitap ve dergi kopyalarının, müşterilerin, ödünç işlemlerinin ve rezervasyonların kaydedilmesine, güncellenmesine ve silinmesine olanak sağlayacaktır.
- Sistem tüm popüler bilgisayar ortamlarında (UNIX, Windows, OS/2, vb.) çalışacak ve modern bir kullanıcı ara yüzüne sahip olacaktır.
- Sistem yeni işlevler eklemek suretiyle genişletilebilir olacaktır.

Örnek Çözümleme: KDS

Detaylı Tasarım

Detaylı Tasarım

- Amaç: Tasarım modellerini, gerçekleştirmeye olanak sağlayacak şekilde detaylandırmak
 - ▶ Odak gerçekleştirmeye detaylarına kayar
 - ▶ Üst düzeyde oluşturulan UML diyagramları somut kararlar alındıkça zenginleştirilir
 - ▶ Sistem mimarisini ve detay davranışını modellemek için farklı UML diyagramları oluşturulur

Ele Alınacak Hususlar

- Gereksinimlerde tanımlanan işi gerçekleştirmek için gereken diğer sınıflar
- Gerçekleştirmede kullanılacak algoritmalar ve veri yapıları
- Kullanılacak tasarım örüntüleri
- Kullanılacak hazır parçalar (“component”)
- Veri saklama yöntemleri (ilişkisel veri tabanı, vb.)
- Uygulamanın üzerinde çalışacağı platform
 - ▶ Ortam dağıtık olacak mı ?
 - ▶ Uygulama nasıl parçalanacak ?
- Performans kısıtları
- ...

Detaylı Tasarım Adımları

1. Sınıf yapısını detaylandırır

- ▶ Amaç: Sınıf diyagramlarını, gerçekleştirmeye yönelik zenginleştirmek
 - ◆ Sınıf tanımlarını ve ilişkileri detaylandırmak ve/veya kısıtları belirlemek
 - Tasarım sınıflarını ve aralarındaki ilişkileri belirlemek
 - Tasarım sınıflarının diğer sınıflarla ilişkilerini belirlemek

2. Davranış modelini detaylandırır

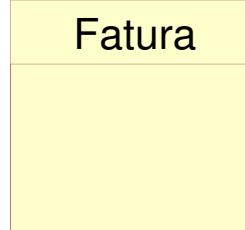
- ▶ Amaç: Sınıfların ve sistemin davranışını daha detaylı olarak modellemek
 - ◆ Nesneler arası işbirliği kapsamında
 - Ardıl-işlem (“sequence”) diyagramı; İşbirliği (“collaboration”) diyagramı
 - ◆ Bir nesne kapsamında
 - Durum (“statechart”) diyagramı; Etkinlik (“activity”) diyagramı

3. Mimariyi modelle

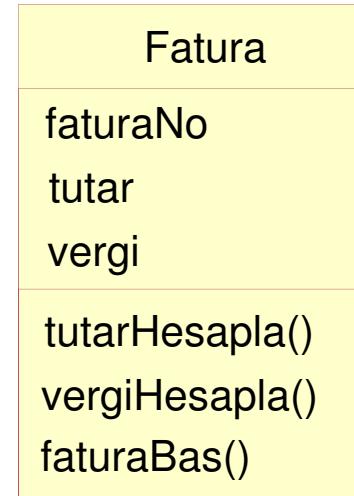
- ▶ Amaç: Yazılım sistemini parçaları (“components”) cinsinden tanımlamak
 - ◆ Diyagramları ve model varlıklarını, paket (“package”) yapısı altında gruplandırmak

(Adım-1. Sınıf Yapısını Detaylandır)

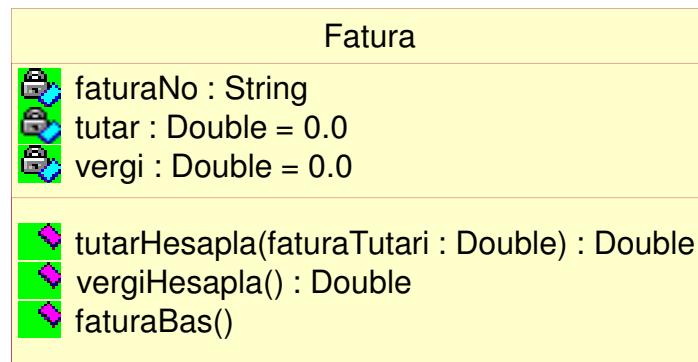
Tasarım Boyunca Sınıf Tanımının Gelişimi: Örnek



Üst Düzey Tasarım
(ön aşamalar)

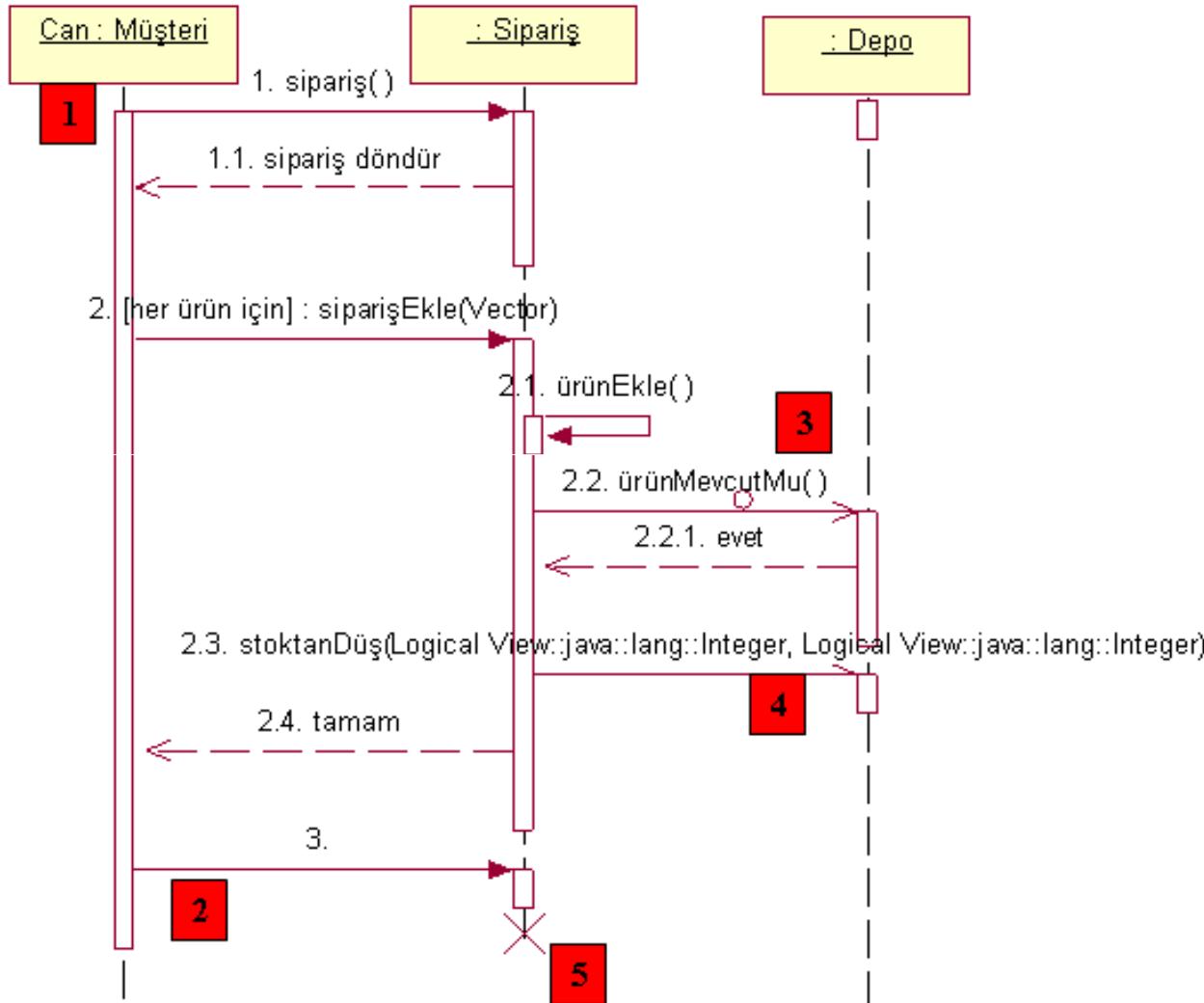


Üst Düzey Tasarım
(ileri aşamalar)

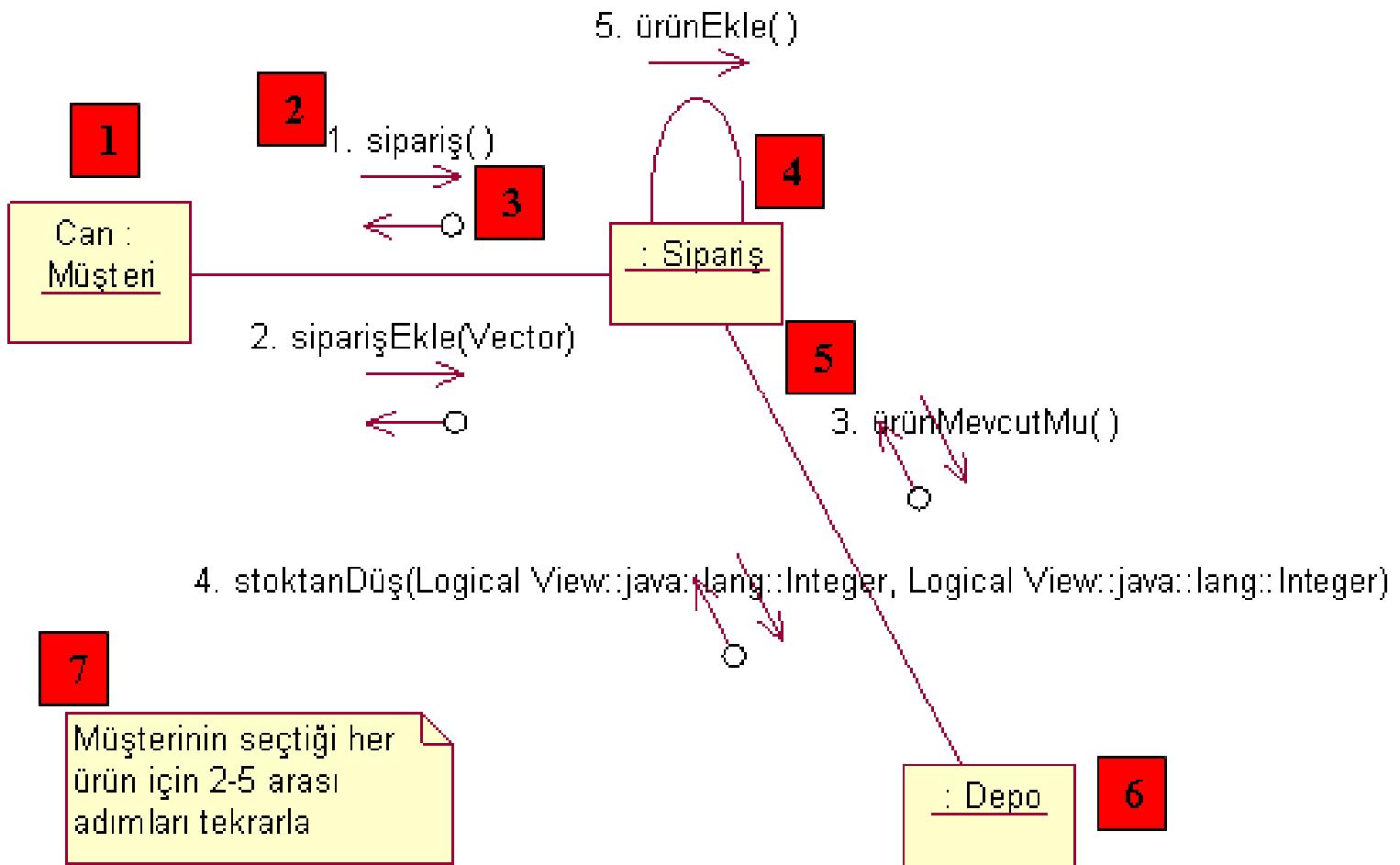


Detay Tasarım

(Adım-2. Davranış modelini detaylandır) Ardıl-İşlem Diyagramı: Örnek



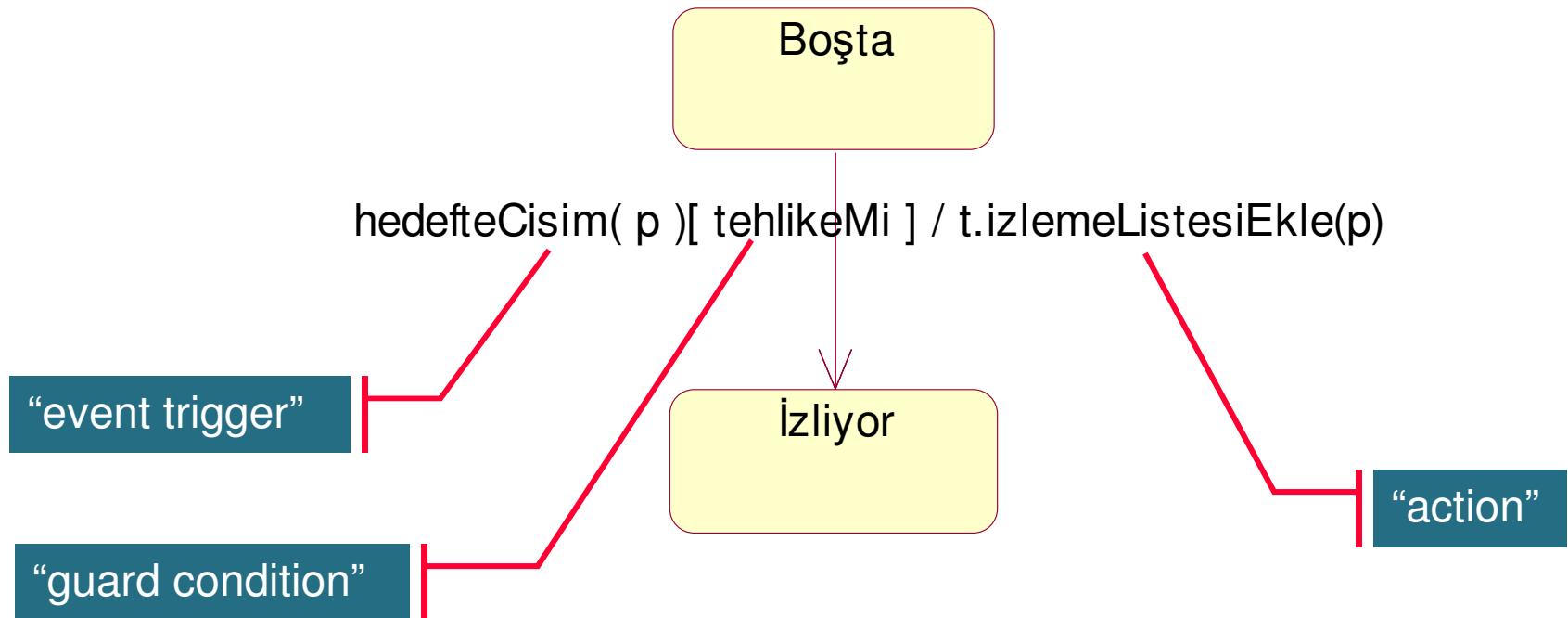
(Adım-2. Davranış modelini detaylandır) İşbirliği Diyagramı: Örnek



(Adım-2. Davranış modelini detaylandır) Durum (“Statechart”) Diyagramı

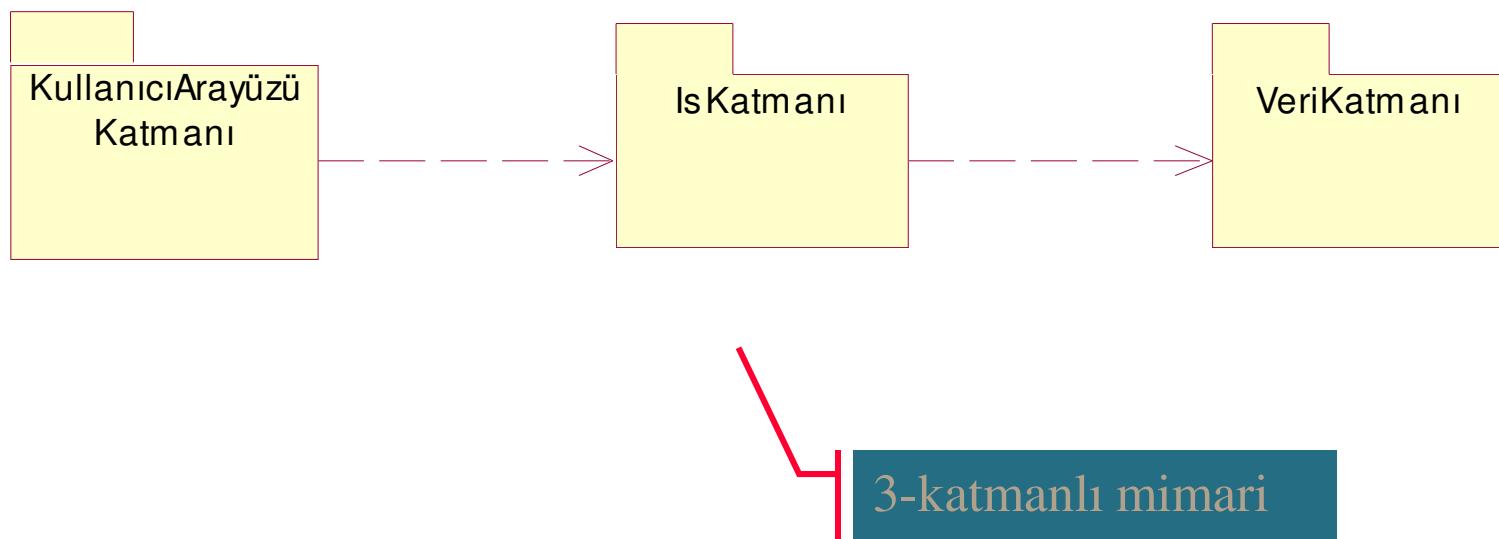
- Nesnelerin dinamik davranışlarını tanımlar
- Davranışı karmaşık olan nesneleri modellemek için kullanılır
- Bir nesnenin farklı durumları (“state”) arasındaki geçişleri modeller
- Özellikle reaktif nesnelerin davranışlarının modellenmesinde etkilidir
 - ▶ Asenkron olaylara göre davranış gösteren nesneler
- Sistemin genel davranışının modellemesi için de kullanılabilir

(Adım-2. Davranış modelini detaylandır) Durum Diyagramı: Örnek



(Adım-3. Mimariyi Modelle) Kavramsal Mimari

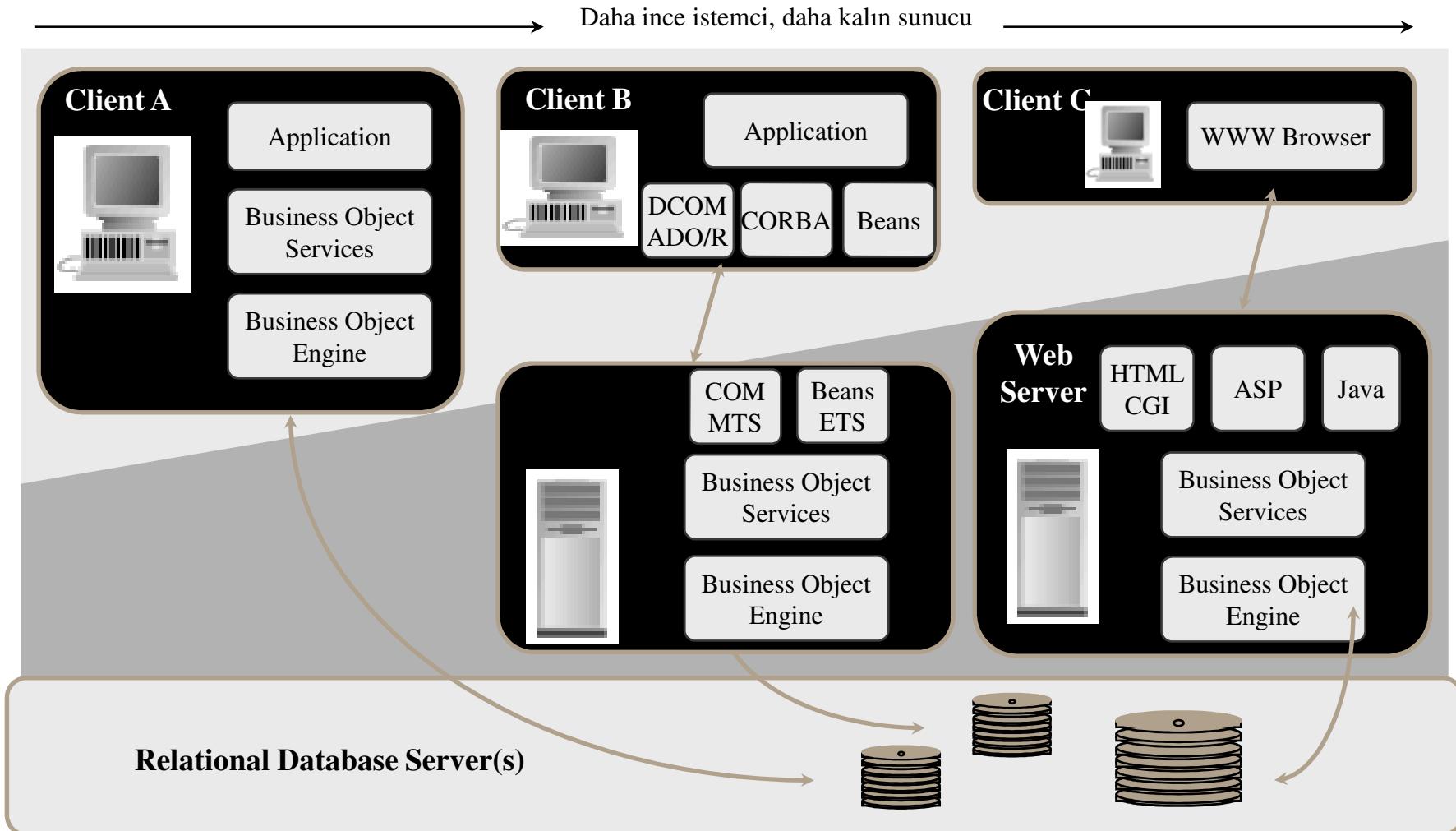
- Uygulamanın fiziksel platformdan bağımsız, farklı katmanlar halinde tasarılanması
 - ▶ Tek bir platformda çalışacak bile olsa farklı katmanlardan oluşmasında fayda var



(Adım-3. Mimariyi Modelle) Fiziksel Mimari

- Uygulamanın fiziksel olarak nasıl parçalanacağını tanımlar
 - ▶ Kavramsal mimari ile aynı olabilir
 - ▶ Kavramsal mimariden farklı olabilir
- Örnekler:
 - ▶ 3-katmanlı mimari
 - ◆ Kullanıcı arayüzü: Client
 - ◆ İş katmanı: Uygulama sunucusu
 - ◆ Veri katmanı: Veri sunucusu
 - ▶ 2-katmanlı mimari
 - ◆ Kullanıcı arayüzü: Client
 - ◆ İş ve Veri katmanları: Uygulama sunucusu

(Adım-3. Mimariyi Modelle) Fiziksel Mimari: Örnek



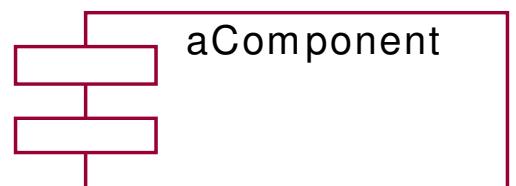
Bileşen (“Component”) Diyagramı

■ Temel kullanımı:

- ▶ Fiziksel yazılım bileşenlerini ve aralarındaki ilişkileri modellemek
- ▶ Kaynak kodlar ve dosyalar arasındaki ilişkileri modellemek
- ▶ Yazılım sürümlerinin yapısını modellemek
- ▶ Çalıştırılabilir kod oluşturan kaynak dosyalar arasındaki ilişkileri modellemek

■ UML’de bileşen (“component”):

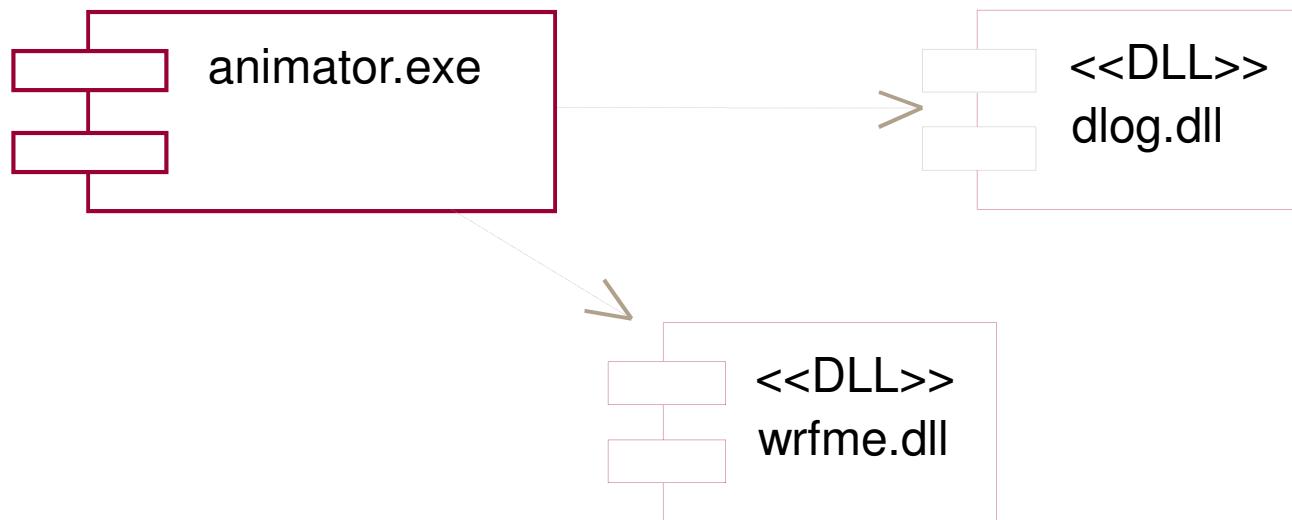
- ▶ Bileşen, işleyen sistemi oluşturan parçalardan biridir
 - ◆ İşleyen kod parçası, kütüphane, tablo, dosya, vb.
- ▶ Sistemin fiziksel ve değiştirilebilen bir parçasıdır
 - ◆ Aynı arayüze sahip farklı parçalar ile değiştirilebilir
- ▶ Bir veya birden fazla sınıf veya paket içerebilir



Bileşen Diyagramı: Örnek

■ Gösterim:

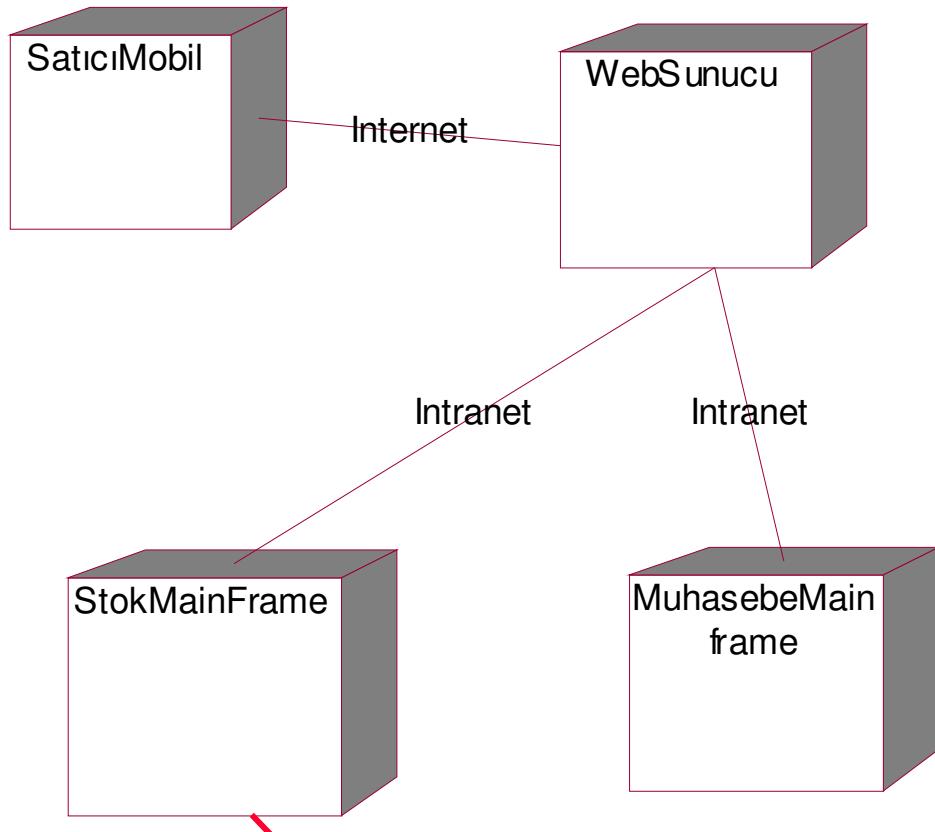
- ▶ Bileşenler ve aralarındaki ilişkileri gösteren oklar
- ▶ Okun yönü bağımlı bileşenden, bağımlı olduğu bileşene doğru
- ▶ İlişki okları “stereotype” kullanarak isimlendirilebilir
 - ◆ Örnek: “include”, “derive”, “friend”, “import”, vb.



Yayılma (“Deployment”) Diyagramı

- Bir yazılım sisteminde yer alan donanım öğelerini ve aralarındaki ilişkileri modeller
- Yayılma diyagramında iletişim ilişkisi ile birleştirilmiş “node”lar yer alır
 - ▶ “Node”lar sistemdeki işlemci kaynakları temsil eder ve küp şeklinde gösterilir
 - ◆ Bilgisayarlar, alıcılar, çevresel veya gömülü sistemler, vb.
 - ▶ İletişim ilişkileri “stereotype” kullanılarak iletişimin tipini gösterebilir

Yayıılma Diyagramı: Örnek



Fiziksel ortamda, uygulamanın bir parçasının üzerinde çalışacağı bir işlemciyi temsil eder



BM306

Yazılım Mühendisliği

DERS 6

Yazılım Mühendisliğinde Ölçme



Dr. Öğr. Üyesi Hasan BADEM
hbadem@ksu.edu.tr



■ Yazılım mühendisliğinde ölçme

- ▶ Ölçme teorisi ve ölçekler
- ▶ Objektif ve sübjektif ölçevler
- ▶ Temel ve türetilmiş ölçevler

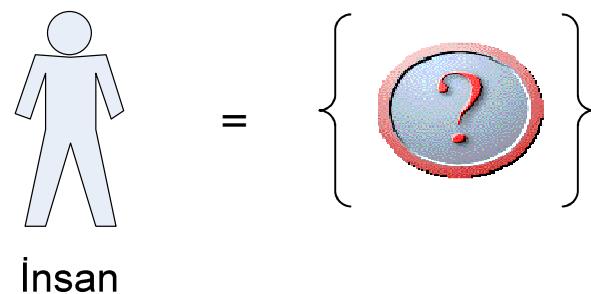
■ Yazılım ürün ölçevleri

■ COSMIC İşlevsel Büyüklük Ölçme Yöntemi

Ölçme Nedir?

- Rakamları ve sembollerini tanımlı kurallara göre gerçek dünyadaki varlıkların özelliklerine atama işidir
 - ▶ Özellikleri nicel olarak tanımlamak amacıyla

$$\text{VARLIK} = \left\{ \begin{array}{l} \text{özellik}_1 (\text{değer}_{11}, \text{değer}_{12}, \dots) \\ \text{özellik}_2 (\text{değer}_{21}, \text{değer}_{22}, \dots) \\ \dots \\ \text{özellik}_n (\text{değer}_{n1}, \text{değer}_{n2}, \dots) \end{array} \right\}$$



A diagram illustrating measurement. On the left is a simple stick figure icon. To its right is an equals sign (=). To the right of the equals sign is a large curly brace enclosing a red circle. Inside the red circle is a large red question mark. Below the entire diagram is the word "İnsan".

İnsan

Neden Ölçeriz?

*“When you can measure what you are speaking about, and express it in numbers,
you know something about it;
but when you cannot measure it, when you cannot express it in numbers, your
knowledge is of meager and unsatisfactory kind.”*

-- *Lord Kelvin, 1824-1907*

“You cannot control what you cannot measure.”

-- *DeMarco, 1982*

Ölçmediğimiz şeyleri kontrol edemeyiz!

Diger bir deyişle;

Kontrol etmek istediğimiz şeyleri ölçmek zorundayız!

Neleri Ölçeriz?

“What is not measurable makes measurable.”

-- Galileo Galilei (1564-1642)

- Geçmişte ölçülemeyeceğini düşündüğümüz varlıkların özelliklerini bugün kolaylıkla ölçübiliyoruz
 - ▶ Zaman, sıcaklık, hız, zeka, enflasyon, vb.
- İlgi duyduğumuz ve anlamak istediğimiz şeylerin özelliklerini ölçme yollarını araştırırız

Ölçme Birimi

- Varlıkların özelliklerini sıkılıkla, gerçek dünyayı algılayışımızı yansıtan rakam ya da semboller kullanarak tanımlarız
 - ▶ Örnek: YTL olarak fiyat, santimetre olarak boy, beden olarak giysilerimiz
- Ölçme Birimi
 - ▶ Bir niceliği ölçmek için kendi cinsinden örnek seçilen değişmez parça
 - ▶ “particular quantity, defined and adopted by convention, with which other quantities of the same kind are compared in order to express their magnitude relative to that quantity” [ISO 2002]

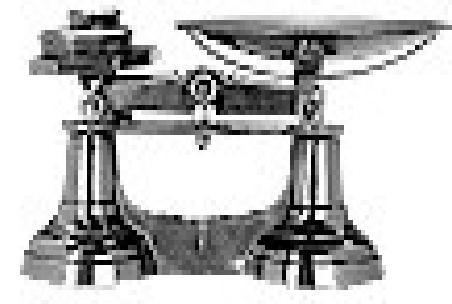


Ölçme ve Kestirme

Ölçme (“measurement”):

- Var olan bir varlığın bir özelliğine değer biçmek

- ▶ Örnek: Yazılım kodlandıktan sonra kodun uzunluğunu (satır sayısını) bulmak



Kestirme (“estimation”):

- Henüz var olmayan bir varlığın bir özelliğini öngörmek

- ▶ Örnek: Proje başlangıcında bir yazılım projesinin ne kadar süreceğini, maliyetini, vb. tahmin edebilmek



Objektif ve Sübjektif Ölçüler

■ Objektif ölçever

- ▶ Ölçme sonucunda farklı kişiler aynı değeri elde eder
 - ◆ Örnek: Metre ile bir duvarın yüksekliğini ölçmek
- ▶ Nicel kontrolde daha etkinler

■ Sübjektif ölçevler

- ▶ Ölçme sonucunda elde edilen değer, ölçmenin yapıldığı ortama göre ve kişiden kişiye değişir
 - ◆ Örnek: Eğitim etkinliğini derecelendirmek
- ▶ Kısıtlarını bildiğimiz sürece faydalananabiliriz

Ölçekler

- Gerçek dünyadan biçimsel dünyaya geçerken yaptığımız ölçmeler arasında farklılıklar vardır

- ▶ Örnek:

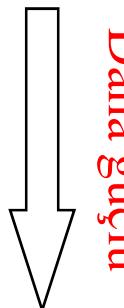
- ◆ İnsanın yaşı tamsayılarla ölçülür
 - ◆ Baş ağrısı derecelendirilerek ölçülür
 - ◆ Müzik türleri sınıflandırılır

- Ölçmede bu farklılıklar ölçeklerle ifade edilir

- ▶ Ölçekler bize ölçümler üzerinde ne tür analizler yapabileceğimizi söyler

- Temel ölçekler:

- ▶ Sembolik (“nominal”)
 - ▶ Dereceli (“ordinal”)
 - ▶ Aralıklı (“interval”)
 - ▶ Oransal (“ratio”)



Daha güçlü

Daha güçlü ölçekler gerçek dünyadaki varlıklar ve ilişkileri hakkında daha fazla bilgi sağlar

Temel ve Türetilmiş Ölçüler

- Bir varlığın özellik ölçüvi, başka bir varlığı veya özelliği içermiyorsa (varlığın özelliğini direkt ölçuyorsa), temel ölçüvdür
 - ▶ Örnek: Duvarın yüksekliği
- Bir ölçev, diğer ölçevler üzerinde yapılan işlemler sonucunda meydana geliyorsa, türetilmiş ölçüvdür
 - ▶ Örnek: Sıvının yoğunluğu (kütle/hacim), arabanın hareket hızı (uzaklık/zaman)

Temel ölçevler ölçmenin yapı taşılarıdır.

Ancak pek çok yararlı ve etkin ölçev, türetilmiş ölçevler arasından çıkar!

■ Temel ölçevler

- ▶ Kod uzunluğu (“satır sayısı” ile ölçülür)
- ▶ Test süresi (“saat” zaman birimi ile ölçülür)
- ▶ Test hata sayısı (test boyunca tespit edilen hatalar sayılarak ölçülür)

■ Türetilmiş ölçevler

- ▶ Programcı üretkenliği (kod uzunluğu / kodlama işgücü)
- ▶ Modül hata yoğunluğu (modül hata sayısı / modül kod uzunluğu)
- ▶ Gereksinim değişiklikleri (değişen gereksinimlerin sayısı / ilk gereksinimlerin sayısı)

Her ölçev, belirli bir amaca hizmet etmek için tanımlanmış olmalıdır.

Bu amaç, verilerin toplanması ve analizi için yol gösterir.

Ölçülebilecek Varlıklar

■ Süreç

- ▶ Belirli bir hedefe ulaşmak için gerçekleştirilen adımlar zinciri
 - ◆ Örnek: Yazılım gereksinim analizi

■ Ürün

- ▶ Bir süreç etkinliği ile üretilen çıktı
 - ◆ Örnek: Yazılım Gereksinimleri Tanımı Belgesi

■ Kaynak

- ▶ Bir süreç etkinliği tarafından kullanılan diğer bir varlık
 - ◆ Örnek: İnsan kaynağı

Yazılımın Büyüklüğü (Ürün, İç Özellik)

4 boyutta ölçülebilir:

■ **Uzunluk:** ürünün fiziksel büyülüğu

- ▶ Doküman:
 - ◆ Sayfa sayısı, karakter sayısı, tablo sayısı, şekil sayısı, vs.
- ▶ Kod:
 - ◆ Satır sayısı (mantıksal veya fiziksel)
 - Sayma tekniği sebebiyle 5 kata kadar farklılık gösterebilir (Capers Jones, 1986)
 - ◆ Halstead uzunluk ölçümü
 - Uzunluk : $N = N_1 + N_2$ (N_1 : toplam operatör sayısı, N_2 : toplam değişken sayısı)

■ **İşlevsellilik:** yazılımın kullanıcıya sunduğu işlevselliğin miktarı

- ▶ İşlev puan (“function points”), özellik puan (“feature points”), vs.

■ **Karmaşıklık:** yazılımın çözdüğü konunun karakteristiği

- ▶ Problem karmaşıklığı, algoritmik karmaşıklık (big-O notasyonu), yapısal karmaşıklık, kavramsal karmaşıklık

■ **Tekrar kullanım:** yazılımın ne kadarı yeni geliştirildi?

- ▶ Örnek: yüzde (%) olarak

Kod Uzunluğu

- Çoku kez; yazılımın kalitesi, geliştirme işgücü ve maliyeti tahmini için temel girdi
- Sayma tekniğine göre kod satır sayısı (KSS), 5 kata kadar farklılık gösterebiliyor (Jones, 1986)
 - ▶ Örnek:

If A>B
then A - B
else A + B;

If A>B
then
begin
 A - B
end
else
begin
 A + B
end;

Fiziksel sayma:
Sol : 3 KSS
Sağ: 9 KSS

Mantıksal sayma:
Sol ve sağ: 1 KSS

Anahtar sözcükleri sayma
(if, then, else, begin, ...):
Sol : 3 KSS
Sağ: 7 KSS

Örnek: Kişisel Yazılım Süreci (“Personal Software Process”)

KSS Sayma Şablonu

Count Type	Type	Comments
Physical/Logical		
Statement Type	Included	Comments
Executable		
Nonexecutable:		
Declarations		
Compiler Directives		
Comments		
On own lines		
With source		
Banners		
Blank lines		
Clarifications		Examples/Cases
Nulls		continues, no-ops, ...
Empty statements		“;;”, lone ;’s, etc.
Generic instantiators		
Begin...end		when executable
Begin...end		when not executable
Test conditions		
Expression evaluation		when used as sub program arguments
End symbols		when terminating executable statements
End symbols		when terminating declarations or bodies
Then, else, otherwise		
Elseif		
Keywords		procedure division, interface, implementation
Labels		branch destinations when on separate lines

“Kod Uzunluğu” Ölçevleri

- Kaynak Kod Satır Sayısı (“Source Lines of Code”)
- Yorumsuz Kod Satır Sayısı (“Un-commented Lines of Code”)
- Yorum Kod Satır Sayısı (“Commented Lines of Code”)
- Yürütülür Deyim (“Executable Statements”)
- Program kaynak kodunun bilgisayarda tutacağı bayt miktarı
- Program kaynak kodunun içerisindeki karakter sayısı
-

Kod Uzunluğu

■ Avantajlar

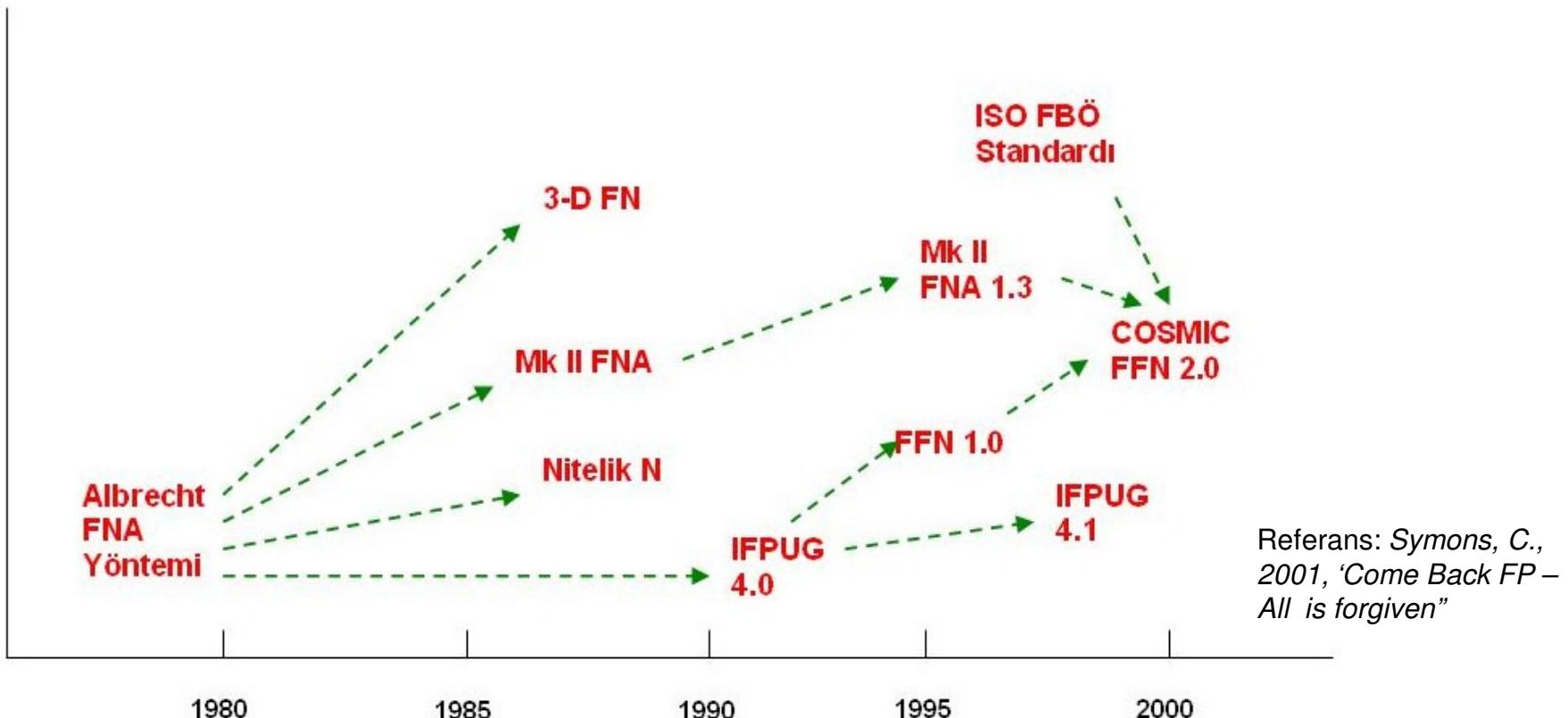
- ▶ Otomatik olarak kolaylıkla sayılabilir
- ▶ Direkt son ürünle ilgili

■ Zorluklar

- ▶ Belirsiz tanım
- ▶ Dil bağımlı
- ▶ Programcı bağımlı
- ▶ Erken planlama için kullanılamıyor
- ▶ Karmaşıklık gibi faktörleri adreslemiyor
 - ◆ Aynı uzunluktaki 2 kod parçasının genellikle aynı karmaşıklıkta olduğu varsayılıyor

İşlevsellik

- İşlevsel büyülüük, özellikle erken aşamalarda işgücü ve takvim tahminleme için tercih ediliyor
- “İşlevsellik” ölçüsüne dayanan yöntemler:



ISO Onaylı İşlevsel Büyüklük Ölçme (İBÖ) Yöntemleri

■ IFPUG FPA (“Function Points Analysis”) Yöntemi

- ▶ ISO/IEC 20926

■ Mark II FPA Yöntemi

- ▶ ISO/IEC 20968

■ COSMIC FFP (“Full Function Points”) Yöntemi

- ▶ ISO/IEC 19761

■ NESMA FPA Yöntemi

- ▶ ISO/IEC 24570

Yazılımın Yapısı (Ürün, İç Özellik)

3 tür altında incelenebilir:

■ Kontrol-akış

- ▶ Programdaki komutların işletiliş şekli
 - ◆ Örnek: sıra (“sequence”), seçim (“selection”), döngü (“iteration”)

■ Veri-akış

- ▶ Programla etkileşim halindeki verinin davranışı

■ Veri-yapı

- ▶ Programdan bağımsız olarak, veri yapıları
 - ◆ Örnek: “liste”, “yığın”, “kuyruk” kullanımı

Kontrol-AKİŞ Ölçevleri: Örnek – Siklomatik Karmaşıklık

(McCabe's Cyclomatic Complexity)

■ Siklomatik karmaşıklık: $v(F)$

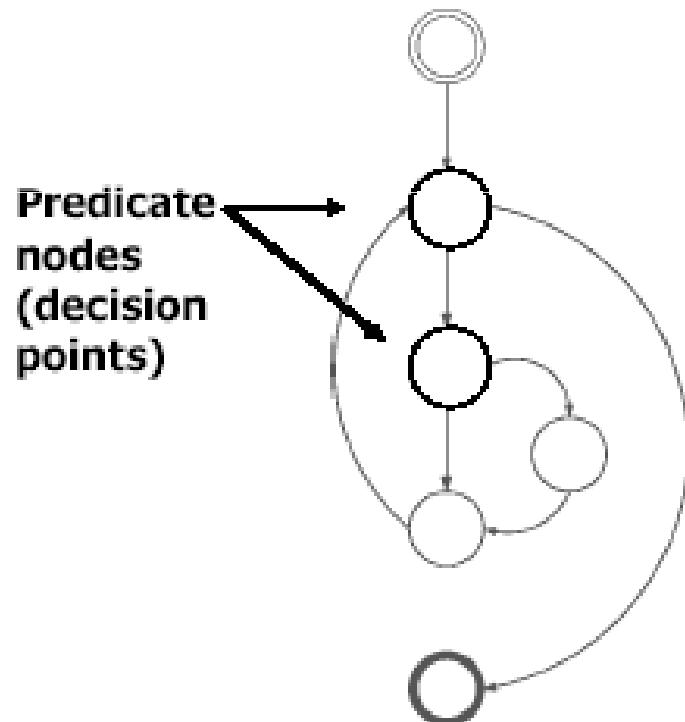
- ▶ $v(F) = e - n + 2$ veya $v(F) = 1 + d$
 - ◆ “n: node” (düğüm)
 - ◆ “e: edge” (geçiş)

■ Örnek:

- ▶ $v(G) = e - n + 2$
- ▶ $v(G) = 7 - 6 + 2$
- ▶ $v(G) = 3$

veya

- ▶ $v(G) = 1 + d$
- ▶ $v(G) = 1 + 2 = 3$



Yazılımın Kalitesi (Ürün, Dış Özellik)

■ Hata-esaslı ölçevler

- ▶ Hata yoğunluğu (“defect density”) = Bilinen hatalar / ürün büyüklüğü
- ▶ Sistem hasarı (“system spoilage”) = Müşteriye teslim sonrası bildirilen hataları düzeltmenin maliyeti / toplam proje maliyeti

■ Bakım-yapılabilirlik ölçevleri

- ▶ Ortalama tamir zamanı (“mean time to repair” - MTTR)

■ Güvenilirlik ölçevleri

- ▶ Ortalama bozulma zamanı (“mean time to failure” - MTTF)
- ▶ Bozulmalar arası ortalama zaman (“mean time between failures” - MTBF)
 - ◆ $MTBF = MTTF + MTTR$
- ▶ Mevcut olma zamanı (“availability” - A)
 - ◆ $A = MTTF / (MTTF + MTTR) * 100$

Yazılım İşlevsel Büyüklük Ölçme Yöntemleri

ISO Onaylı İBÖ Yöntemlerinin Ortak Özellikleri

- Yazılımın işlevsel büyülüüğünü “İşlevsel Kullanıcı Gereksinimleri”ni kullanarak hesaplarlar.
- İşlevsel Kullanıcı Gereksinimleri (İKG):
 - ▶ Kullanıcı gereksinimlerinin alt kümesidir
 - ▶ Kullanıcının ihtiyaç duyduğu ve yazılımın gerçekleştireceği kullanıcı etkinliklerini ve prosedürlerini ifade eder
 - ▶ Kalite ve teknik gereksinimleri dışında bırakır
- İşlevsel büyülüüğü ölçülecek yazılımın:
 - ▶ Geliştirilmesi / desteklenmesi için gereken yöntemlerden bağımsızdır
 - ▶ Fiziksel ve teknolojik bileşenlerinden bağımsızdır

IFPUG İşlev Puan Analiz Yöntemi

- Albrecht'in işlev puan analiz yöntemine dayanır [Albrecht, 1983].
- Genel tanımı verilen bir sistemin içerdiği işlevselligi tanımlar.
- İki adımda hesaplanır:
 - ▶ Ham işlev puan değerini hesaplama (HİP)
 - ▶ Teknik karmaşıklık faktörünü hesaplama (TKF)
 - ▶ Nihai işlev puan değerini hesaplama (İP)
$$\text{İP} = \text{HİP} * \text{TKF}$$

Albrecht İşlev Puan Analiz Yöntemi (1)

■ Ham işlev puan değeri, aşağıdaki bileşenler cinsinden hesaplanır

- ▶ Sistem girdilerinin sayısı (N_i)
 - ◆ Örnek: dosya isimleri, menü seçimleri, vs.
- ▶ Sistem çıktılarının sayısı (N_o)
 - ◆ Örnek: Mesajlar, raporlar, vs.
- ▶ Sistemin yanıtladığı soru sayısı (N_q)
 - ◆ Örnek: Hizmet fonksiyonları
- ▶ Sistemin dış arayüz sayısı (N_{ef})
 - ◆ Örnek: Diğer sistemlerle etkileşim
- ▶ Sistemin eriştiği iç varlık sayısı (N_{if})
 - ◆ Örnek: Veritabanı, excel dosyası, vs.

Item	Weighting Factor		
	Simple	Average	Complex
External inputs (N_i)	3	4	6
External outputs (N_o)	4	5	7
External inquiries (N_q)	3	4	6
External interface files (N_{ef})	7	10	15
Internal files (N_{if})	5	7	10

■ Örnek: Orta karmaşıklık → $HFN = 4 N_i + 5 N_o + 4 N_q + 10 N_{ef} + 7 N_{if}$

Sayı	Bileşen Tipi	Ağırlık	Toplam
8	Girdi	8×4	32
12	Çıktı	12×5	60
4	Sorgu	4×4	16
2	Dış Arayüz	2×10	20
1	İç Varlık	1×7	7
Ham Puan			135

Albrecht İşlev Puan Analiz Yöntemi (2)

■ Teknik karmaşıklık faktörü, 14 ögenin ağırlıklı değerlerinin toplamıdır.

► Her öğe 0'dan 5'e derecelendirilir

- ◆ 0: ilişkisiz, 3: orta derecede ilişkisi, 5: yakın ilişkili

$$TCF = 0.65 + 0.01 \sum_{j=1}^{14} F_j$$

F1	Reliable back-up and recovery	F2	Data communications
F3	Distributed functions	F4	Performance
F5	Heavily used configuration	F6	Online data entry
F7	Operational ease	F8	Online update
F9	Complex interface	F10	Complex processing
F11	Reusability	F12	Installation ease
F13	Multiple sites	F14	Facilitate change

■ Karmaşıklık faktörleri için belirlenen ağırlık değerlerine göre, final işlev puan değeri %35 fark edebilir.

Albrecht İşlev Puan Analiz Yöntemi (3)

- Veri işleme uygulamaları için daha uygundur.
 - ▶ Gerçek zamanlı ve gömülü uygulamalar için Özellik Puan (“Feature Points”) yöntemi oluşturulmuştur.
- Kestirim yöntemi, Mark II ve COSMIC yöntemlerinininkine göre daha kabadır.

Mark II İşlev Puan Analiz Yöntemi (1)

- Bir bilgi sistemi içindeki bilgi işleme yoğunluğunu, mantıksal işlembilgiler (Mİ) cinsinden ölçer [1998].
 - ▶ Metrics Practices Committee (MPC), UK Software Metrics Association
 - ◆ MK II Function Point Analysis Counting Practices Manual Version 1.3.1
- Bir Mİ, yazılım uygulaması tarafından desteklenecek en alt seviyedeki iş sürecidir. Üç bileşenden oluşur:
 - ▶ Uygulamaya veri sağlayan “girdi bileşeni”
 - ▶ Uygulama içindeki işlemleri yapan “işlem bileşeni”
 - ▶ Uygulama sonucunu sunan “çıkıtı bileşeni”
- Bir uygulamanın işlevsel büyülüğu, bütün Mİ’lerinin işlevsel büyülüklüklerinin toplamıdır.
- Bir Mİ kullanıcıyla ilişkili olan tek bir olay tarafından tetiklenir ve tamamlandığı zaman uygulamayı o olaya göre kendi içinde tutarlı bir durumda bırakır.

Mantıksal İşlembilgi

■ “Müşteriye ait bilgilerin bakımını yap” bir Mİ midir? HAYIR

- ▶ En alt düzeyde değildir, daha alt düzeydeki iş süreçlerinden oluşmuştur
 - ◆ Mİ: “Bir müşteriye ait bilgileri görüntüle”
 - ◆ Mİ: “Yeni bir müşteri ve o müşteriye ait bilgileri ekle”
 - ◆ Mİ: “Bir müşterinin görüntülenen bilgilerini sil”
 - ◆ Mİ: “Bütün müşterileri listele”

■ “Bankamatikten para çekilmesi” bir Mİ midir? EVET

- ▶ Tek bir olay tarafından tetikleniyor
- ▶ Kullanıcıdan uygulama içine girdi var (müşteri kodu, para miktarı), içerisinde işlemler yapılıyor (çekilecek miktar hesaptan düşülüyor) ve kullanıcıya bir çıktı (makbuz) sunuluyor
- ▶ Tamamladığı zaman uygulamayı tutarlı bir durumda bırakıyor

■ Aşağıdakiler birer Mİ değil, çünkü iş süreci degiller

- ▶ “Çekilecek para miktarının kullanıcı arayüzünden girilmesi”
- ▶ “Hesaptan çekilecek para miktarının düşülmesi”
- ▶ “Para sayma makinesine tetikleyici mesajın gönderilmesi”

Mark II İşlev Puan Analiz Yöntemi (2)

■ Ham işlev puan değeri aşağıdaki gibi hesaplanır

$$\triangleright \text{HIP} = 0.58 * \Sigma \text{NI} + 1.66 * \Sigma \text{NE} + 0.26 * \Sigma \text{NO}$$

- ◆ NI : Girdi bileşen tipi
- ◆ NE : İşlem bileşenin (işi yapmak için) ulaştığı varlık tipi
- ◆ NO : Çıktı bileşen tipi

■ Temel işlev puan karmaşıklık faktörlerine 6 tane ekler

$$\triangleright 20 \text{ faktör } 0 \text{'dan } 5 \text{'e derecelendirilerek ağırlıklı değerler toplanır (TDI)}$$

- ◆ Ek faktörler: Interfaces with other applications, special security features, direct use by third parties, documentation requirements, special user training facilities, user defined characteristics

$$\boxed{\text{■ Teknik Karmaşıklık Faktörü: } \text{TKF} = (\text{TDI} * C) + 0.65}$$

(C'nin sektör ortalaması 0.005'tir)

$$\boxed{\text{■ Final İşlev Puan Değeri: } \text{IP} = \text{HIP} * \text{TKF}}$$

Mark II İşlev Puan Analiz Yöntemi (3)

■ Örnek:

ÜYE KAYIT EKRANI

Uye No :	<input type="text"/>	EKLE
Uye Adı / Soyadı :	<input type="text"/> <input type="text"/>	GÜNCELLE
Adres :	<input type="text"/>	SİL
Tel :	<input type="text"/>	
Faks :	<input type="text"/>	
KAPAT		

Albrecht İşlev Puan Analiz Yöntemi (3)

- Veri işleme uygulamaları için daha uygundur.
 - ▶ Gerçek zamanlı sistemler için kullanılması önerilmez.
- Sadece kullanıcı bakış açısını destekler.
 - ▶ Uygulamanın içindeki katmanları ve içteki İşlevselligi hesaba katmaz.
- Hesaplamadaki detay seviyesi sebebiyle, yazılım geliştirmenin erken aşamalarında (proje başlangıcında) kullanmak güçtür.
 - ▶ Ancak yazılım gereksinimleri analizi tamamlandıktan sonra bu detay yakalanabilir.

COSMIC İşlev Puan Analiz Yöntemi (1)

- 1999 yılında Ortak Yazılım Ölçme Uluslararası Konsorsiyum'u (Common Software Measurement International Consortium-COSMIC) tarafından geliştirilmiştir.
- Yazılım uygulamasının fonksiyonel büyüklüğünü "Fonksiyonel Kullanıcı Gereksinimleri"ni temel alarak ölçmek üzere tasarlanmıştır.
- COSMIC İşlevsel Büyüklük Ölçme Süreci
 - ▶ Eşleme ("Mapping") Aşaması - Kurallar
 - ▶ Ölçme Aşaması - Kurallar

COSMIC İşlev Puan Analiz Yöntemi (2)

■ Uygulanabilirliği:

- ▶ Veri-güçlü sistemler
 - ◆ Yönetim Bilgi Sistemleri, banka, muhasebe, personel, satın alma, vb.
- ▶ Kontrol-güçlü sistemler (gerçek-zamanlı sistemler)
 - ◆ Avionik sistemler, çeşitli makineleri kontrol etmek üzere geliştirilen yazılım gömülü cihazlar (asansörler, çamaşır makineleri, vs.)
- ▶ Melez (hibrid) sistemler
 - ◆ Gerçek zamanlı havayolu rezervasyon sistemleri, vs.

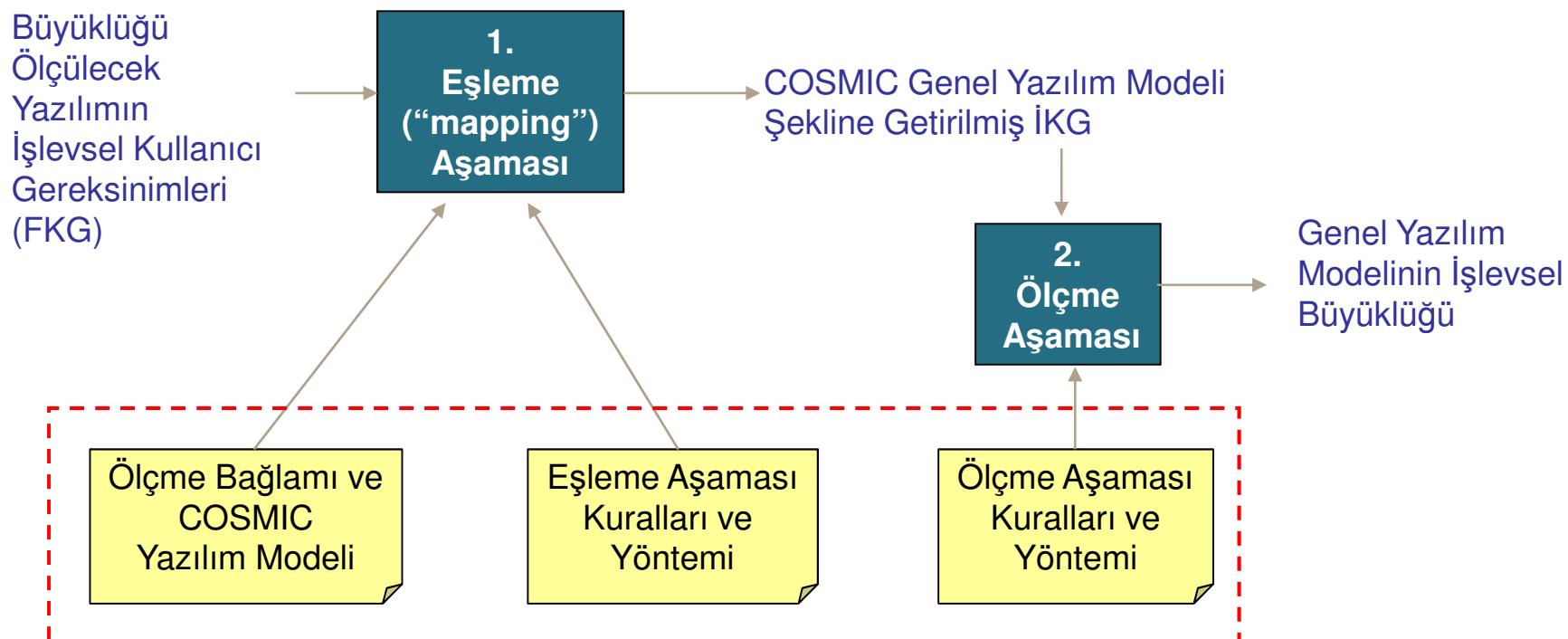
■ COSMIC yöntemi ile ne ölçülebilir?

- ▶ Yeni bir yazılım uygulamasının işlevsel kullanıcı gereksinimleri
- ▶ Var olan bir yazılımda yapılacak değişikliklerin işlevsel kullanıcı gereksinimleri
- ▶ Var olan ve şu anda kullanımda olan bir uygulamanın karşılıladığı işlevsel kullanıcı gereksinimleri

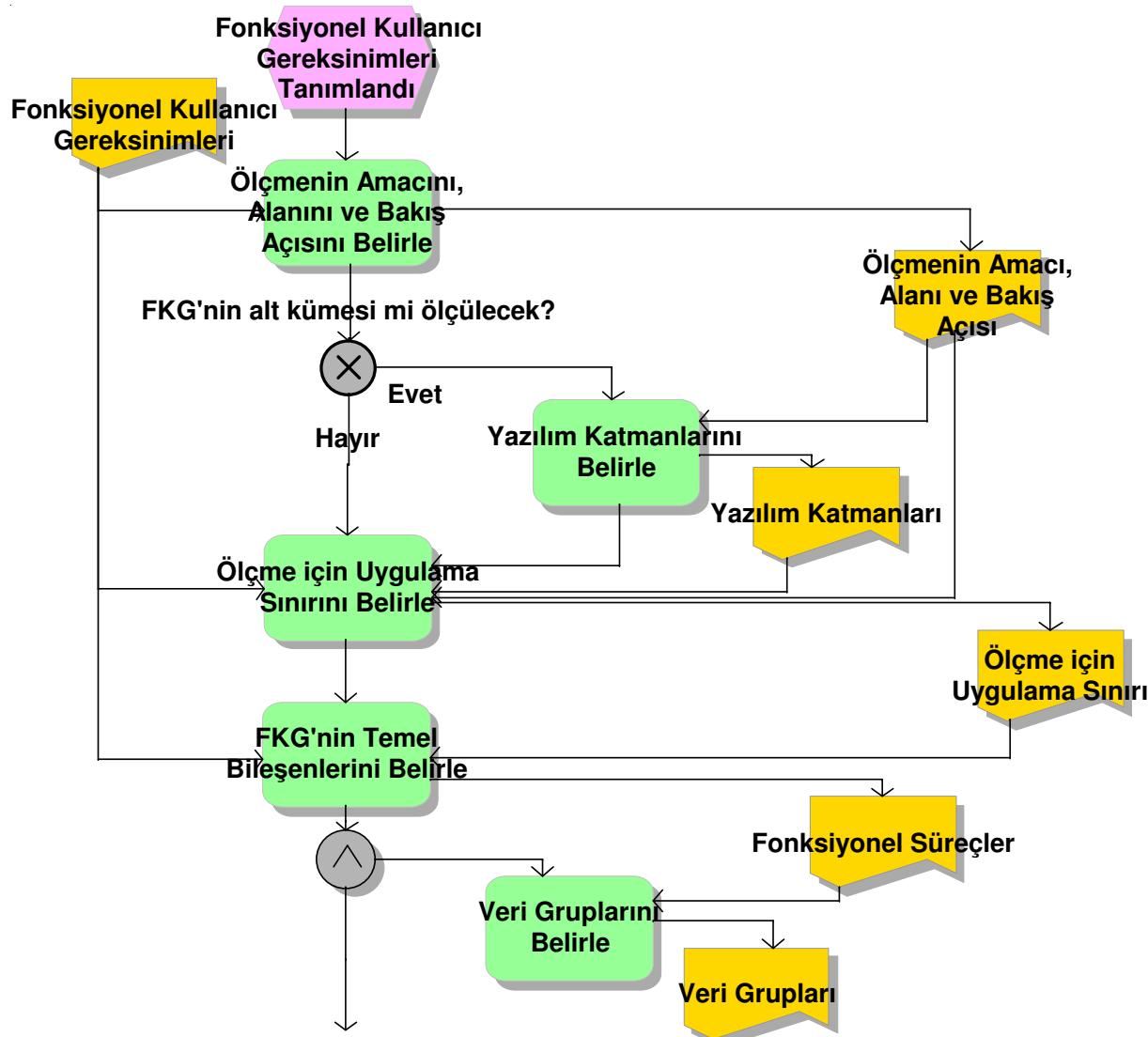
COSMIC İşlev Puan Analiz Yöntemi (3)

- Büyüklük kestirimi yapan kişinin şunu gözönünde tutması gereklidir:
 - ▶ Ölçme kuralları, karmaşık algoritmaların büyüklüğe etkisini hesaba katmaz (uzman sistemleri, simülasyon yazılımları, hava tahminleme sistemleri, vb.)
 - ▶ Ölçme kuralları video imgelerinin büyüklüğe etkisini hesaba katmaz (bilgisayar oyun yazılımları, müzik aletleri, vb.)
- Yazılım yaşam döngüsünden bağımsızdır
 - ▶ Çağlayan (“waterfall”), Spiral, Artımlı (“incremental”), vb.
- Geliştirme metodolojisinden bağımsızdır.
 - ▶ Nesneye yönelik, vb.
- İşlevsel Kullanıcı Gereksinimleri (İKG)’nin bir ölçüvidir, ancak bunların nasıl gerçekleştirildiğinden (“implementation”) bağımsızdır.

COSMIC Yöntemi Ölçme Süreci Modeli



1. Eşleme Aşaması



“Ölçmenin Amacını, Alanını, Bakış Açısını Belirle”

- Ölçmenin amacı, ölçmenin neden yapıldığını ve sonucunun ne şekilde kullanılacağını anlatan bir cümledir. Örneğin;
 - ▶ Yazılım kestirim sürecine girdi olmak üzere,
 - ▶ İKG üzerinde anlaşmaya varıldıktan sonra, zaman içinde İKG ile ilgili yapılan değişikliklerin büyüklüğünün hesaplanması ve yönetilmesi,
 - ▶ Geliştiricilerin üretkenliğinin ölçülmesinde temel birim olarak,
 - ▶ ...
- Ölçme alanı, bir ölçme sürecinin kapsayacağı İKG kümesidir.
 - ▶ Bir yazılım uygulaması
 - ▶ Bir organizasyonun yazılım portföyü
 - ▶ ...
- Ölçmenin bakış açıları:
 - ▶ Son Kullanıcı Bakış Açısı: Yazılımın karşılaşacağı ve/veya son kullanıcıya sağlayacağı işlevsellik miktarı.
 - ▶ Geliştirici Bakış Açısı: Yazılımın her bileşeninin karşılaşacağı ve/veya sağlayacağı işlevsellik miktarı.

“Ölçme İçin Uygulama Sınırlını Belirle”

- Yazılım ile “kullanıcı” arasında kavramsal bir sınır tanımlar.
 - ▶ İş kullanıcısı – Bilgi girişi yapan ya da da çıktıları alan kişi (örneğin; müdür, terminal başında herhangi bir kişi)
 - ▶ Otomatikleştirilmiş kullanıcı – Bir başka uygulama yazılımı veya büyülüğu ölçülen yazılım uygulamasından veri alan veya o yazılıma veri iletan mühendislik cihazları (örneğin; sensörler).
- Hangi işlevsel süreçlerin ölçüleceğinin ve diğer yazılım uygulamaları ile olan arabirimlerinin belirlenmesini sağlar.
- Ölçme sırasında hangi işlevselliklerin hesaba katılacağını hangilerinin ise dışında bırakılacağını belirlenmesini sağlar.

Örnek: XYZ Sistemi

- Amaç: XYZ sistemini geliştirmeye yönelik kestirimlerin yapılabilmesi için işlevsel büyülüğünün hesaplanması
- Alan:
 - ▶ Uygulama - XYZ
 - ▶ Bir modül – Muhasebe işlemleri
- Bakış Açısı: Son kullanıcı
- Sınır:
 - ▶ Uygulama XYZ'nin etrafından
 - ▶ Modülün etrafından

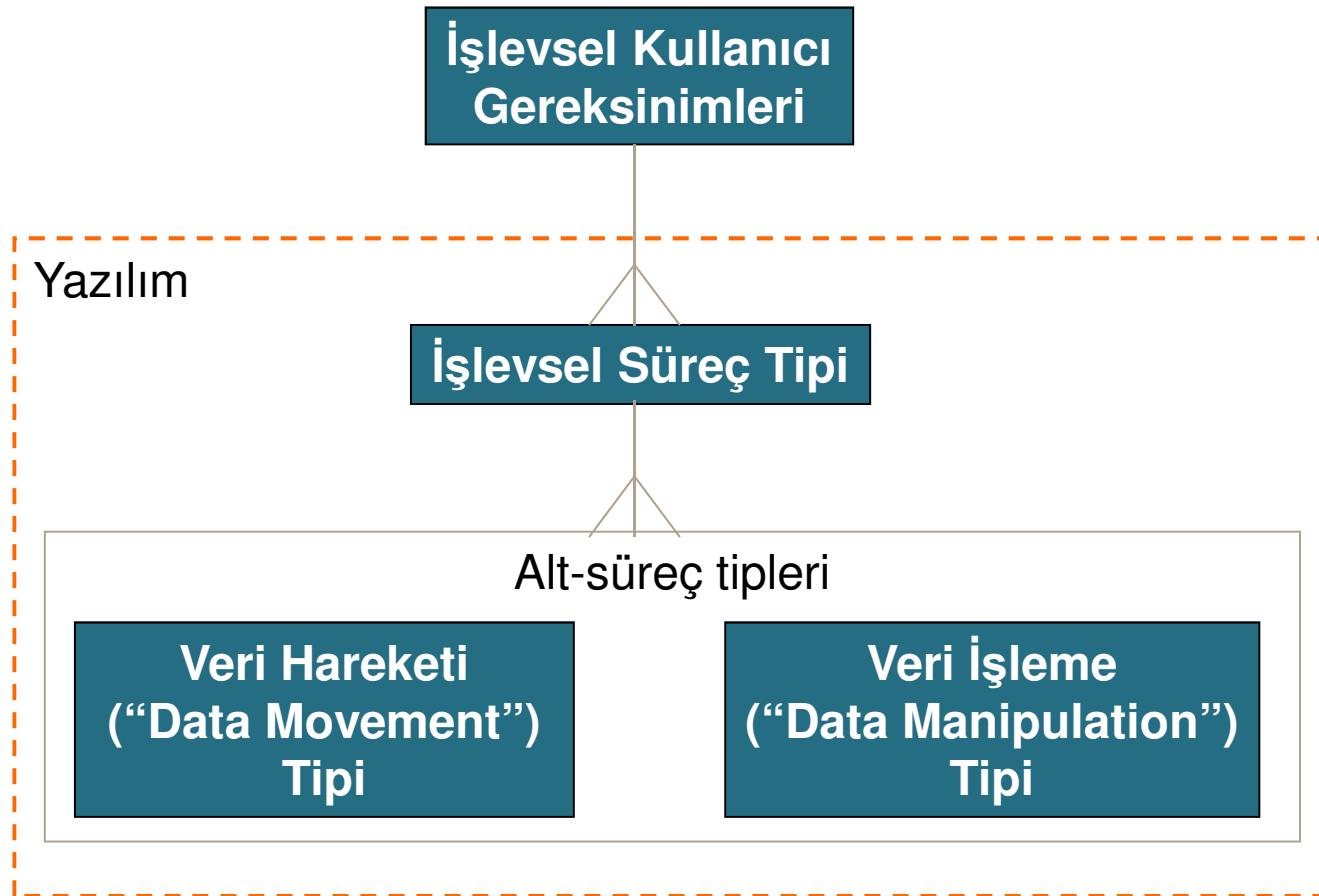
COSMIC Genel Yazılım Modeli (1)

- COSMIC genel yazılım modeli şu ilkelerin doğru olduğu varsayıımına dayanır:

1. Eşlenecek ve ölçülecek yazılım, girdilerle beslenir ve kullanıcı için çıktılar üretir.
2. Eşlenecek ve ölçülecek yazılım, verileri çeşitli şekillerde işler.

- Bu modele göre, yazılım **İşlevsel Kullanıcı Gereksinimleri**, bir seri **İşlevsel Süreç**'e ayrıstırılır.
- Her İşlevsel Süreç veri hareketi ve/veya veri işleme (manipülasyon) icra eden tek ve eşsiz alt-süreçtir.

COSMIC Genel Yazılım Modeli (2)



COSMIC Genel Yazılım Modeli (3)

- COSMIC yöntemi; “veri işleme” tipindeki alt-süreçleri ayrı olarak tanımaz, bunları “veri hareketi” tipindeki alt-süreçlerin bir parçası olarak ele alır.
 - ▶ Bunun için, COSMIC-FFN Yöntemi veri hareketi ağırlıklı yazılımların (örneğin iş uygulamaları ve gerçek-zamanlı sistemlerin çoğu) büyülüklerini ölçmek için uygunudur.
 - ▶ Ancak algoritma ağırlıklı yazılımları (örneğin bilimsel yazılımlar) ölçmek için uygun değildir.

“İşlevsel Süreçleri Belirle” (1)

■ İşlevsel Süreç (“Functional Process”):

- ▶ Bir set İKG'nin, kendi içinde bir bütün olan ve bağımsız olarak uygulanabilecek veri hareketlerinden oluşan temel bileşenidir.
- ▶ Bir aktör tarafından dolaylı olarak veya bir ya da daha fazla olay tarafından doğrudan tetiklenebilir.
- ▶ Tetikleyicinin ihtiyaç duyduğu bütün gereksinimi karşıladığında tamamlanmış olur.

■ Tetikleyici olay: Uygulama sınırı dışında olan ve bir ya da daha fazla İşlevsel Süreci başlatan olaydır.

“İşlevsel Süreçleri Belirle” (2)

■ Kurallar:

- ▶ Önce Tetikleyici Olayları belirler, sonra bu olaylar tarafından tetiklenen İşlevsel Süreçleri buluruz.
- ▶ Tetikleyici Olay, ölçülen yazılımın “sınırları” dışında oluşur ve **bir ya da daha fazla İşlevsel Süreci başlatır**. “Sınır”, Tetikleyici Olaylar ile İşlevsel Süreçler arasındadır.
- ▶ Bir İşlevsel Süreci başlatan Tetikleyici Olay, daha alt seviyedeki olaylara bölünemez.
 - ◆ **Saat ve zamanlamaya** ilişkin olaylar, Tetikleyici Olay olabilirler.
 - ◆ Yazılım göz önünde bulundurulduğunda, bir Tetikleyici Olay ya olmuştur, ya da olmamıştır; yani **anlıktır**.

■ Bir İşlevsel Süreç:

- ▶ **en azından bir adet İşlevsel Kullanıcı Gereksinimi’nden çıkarılmıştır**
- ▶ bir **tetikleyici olay** olduğu zaman icra edilir
- ▶ en azından **iki adet veri hareketi** içerir;
 - ◆ 1 Giriş + 1 Çıkış / 1 Yazma

“İşlevsel Süreçleri Belirle”: Örnek-1

- “Yazılım; 35 yaşından büyük olan çalışanların listesinin personel veritabanından okunarak kullanıcıya gösterilmesine olanak sağlayacaktır”.
 1. Eşsiz ve sıralı bir seri veri hareketi içeriyor mu? Evet
 2. Tek ve eşsiz bir Olay tarafından mı tetikleniyor? Evet
 3. Tetikleyici Olay, Yazılımın Sınırı dışında mı oluşuyor? Evet
- Öyleyse, bu bir İşlevsel Süreç’tir!

“İşlevsel Süreçleri Belirle”: Örnek-2

■ “Bankamatikten para çekilmesi” bir İşlevsel Süreç midir?

► EVET

- ◆ Tek ve eşsiz bir olay tarafından tetikleniyor.
- ◆ Uygulama sınırının içine doğru bir veri girişi var (para çekme isteği tetikleyicisi, müşteri kodu, para miktarı), içerisinde işlemler yapılıyor (çekilecek miktar hesaptan düşülüyor) ve kullanıcıya bir veri cıkışı (makbuz) var.
- ◆ Tamamlandığı zaman Uygulamayı tutarlı bir durumda bırakıyor.

- “Çekilecek para miktarının kullanıcı arayüzünden girilmesi” bir İşlevsel Süreç değildir.
- “Hesaptan çekilecek para miktarının düşülmesi” bir İşlevsel Süreç değildir.
- “Para sayma makinesine tetikleyici mesajın gönderilmesi” bir İşlevsel Süreç değildir.

“İşlevsel Süreçleri Belirle”: Örnek-3

- Bir İKG'nin A ve B olarak iki kısmı var ve görünüşte bağımsız olarak uygulanabiliyorlar. Ancak A, Uygulama Sınırı dışındaki bir olay tarafından tetiklenirken, B sadece A tarafından tetikleniyor.
 - ▶ Bu durumda, İşlevsel Süreç; A+B biçimindedir. A ve B olarak iki ayrı İşlevsel Süreç yoktur.
- Örnek: Yazılım, yeni bir müşteri (müşteri adı, hesap numarası, hesap tipi, adresi, telefonu, e-mail adresi) eklendiğinde, müşteriye bilgilendirme için bir e-posta gönderecektir.
 - ▶ Bu İKG sadece 1 İşlevsel Süreç'den oluşuyor. E-posta gönderme işlemi ancak müşteri eklendiği zaman tetikleniyor.

İşlevsel Süreç ile ilgili değişik durumlar ..

■ CRUDL (Yarat, Oku,Güncelle, Sil, Listele) işlemleri

- ▶ İş Uygulamaları'nda, hakkında kalıcı veri depolanan Nesneleri ayırt etmekte fayda vardır.
- ▶ Çünkü genellikle, bu Nesne ile ilgili verilerin Yaratılması, Güncellenmesi, Silinmesi, Okunması ve Listelenmesi ayrı birer İşlevsel Süreç'tir.

İşlevsel Süreç ile ilgili değişik durumlar ..

■ Ekran tasarımı

- ▶ Tek bir fiziksel ekranda birden fazla İşlevsel Süreç sunuluyor olabilir.
- ▶ Örneğin tek bir ekranda Yarat ve Güncelle İşlevsel süreçleri birlikte bulunabilirler.
- ▶ Çünkü, bu iki süreç çok fazla ortak işlevsellik içerir.

■ Fiziksel ekran kısıtları

- ▶ Bir önceki durumun tersine bazen tek bir İşlevsel Süreç birden fazla ekrana yayılmış olabilir.
- ▶ Bu durumda ekran kısıtları ve aralardaki geçiş amaçlı kontroller göz ardı edilmelidir.

İşlevsel Süreç ile ilgili değişik durumlar ..

■ Güncelleme öncesi okuma

- ▶ İş uygulama yazılımlarının büyük bölümü kalıcı veri gruplarını güncellemek için iki ayrı İşlevsel Süreç içerir.
 - ◆ Birincisi, hakkında tutulan veri grubu güncellenecek Nesne'nin okumasını içeren '**Güncelme öncesi okuma**' işlevsel süreci: Bu süreç, kullanıcıya güncellemeye öncesi doğru verilerin getirilip doğru verilerin seçilmesini sağlar.
 - ◆ İkincisi, güncellenen veri grubunun veri tabanına kaydedilmesini içeren '**Güncelme**' işlevsel süreci: Bu süreç kullanıcıya değiştirilecek, eklenecek veya silinecek veri girişini sağlar.

“İşlevsel Süreçleri Belirle”: Örnek-4 (Güncelleme Öncesi Okuma)

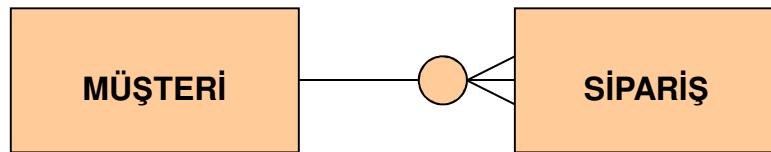
- ‘Güncelleme öncesi okuma’ bazı durumlarda iki ayrı İşlevsel Süreç’ten de oluşuyor olabilir.
- Örnek İKG:
 - ▶ Bir Çalışanın kayıtları güncellenebilir. Kullanıcının Çalışan’ın adını bilmekte ancak Çalışan numarasını bilmemektedir. Bunun için önce Çalışan bütün kişileri görüntüleyip sonra ilgili kişiyi seçebilmelidir. Seçilen kişinin güncellenebilecek kayıtları görüntülenmelidir.
- Bu durumda;
 - İS1: Kullanıcı önce isme göre Çalışanlar listesini görüntüler.
 - İS2: Kullanıcı belirli bir Çalışan’ı seçer ve onunla ilgili bilgileri görüntüler.
 - İS3: Kullanıcı Çalışan’ın bilgilerini günceller.

“Veri Gruplarını Belirle”

- Veri Grubu: Aynı nesne ile ilişkili, farklı, boş olmayan ve sıralı olmayan veri özellikleri setidir.
 - ▶ Örnek: Müşteri (numarası, adı, soyadı, hesap numarası, vs.)
- COSMIC Yöntemi her İşlevsel Süreç'teki, her biri bir Nesne hakkındaki bir grup veri özelliğini (Veri Grubu) hareket ettiren Veri Hareketi'ni belirlemeye dayanır.
- Bunun için öncelikle Nesne'leri ve veri hareketlerini belirlemek için Varlık Analiz Yöntemlerinden birisini kullanmak gereklidir.
 - ▶ Örnek: E/R diyagramları, Sınıf Diyagramları, vs.

E/R Diyagramları (1)

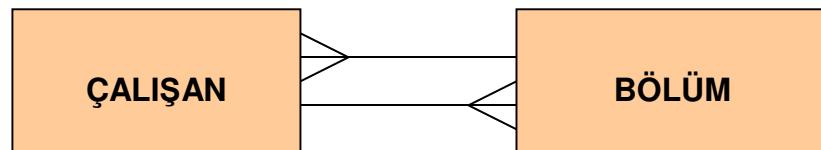
- Örneğin, Fonksiyonel Kullanıcı gereksinimi şu şekilde olsun:
 - ▶ ‘Sipariş işleme sistemi, müşteriler ve müşterilerin verdikleri siparişlerle ilgili verileri depolayacaktır.’
- Bu gereksinime göre çizilecek bir E/R diyagramı şu varlıklarını gösterir:



Müşteri ve Sipariş varlıkları, COSMIC yönteminde hakkında bilgi tutulacak Nesne ve dolayısıyla Veri Grupları'na karşılık gelir.

E/R Diyagramları (2)

- Başka bir Fonksiyonel Kullanıcı Gereksinimi şu şekilde olsun:
 - ▶ ‘Yazılım, çalışanlar ve bu kişilerin çalıştığı bölümler ile ilgili verileri depolayacak ve bakımını yapacaktır. İstendiğinde, bu verilerden bir çalışanın geçmişte çalıştığı bölümlerin listesini başlama ve bitiş tarihleri ile birlikte gösterecektir.’
- Bir önceki örnektenden farkı bu iki varlık arasında 1-N ilişki yerine M-N ilişki vardır.



Bu durumda, E/R der ki, iki varlık arasında M-N ilişki bulunuyorsa, bu ilişki bir ara varlık eklenerek iki 1-N ilişki oluşturacak şekilde dönüştürülmelidir.



Bu durumda COSMIC yöntemi için Nesneler ve bu Nesneler hakkında veri tutacak Veri Grupları; Çalışan, Bölüm ve Çalışan/Bölüm Tarihçe'dir.

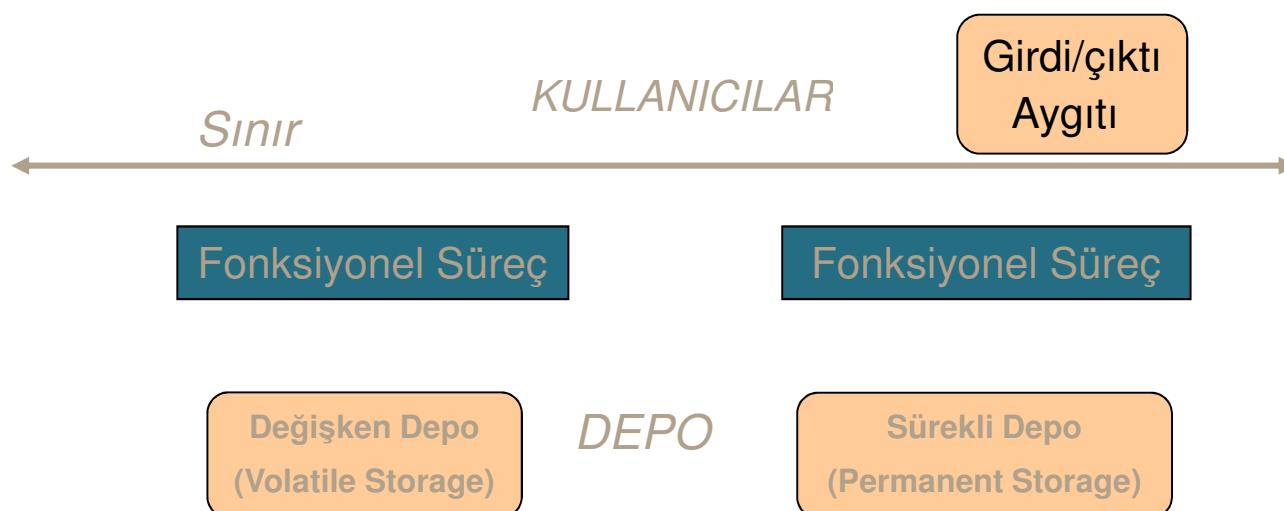
UML Sınıf Diyagramları (ve Use Case'ler)

- COSMIC yöntemi ile doğru ölçme yapabilmek için hem İşlevsel Süreçler hem de her birindeki veri etkileşimleri bilinmelidir.
- Bu bağlamda, UML “use-case” leri yaygın olarak kullanılmaktadır. Fakat burada önemli noktalardan birisi, bir “use-case”的in her zaman bir İşlevsel Süreç'e denk gelmeyeceğidir.
 - ▶ Eğer “use-case”lerden işlevsel büyülü ölçümü yapmak istenirse, “use-case”lerin soyutlama seviyeleri ve detayları, COSMIC yönteminde tanımlı İşlevsel Süreç'e eşlenebilecek şekilde tanımlanmalıdır.
- Analiz sınıflarını gösteren sınıf diyagramındaki her UML sınıfı, tipik olarak COSMIC yöntemi için Nesne'ye karşılık gelir.
 - ▶ Her UML sınıfı için tanımlanmış metod da, her sınıf ile ilgili işlevselliği tanımladığı için İşlevsel Süreç olmaya adaydır.

Veri Grubu - İlkeleri

1. Bir yazılımda Veri Grubu üç şekilde gerçekleşebilir:

- a. Sürekli depoda (persistent storage) (dosya, veritabanı tabloları, vb.);
- b. Değişken depoda (volatile storage)
- c. Girdi/Çıktı aygıtında (ekran, basılı rapor, vb.),



Veri Grubu – İlkeleri (devam)

2. Belirlenen her Veri Grubu tek ve eşsiz olmalı ve içерdiği veri özellikleri ile ayırt edilmelidir.
3. Her Veri grubu, yazılımın İşlevsel Kullanıcı Gereksinimleri’nde tanımlı bir Nesne ile ilişkili olmalıdır.
 - ▶ Üç tipi vardır:
 - ◆ Geçici (Transient)
 - ◆ Sınırsız (Indefinite)
 - ◆ Kısa süreli (Short)

İş Uygulama Yazılımlarındaki Veri Grupları

■ ‘Sınırsız’ biçiminde olan Veri Grupları:

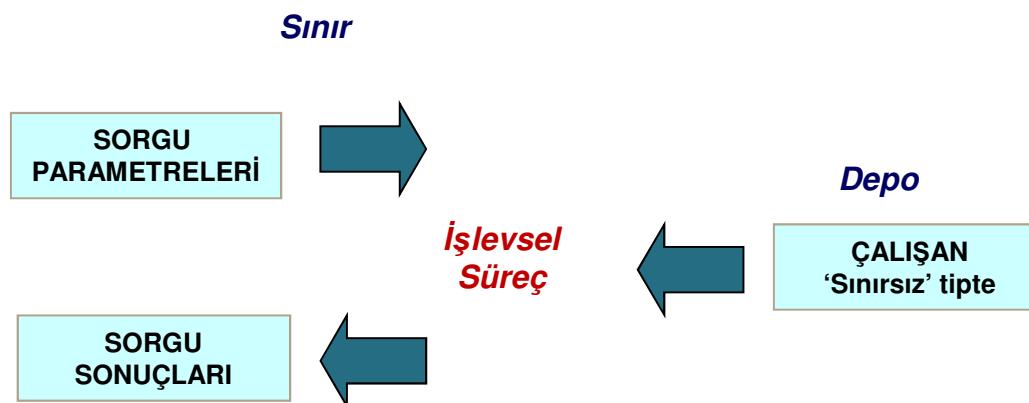
- ▶ Örneğin ‘Çalışan’ (fiziksel) veya ‘Sipariş’ (kavramsal) -Yazılımın bu nesneler hakkında bilgi depolama gereksinimleri olabilir.

■ ‘Geçici’ biçiminde olan Veri Grupları (Geçici bir Nesne ile ilişkili Veri Grupları) genelliklelarındaki bilgi ‘sınırsız’ olarak tutulmayan bir ‘şey’ hakkında herhangi bir soru yapıldığında oluşur.

- ▶ Ancak, bu Veri Grupları ‘sınırsız’ olarak depolanan Veri Grupları sorgulanarak oluşturulur.
- ▶ Örneğin Yaşı 35’den büyük olan Çalışan kişiler listesi

Örnek

- İşlevsel Süreç: “Yazılım yaşı 35’den büyük olan çalışanların listesinin personel veritabanından okunarak kullanıcıya gösterilmesine olanak sağlayacaktır”



■ İKG: Kullanıcı;

- ▶ Belirli bir zaman aralığında satılmış malların toplam değerini Müşteri tipine (K= Kişisel, P= Perakendeci veya T= Toptancı) göre hesaplayabilecek ve görüntüleyebilecektir.
- ▶ Belirli bir zaman aralığında satılmış malların toplam değerini hesaplayabilecek ve görüntüleyebilecektir.

■ Bu iki farklı şekilde bir araya getirilen ve gösterilen satış verisi iki ayrı Nesne hakkındaki ‘geçici’ veri gruplarını göstermektedir.

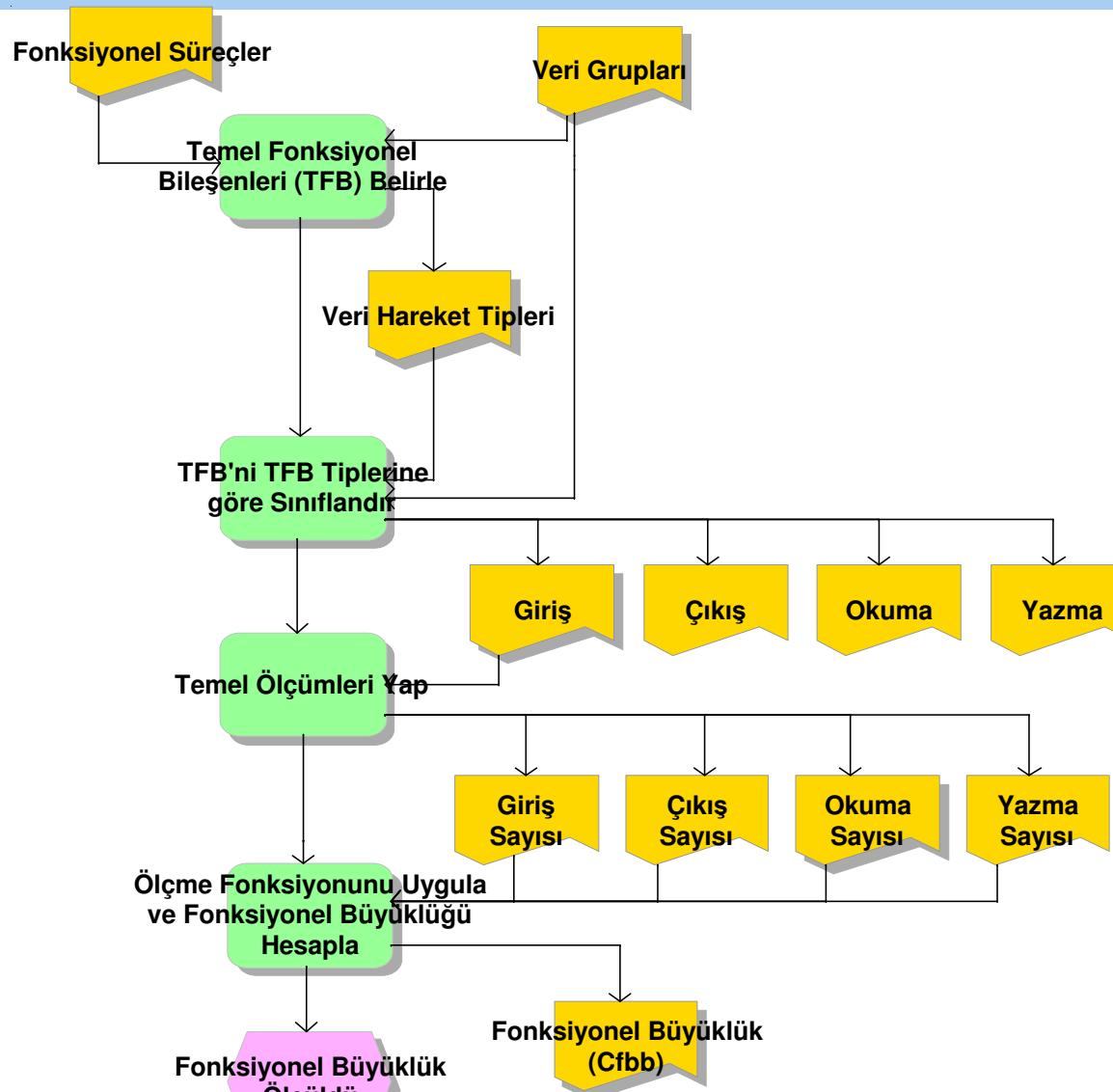
■ Bunun için bu soru sonrasında iki ‘Çıkış’ hesaplanır. Bu iki Nesne şunlardır:

- ▶ Müşteri tipine göre belirli bir zaman aralığında satılmış mallar.
- ▶ Belirli bir zaman aralığında tüm müşterilere satılmış mallar.

Gerçek-Zamanlı Yazılımlardaki Veri Grupları

- **Fiziksel aygıtlardan** Yazılımğa girdi olan ve Tek bir Nesne ile ilişkili verileri taşıyan Veri hareketleridir. Örneğin, bir Valf'in açık veya kapalı olduğu, ya da Geçici Depo'nun geçerli veya geçersiz olduğu, vb.
- İşlevsel Kullanıcı Gereksinimleri'nde tanımlı Nesneleri temsil eden ve Geçici Depo'da tutularak Yazılımdaki birçok Fonksiyonel Süreç tarafından erişilen **Ortak veri yapıları**,
- İşlevsel Kullanıcı Gereksinimleri'ndeki tabloları veya grafikleri temsil eden ve Sürekli Depo'da (örneğin ROM bellek) tutularak Yazılımdaki birçok İşlevsel Süreç tarafından erişilen **Referans Veri Yapıları**,
- İşlevsel Kullanıcı Gereksinimleri'ndeki bir Nesneyi temsil eden ve Sürekli Depo'da tutulan **Dosyalar**.

2. Ölçme Aşaması



Temel İşlevsel Bileşenleri Belirle ve Sınıflandır

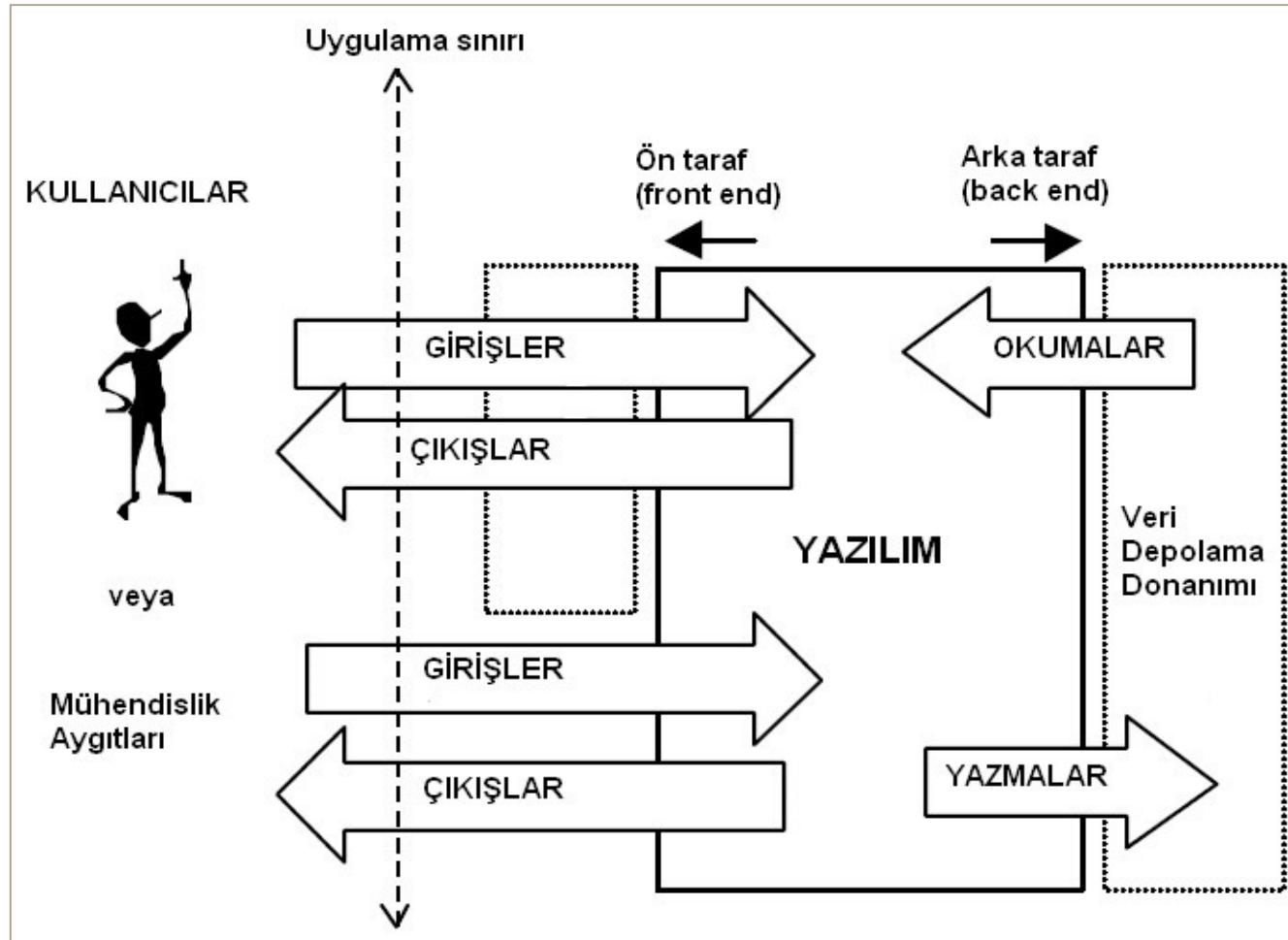
- Temel İşlevsel Bileşen: Veri Hareketi (Data Movement):
 - ▶ Tek bir Veri Grubu Tipine dahil olan bir ya da daha fazla Veri Özelliği Tipini (Data Attribute Type) hareket ettiren Temel İşlevsel Bileşendir.
 - ▶ Bir İşlevsel Süreç yürütüldüğü sırada oluşur.

- Veri Hareketi Tipi dört çeşittir:
 - ▶ Giriş, Çıkış, Okuma, Yazma

Veri Hareketi Tipleri

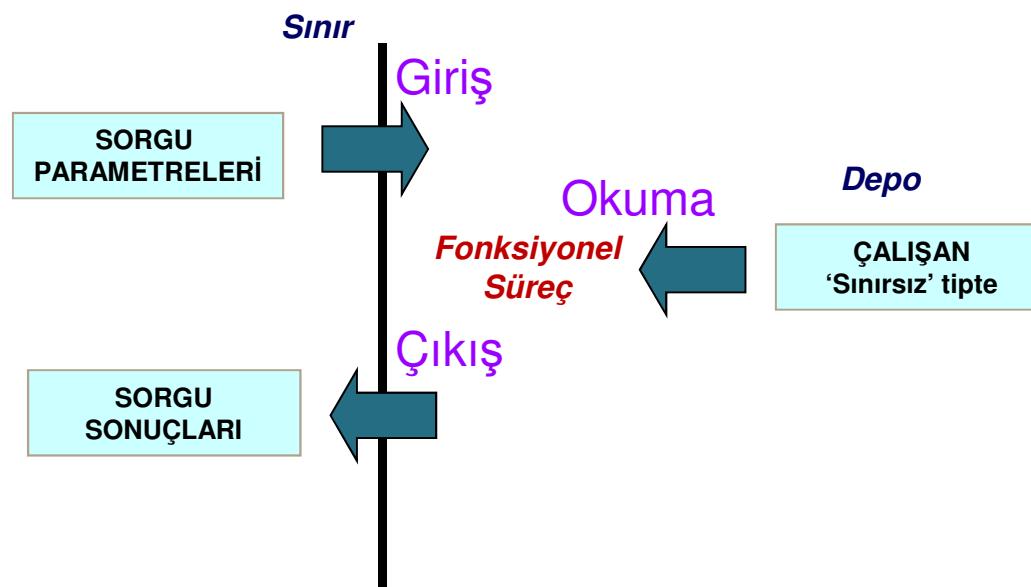
- **Giriş:** Bir Veri Grubu'nu kullanıcının Uygulama Sınırı'ndan içeriye ihtiyaç duyulan İşlevsel Süreç'e doğru hareket ettiren Veri Hareketi Tipi'dir.
- **Cıkış:** Bir Veri Grubu'nu İşlevsel Süreç'ten Uygulama Sınırı'ndan dışarıya ihtiyaç duyan Kullanıcı'ya doğru hareket ettiren Veri Hareketi Tipi'dir.
- **Okuma:** Bir Veri Grubu'nu Sürekli Depo'dan (persistent storage) ihtiyaç duyulan İşlevsel Süreç'e doğru hareket ettiren Veri Hareketi Tipi'dir.
- **Yazma:** Bir Veri Grubu'nu içinde barındıran İşlevsel Süreç'ten Sürekli Depo'ya doğru hareket ettiren Veri Hareketi Tipi'dir.

Veri Hareket Tipleri



Örnek

- İşlevsel Süreç: “Yazılım yaşı 35’den büyük olan çalışanların listesinin personel veritabanından okunarak kullanıcıya gösterilmesine olanak sağlayacaktır”



Temel Ölçümleri Yap

- Yazılım uygulamasının toplam işlevsel büyütüğü, bütün İşlevsel Süreçlerin büyütüklerin toplanmasıyla bulunur.
 - ▶ Bir İşlevsel Süreci oluşturan her veri hareketinin ne tipte olduğu belirlenir
 - ◆ (Giriş, Çıkış, Okuma, Yazma)
 - ▶ *İşlevsel Büyüklük Birimi*: COSMIC İşlevsel Büyüklük Birimi – Cfbb
 - ▶ Her veri hareketi tipinin büyütüğü = 1 Cfbb

Ölçme Fonksiyonunu Uygula

- Bir İşlevsel Sürecin toplam işlevsel büyüklüğü, bütün veri hareketi tiplerinin işlevsel büyüklüklerinin toplanması ile bulunur.

Büyüklük_{Cfbb} (FS_i) = Σ büyüklük(girişler_i) + Σ büyüklük(çıkışlar_i) +
 Σ büyüklük(okumalar_i) + Σ büyüklük(yazmalar_i)

COSMIC Ölçme Sonucunu Raporla

- Bir COSMIC FFN ölçme sonucu “**x** Cibb (v. **y**)” şeklinde raporlanır,
 - ▶ “x” fonksiyonel büyüklüğü temsil eder, “v.y” ise ölçmede kullanılan COSMIC-FFN yönteminin sürümünü gösterir.
 - ▶ Cibb: COSMIC işlevsel büyüklük birimi (*COSMIC functional size unit – Cfsu*)
- Örnek: **1250** Cibb (COSMIC-FFN v2.2)

COSMIC Ölçme: Örnek

'Güncelleme öncesi okuma' bazı durumlarda iki ayrı İşlevsel Süreç'ten de oluşuyor olabilir.

Örnek FKG: Bir Çalışanın kayıtları güncellenebilir. Kullanıcı, Çalışan'ın adını bilmekte ancak Çalışan numarasını bilmemektedir. Bunun için önce Çalışan bütün kişileri görüntüleyip sonra ilgili kişiyi seçebilmelidir. Seçilen kişinin güncellenebilecek kayıtları görüntülenmelidir.

İS1: Kullanıcı önce isme göre Çalışanlar listesini görüntüler.

E 'Çalışanları liste' isteği

R Çalışan

X Çalışan verileri (istenen Çalışanı seçmek için gerekli veri özellikleri)

X Hata mesajı

İS1 işlevsel büyülüğu = 4 Cfbb

İS2: Kullanıcı belirli bir Çalışan'ı seçer ve onunla ilgili bilgileri görüntüler.

E Çalışan No (= istenen çalışanı seçerek)

R Çalışan

X Çalışan verileri (Çalışan hakkındaki bütün veriler)

X Hata mesajı

İS2 işlevsel büyülüğu = 4 Cfbb

İS3: Kullanıcı Çalışan'ın bilgilerini günceller.

E Güncellenmiş Çalışan verileri

W Çalışan

X Hata/Onay mesajı

İS3 işlevsel büyülüğu = 3 Cfbb

İKG fonksiyonel büyülüğu =

İşlevsel büyülüük (FS1)

+

İşlevsel büyülüük (FS2)

+

İşlevsel büyülüük (FS3)

=

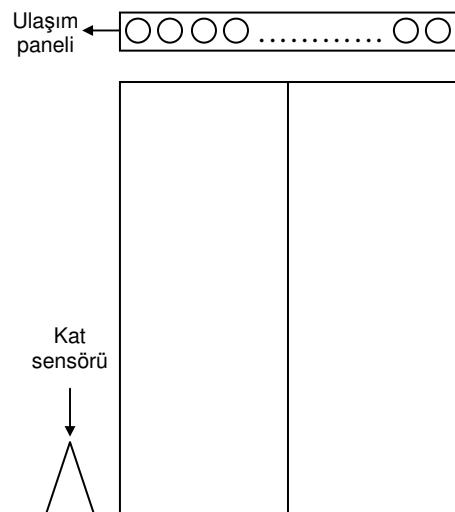
4Cibb + 4Cibb + 3Cibb = 11 Cibb

Asansör Kontrol Sistemi

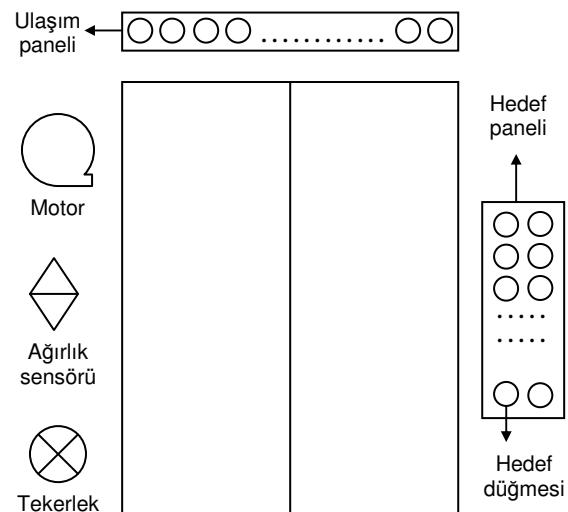
40 katlı bir binadaki 4 asansörü zamanlamak ve kontrol etmek için bir sistem geliştirilecektir. Asansörler tipik olarak katlar arasında yolcu taşımak için kullanılacaktır.

Asansörler etkin şekilde zamanlanacaktır. Örneğin, bir yolcu 4. kattaki çağrı panelinden aşağı düğmesine basarak asansör çağrılığında, aşağı yönde ilerleyen ve dördüncü kata ulaşan ilk asansör yolcuyu almak için katta duracaktır. Asansör yolcu taşımıyor ve herhangi bir hedef talebi bulunmuyorsa, tekrar ihtiyaç duyulana kadar ulaştığı son katta bekleyecektir. Asansör, ilerlediği yönde talepte bulunan tüm yolcular katlarına ulaşana kadar, yönünü değiştirmeyecektir. Yük taşıma kapasitesini dolduran bir asansör, yeni çağrırlara yanıt vermeyecektir.

Katın yapısı



Asansörün yapısı



Ağırlık sensörü: Her asansörün bir ağırlık sensörü bulunacak ve sistem bu sensörleri sorgulayabilecektir.

Kat sensörü: Her kattaki her asansör boşluğununda bir kat sensörü bulunacaktır. Bir asansör kata 10 cm yaklaşlığında, asansördeki tekerlek kat sensörüne ait anahtarı kapatacak ve sisteme asansörün kata ulaştığına dair sinyal gönderecektir.

Hedef paneli ve düğmeleri: Her asansörün içinde 40 farklı kat için 1'den 40'a kadar numaralandırılmış hedef düğmelerini barındıran bir hedef paneli bulunacaktır. Hedef düğmeleri sistem tarafından ışıklandırılacaktır.

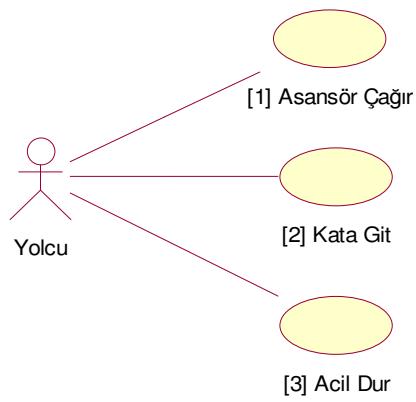
Ulaşım paneli ve ışıkları: Her asansörün içinde ve her kattaki asansör kapısının üstünde, 40 farklı kat için 1'den 40'a kadar numaralandırılmış ışıklarını barındıran bir ulaşım paneli bulunacaktır. Sistem, her asansörün belirli bir zamanda hangi katta bulunduğu bilgisini, ilgili ışığı aktif hale getirerek gösterecektir.

Çağrı paneli ve düğmeleri: Her katta, içinde "yukarı" ve "aşağı" yönleri için çağrı düğmelerini barındıran bir çağrı paneli bulunacaktır. Yolcu bu düğmeleri, gitmek istediği yöne göre asansörü çağrımak için kullanacaktır.

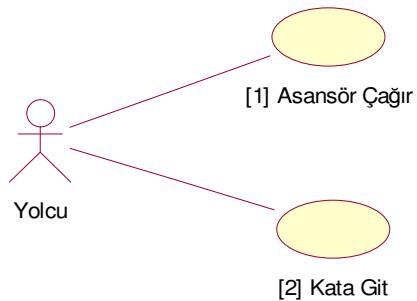
Asansör motoru: Her asansörün yukarı veya aşağı yönde hareket etmesini veya katta durmasını sağlayan bir motoru bulunacaktır.

Sistem uygun ve güvenlikli olmayan komutlara yanıt vermeyecektir. Yukarıda tanımı verilmeyen, asansör kaplarının açılması ve asansörün katın tam hızında durması gibi işlemler, asansör üretici firma tarafından mekanik olarak gerçekleştirileceğinden sistem tarafından kontrol edilmeyecektir. Ayrıca her asansörün içinde yine mekanik olarak işleyen bir acil durma düğmesi olacaktır.

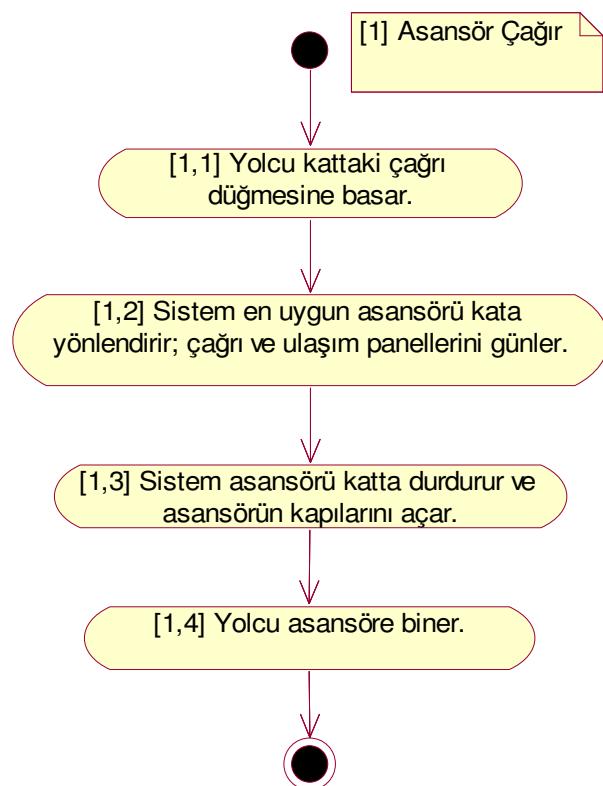
Sistem Geliştirme Use-Case Modeli:



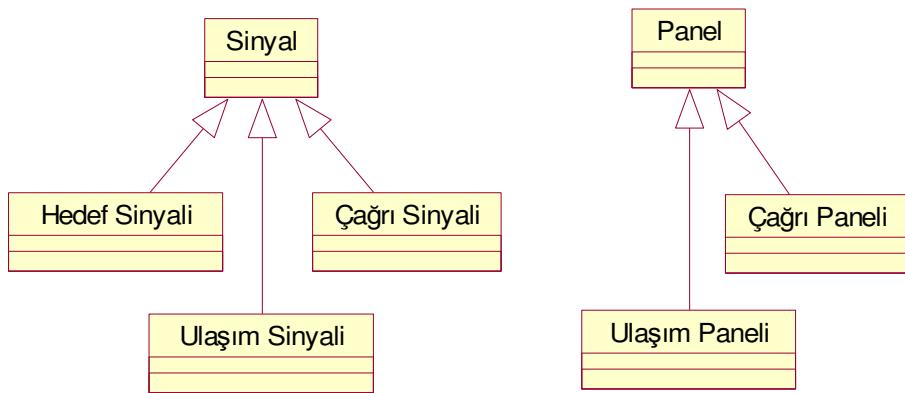
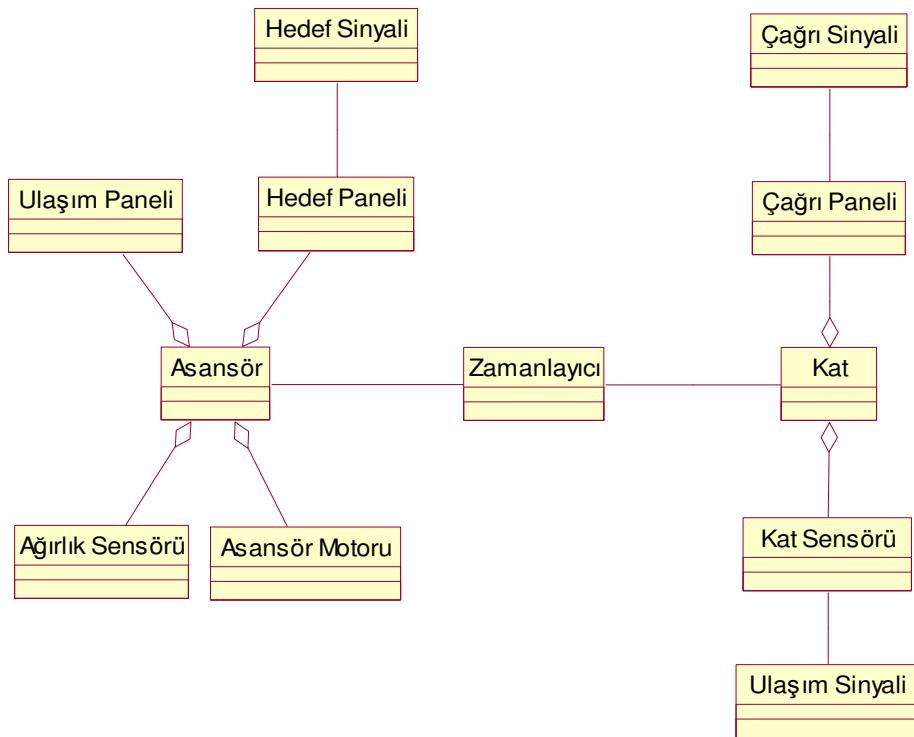
Yazılım Geliştirme Use-Case Modeli:



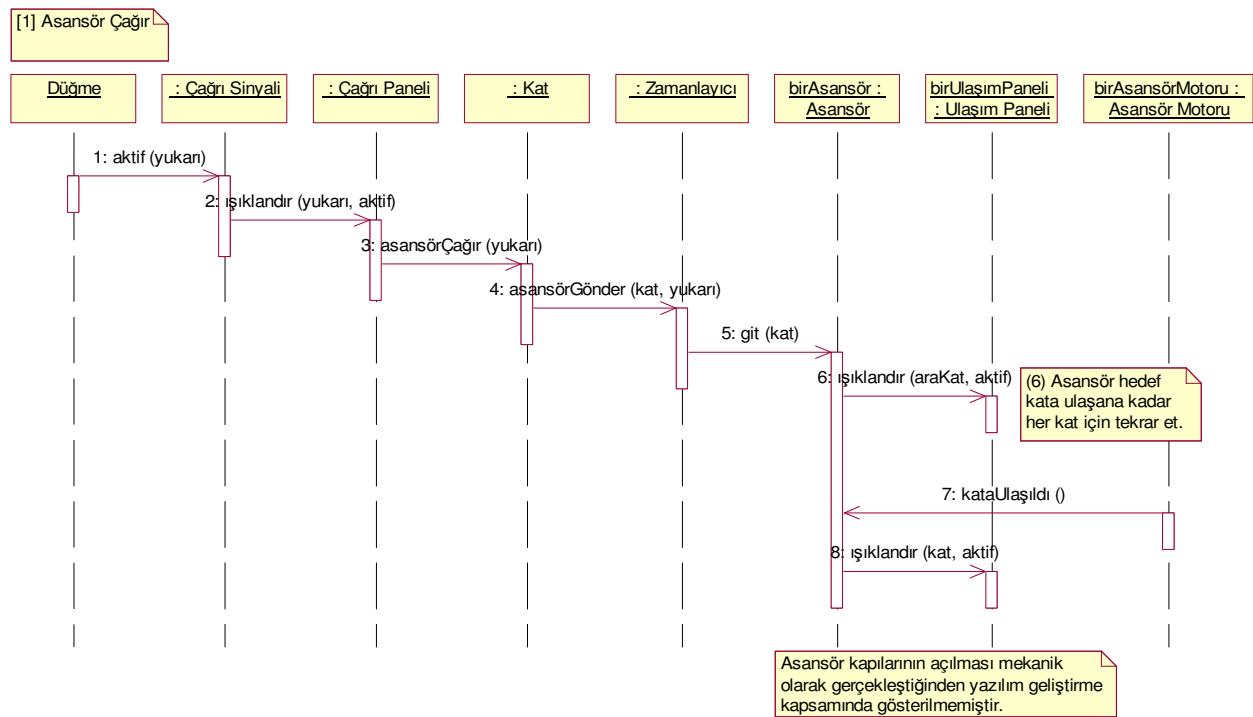
“[1] Asansör Çağır” Use-Case Detaylı Etkinlik Akışı:



Yazılım Geliştirme Analiz Sınıfları, Sınıf Diyagramları:



“[1] Asansör Çağır” Use-Case Detaylı Etkinlik Akışı için Sınıfların Davranışı
(Ardıl-İşlem Diyagramı ile):





BM306

Yazılım Mühendisliği



DERS 8

Yazılım Kalite Güvence
ve İlişkili Kavramlar

Dr. Öğr. Üyesi Hasan BADEM
hbadem@ksu.edu.tr

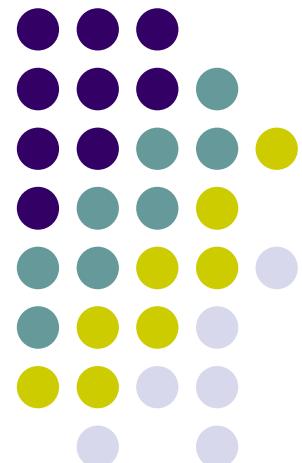




Ders İçeriği

- Kalite kavramı ve yazılımda kalite ihtiyacı
 - Yazılım kalite güvence ve öğeleri
 - ISO 12207 standardı ve Yazılım Kalite Güvence
 - Yazılım yaşam döngüsü boyunca kalite güvence
 - Kalite maliyeti
 - Yazılım kalite metrikleri
-
- Ödev
 - KDS İşlevsel Büyüklük Kestirimi

Kalite Kavramı ve Yazılımda Kalite İhtiyacı





“Kalite” Nedir? - 1

- Deming, 1968
 - “Ability to satisfy requirements”
→ İhtiyaçları karşılama yeteneğidir.
- Crosby, 1979
 - “Conformance to user requirements”
→ Kullanıcı gereksinimlerine uygunluktur.
- Price, 1985
 - “Doing right it the first time”
→ İlk seferde doğrusunu yapmaktadır.
- Juran, 1988
 - “Fitness for intended use”
→ Amaçlanan kullanıma uygunluktur.



“Kalite” Nedir? - 2

- ISO 9001:2000
 - “the degree to which a set of *inherent* characteristics fulfills *requirements*”
 - (Nesnenin) tabiatında var olan özelliklerin gereksinimleri karşılama derecesidir.
- National Institute of Standards and Technology, 2003
 - Baldrige National Quality Program
 - “Customer-driven quality”
 - Müşteri güdümlü kalite



“Kalite” Mitleri

- “Kalite soyut bir kavramdır ve ölçülemez”
 - Kalite, uygunsuzluğun maliyeti (hata maliyeti) ile ölçülebilir.
- “Bizim işimiz diğerlerinden farklı”,
“Daha iyisini yapmanın maliyetini şu an için karşılayamayız”
 - Kaliteyi baştan sağlanmanın (hata önleme) maliyeti, sonradan sağlamaya çalışmanın (hata bulma ve düzeltme) maliyetinden daha azdır.
- “Kalite problemlerine en alt kademedede çalışanlar sebep olur”
 - En alt kademedede çalışanlar, daha üst kademedede çalışanlardan daha az hata maliyetine sebep olur.
- “Kaliteyi, kurumların Kalite Bölümleri başlatır”
 - Kalite anlayışı üst yönetim seviyesinde başlar ve tüm çalışanlar tarafından benimsenmelidir.



Kalite Nasıl Sağlanır?

- Geleneksel anlayış: Hata ayıklama
 - “Kalite kontrol”
 - Bir ürün veya hizmetin tanımlanmış gereksinimleri karşılayıp karşılamadığının tespitinde kullanılan teknikler ve uygulanan faaliyetler
 - Ürün/hizmet üzerinden kaliteyi sağlama anlayışı
- Gelişmiş anlayış: Hata önleme
 - “Kalite güvence”
 - Bir ürün veya hizmetin tanımlanmış gereksinimleri karşılanması yeterli derecede güvence altına almak için gerekli olan, tüm planlanmış ve sistematik faaliyetler
 - Ürünü/hizmeti oluşturan sistem üzerinden kaliteyi sağlama anlayışı



Ürün Yoluyla Kalite

- Üretim hattını durdurmak yerine; kaza eseri olduğunu varsayıarak hatalı ürünler birleştirilir.
 - Hataların giderilmesi sonraya bırakılır.
- Çalışanı hatayı tekrarlamamaya zorlar.
 - Çalışanların hataları saklamasına neden olur.
 - Yönetimin çalışanlara karşı tavır almasına sebep olur.



Süreç Yoluyla Kalite

- Hatalı ürün tespit edildiğinde; bunun ürünü geliştiren sürecin bir problemi olduğu varsayılarak üretim hattı durdurulur.
 - Hatanın kaynağı bulunur ve iyileştirilir.
- Çalışanları hataları bulmaya ve sistemi iyileştirmeye teşvik eder.
 - Yönetsel rollerde çalışanların görev almasını sağlar.
 - Alt kademedede çalışanların görev almasını sağlar.



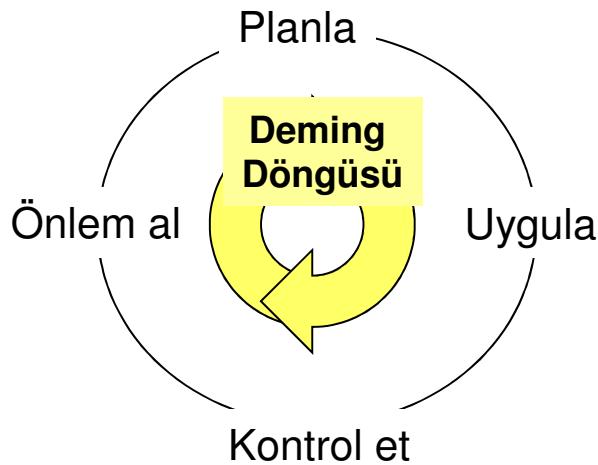
Süreç Yönetimi - 1

- Shewhart, 1930: “İstatistiksel süreç kontrolü”
 - Değişkenlik endüstriyel yaşamın bir gerçeğidir ve istatistiksel yöntemlerle yönetilebilir.
 - Değişkenliğin sebepleri:
 - Kabul edilebilir değişkenlik (“common causes”)
 - Her üretim sistemi, iç bileşenleri sebebiyle belirli seviyede değişkenliğe sebep olur.
 - Kabul edilemez değişkenlik (“special causes”)
 - Sistemin işleyişindeki özel durumlar, ürün özelliklerinde önemli değişkenliğe sebep olur
 - Tecrübeli çalışanlar tarafından tespit edilerek giderilebilir.



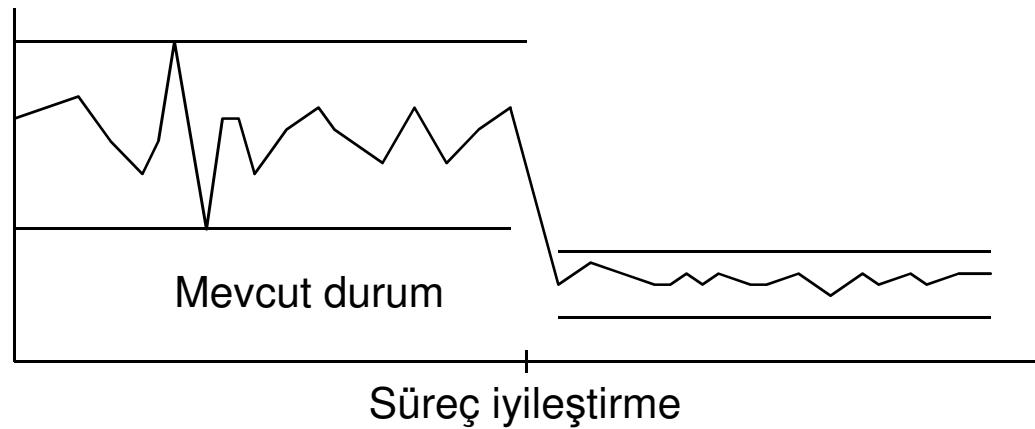
Süreç Yönetimi - 2

"Süreç Yönetimi"



Kalitesizlik maliyeti
(örnek: % takrar çalışma işgücü)

Süreç: Belirli bir hedefe ulaşmak için gerçekleştirilen (ve kullandığı girdileri çıktılara dönüştüren) adımlar zinciridir.





Kalite Kontrol ve Kalite Güvence

- Kalite kontrol
 - Ayıklamaya yönelik, sistematik, belirli bir anda
- Kalite güvence
 - Önlemeye yönelik, sistematik, zamana yayılmış
- Geliştirme sürecinin kalitesini iyileştirdirsek; hem ürün kalitesini iyileştirmiş, hem de geliştirme maliyetini ve zamanını azaltmış oluruz.
 - “Kalite bedavadır”



Yazılım Kalitesi

- Geliştirdiğimiz yazılımın kalitesi, yazılımı nasıl geliştirdiğimize büyük ölçüde bağlıdır.
 - Yazılım geliştirme süreci, yazılımı nasıl geliştirdiğimizi tanımlar.
 - Örnek: Çağlayan modeli
 - Analiz → tasarım → gerçekleştirmeye → test → bakım
 - Yazılım geliştirme sürecinin temel aşamalarını tanımlar.
- Kaliteyi, yazılım geliştirme aşamaları boyunca yazılım ürününe yerleştirmek zorundayız.
 - Kaliteyi en sonunda sağlamaya çalışmak hem zordur, hem de maliyet çok yükselir.



Kaliteli Yazılım Ne Demektir?

- Kabullenilebilir hata seviyesinde,
- Gereksinimleri karşılayan,
- Amaçlanan kullanıma uygun,
- Zamanında tamamlanmış,
- Belirlenen bütçe sınırları içinde gerçekleştirilmiş,
- Standartlara uyumlu,
- Bakımı sağlanabilen yazılımdır.



Yazılımda Kalitesizliğin Sorunları

- Müşteri tarafındanki sorunlar:
 - Gereksinimlerin sağlanamayışı
 - Kolay anlaşılabilir ve kullanılabılır olmaması
 - İstenilen zamanda bakım yapılabılır olmaması
 - Eğitim desteğinin yetersiz olması
- Yazılım firması tarafındanki sorunlar:
 - Geciken ya da bitmeyen projeler
 - Yüksek maliyet
 - Çalışanların tatminsizliği
 - Firmaya olan güven kaybı

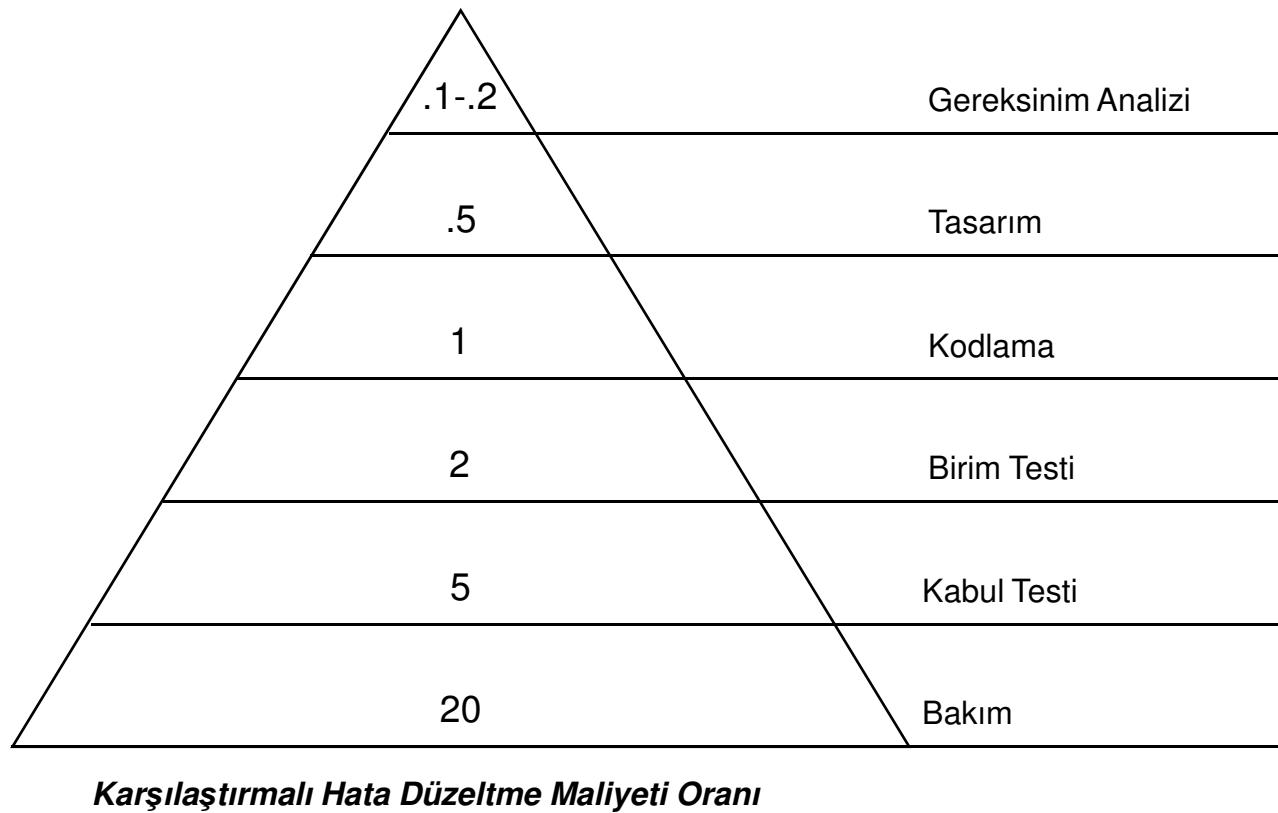


Yazılımda Kalite Neden Gerekli?

- Tecrübeli bir mühendis her 7-10 satırda bir hata üretmektedir.
 - Orta ölçekli bir projede binlerce satıra karşılık gelir.
- Hataların çoğunun test aşamasında düzeltilmesi gerekmektedir.
- Testler uzadıkça maliyet artmakta, teslimat gecikmekte, ürün kalitesi düşmekte ve bakım maliyeti geliştirme maliyetini aşmaktadır.



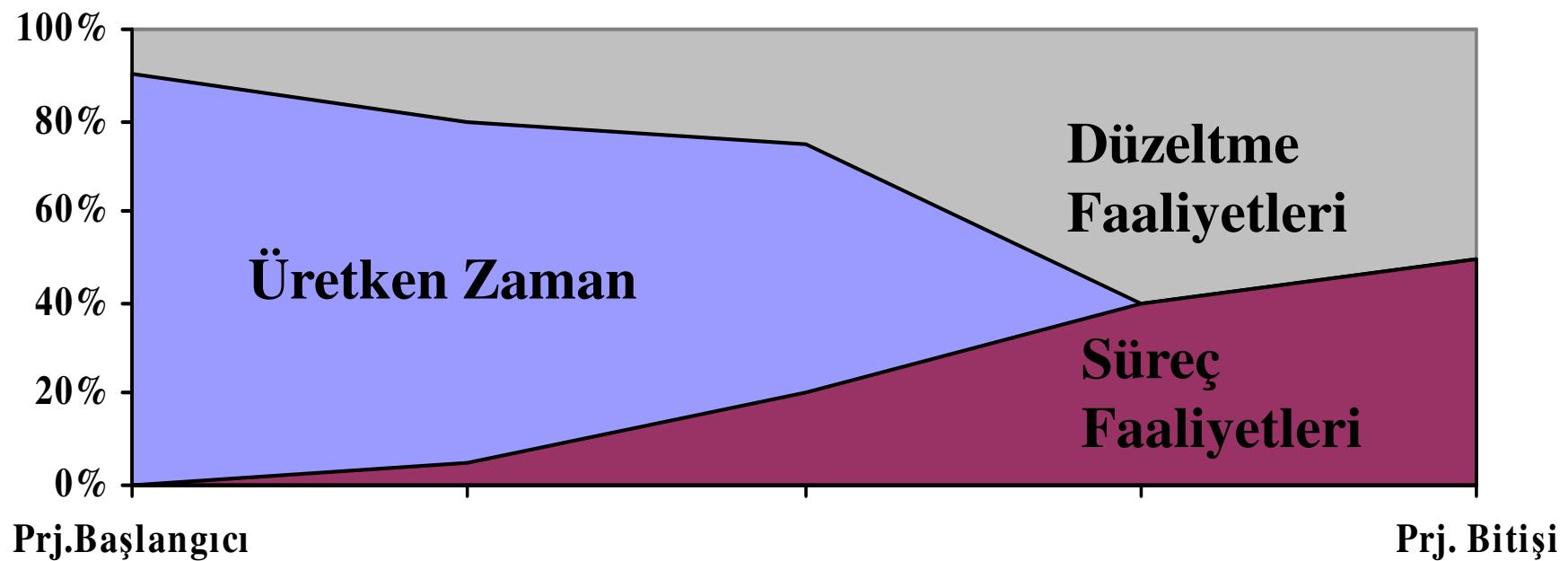
Yazılım Hatalarını Düzeltme Maliyeti



Referans: *Managing Software Requirements – A Unified Approach*, Leffingwell & Widrig, 2000

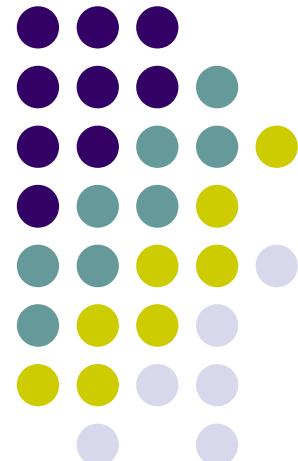


Yazılımda Kalitesizliğin Proje Zamanına Etkisi



Referans: *Software Project Survival Guide*, Steve McConnell, 1997.

Yazılım Kalite Güvence ve Öğeleri



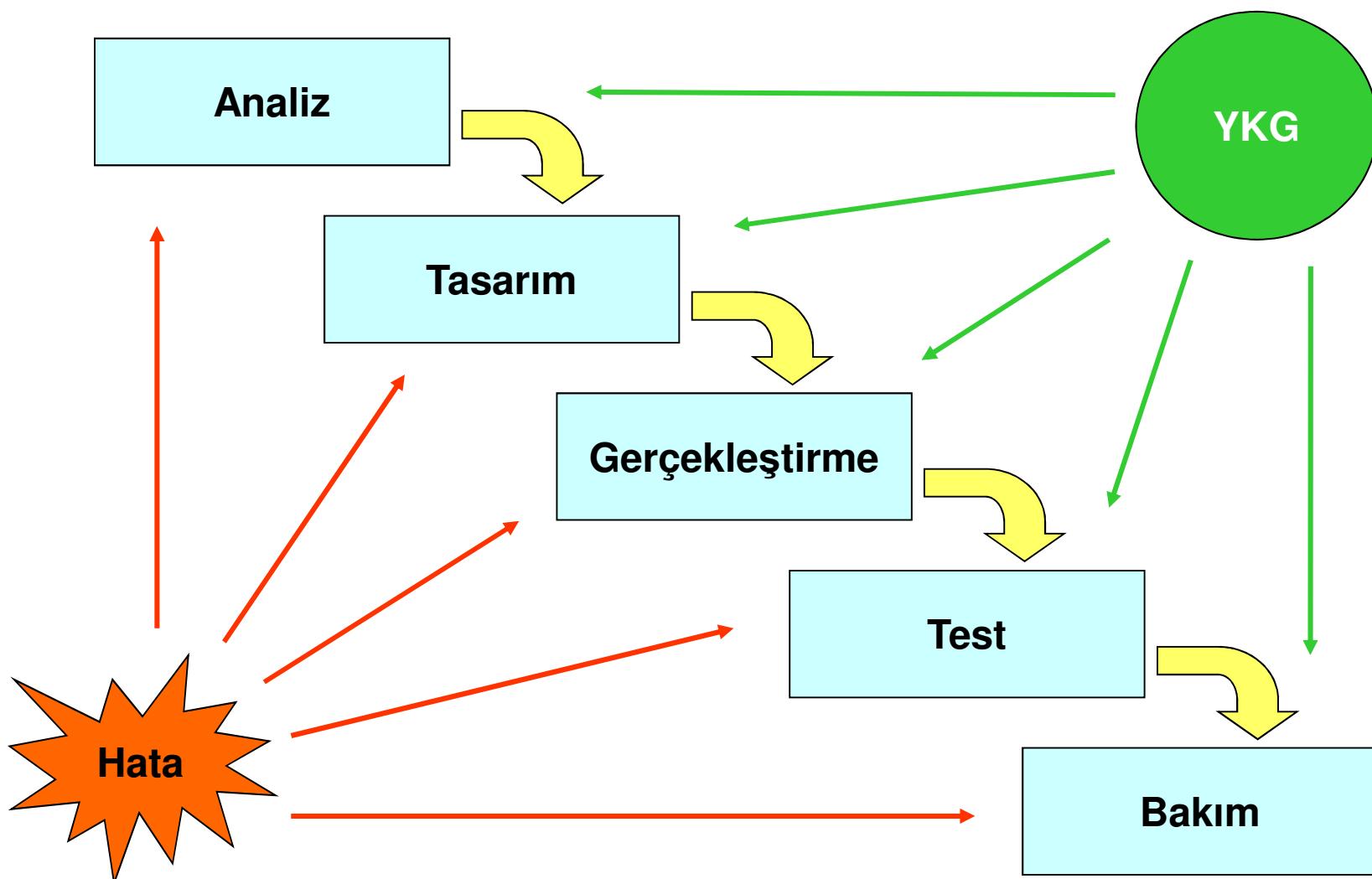


Yazılım Kalite Güvence - 1

- Yazılımın geliştirmesi ve bakımı boyunca kaliteyi sağlamaının maliyet-etkin yoludur.
- Kalitenin yazılım ürünüğe yerleştirilmesi için, proje yaşam döngüsü boyunca üretilen ürünlerinin ve uygulanan süreçlerinin, tanımlı gereksinimlere uyduğunu güvence altına almayı hedefler.
- Proje yaşam döngüsü boyunca çeşitli etkinlikleri planlayarak, işleteerek ve izleyerek hataların erken tespit edilmesine hizmet eder.



Yazılım Kalite Güvence - 2





Yazılım Kalite Güvence - 3

- Yazılım Kalite Güvence (YKG), yazılım ürünün istenilen kalitede olmasını güvence altına alan ve geliştirme süreci boyunca uygulanan etkinlikleri içerir.
 - “Software quality assurance”
- Doğrulama ve Geçerleme (D&G), yazılım geliştirme sürecinin her aşamasındaki çıktıların, istenilen özelliklere uygunluğunu kontrol etmeye yarayan etkinliklerdir.
 - “Verification and validation”
- Test, bir üründeki hataların ortaya çıkarılması etkinliğidir.

D&G, YKG'nin bir parçasıdır ve Test'i içerir.

YKG > D&G > Test



Yazılım Kalite Güvence: Yöntemler

- **Statik yöntemler:** Kodu çalıştırmadan yapılır.
 - Örnek: Gereksinim ve tasarım belgeleri ile kodun denetimi veya gözden geçirilmesi
 - İnceleme (“inspection”)
 - Gözden geçirme (“review”)
 - Denetleme (“audit”)
- **Dinamik yöntemler:** Kodu çalıştırarak yapılır.
 - Örnek: Ürünün veya bileşenlerinin gerçeğe yakın testi
 - Birim test (“unit test”)
 - Tümleştirme test (“integration test”)
 - Sistem test (“system test” / “functional test” / “qualification test”)
 - Kabul test (“acceptance test”)



Doğrulama ve Geçerleme

- **Doğrulama (“verification”):** Müşterinin istediği ürünü, doğru şekilde geliştirmeyi güvence altına alır.
 - Mühendislik adımlarının doğru uygulanıp uygulanmadığının kontrolüdür.
 - Her aşamada, bir önceki aşamada yapmayı taahhüt ettiklerimizi yaptık mı? Bir önceki belgede tanımladıklarımıza göre uyumsuzluk veya eksik var mı?
 - “**Are we building the product right?**”
- **Geçerleme (“validation”):** Müşterinin istediği, (doğru) ürünü geliştirmeyi güvence altına alır.
 - Mühendislik ürününün müşteri ihtiyaçlarına uyup uymadığının kontrolüdür.
 - Her aşamanın çıktısı müşteri gereksinimlerini karşılıyor mu?
 - “**Are we building the right product?**”



Test

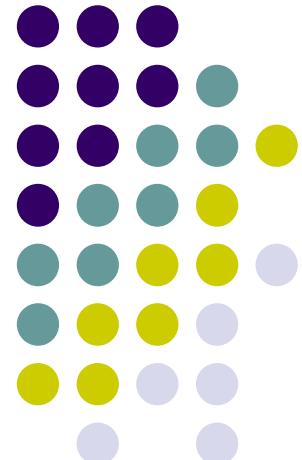
- Bir programın davranışını; beklenen davranışa uymadığı durumları bulma amacı ile, sonsuz bir küme içinden sınırlı sayıda seçilen test vakalarını kullanarak, dinamik yöntemlerle sınama işlemidir. [SWEBOK 2004].
 - Beklenen : Tanımlanmış gereksinimlere uyan
 - Sınırlı : Yeterli sayıda
 - Seçilmiş : Uygun test vakaları
 - Dinamik : Kod çalıştırılarak



Amaç: Hata Önleme veya Bulma

- YKG'nin amacı hataları bulmaktan çok önlemektir.
 - En uygun yazılım geliştirme ortamının kurulması
 - Gerekli süreçlerin ve talimatların tanımlanması
 - Süreç uygulamalarının etkinliğini ölçme yöntemlerinin belirlenmesi
 - Uyulacak standartların ve kullanılacak şablonların tanımlanması
 - ...
- D&G ve Test hata bulmaya odaklanır.
 - Denetleme, Gözden Geçirme, İnceleme (statik)
 - Birim, Tümleştirme, Sistem, ve Kabul Test (dinamik)

ISO 12207 Standardı ve Yazılım Kalite Güvence





ISO/IEC 12207-2008 (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 Kapsamı

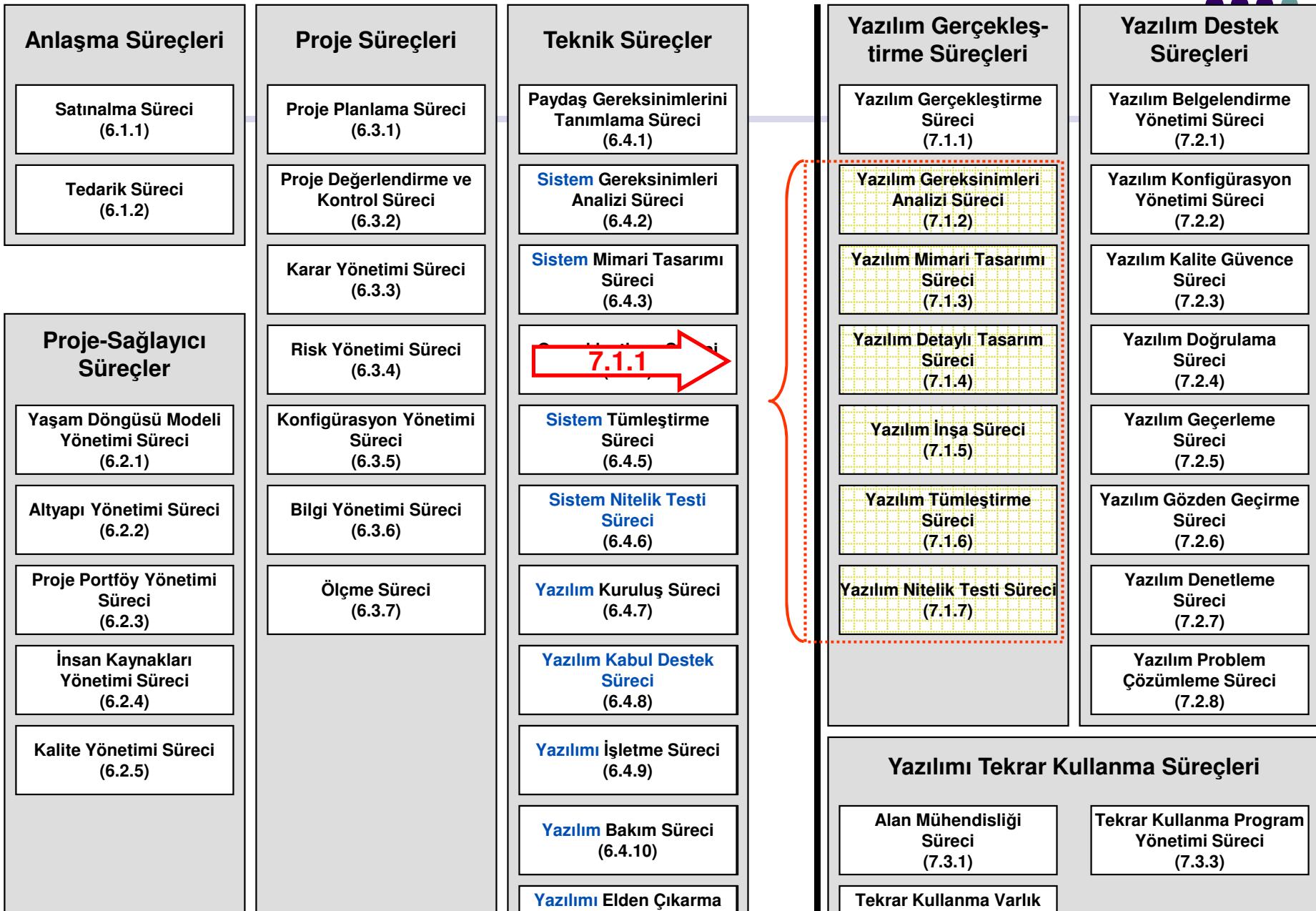
- Daha büyük bir sistemin parçası olarak veya tek başına yazılım ürünü veya hizmeti için, bir grup yaşam döngüsü süreci ile bunların altındaki etkinlikleri ve görevleri tanımlar.
- Yazılım ürününün veya hizmetinin satın alınması, tedarıği, geliştirilmesi, işletilmesi, bakımı ve elden çıkarılması boyunca 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



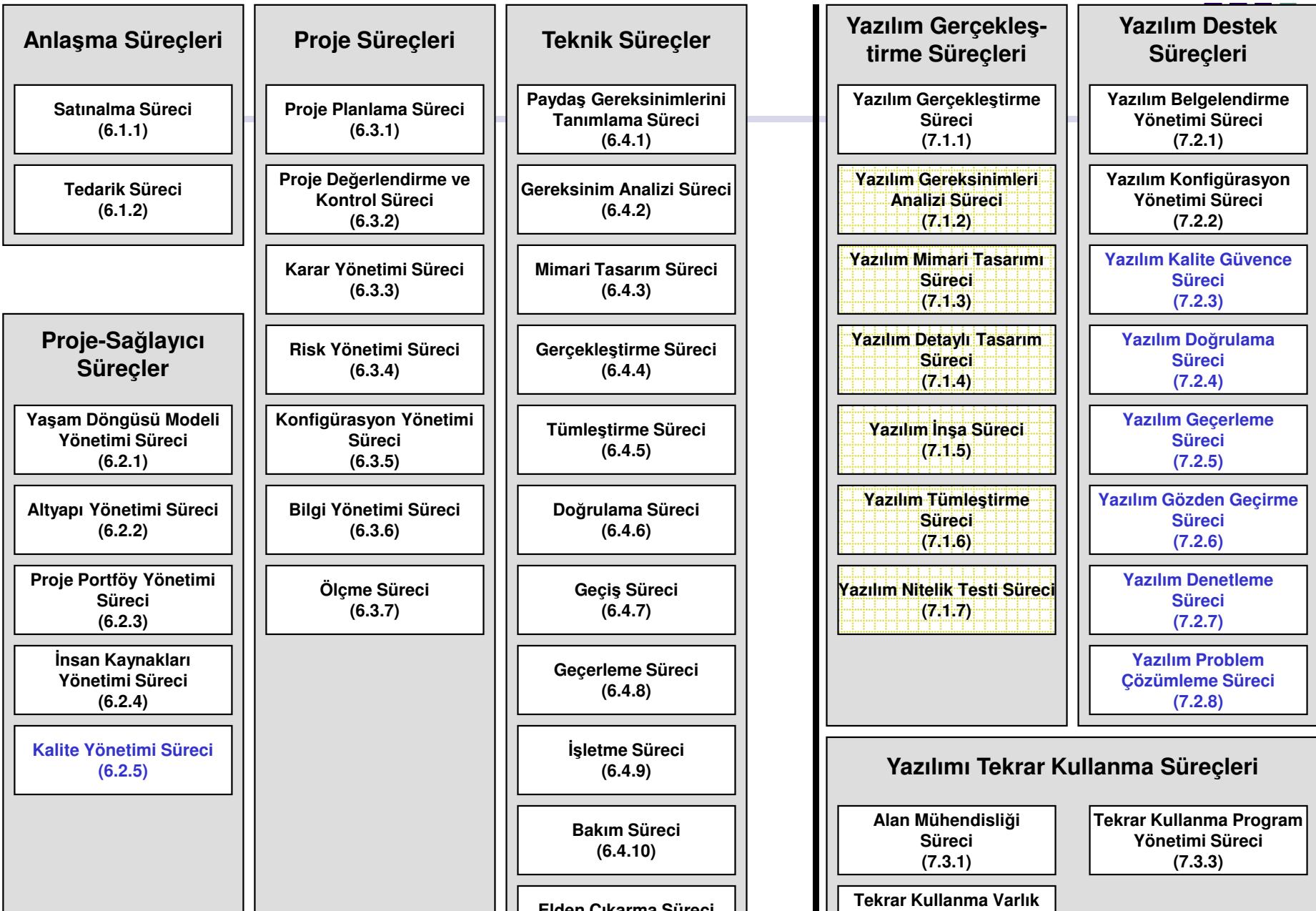
ISO/IEC 12207 Kısıtları

- Herhangi bir yaşam döngüsü modelinin, geliştirme yaklaşımının veya yönteminin kullanılmasını öngörmez.
- Her sürecin beklenen çıktıları üretecek amacıyla 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 (1528)



Sistem Kapsamlı Süreçler



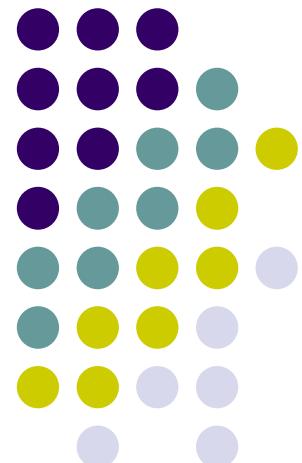


ISO/IEC 12207 – Kalite Güvence ile İlgili Süreçler

Süreç	Sürecin Amacı
Kalite Yönetimi	Ürünlerin, hizmetlerin ve yaşam döngüsü süreçlerine ait uygulamaların; kurumsal kalite hedeflerini karşıladığı ve müşteri memnuniyetini sağladığını güvence altına almak
Yazılım Kalite Güvence	İş ürünlerinin ve süreçlerin, tanımlanmış planlara uygunluğunu güvence altına almak
Yazılım Doğrulama	Sürece veya projeye ait her yazılım iş ürününün ve/veya hizmetinin, tanımlı gereksinimleri karşıladığı onaylamak
Yazılım Geçerleme	Yazılım iş ürününün, amaca özel kullanımına yönelik gereksinimleri karşıladığı onaylamak
Yazılım Gözden Geçirme	Anlaşmanın hedeflerine ve paydaşları tatmin edecek bir ürünün geliştirilmesini güvence altına almak için yapılması gerekenlere göre, gelişim hakkında paydaşlarla ortak bir anlayışı oluşturmak ve sürdürmek
Yazılım Denetleme	Seçilen ürünlerin ve süreçlerin; gereksinimlere, planlara ve anlaşmaya uygunluğunu bağımsız olarak tespit etmek
Yazılım Problem Çözümleme	Tespit edilen tüm problemlerin tanımlandığını, analiz edildiğini, yönetildiğini ve çözüme dek kontrol edildiğini güvence altına almak

ISO/IEC 12207

Yazılım Kalite Güvence Süreci





Yazılım Kalite Güvence Süreci (7.2.3)

- Amaç:
 - İş ürünlerinin ve süreçlerin, tanımlanmış planlara uygunluğunu güvence altına almak
- Çıktılar:
 - Kalite güvenceyi sağlamak için strateji geliştirilir.
 - Yazılım kalite güvencenin kanıtları üretilir ve saklanır.
 - Gereksinimlere göre problemler ve uygunsuzluklar belirlenir ve kaydedilir.
 - Ürünlerin, süreçlerin ve etkinliklerin; standartlara, prosedürlere ve gereksinimlere uyumluluğu doğrulanır.



Yazılım Kalite Güvence Süreci – Etkinlikler ve Görevler

7.2.3.3.1 – Süreç uygulaması

1. Projeye uygun bir kalite güvence süreci oluşturulacaktır.
2. Kalite güvence süreci; ilişkili **Yazılım Doğrulama**, **Yazılım Geçerleme**, **Yazılım Gözden Geçirme** ve **Yazılım Denetleme** süreçleriyle koordine edilecektir.
3. Kalite güvence sürecinin etkinliklerini ve görevlerini gerçekleştirmek için bir plan geliştirilecek, belgelendirilecek, uygulanacak ve sözleşme süresi boyunca idame ettirilecektir.
4. Planlanan kalite güvence etkinlikleri ve görevleri işletilecektir. Gereksinimlere göre problemler veya uygunsuzluklar tespit edildiğinde belgelendirilecek ve bu belgeler **Yazılım Problem Çözümleme** sürecine girdi olacaktır.
5. Kalite güvence etkinliklerinin ve görevlerinin kayıtları, sözleşmede tanımlandığı şekilde satın alan kurumun erişimine sunulacaktır.
6. Sözleşme gereksinimlerine uygunluğu güvence altına almakta sorumlu kişilerin; kurumsal bağımsızlığa, kaynaklara ve yetkiye sahip olduğu güvence edilecektir.



Yazılım Kalite Güvence Süreci – Etkinlikler ve Görevler (.. devamı)

7.2.3.3.2 – Ürün güvence

1. Sözleşmenin gerektirdiği tüm planların; belgelendirildiği, anlaşmaya uyumlu olduğu, ortak tutarlılığının bulunduğu ve gerektiği şekilde işletildiği güvence edilecektir.
2. Yazılım ürünlerinin ve ilişkili belgelerin sözleşmeyle uyumu ve planlara uygunluğu güvence edilecektir.
3. Yazılım ürünlerini teslime hazırlarken, sözleşme gereksinimlerinin tam olarak karşılandığı ve ürünün satın alan kurum tarafından kabul edilebilir olduğu güvence edilecektir.

7.2.3.3.3 – Süreç güvence

1. Proje kapsamında uygulanan yazılım yaşam döngüsü süreçlerinin sözleşmeye uyumu ve planlara uygunluğu güvence edilecektir.
2. Genel yazılım mühendisliği pratiklerinin, geliştirme ortamının, test ortamının ve kullanılan kütüphanelerin; sözleşmeye uyumu güvence edilecektir.
3. Sözleşme gereksinimlerinden uygulanabilir olanların alt yükleniciye geçirildiği ve altyüklenicinin yazılım ürünlerinin, sözleşme gereksinimlerini karşıladığı güvence edilecektir.
4. Satın alan kuruma ve işe dahil olan diğer birimlere; sözleşmeye, anlaşmalara ve planlara göre gereken desteğin ve işbirliğinin sağlandığı güvence edilecektir.
5. Yazılım ürün ve süreç ölçümlerinin, tanımlı standartlara ve prosedürlere uygunluğu güvence edilecektir.
6. Çalışanların, proje gereksinimlerini karşılayacak yetkinlik ve bilgiye sahip olduğu ve gerekli eğitimleri aldığı güvence edilecektir.

7.2.3.3.4 – Kalite sisteminin güvencesi

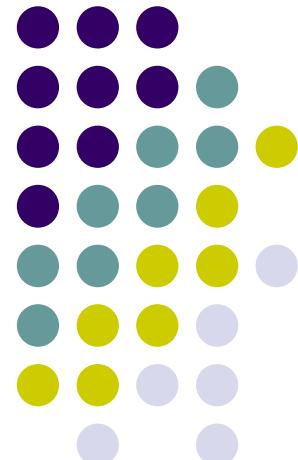
1. İlave kalite yönetimi etkinlikleri, ISO 9001 standardının maddelerine göre güvence edilebilir.



Kalite Güvence Planı İçeriği

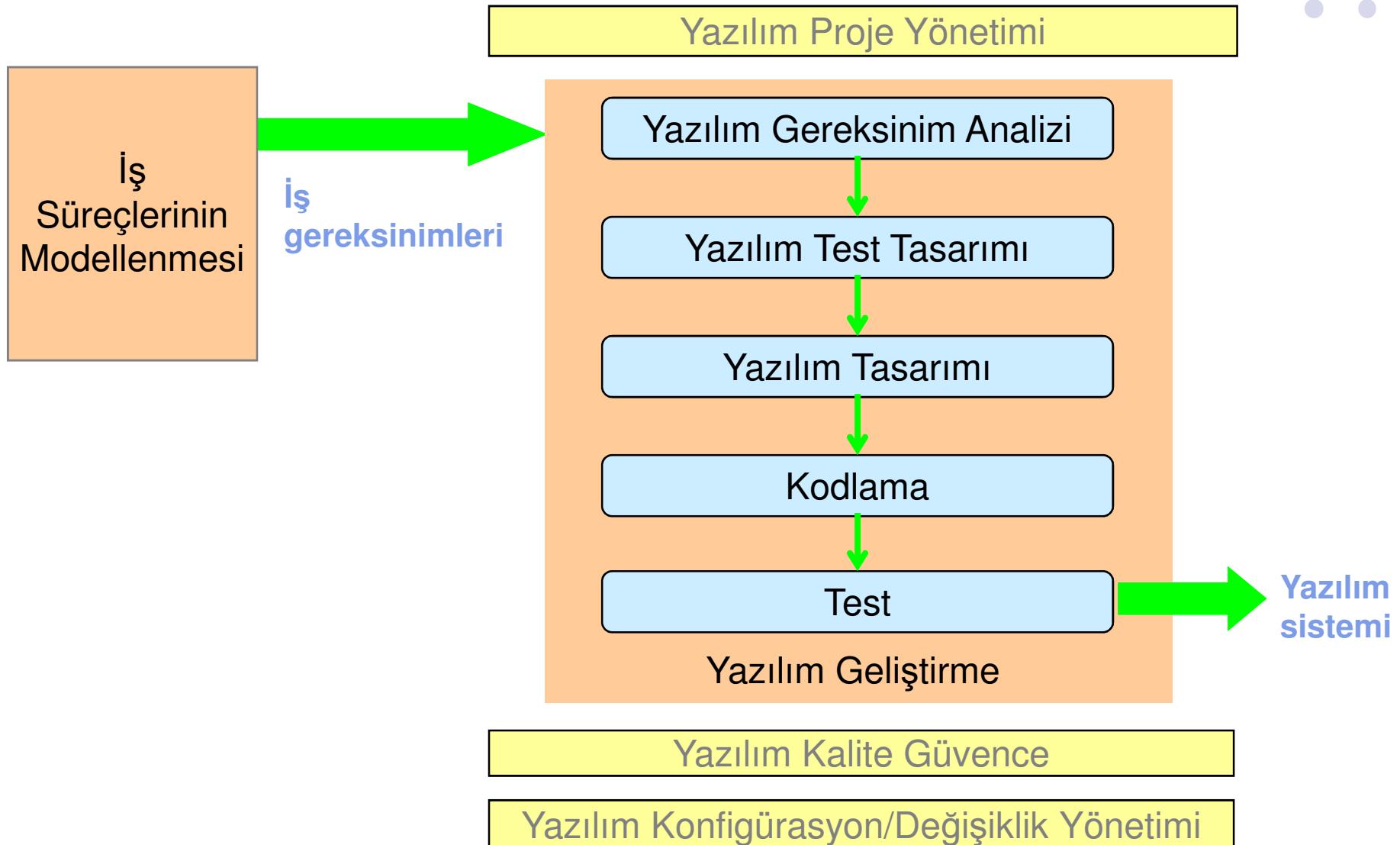
- Kalite güvence etkinliklerini gerçekleştirmek için kullanılacak kalite standartları, metodolojileri, prosedürleri ve araçları
- Sözleşme gözden geçirme ve koordinasyonu için prosedürler
- Kalite kayıtlarının belirlenmesi, toplanması, doldurulması, bakımı ve imhası için prosedürler
- Kalite güvence etkinliklerinin uygulamasına yönelik kaynaklar, takvim ve sorumluluklar
- Yazılım Doğrulama, Yazılım Geçerleme, Yazılım Gözden Geçirme, Yazılım Denetleme ve Yazılım Problem Çözümleme gibi destek süreçlerinden seçilen etkinlik ve görevler

Yazılım Yaşam Döngüsü Boyunca Kalite Güvence

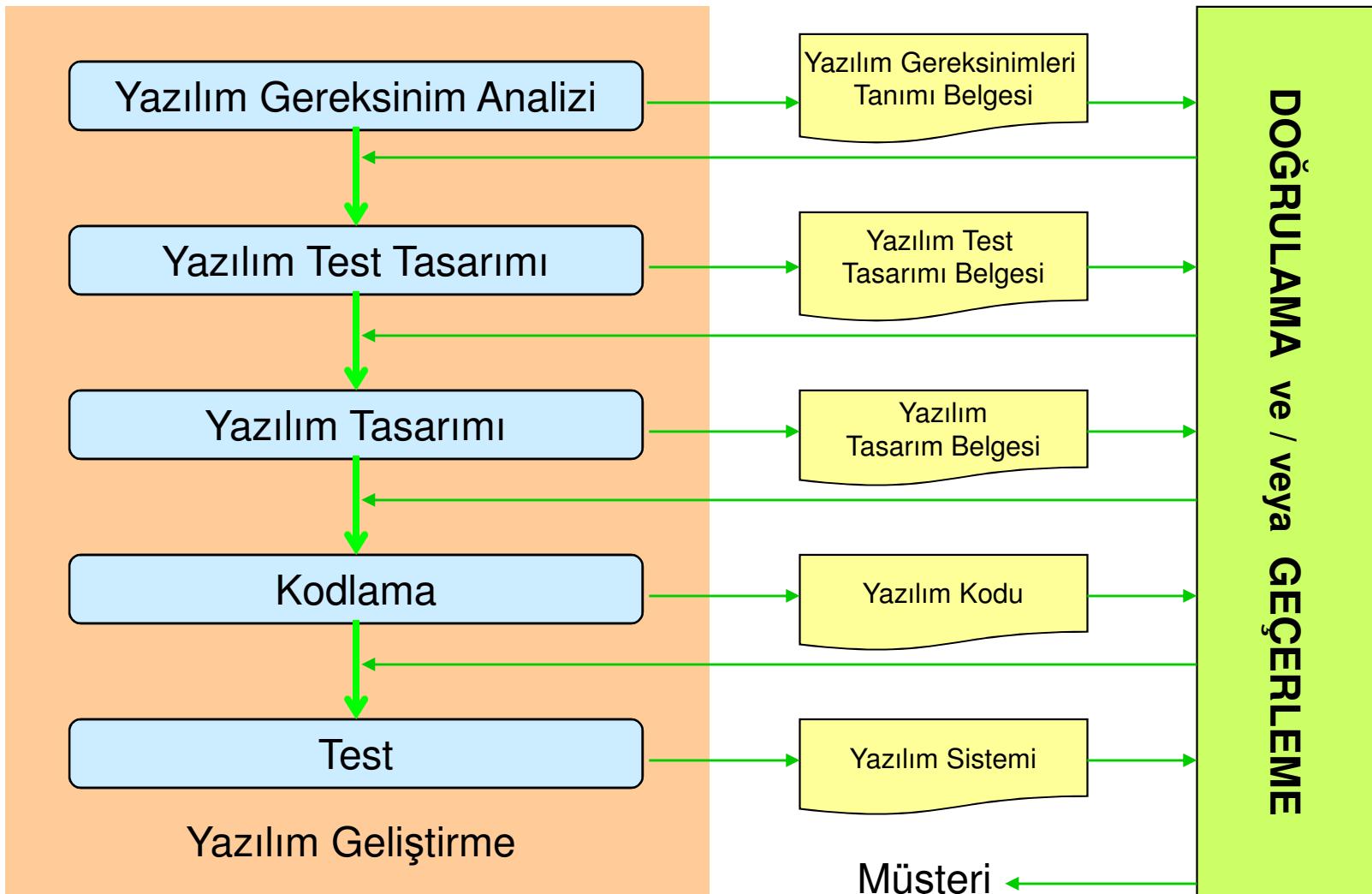




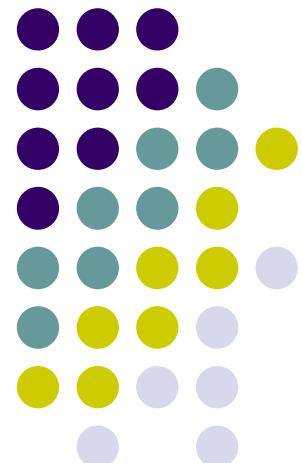
Temel Yazılım Geliştirme Yaşam Döngüsü



Yazılım Geliştirme ve Doğrulama ve Geçerleme



Kalite Maliyeti





Kalite Maliyeti (“Cost of Quality”)

- Ürünler ve onları üreten süreçler mükemmel olduğunda ortadan kalkacak toplam maliyettir.
- Üretim ve hizmet sektörlerinde yaygın olarak kullanılmaktadır.
- Süreçteki zayıf halkaların tespit edilerek kalite maliyetinin düşürülmesini hedefler.



Kalite Maliyeti Kategorileri

- Uygunluk maliyeti (“cost of non-conformance”)
 - Kaliteyi sağlamayı başaramadığımızda ödediğimiz bedeldir.
 - “Failure costs”: hatanın sebep olduğu maliyet
 - “Internal failures”: teslimattan önceki
 - “External failures”: teslimattan sonraki
- Uygunluğu sağlanmanın maliyeti (“cost of conformance”)
 - Kaliteyi sağlamak için ödediğimiz bedeldir.
 - “Appraisal costs”: kalitesizliğin tespit etmenin maliyeti
 - “Prevention costs”: kalitesizliği önlenmenin maliyeti



Hata Maliyeti (“Failure Costs”)

- Ürünün sağlaması beklenen özelliklerle ilişkilidir.
 - Müşteri ihtiyaçlarını karşılayamama
- İç ve dış hata maliyetleri toplanarak hesaplanır.
 - İç hata maliyeti: tekrar çalışma (“rework”), tekrar test (“re-test”)
 - Dış hata maliyeti: müşteriden dönen ürün, müşteri şikayetleri
- Kalite maliyetinin düşürülmesinde anahtar kategoridir.



Hata Bulma Maliyeti (“Appraisal Costs”)

- Ürün veya hizmetin, kalite standartlarına ve performans gereksinimlerine uyduğunu ölçmek, değerlendirmek veya denetlemek ile ilgilidir.
- Hata giderme etkinlikleri:
 - Gözden geçirmeler, ürün denetlemeler, beta testler

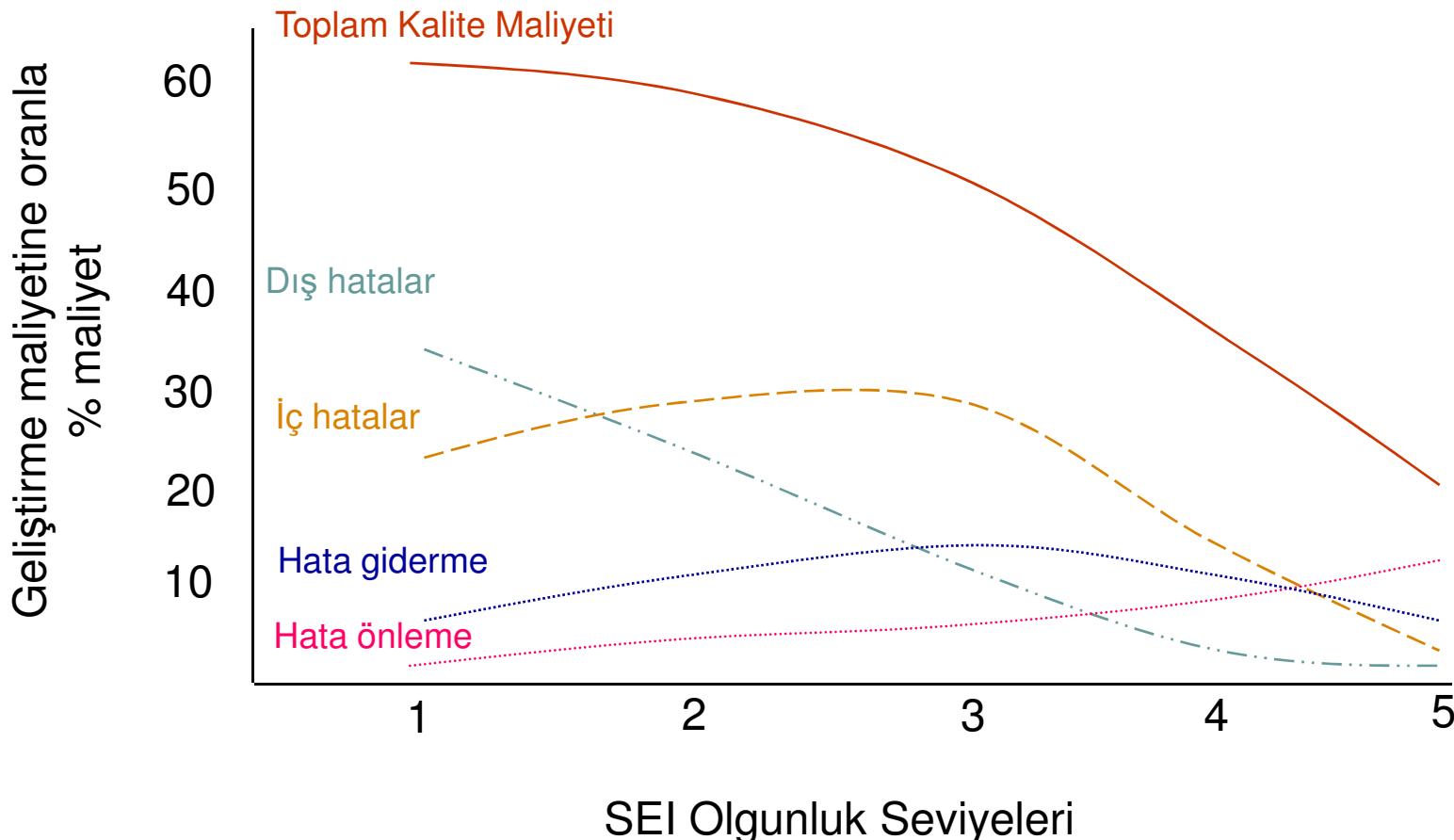


Hata Önleme Maliyeti (“Prevention Costs”)

- Bir ürün veya hizmetin hata maliyetini (“failure costs”) ve hata giderme maliyetini (“appraisal costs”) en aza indirmek üzere tasarlanmış etkinliklerle ilişkilidir.
 - Hata önleme yaklaşımlarının keşfi, uygulaması ve iyileştirilmesi için yatırım gerektirir.
- Hata önleme etkinlikleri:
 - Eğitim, araç ve yöntemler, kalite planlama, süreç yönetimi



Kurumsal Olgunluk ve Kalite Maliyeti

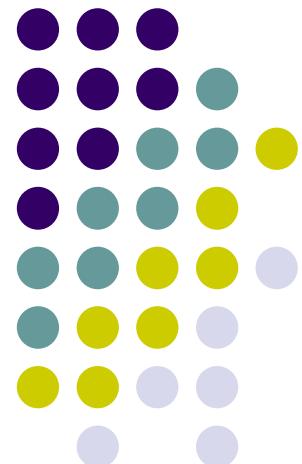




Donanım vs. Yazılım Kalite Maliyeti

- Donanım kalite maliyeti dağılımı, tasarım sonrasında yoğunlaşır.
 - Tasarım : %10
 - Yıpranma : %60
 - Üretim : %30
- Yazılım kalite maliyeti, geliştirme aşamasında yoğunlaşır.
 - Geliştirme : %99
 - Yıpranma : %0
 - Üretim : %1

Yazılım Kalite Metrikleri





Yazılımın Kalitesi (Ürün, Dış Özellik)

- Hata-esaslı metrikler
 - Hata yoğunluğu (“defect density”) = Bilinen hatalar / ürün büyüklüğü
 - Sistem hasarı (“system spoilage”) = Müşteriye teslim sonrası bildirilen hataları düzeltmenin maliyeti / toplam proje maliyeti
- Bakım-yapılabilirlik (“maintainability”) metrikleri
 - Ortalama tamir zamanı (“mean time to repair” - MTTR)
- Güvenilirlik (“reliability”) metrikleri
 - Ortalama bozulma zamanı (“mean time to failure” - MTTF)
 - Bozulmalar arası ortalama zaman (“mean time between failures” - MTBF)
 - $MTBF = MTTF + MTTR$
 - Erişilebilirlik (“availability” - A)
 - $A = MTTF / [(MTTF + MTTR) * 100]$



Güvenilirlik İçin Kalite Gereksinimi: Örnek

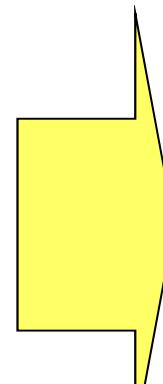
- Kalite gereksinimi:
 - Web-tabanlı ABC yazılımı, kullanıma açılmadan önce ve kabule esas olarak, 720 saat (30 gün) süre ile test edilecektir.
 - ABC yazılımı için, ortalama bozulma zamanı (MTTF) en az 96 saat (4 gün) ve ortalama tamir zamanı (MTTR) en çok 4 saat olacaktır.
- Soru: ABC yazılımının erişilebilirliği (“availability”) test süresi boyunca % kaç olmalıdır?

$$A = \text{MTTF} / [(\text{MTTF} + \text{MTTR}) * 100]$$

$$A = 96 / (96 + 4)$$

$$A = 96 / 100$$

$$A = \%96$$



720 saatlik test süresi boyunca, en çok 7 kez bozulabilir.



Gözden Geçirme ve Test Metrikleri

- Gözden geçirme veya test kazancı (“review or test yield”)
 - $100 * (\text{bulunan hatalar}) / (\text{bulunan hatalar} + \text{bulunamayan hatalar})$
 - Örnek:
 - Tasarım GG : 3 hata
 - Kodlama : 1 hata
 - Kod GG : 8 hata
 - Derleme : 6 hata
- Birim zamanda bulunan hata sayısı (hata sayısı / saat)
 - Gözden geçirme veya test zamanının etkinliğini ölçer.
 - Değerlendirmede hata yoğunluğu (hata sayısı / kod satır sayısı (KSS)) hesaba katılmalıdır.
- Birim zamanda gözden geçirilen ürün büyütüğü (KSS / saat)
 - Gözden geçirme veya test hızınızı ölçer.



Ödev: COSMIC İşlevsel Büyüklük Kestirimi

- Kütüphane Destek Sisteminin tanımını ve use-case diyagramını , ayrıca 6.dersin notlarını temel alarak; sistemin işlevsel büyüğünü, COSMIC İşlevsel Büyüklük Kestirim Yöntemine göre kabaca hesaplayın.



BM306

Yazılım Mühendisliği

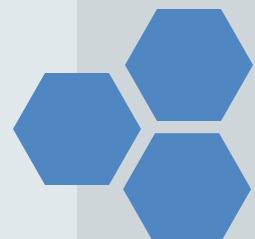


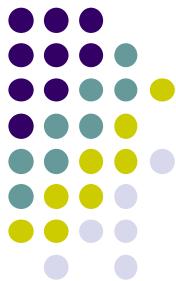
DERS 9

Yazılım Kalite Güvencesi

-devam

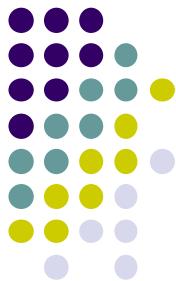
Dr. Öğr. Üyesi Hasan BADEM
hbadem@ksu.edu.tr



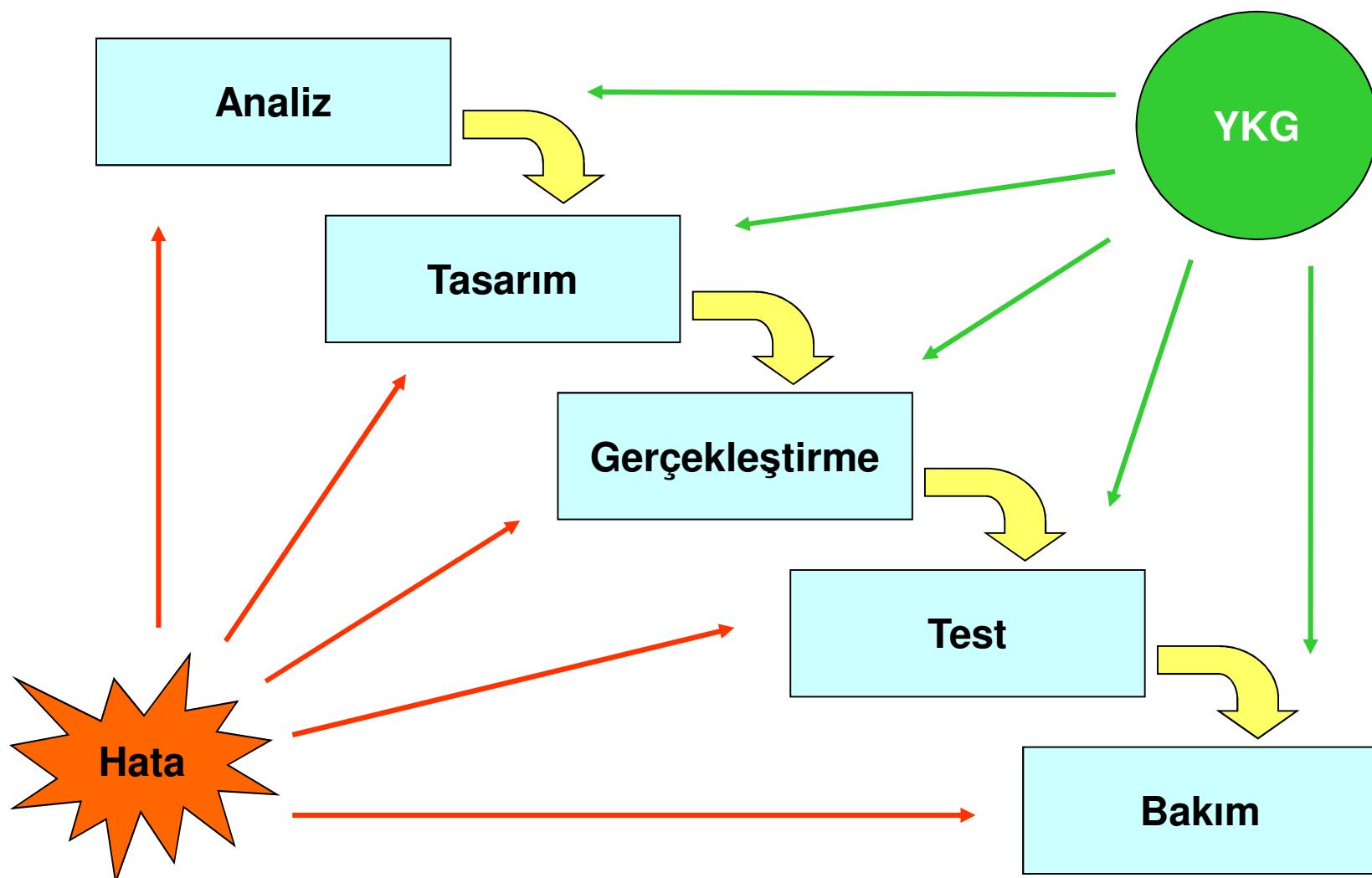


İçerik

- Yazılım test amacı ve kapsamı
- ISO 12207 yazılım doğrulama süreci
- ISO 12207 yazılım geçerleme süreci
- Doğrulama ve geçerleme türleri ve teknikleri
- Yazılım yaşam döngüsü boyunca test
- Yazılım test süreci ve çıktıları



Yazılım Kalite Güvence





Yazılım Kalite Güvence Öğeleri

- Yazılım Kalite Güvence (YKG), yazılım ürünün istenilen kalitede olmasını güvence altına alan ve geliştirme süreci boyunca uygulanan etkinlikleri içerir.
 - “Software quality assurance”
- Doğrulama ve Geçerleme (D&G), yazılım geliştirme sürecinin her aşamasındaki çıktıların, istenilen özelliklere uygunluğunu kontrol etmeye yarayan etkinliklerdir.
 - “Verification and validation”
- Test, bir üründeki hataların ortaya çıkarılması etkinliğidir.

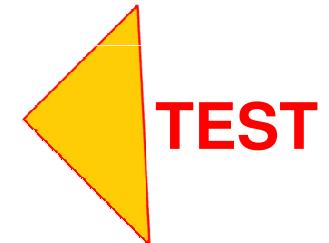
D&G, YKG'nin bir parçasıdır ve Test'i içerir.

YKG > D&G > Test



Yazılım Kalite Güvence: Yöntemler

- Statik yöntemler: Kodu çalıştırmadan yapılır.
 - Örnek: Gereksinim ve tasarım belgeleri ile kodun denetimi veya gözden geçirilmesi
 - İnceleme (“inspection”)
 - Gözden geçirme (“review”)
 - Denetleme (“audit”)
- Dinamik yöntemler: Kodu çalıştırarak yapılır.
 - Örnek: Ürünün veya bileşenlerinin gerçeğe yakın testi
 - Birim test (“unit test”)
 - Tümllestirme test (“integration test”)
 - Sistem test (“system test” / “functional test” / “qualification test”)
 - Kabul test (“acceptance test”)





Doğrulama ve Geçerleme

- **Doğrulama (“verification”):** Müşterinin istediği ürünü, doğru şekilde geliştirmeyi güvence altına alır.
 - Mühendislik adımlarının doğru uygulanıp uygulanmadığının kontrolüdür.
 - Her aşamada, bir önceki aşamada yapmayı taahhüt ettiklerimizi yaptık mı? Bir önceki belgede tanımladıklarımıza göre uyumsuzluk veya eksik var mı?
 - **“Are we building the product right?”**
- **Geçerleme (“validation”):** Müşterinin istediği, (doğru) ürünü geliştirmeyi güvence altına alır.
 - Mühendislik ürününün müşteri ihtiyaçlarına uyup uymadığının kontrolüdür.
 - Her aşamanın çıktısı müşteri gereksinimlerini karşılıyor mu?
 - **“Are we building the right product?”**



Test

- Bir programın davranışını; beklenen davranışa uymadığı durumları bulma amacı ile, sonsuz bir küme içinden sınırlı sayıda seçilen test vakalarını kullanarak, dinamik yöntemlerle sınama işlemidir. [SWEBOK 2004].
 - Beklenen : Tanımlanmış gereksinimlere uyan
 - Sınırlı : Yeterli sayıda
 - Seçilmiş : Uygun test vakaları
 - Dinamik : Kod çalıştırılarak

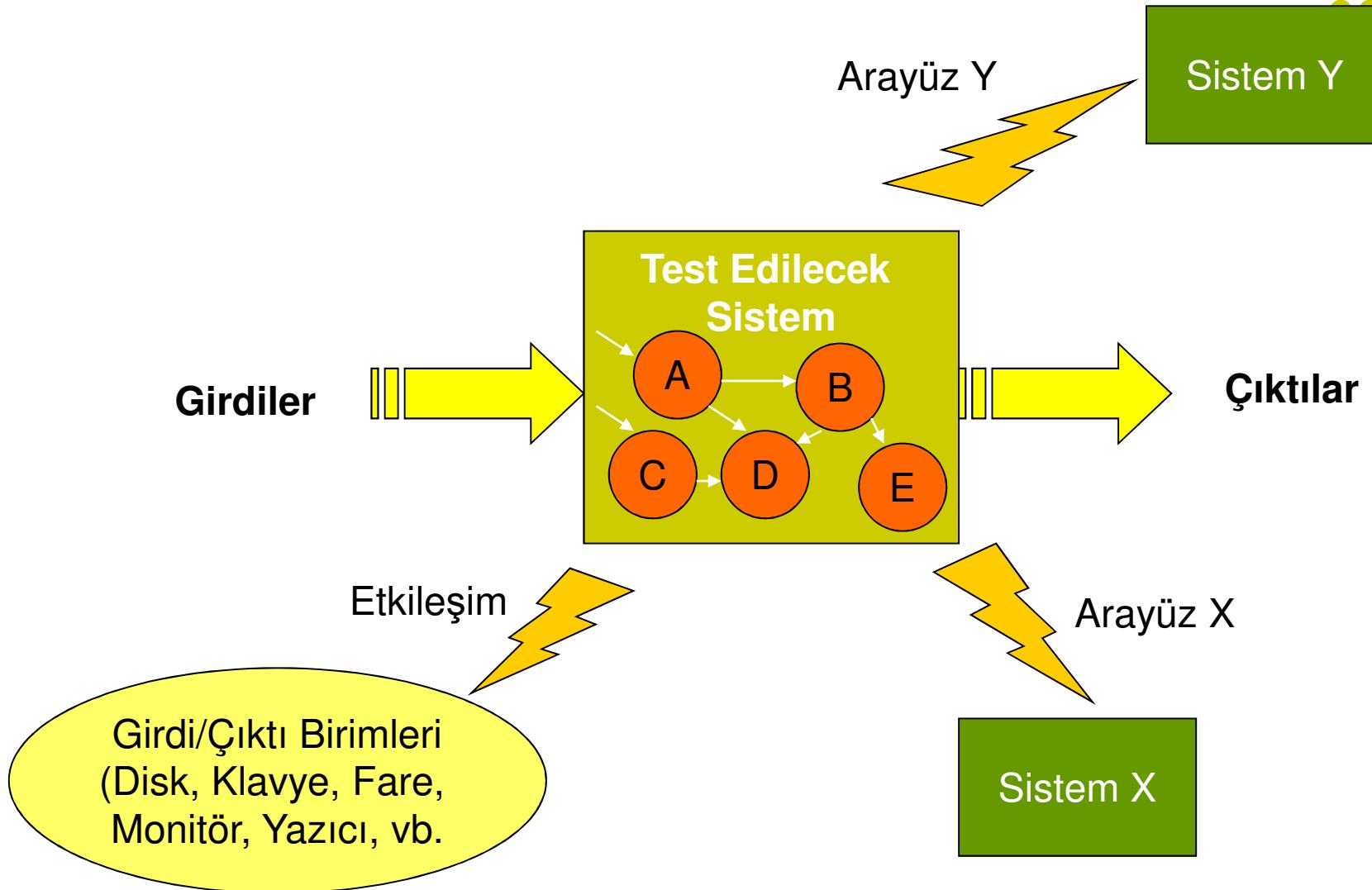


Test: Amaç ve Kapsam

- Testin amacı hata bulmaktır. Hata bulunamamışsa, test süreci başarısız sonuçlanmıştır.
- Başarılı bir test süreci sadece beklenen senaryoları değil, olağan dışı senaryoları da kapsamalıdır.
- Test süreci, gereksinimlerin belirlenmesi aşamasında başlar.

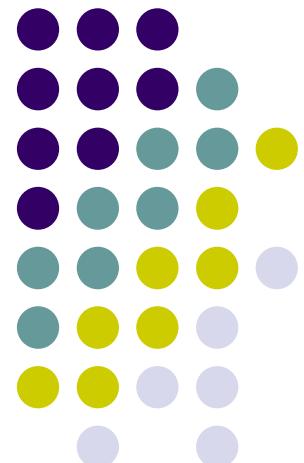


Test Edilecek Sistem

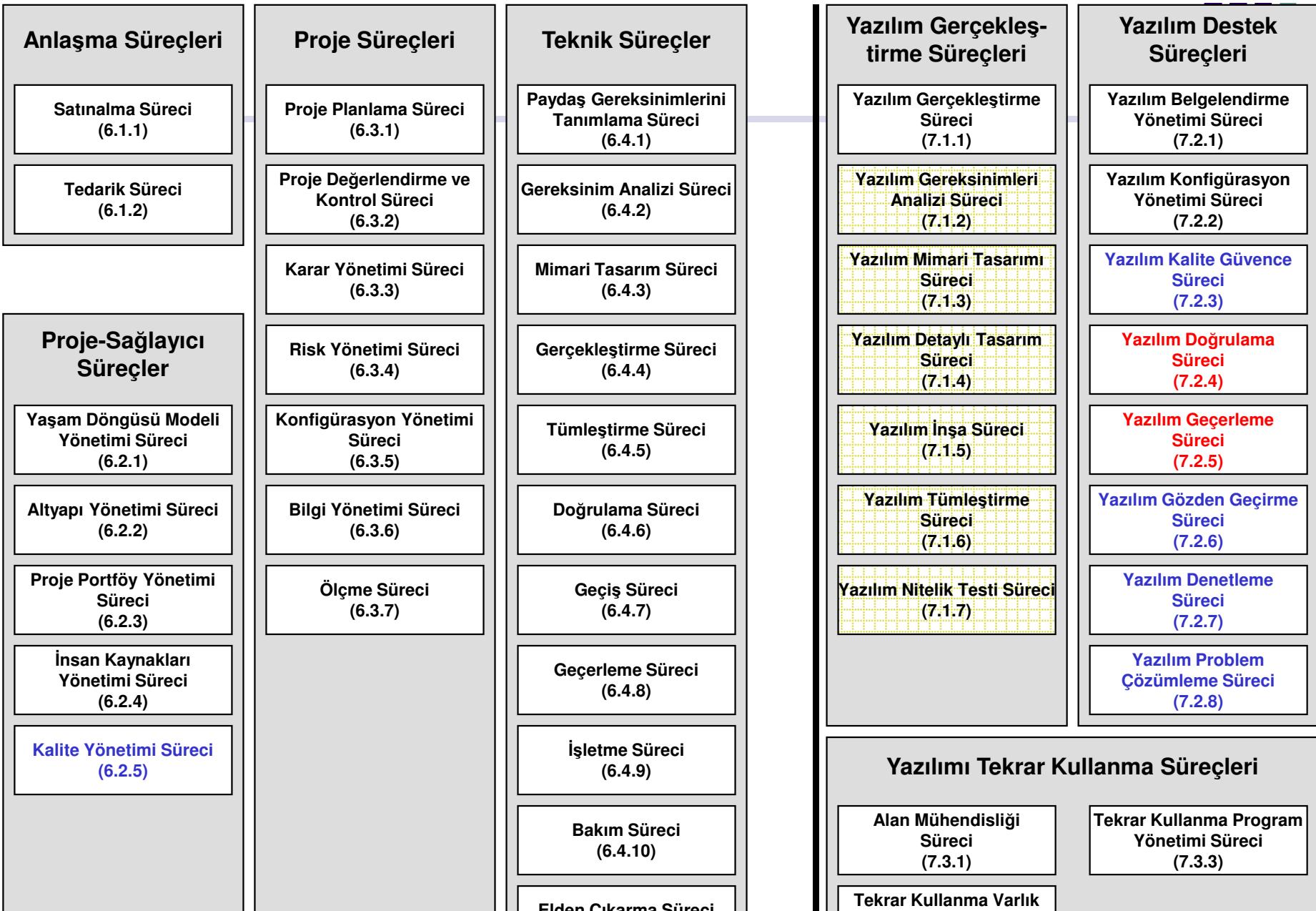


ISO/IEC 12207

Yazılım Doğrulama Süreci



Sistem Kapsamlı Süreçler





Yazılım Doğrulama Süreci (7.2.4)

- Amaç:
 - Sürece veya projeye ait her yazılım iş ürününün ve/veya hizmetinin, tanımlı gereksinimleri karşıladığı onaylamak
- Çıktılar:
 - Doğrulama stratejisi geliştirilir ve uygulanır.
 - Gereken yazılım iş ürünlerinin tümü için doğrulama kriterleri belirlenir.
 - Gereken doğrulama etkinlikleri uygulanır.
 - Hatalar tespit edilir ve kaydedilir.
 - Doğrulama etkinliklerinin sonuçları, müşterinin ve işe dahil olan diğer birimlerin erişimine açılır.



Yazılım Doğrulama Süreci – Etkinlikler ve Görevler

7.2.4.3.1 – Süreç uygulaması

1. Projenin doğrulama işgürünü garanti edip edemeyeceği ve bu işgürünün ne derecede kurumsal bağımsızlık gerektirdiği belirlenecektir. Proje gereksinimleri; kritikliğe göre analiz edilecektir.
2. Proje doğrulama işgürünü garanti ediyorsa, bir doğrulama süreci oluşturulacaktır.
3. Proje bağımsız bir doğrulama işgürünü garanti ediyorsa, doğrulamayı gerçekleştirmekten sorumlu olacak yetkin bir birim seçilecektir.
4. Projenin kapsamına, büyülüğüne, karmaşıklığına ve kritikliğine göre; hedef yaşam döngüsü etkinlikleri ve doğrulama gerektiren yazılım ürünleri belirlenecektir. Yapılacak doğrulamaları destekleyecek yöntemler, teknikler ve araçlar seçilecektir.
5. Belirlenen doğrulama görevleri esas alınarak, doğrulama planı geliştirilecek ve belgelendirilecektir.
6. Doğrulama planı uygulanacaktır. Doğrulama sırasında belirlenen problemler ve uygunsuzluklar, **Yazılım Problem Çözümleme** sürecine girdi olacaktır. Doğrulama etkinliklerinin sonuçları, müşterinin ve işe dahil olan diğer birimlerin erişimine sunulacaktır.



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

7.2.4.3.2 – Doğrulama

1. Gereksinim doğrulama
2. Tasarım doğrulama
3. Kod doğrulama
4. Tümleştirme doğrulama
5. Belgelendirme doğrulama



Doğrulama Planı

- Doğrulamaya esas yaşam döngüsü süreçleri ve yazılım ürünleri
- Her yaşam döngüsü süreci veya yazılım ürünü için gereken doğrulama görevleri
- Doğrulama yöntem, teknik ve araçlarının tanımı
- Kaynaklar
- Sorumluluklar
- Takvim
- Doğrulama raporlarının satın alan kuruma veya işe dahil olan diğer birimlere iletilmesi için prosedürler



Doğrulama Kriterleri - 1

- **Gereksinim doğrulama kriterleri:**
 - Sistem gereksinimleri tutarlı, gerçekleştirilebilir, test edilebilirdir.
 - Sistem gereksinimleri; tasarım kriterlerine göre; donanım öğelerine, yazılım öğelerine ve manüel işlemlere uygun şekilde atanmıştır.
 - Yazılım gereksinimleri tutarlı, gerçekleştirilebilir, test edilebilirdir ve sistem gereksinimlerini net olarak yansıtmaktadır.
 - Güvenliğe ve kritikliğe ilişkin yazılım gereksinimleri, uygun yöntemlerle gösterildiği gibi doğrudur.
- **Tasarım doğrulama kriterleri:**
 - Tasarım doğru, gereksinimlerle tutarlı ve gereksinimlere izlenebilirdir.
 - Tasarım; olayları, girdileri, çıktıları, arayüzleri ve mantık akışını uygun sırada gerçekleştirir.
 - Seçilen tasarım, gereksinimlerden türetilebilir.
 - Tasarım; güvenlik ve kritikliğe ilişkin gereksinimleri, uygun yöntemlerle gösterildiği gibi gerçekleştirir.

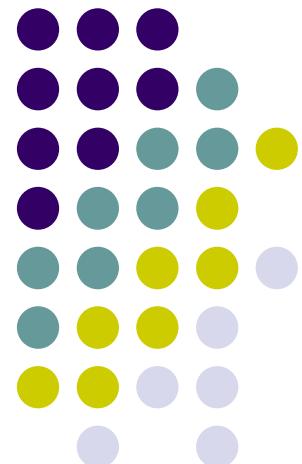


Doğrulama Kriterleri - 2

- **Kod doğrulama kriterleri:**
 - Kod; tasarım ve gereksinimlere izlenebilir, test edilebilir ve doğrudur.
 - Kod, gereksinimlere ve kodlama standartlarına uygundur.
 - Kod; uygun olay sırasını, tutarlı arayüzleri, doğru veri ve kontrol akışını gerçekleştirir.
 - Seçilen kod, tasarımdan veya gereksinimlerden türetilenbilir.
 - Kod; güvenliğe ve kritikliğe ilişkin gereksinimleri, uygun yöntemlerle gösterildiği gibi gerçekleştirir.
- **Tümleştirme doğrulama kriterleri:**
 - Yazılım bileşenleri ve her yazılım öğesinin birimleri, yazılım öğesine tam ve doğru olarak tümleştirilmiştir.
 - Sistemin donanım öğeleri, yazılım öğeleri ve manüel işlemleri, sisteme tam ve doğru olarak tümleştirilmiştir.
 - Tümleştirme görevleri, bir tümleştirme planına uygun olarak gerçekleştirilmiştir.
- **Belgelendirme doğrulama kriterleri:**
 - Belgelendirme uygun, tam ve tutarlıdır.
 - Belgeler zamanında hazırlanmıştır.
 - Belgelerin konfigürasyon yönetimi, tanımlı prosedürlere göre yapılmıştır.

ISO/IEC 12207

Yazılım Geçerleme Süreci





Yazılım Geçerleme Süreci (7.2.5)

- Amaç:
 - Yazılım iş ürününün, amaca özel kullanımına yönelik gereksinimleri karşıladığı 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 işgürünü garanti edip edemeyeceği ve bu işgürünün ne derecede kurumsal bağımsızlık gerektirdiği belirlenecektir.
2. Proje geçerleme işgürünü garanti ediyorsa, bir geçerleme süreci oluşturulacaktır. Geçerleme görevleri; ilişkili yöntemler, teknikler ve araçlarla birlikte, seçilecektir.
3. Proje bağımsız bir geçerleme işgürünü garanti ediyorsa, gerçeklemeyi gerçekleştirmekten sorumlu olacak yetkin bir birim seçilecektir. Birimin gerçeklemeyi 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

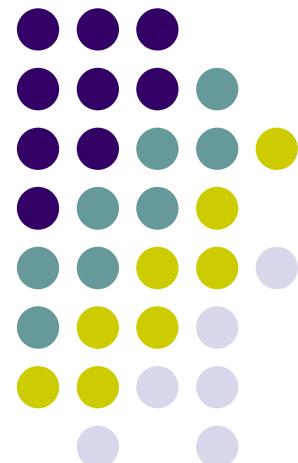
1. Seçilen test gereksinimleri, test durumları ve test tanımları hazırlanacaktır.
2. Tanımlanan test gereksinimlerinin, test durumlarının ve test tanımlarının, amaçlanan kullanıma özel gereksinimleri yansındığı güvence edilecektir.
3. Tanımlanan testler gerçekleştirilecektir.
4. Yazılım ürününün amaçlanan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



Geçerleme Planı

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

Doğrulama ve Geçerleme Türleri ve Teknikleri





Doğrulama ve Geçerleme Türleri

- IEEE Std 1012-2004 IEEE Standard for Software Verification and Validation (V&V), 6 tür doğrulama ve geçerleme etkinliği tanımlar:
 - Kavram doğrulama ve geçerleme (“concept V&V”)
 - Gereksinim doğrulama ve geçerleme (“requirements V&V”)
 - Tasarım doğrulama ve geçerleme (“design V&V”)
 - Gerçekleştirme doğrulama ve geçerleme (“implementation V&V”)
 - Test doğrulama ve geçerleme (“test V&V”)
 - Kuruluş ve kontrol doğrulama ve geçerleme (“installation and checkout V&V”)



Kavram D&G

- The concept activity represents the delineation of a specific implementation solution to solve the user's problem.
- During the concept activity, the system architecture is selected and system requirements are allocated to hardware, software, and user interface components.
 - The Concept V&V activity addresses system architectural design and system requirements analysis.
- The objective of Concept V&V is to verify the allocation of system requirements, validate the selected solution, and ensure that no false assumptions have been incorporated in the solution.



Kavram D&G: Görevler

- Concept documentation evaluation
- Criticality analysis
- Hardware/software/user requirements allocation analysis
- Traceability analysis
- Hazard analysis
- Security analysis
- Risk analysis



Gereksinim D&G

- The Requirements V&V activity addresses software requirements analysis of the functional and performance requirements, interfaces external to the software, and requirements for qualification, safety and security, human factors engineering, data definitions, user documentation for the software, installation and acceptance, user operation and execution, and user maintenance.
- V&V test planning begins during the Requirements V&V activity and spans several V&V activities.
- The objective of Requirements V&V is to ensure the correctness, completeness, accuracy, testability, and consistency of the system software requirements.



Gereksinim D&G: Görevler

- Traceability analysis
- Software requirements evaluation
- Interface analysis
- Criticality analysis
- System V&V test plan generation
- Acceptance V&V test plan generation
- Configuration management assessment
- Hazard analysis
- Security analysis
- Risk analysis



Tasarım D&G

- In software design, software requirements are transformed into an architecture and a detailed design for each software component.
 - The design includes databases and system interfaces (e.g., hardware, operator/user, software components, and subsystems).
- The Design V&V activity addresses software architectural design and software detailed design.
- V&V test planning continues during the Design V&V activity.
- The objective of Design V&V is to demonstrate that the design is a correct, accurate, and complete transformation of the software requirements and that no unintended features are introduced.



Tasarım D&G: Görevler

- Traceability analysis
- Software design evaluation
- Interface analysis
- Criticality analysis
- Component V&V test plan generation
- Integration V&V test plan generation
- Component V&V test design generation
- Integration V&V test design generation
- System V&V test design generation
- Acceptance V&V test design generation
- Hazard analysis
- Security analysis
- Risk analysis



Gerçekleştirme D&G

- In software implementation, the system design is transformed into code, database structures, and related machine executable representations.
- The Implementation V&V activity addresses software coding and testing, including the incorporation of reused software products.
- The objective of Implementation V&V is to verify and validate that these transformations are correct, accurate, and complete.



Gerçekleştirme D&G: Görevler

- Traceability analysis
- Source code and source code documentation evaluation
- Interface analysis
- Criticality analysis
- Component V&V test case generation
- Integration V&V test case generation
- System V&V test case generation
- Acceptance V&V test case generation
- Component V&V test procedure generation
- Integration V&V test procedure generation
- System V&V test procedure generation
- Component V&V test execution
- Hazard analysis
- Security analysis
- Risk analysis



Test D&G

- Testing includes software testing, software integration testing, software qualification testing, system integration testing, and system qualification testing.
- The objective of Test V&V is to ensure that the software requirements and system requirements allocated to software are validated by execution of integration, system, and acceptance tests.



Test D&G: Görevler

- Traceability analysis
- Acceptance V&V test procedure generation
- Integration V&V test execution
- System V&V test execution
- Acceptance V&V test execution
- Hazard analysis
- Security analysis
- Risk analysis



Kuruluş ve Kontrol D&G

- In installation and checkout, the software product is installed and tested in the target environment.
- The installation and checkout V&V activity supports the software system installation activities.
- The objective of Installation and checkout V&V is to verify and validate the correctness of the software installation in the target environment.



Kuruluş ve Kontrol D&G: Görevler

- Installation configuration audit
- Installation checkout
- Hazard analysis
- Security analysis
- Risk analysis
- V&V final report generation



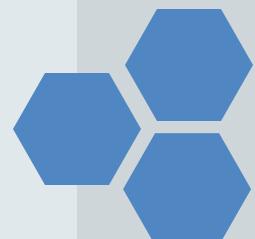
BM306

Yazılım Mühendisliği



DERS 10
Yazılım Gözden Geçirme
ve Testi

Dr. Öğr. Üyesi Hasan BADEM
hbadem@ksu.edu.tr



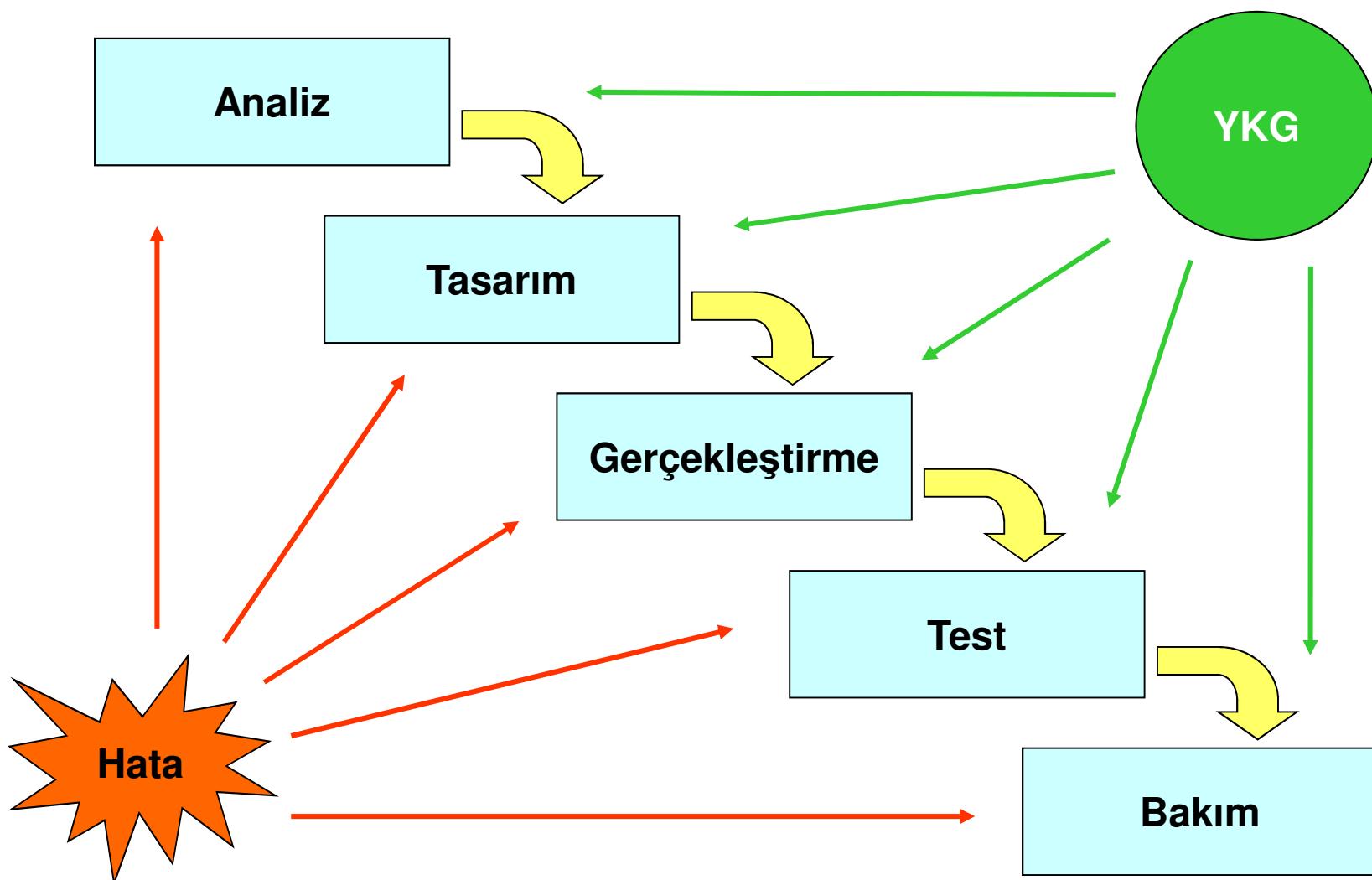


İçerik

- Yazılım gözden geçirme
 - ISO 12207 Yazılım gözden geçirme süreci
- Yazılım test
 - Yazılım test amacı ve kapsamı
 - Yazılım yaşam döngüsü boyunca test
 - Yazılım test süreci ve çıktıları



Yazılım Kalite Güvence





Yazılım Kalite Güvence Öğeleri

- Yazılım Kalite Güvence (YKG), yazılım ürünün istenilen kalitede olmasını güvence altına alan ve geliştirme süreci boyunca uygulanan etkinlikleri içerir.
 - “Software quality assurance”
- Doğrulama ve Geçerleme (D&G), yazılım geliştirme sürecinin her aşamasındaki çıktıların, istenilen özelliklere uygunluğunu kontrol etmeye yarayan etkinliklerdir.
 - “Verification and validation”
- Test, bir üründeki hataların ortaya çıkarılması etkinliğidir.

D&G, YKG'nin bir parçasıdır ve Test'i içerir.

YKG > D&G > Test



Yazılım Kalite Güvence: Yöntemler

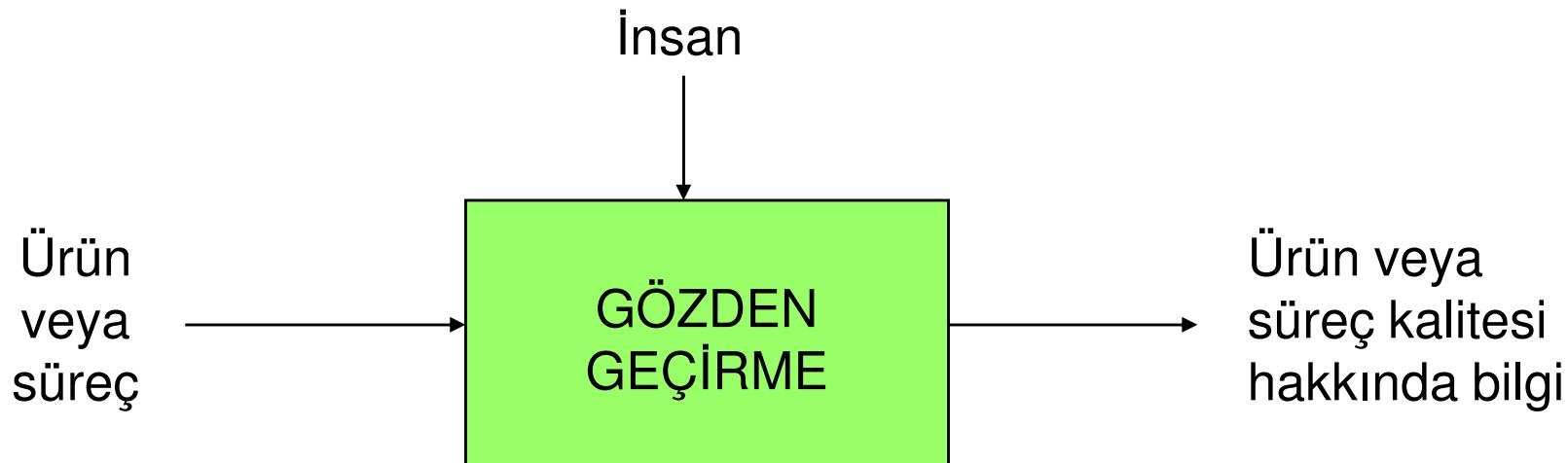
- Statik yöntemler: Kodu çalıştırmadan yapılır.
 - Örnek: Gereksinim ve tasarım belgeleri ile kodun denetimi veya gözden geçirilmesi
 - İnceleme (“inspection”)
 - Gözden geçirme (“review”)
 - Denetleme (“audit”)
- Dinamik yöntemler: Kodu çalıştırarak yapılır.
 - Örnek: Ürünün veya bileşenlerinin gerçeğe yakın testi
 - Birim test (“unit test”)
 - Tümleştirme test (“integration test”)
 - Sistem test (“system test” / “functional test” / “qualification test”)
 - Kabul test (“acceptance test”)



Gözden Geçirme



Gözden Geçirme



[IEEE Std 1028-2008: IEEE Standard for Software Reviews and Audits](#)

“A process or meeting during which a software product, set of software products, or a software process is presented to project personnel, managers, users, customers, user representatives, auditors or other interested parties for examination, comment or approval.”



Neden Gözden Geçirme?

- Ürününün veya sürecin farklı bakış açılarıyla sistematik olarak değerlendirilmesini sağlar.
- Proje zamanlamasını ve maliyetini iyileştirir.
- Test etkinliğini destekler ve maliyetini düşürür.
- Yatırım geri dönüşü yüksektir.
- Bir çeşit eğitim yöntemidir.



Gözden Geçirme Türleri

- IEEE Std 1028-2008 beş tür gözden geçirme tanımlar:
 - Management reviews (“yönetim gözden geçirme”)
 - Technical reviews (“teknik gözden geçirme”)
 - Inspections (“detaylı inceleme”)
 - Walkthroughs (“genel gözden geçirme”)
 - Audits (“denetleme”)

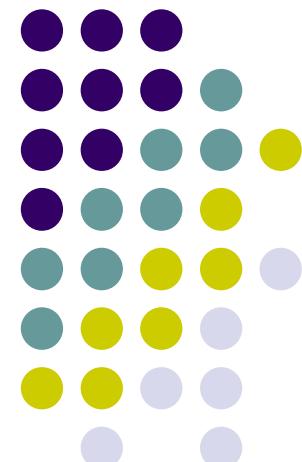


Eş Gözden Geçirme (“Peer Review”)

- Diğer bir tür gözden geçirme; eş gözden geçirmedir.
 - Yetkin bir kişi ürünü/dokümanı gözden geçirir.
 - Amaç, maliyeti büyümeden hataları yakalamaktır.
- Eş gözden geçirme; analiz dokümanı, tasarım dokümanı, kod, vb. için uygulanabilir.
 - Önce ekip içinde eş gözden geçirme, sonra diğer paydaşlarla birlikte genel gözden geçirme de planlanabilir.

ISO/IEC 12207

Yazılım Gözden Geçirme Süreci





Yazılım Gözden Geçirme Süreci (7.2.6)

- Amaç:
 - Anlaşmanın hedeflerine ve paydaşları tatmin edecek bir ürünün geliştirilmesini güvence altına almak için yapılması gerekenlere göre, gelişim hakkında paydaşlarla ortak bir anlayışı oluşturmak ve sürdürmek
- Çıktılar:
 - Projenin ihtiyaçlarına göre yönetim ve teknik gözden geçirmeler yapılır.
 - Bir sürecin etkinliğine ait durum ve ürünler, gözden geçirme etkinlikleri ile değerlendirilir.
 - Gözden geçirme sonuçları, etkilenen tüm birimlere duyurulur.
 - Gözden geçirmeler sonucunda oluşan düzeltici faaliyetler, kapanana kadar izlenir.
 - Riskler ve problemler belirlenir ve kaydedilir.



Yazılım Gözden Geçirme Süreci – Etkinlikler ve Görevler

7.2.6.3.1 – Süreç uygulaması

1. Proje planlarında tanımlandığı gibi, belirlenen kilometre-taşlarında periyodik gözden geçirmeler yapılacaktır. Bunların dışında yapılacak gözden geçirmeler, ihtiyaç duyulması halinde paydaşlar tarafından belirlenecektir.
2. Gözden geçirmenin yapılması için gereken tüm kaynaklar (çalışanlar, yer, donanım, yazılım ve araçlar) sağlanacaktır.
3. Gözden geçirmeye katılanlar, her gözden geçirmede; toplantı gündemi, yazılım ürünleri, gözden geçirilecek problemler, kapsam ve prosedürler, gözden geçirmeye giriş ve çıkış kriterleri konularında fikir birliğine varmalıdır.
4. Gözden geçirme sırasında tespit edilen problemler ve uygunsuzluklar kaydedilecek ve Yazılım Problem Çözümleme sürecine girdi olacaktır.
5. Gözden geçirme sonuçları belgelendirilecek ve duyurulacaktır.
6. Gözden geçirmeye katılanlar, gözden geçirmenin çıktıları ve düzeltici faaliyetlerin sorumlulukları ve kapatılma kriterleri hakkında fikir birliğine varacaktır.



Yazılım Gözden Geçirme Süreci – Etkinlikler ve Görevler (.. devamı)

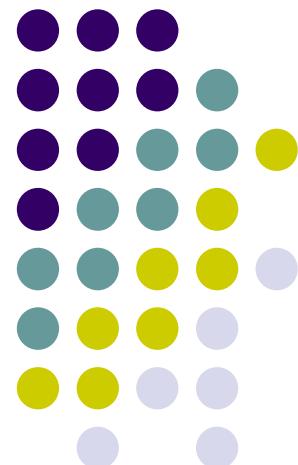
7.2.6.3.2 – Proje yönetim gözden geçirmeleri

1. Proje gelişiminin durumu; proje planlarına, takvime, standartlara ve yönergelere kıyasla değerlendirilecektir. Değerlendirmenin çıktısı, ilişkili yönetim tarafından dikkate alınmalı ve aşağıdakileri sağlamalıdır:
 - Etkinliklerin plana göre gelişmesini güvence etmek,
 - Kaynakları uygun şekilde atayarak projenin genel kontrolünü korumak,
 - Projenin yönünü değiştirmek veya alternatif planlama için ihtiyacı belirlemek,
 - Proje başarısını olumsuz etkileyebilecek riskleri değerlendirmek ve yönetmek.

7.2.6.3.3 – Teknik gözden geçirmeler

1. Yazılım ürünlerini veya hizmetlerini değerlendirmek için teknik gözden geçirmeler yapılacak ve aşağıdakiler hakkında kanıtlar sağlayacaktır:
 - Ürün veya hizmet tamdır.
 - Ürün veya hizmet standartlara ve tanımlara uygundur.
 - Ürün veya hizmete değişiklikler uygun şekilde gerçekleştirilir ve yalnız Konfigürasyon Yönetimi süreci ile tanımlanmış alanları etkiler.
 - Ürün veya hizmet tanımlı takvime uymaktadır.
 - Ürün veya hizmet, planlanmış bir sonraki etkinlik için hazırlıdır.
 - Ürün veya hizmetin geliştirilmesi, işletilmesi ve bakımı; projenin tanımlı planlarına, takvimine, standartlarına ve talimatlarına göre yapılmaktadır.

Gözden Geçirme Süreci



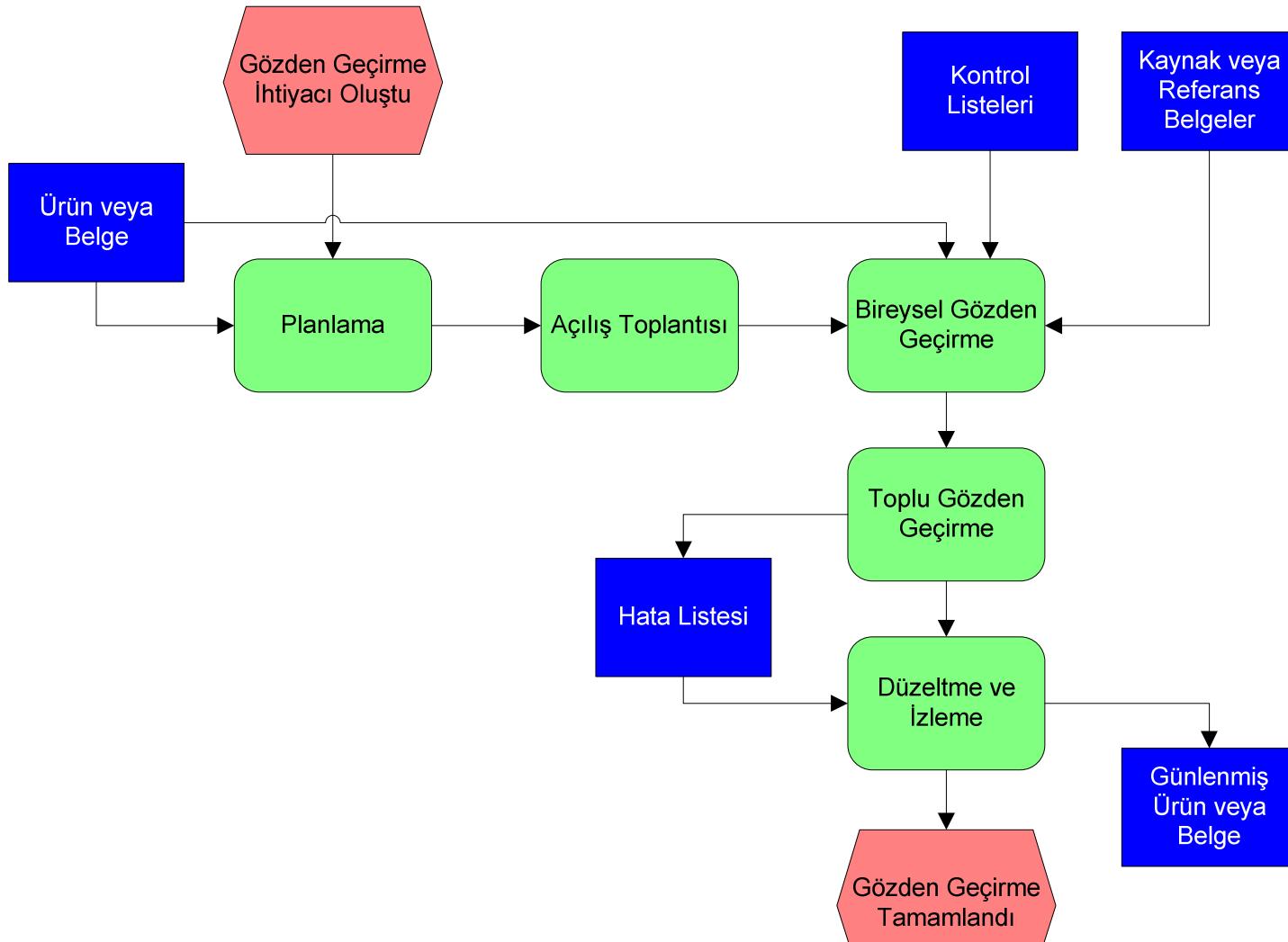


Gözden Geçirme Süreci

- Roller
 - Gözden geçirme lideri (“review leader”)
 - Gözden geçirici (“reviewer”)
 - Kayıt sorumlusu (“recorder”)
 - Yazar (“author”)
- Adımlar
 - Planlama (“planning”)
 - Açıılış toplantısı (“kickoff meeting”)
 - Bireysel gözden geçirme (“individual checking”)
 - Toplu gözden geçirme (“logging meeting”)
 - Düzeltme ve izleme (“rework and follow up”)



Gözden Geçirme Süreci





Gözden Geçirme Kontrol Listeleri

- Gözden geçirilecek her tür doküman için gözden geçirme kriterlerini tanımlar.
 - Analiz dokümanı, tasarım dokümanı, kod, proje planı, kalite planı, vb.
- Gözden geçirmeyi bir kontrol listesi üzerinden yapmak, gözden geçirmenin etkinliğini arttırmır.



Yazılım Gereksinimleri Tanımı Belgesi

Örnek Kontrol Listesi

	(E=Evet, H=Hayır, U=Uygulanamaz)	(E / H / U)	Açıklamalar
Standardlara Uygunluk			
Doküman için standartlar/kılavuzlar ve isimlendirme kuralları belirlenmiş mi?			
Doküman formatı tanımlı standart/kılavuz ile uyumlu mu?			
Doküman boyunca standartlar ve isimlendirme kuralları takip edilmiş mi?			
Doküman İçeriği			
Üst düzey sistem genel bakışı var mı?			
Üst düzey sistem diyagramları iç ve dış arayüzleri ve veri akışlarını gösteriyor mu?			
Sistemin fonksiyonel akışı açıkça ve tam olarak tanımlanmış mı?			
Yazılım çevresi (ör, donanım, yazılım kaynakları, kullanıcılar) tanımlanmış mı?			
Tüm referans dokümanları listelenmiş mi?			
Tüm tanımlamalar, kısadalar ve kısaltmalar konmuş mu?			
Kullanıcı karakteristikleri tanımlanmış mı?			
Genel tasarım ve uygulama kısıtları not edilmiş mi?			
Uygulamayı etkileyen genel varsayımlar belirtilmiş mi?			
Hafıza gereksinimleri sağlanmış mı?			
Zamanlama ve hafıza limitleri donanım kısıtlıyla uyumlu mu?			
Yazılım performansı üzerindeki tüm limitler ve sınırlamalar tanımlanmış mı?			
Her bir fonksiyon ayrı ayrı tanımlanmış mı?			
Her bir fonksiyon için herhangi yazılım sınırlamaları tartışılmış mı?			
Emniyet-kritik yazılım gereksinimleri tanımlanmış mı?			
Güvenlik gereksinimleri tanımlanmış mı?			
Gizlilik gereksinimleri tanımlanmış mı?			
Yazılım kalite gereksinimleri (güvenilirlik, taşınabilirlik, tekrar kullanabilirlik, bakılabilirlik) tanımlanmış mı?			
Çevresel gereksinimler ve durumlar tanımlanmış mı?			
Tüm paketleme gereksinimleri tanımlanmış mı?			
Tüm teslim gereksinimleri tanımlanmış mı?			
İşlemsel bilgisayar donanımı için gereksinimler sağlanmış mı?			
Bilgisayar yazılım kaynakları (işletim sistemi, ağ yazılımı, veritabanları, test yazılımı) tanımlanmış mı?			
Tam bütünlştirme, test ve kabul kriterleri belirlenmiş mi?			
Gereksinimler için test metotları (test, gösterim, analiz veya denetleme) tanımlanmış mı?			
Fonksiyonel gereksinimler özgün numaralandırılmış mı?			



Yazılım Tasarım Belgesi Örnek Kontrol Listesi

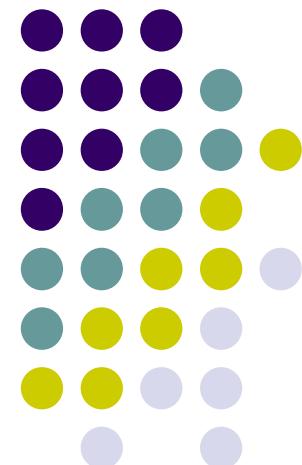
	(E=Evet, H=Hayır, U=Uygulanamaz)	(E / H / U)	Açıklamalar
Standartlara Uygunluk			
Doküman için standartlar/kılavuzlar ve isimlendirme kuralları belirlenmiş mi?			
Doküman formatı tanımlı standart/kılavuz ile uyumlu mu?			
Doküman boyunca standartlar ve isimlendirme kuralları takip edilmiş mi?			
Referans verilen dokümanlar listelenmiş mi?			
Listelenen her referansa doküman içinden atıfta bulunulmuş mu?			
Doküman İçeriği			
Her bir YKB için girdi/çıktı kriterleri tanımlanmış mı?			
Her bir YKB için davranış Özellikleri tanımlanmış mı?			
Emniyet, güvenlik ve gizlilik gereksinimleri ile ilgili genel kararlar belirlenmiş mi?			
Her bir YKB'yi oluşturan yazılım birimleri tanımlanmış mı?			
Her bir yazılım birimi arasındaki statik ve dinamik ilişkiler tanımlanmış mı?			
Her bir yazılım birimi için atanması gereken kaynaklar tanımlanmış mı?			
Her bir yazılım biriminin çalışma prensibi tanımlanmış mı?			
Yazılım birimleri arası arayüz tasarım kararları tanımlanmış mı?			
Her bir arayüze tanımlama kodu atanmış mı?			
Her bir arayüze öncelik değeri atanmış mı?			
Her bir arayüzün tipi tanımlanmış mı?			
Arayüzler arası iletişimı sağlayan veri paketleri tanımlanmış mı?			
Arayüzler arası iletişim altyapı gereksinimleri tanımlanmış mı?			
Her bir YKB için detaylı tasarım tanımları yapılmış mı?			
Yazılım birimleri anlamlı gruplar halinde sınıflandırılmış mı?			
Yazılım birimi ile ilgili algoritmalar tanımlanmış mı?			
Yazılım birimi ile ilgili varsayımlar ve kısıtlamalar tanımlanmış mı?			
Yazılım birimleri arasındaki iletişimde kullanılacak veri paketleri tanımlanmış mı?			
Yazılım biriminin kontrol akışı tanımlanmış mı?			
Yazılım birimi ile ilgili beklenmeyen durumlar ve hatalar tanımlanmış mı?			
Tanımlanan her bir yazılım biriminin hangi gereksinimleri karşıladığına dair izlenebilirlik ilişkileri tanımlanmış mı?			



Gözden Geçirme İçin Genel Kriterler

- Kontrol Listesi olmayan ürünlerin gözden geçirmesinde kullanılabilecek kriterler:
 - Belge için belirlenen biçimde uygunluk
 - Belgenin kendi içindeki tutarlılık ve uyum
 - Anlam ve kapsam bütünlüğü
 - Anlaşılabilirlik
 - Yapılabılırlik
 - Teknik yeterlik
 - Detay seviyesinin uygunluğu
 - İçeriğin müşteriye uygunluğu
 - Diğer belgelerle tutarlılık ve uyum
 - Diğer belgelere izlenebilirlik
 - Yazım hataları

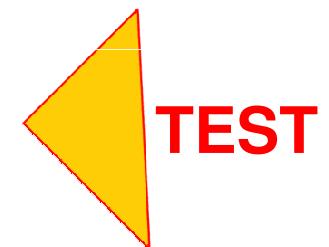
Yazılım Test





Yazılım Kalite Güvence: Yöntemler

- Statik yöntemler: Kodu çalıştırmadan yapılır.
 - Örnek: Gereksinim ve tasarım belgeleri ile kodun denetimi veya gözden geçirilmesi
 - İnceleme (“inspection”)
 - Gözden geçirme (“review”)
 - Denetleme (“audit”)
- Dinamik yöntemler: Kodu çalıştırarak yapılır.
 - Örnek: Ürünün veya bileşenlerinin gerçeğe yakın testi
 - Birim test (“unit test”)
 - Tümleştirme test (“integration test”)
 - Sistem test (“system test” / “functional test” / “qualification test”)
 - Kabul test (“acceptance test”)





Test – Tekrar Ziyaret

- Bir programın davranışını; beklenen davranışa uymadığı durumları bulma amacı ile, sonsuz bir küme içinden sınırlı sayıda seçilen test vakalarını kullanarak, dinamik yöntemlerle sınama işlemidir.
[SWEBOK 2004].
 - Beklenen : Tanımlanmış gereksinimlere uyan
 - Sınırlı : Yeterli sayıda
 - Seçilmiş : Uygun test vakaları
 - Dinamik : Kod çalıştırılarak

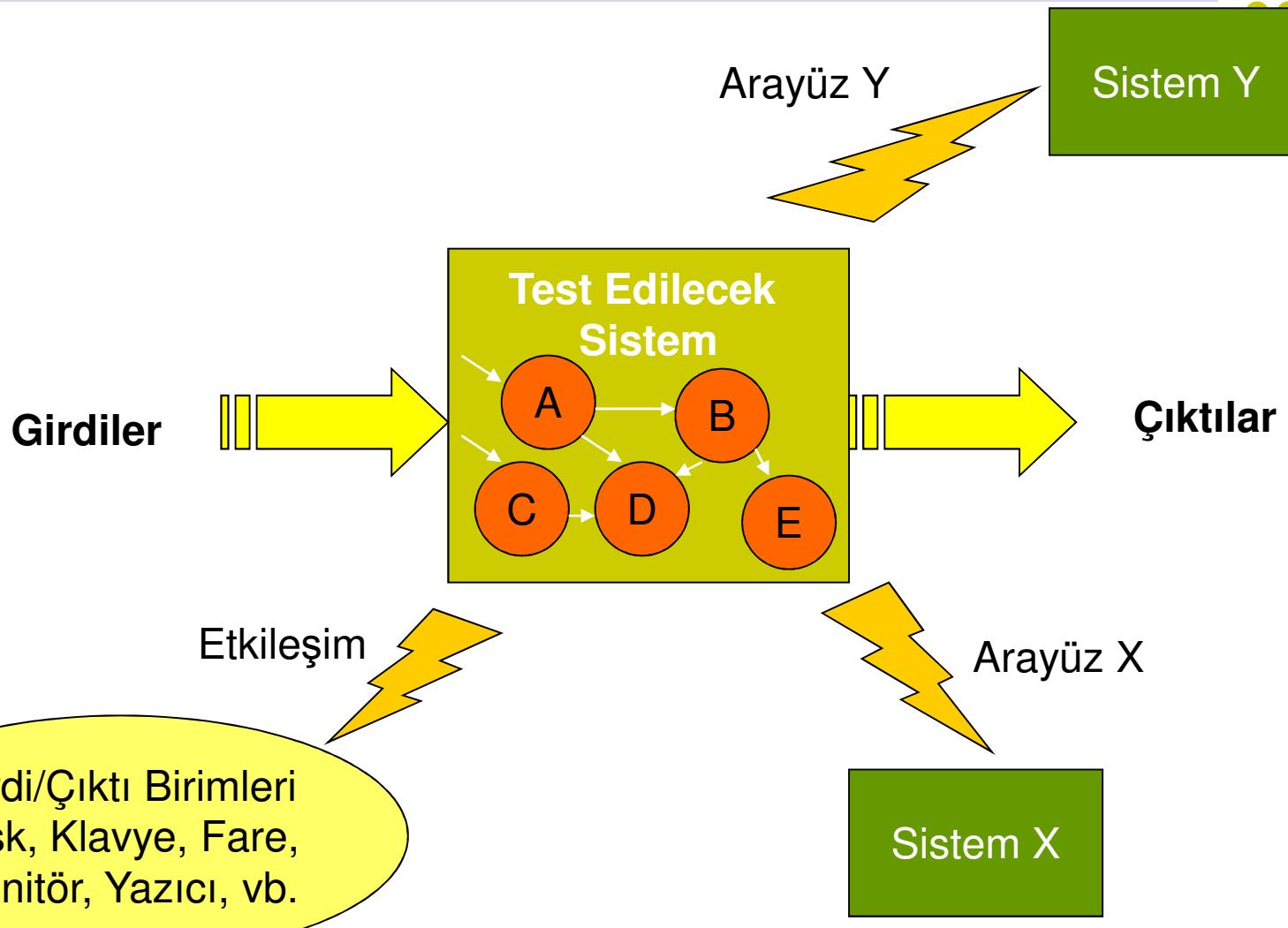


Test: Amaç ve Kapsam

- Testin amacı hata bulmaktır. Hata bulunamamışsa, test süreci başarısız sonuçlanmıştır.
- Başarılı bir test süreci sadece beklenen senaryoları değil, olağan dışı senaryoları da kapsamalıdır.
- Test süreci, gereksinimlerin belirlenmesi aşamasında başlar.



Test Edilecek Sistem





Test Türleri - 1

- Yükleme (“load”)
 - Sistemin kullanılabilmesi için uygun şekilde yüklenmesi gereklidir. İşletme, yönetim ve bakım (yedekleme, geri yükleme, kullanıcı yönetimi, vb.) görevlerini içerir.
- İşlevsel (“functional”)
 - Testin can alıcı noktasıdır. Sistemin tanımlı gereksinimleri yerine getirdiği sınanır.
- Alfa & Beta
 - Kullanıcılar tarafından yapılan test türüdür. Belirli bir test planı yoktur, kullanıcılar hataları bildirir.
- Zorlama (“stress”)
 - Sistem çökene kadar test yapılır. Yük, sistem çalışmaz hale gelinceye kadar artırılır.
- Kurtarma (“recovery”)
 - Sistem çöktükten sonra geri döndürülebilirliği sınanır (veri kaybı var mı, sistem tutarlı durumda kaldı mı? vb.)



Test Türleri - 2

- Yapılandırma (“configuration”)
 - Sistem değişik çalışma ortamlarında test edilir. (Unix/Windows, ORACLE/SYBASE, vb.)
- Performans (“performance”)
 - Sistemin beklenen işlevselligi belirli zaman sınırları içinde yerine getirdiği sınanır. Çeşitli yük altında yanıtlama süresi ölçülür.
- Kullanılabilirlik (“usability”)
 - Sistemin kullanılabilirliği sınanır (çevrim içi yardım, yazmak yerine seçmek, anlaşılır hata mesajları, vb.)
- Güvenilirlik (“reliability”)
 - Sistemin farklı girdilerle aynı şekilde davranışının (tutarlılığı) sınanır.
- Bağlanma (“regression”)
 - Üst üste gelen kurulumların (“build”) sistemin beklenen işlevselligini bozmadığı sınanır.
- Kabul (“acceptance”)
 - Müşteri kabulüne (ve ödemeye) esas olarak yapılan testtir.



Test Teknikleri - 1

- İçgündüye dayalı
 - Herhangi bir tasarıma dayanmadan test (“ad hoc”)
- Gereksinimlere dayalı
 - Eş değerlere bölme (“equivalence partitioning”)
 - Uç değerler analizi (“boundary-value analysis”)
 - Karar tablosu (“decision table”)
 - Belgelenmiş özelliklere göre test
 - Rastgele test



Test Teknikleri - 2

- Koda dayalı
 - Kontrol akışına göre (“control flow-based”)
 - Veri akışına göre (“data flow-based”)
- Uygulamanın türüne dayalı
 - Nesne tabanlı (“object-oriented”)
 - Web tabanlı
 - Grafik kullanıcı arayüzü (“GUI”) esaslı
 - Eşzamanlı / paralel (“concurrent”)
 - Dağıtık (“distributed”)
 - Gerçek zamanlı (“real-time”)



Kara-Kutu / Beyaz-Kutu Test

- Kara kutu (“black-box”) test
 - Eş değerlere bölme (“equivalence partitioning”)
 - Uç değerler analizi (“boundary-value analysis”)
 - Karar tablosu (“decision table”)
 - Belgelenmiş özelliklere göre test
 - Rastgele test
- Beyaz kutu (“white-box”) test
 - Kontrol akışına göre (“control-flow based”)
 - Veri akışına göre (“data-flow based”)



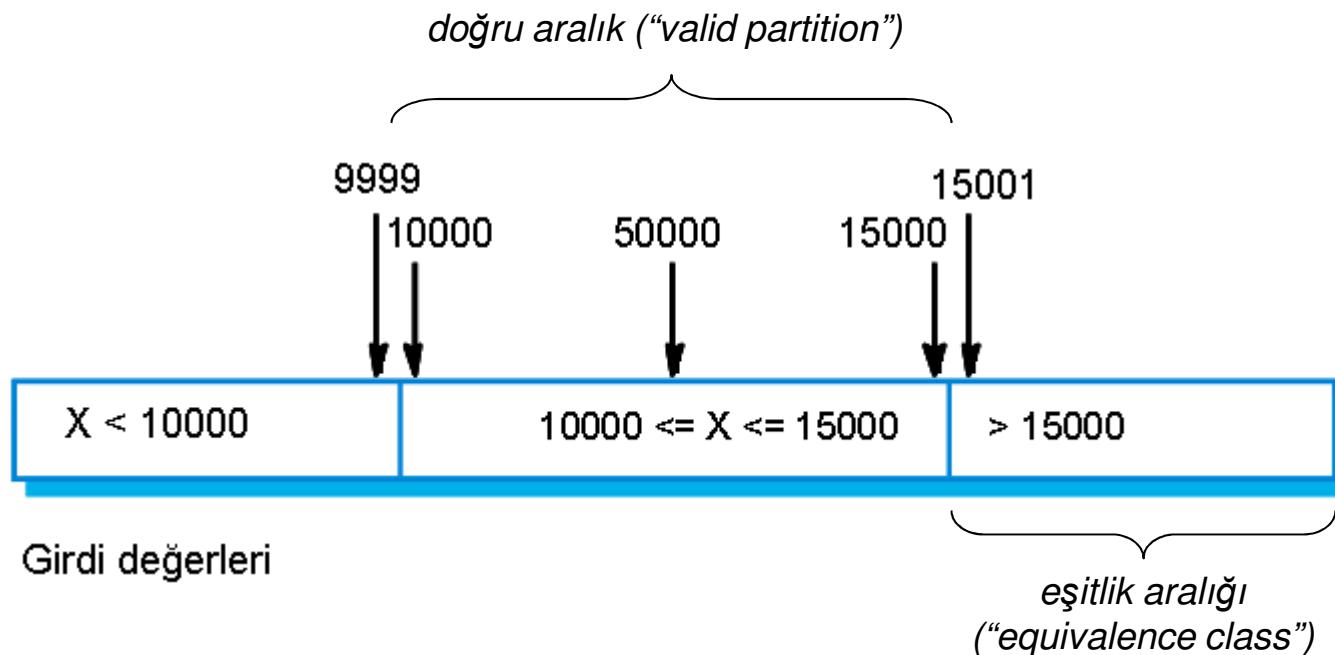
Eş Değerlere Bölme (Kara Kutu): Örnek

10000 ile 15000 TL arasındaki kredi değerlerini işleyen bir program için, değer uzayı 3'e bölünür (eşitlik aralığı - E.A.):

EA-1: $x < 10000$;

EA-2: $10000 \leq x \leq 15000$;

EA3: $x > 15000$





Karar Tablosu (Kara Kutu): Örnek

Ev Alarm Sistemi

1. Sistem güvenlik kodu girilerek aktif hale veya beklemeye geçirilir
2. Sisteme üç algılayıcı (sensor) bağlıdır; biri giriş kapısına, biri oturma odası penceresine, ve sonuncusu ise yatak odası penceresine.
3. Algılayıcılarından herhangi biri devreyi kapatırsa, sistem aktif halde ise, alarm çalışmaya başlar.
4. Eğer sistem beklemedeyse alarm hiçbir durumda çalışmaz.

SA	KP	P1	P2	Alarm
H	H	H	H	Kapalı
H	H	H	E	Kapalı
H	H	E	H	Kapalı
H	H	E	E	Kapalı
H	E	H	H	Kapalı
H	E	H	E	Kapalı
H	E	E	H	Kapalı
H	E	E	E	Kapalı
E	H	H	H	Kapalı
E	H	H	E	Çalar
E	H	E	H	Çalar
E	H	E	E	Çalar
E	E	H	H	Çalar
E	E	H	E	Çalar
E	E	E	H	Çalar
E	E	E	E	Çalar



Kontrol Akışı (Beyaz Kutu): Örnek

"Komisyon Hesaplama" programı

En az alınacak komisyon 1 TL

Tutar > 500 TL, komisyon oranı %0.2

Tutar > 5000 TL, komisyon oranı %0.15

Tutar > 50000, komisyon oranı %0.1

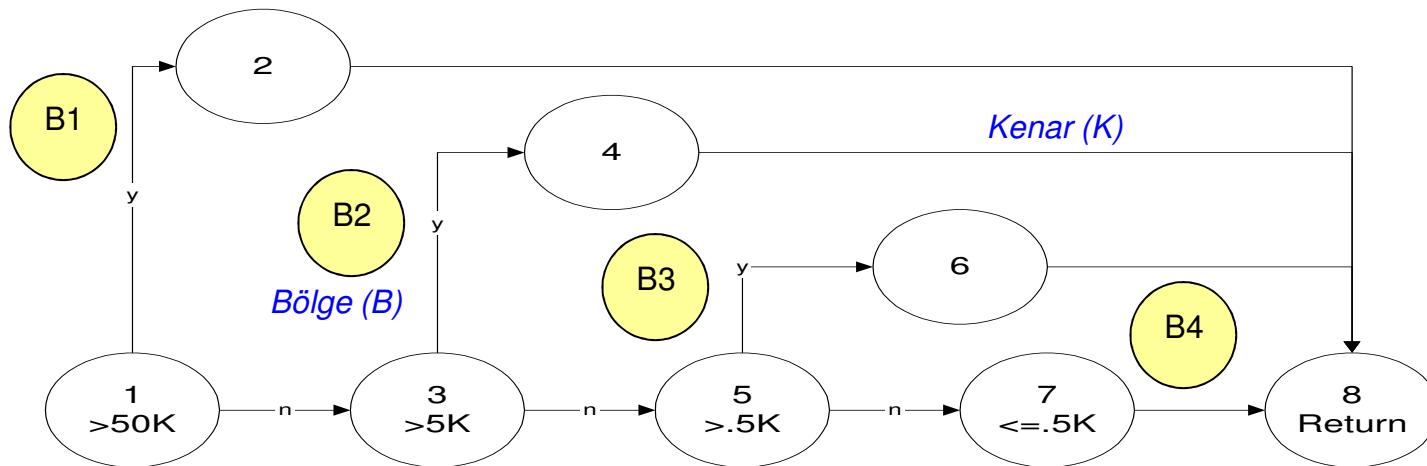
Kaynak Kod

```

1 If Tutar > 50000 then
2   Komisyon = Tutar * 0.001
3 Else If Tutar > 5000 then
4   Komisyon = Tutar * 0.0015
5 Else If Tutar > 500 then
6   Komisyon = Tutar * 0.002
7 Else Komisyon = 1
8 Return Komisyon

```

Düğüm (D)



Kenar (K)

Siklomatik karmaşıklık
("cyclomatic complexity")

$$V(G) = K - D + 2 \\ = 10 - 8 + 2 = 4$$

$$V(G) = B = 4$$





Uç Değerler (Kara Kutu): Örnek

“Komisyon Hesaplama” programı

En az alınacak komisyon 1 TL

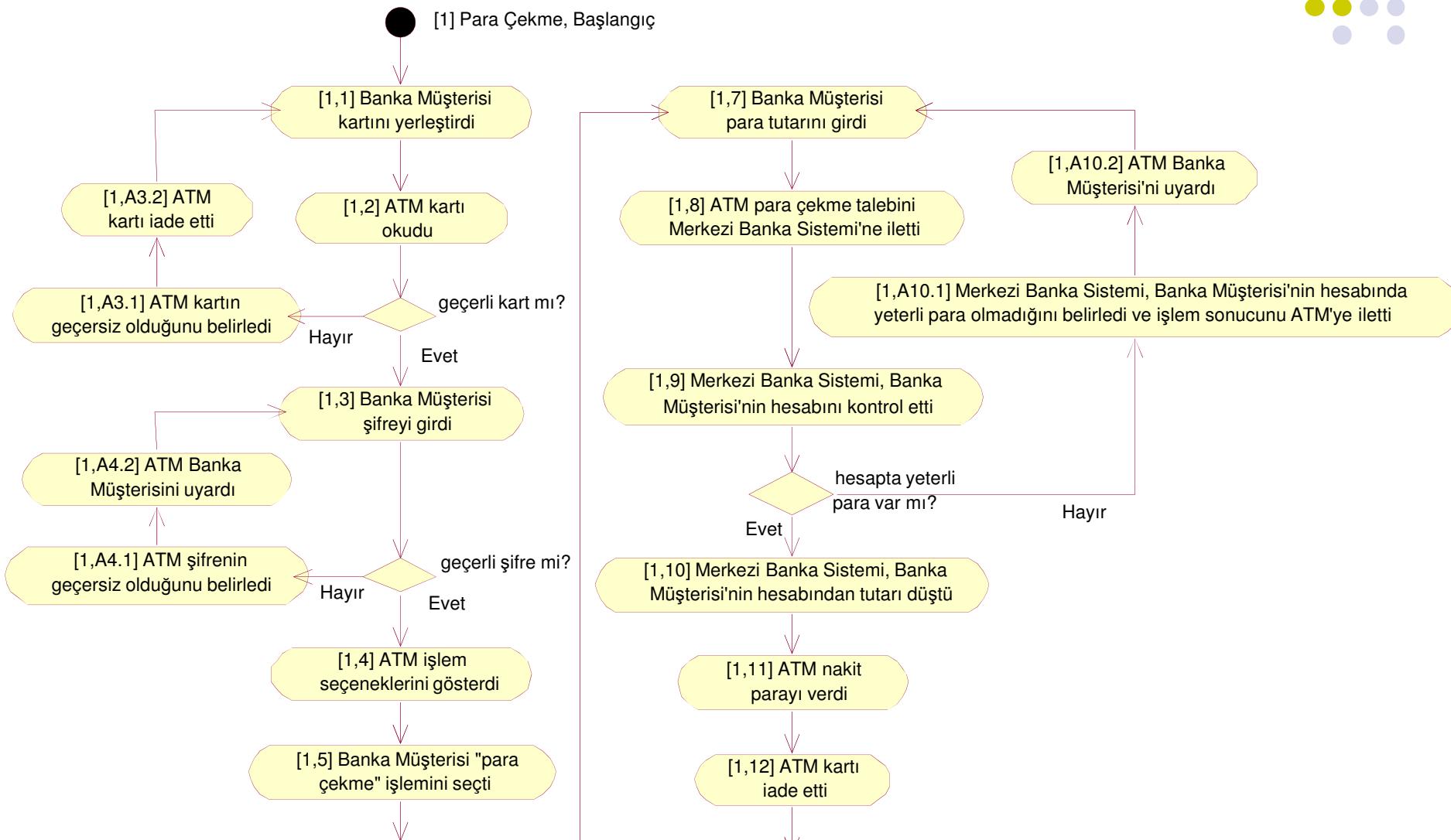
- | | |
|------------------|----------------------|
| Tutar > 500 TL, | komisyon oranı %0.2 |
| Tutar > 5000 TL, | komisyon oranı %0.15 |
| Tutar > 50000, | komisyon oranı %0.1 |

Test No	Girdi (Bin TL)	Beklenen Sonuç (TL)
KH-FN-t1	0.499	1.000
KH-FN-t2	0.500	1.000
KH-FN-t3	0.501	1.002
KH-FN-t4	4.999	9.998
KH-FN-t5	5.000	10.000
KH-FN-t6	5.001	7.5015
KH-FN-t7	49.999	74.9985
KH-FN-t8	50.000	75.000
KH-FN-t9	50.001	50.001

9 Test



Örnek: Belgelenmiş Özelliklere Göre Test (ATM - Para Çekme)

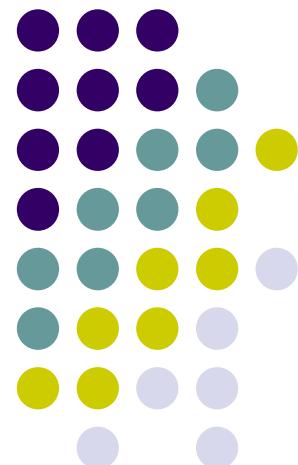




Gereksinimler ve Test

- İşlevsel test durumlarını düşünürken sistemin nasıl davranışacağı netleşir.
- Test bakış açısı ile yaklaşıldığından gereksinimlerdeki problemler çok daha rahat fark edilebilir.
 - Beklenen davranışı/çıktıyı tanımlayamayacaksınız
- Analiz ve test ekiplerinin müşteri ile test senaryolarının üzerinden geçmesi, ortak bir vizyon için çok yararlı olacaktır.

Yazılım Yaşam Döngüsü Boyunca Test



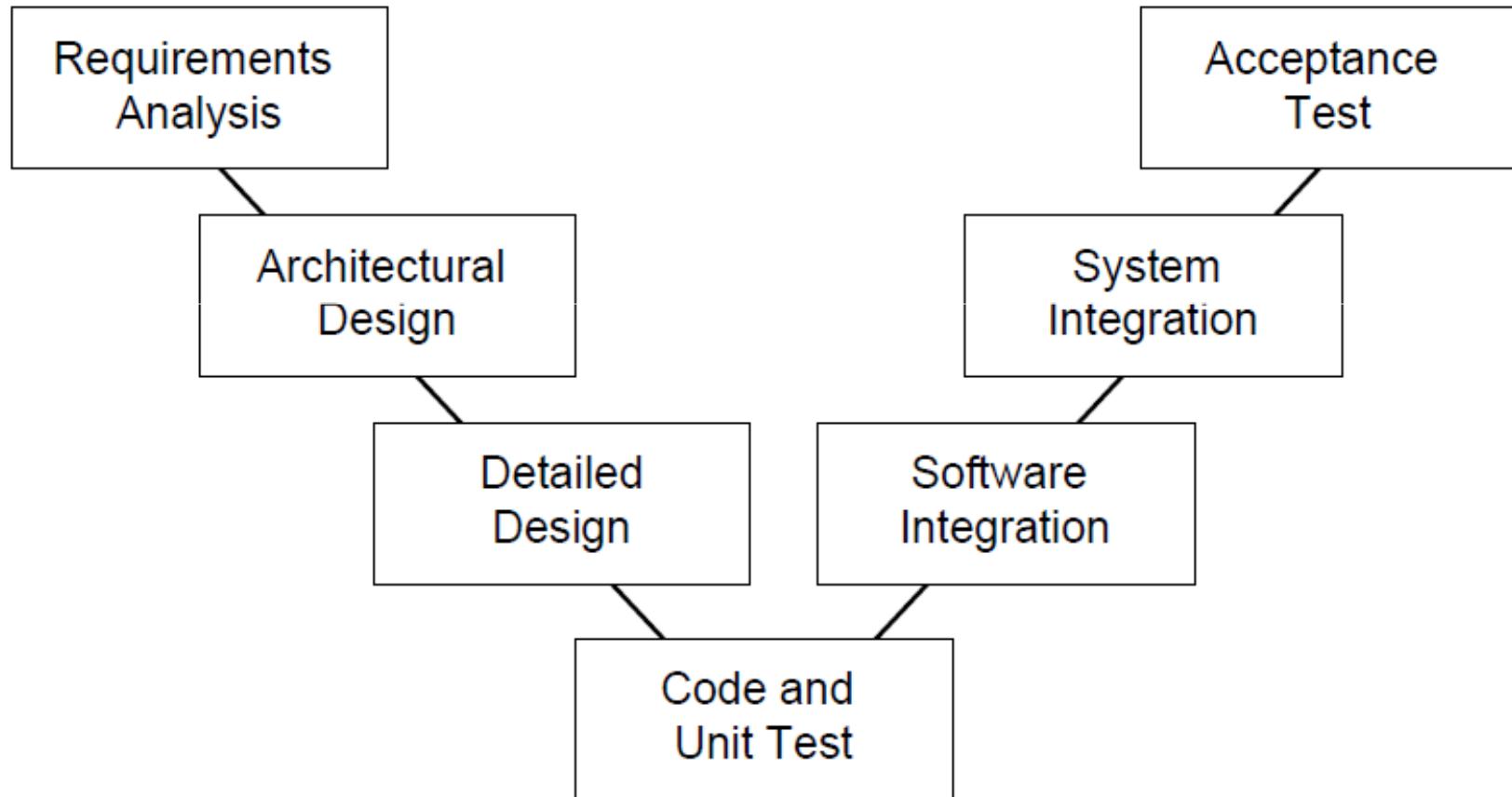


Test Seviyeleri

- **Birim test**
 - En küçük kod birimi için yapılır
 - Örnek: Modül, sınıf, vb.
 - Birimin istenilen gereksinimleri yerine getirdiği güvence edilir.
- **Tümleştirme test**
 - Birkaç birimin oluşturduğu bütün için yapılır.
 - Örnek: "use case", paket, vb.
 - Birleştirilmiş birimlerin birlikte düzgün çalışıyor olduğu güvence edilir.
- **Sistem test**
 - Yazılım sisteminin bütünü için yapılır.
 - Geliştirilen yazılımın müşteri gereksinimlerini sağladığı güvence edilir.
- **Kabul test**
 - Genellikle sözleşmeye esas olarak, yazılım sisteminin bütünü için yapılır.
 - Geliştirilen yazılımın, kullanılacağı gerçek (veya gerçeğe yakın) ortamda, müşteri gereksinimlerini sağladığı güvence edilir.

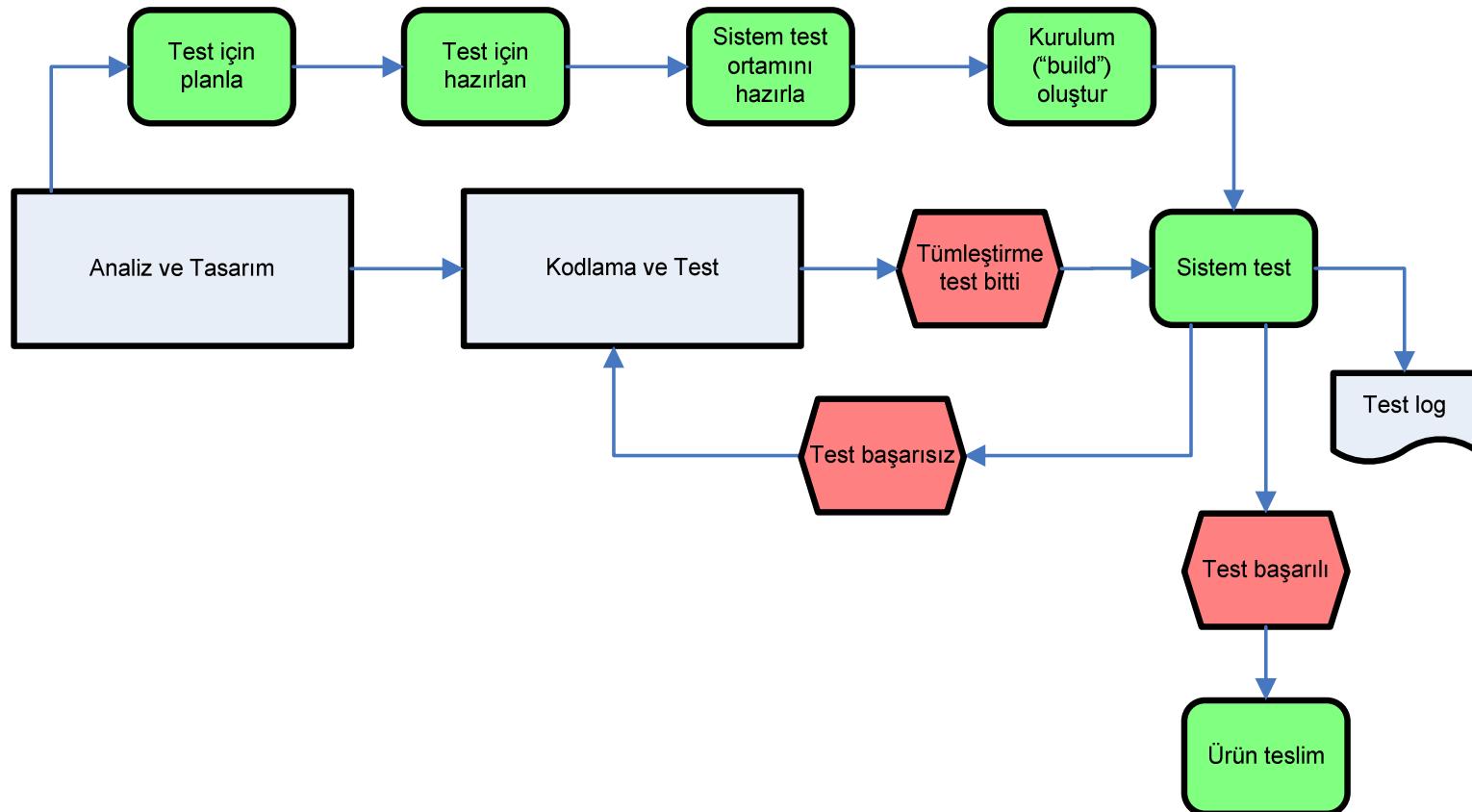


“V” Yaşam Döngüsü Modeli



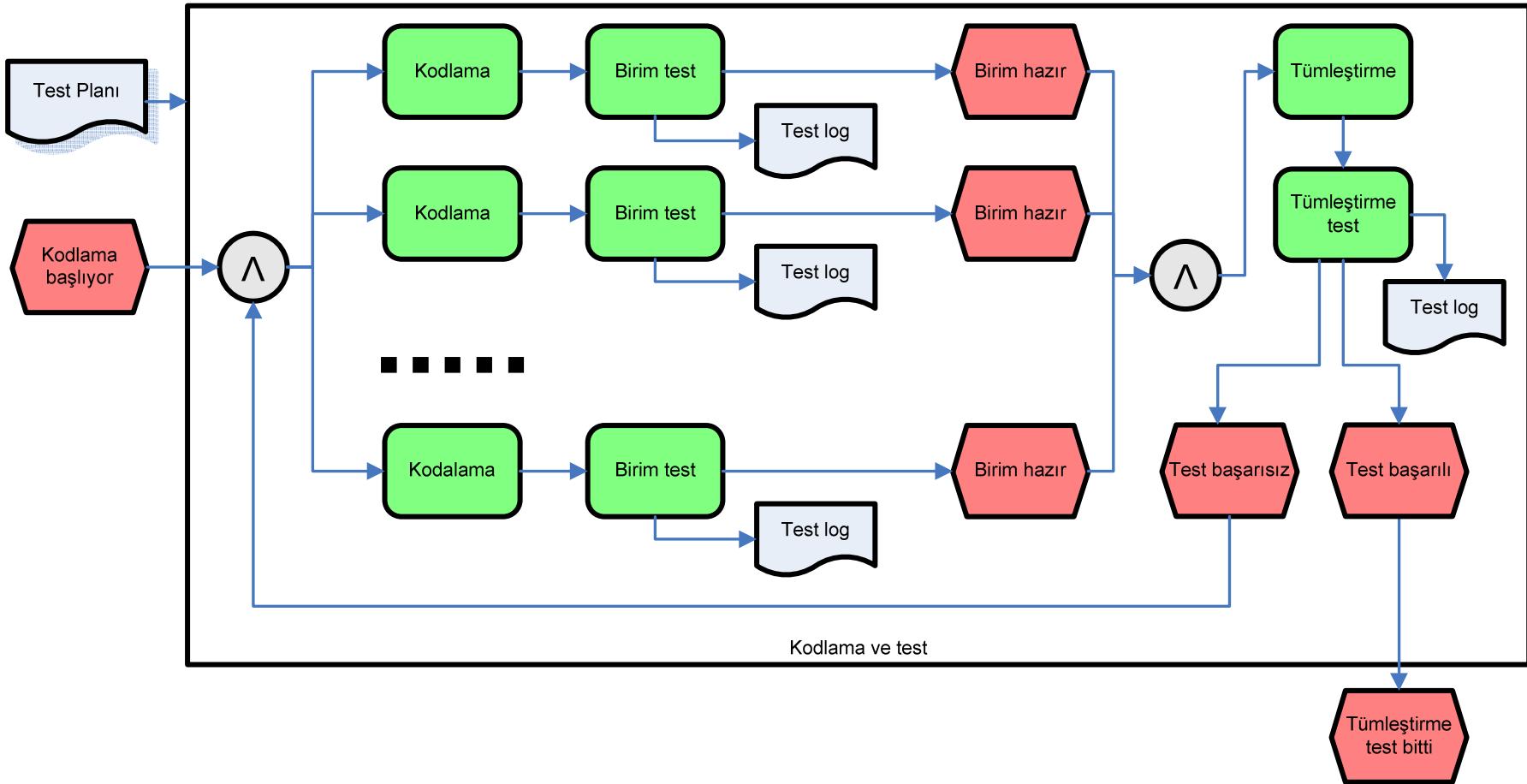


Yazılım Geliştirme ve Test - 1





Yazılım Geliştirme ve Test - 2





Test Otomasyonu

- Test, pahalı bir yazılım geliştirme aşamasıdır.
- Test ortamları toplam test maliyetini azaltabilecek araçlar ve özellikler sunmaktadır.
 - Örnek: Junit – birim testlerin otomatik işletimini destekler.
- Test ortamlarının kapalı analiz ve tasarım araçlarıyla tımlaştırılması zor olabilmektedir.
- Bazı test araçları:
 - Test Director – Mercury Interactive
 - Test Manager – IBM Rational
 - Cactus – Açık kaynak

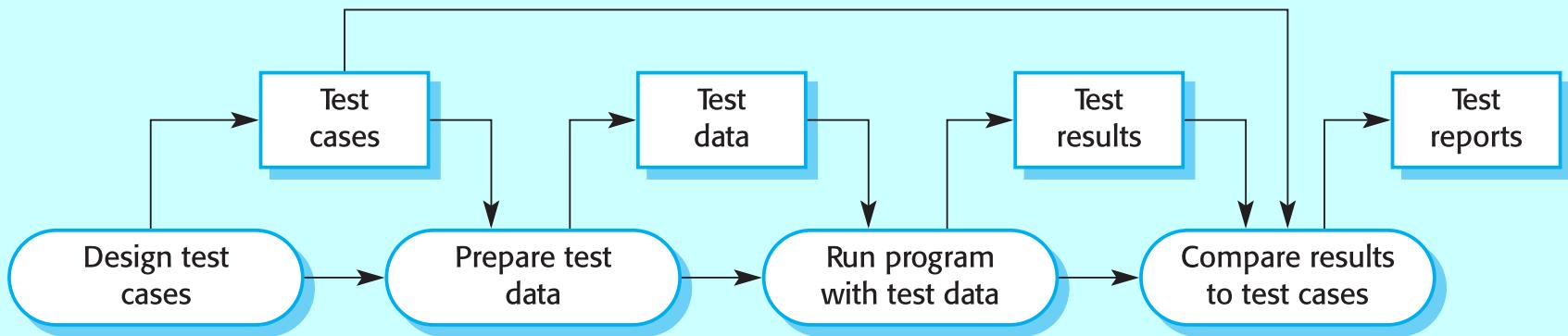


Test Süreci

- Test planlama
- Test tasarıımı
- Test ortamının ve verilerinin hazırlanması
- Testlerin yapılması
- Test sonuçlarının raporlanması
- Testlerin doğrulanması



Yazılım Test Süreci [Sommerville]





Test Stratejisi

- Test stratejisi
 - Neler test edilecek ? (işletme senaryoları, ürün parçaları, use case'ler, vb.)
 - Test edilecek birimlerin öncelik sırası
 - Hangi test teknikleri kullanılacak?
 - Test tamamlama kriterleri neler olacak?
- Test planlama için dikkate alınacaklar
 - Gereksinimlerin doğrulama kriterleri
 - Sistem kabul kriterleri
 - Gereksinimlerin öncelikleri
 - Gereksinimler ve tasarım çözümü ile ilgili riskler
 - Tasarımın karmaşıklığı
 - Test süreci, standartları ve/veya verileri ile ilgili gereksinimler
 - BT altyapısı ile ilgili gereksinimler
 - Güvenirlik, performans, vb. kalite gereksinimleri



(1) Test Planlama

- Test aktiviteleri için araç, personel ve yetkinlik planlanır.
- Test ortamı gereksinimleri belirlenir ve bu ortamın oluşturulması aktiviteleri planlanır.
- Test verilerinin hazırlık aktiviteleri de planlanır.
- Test planı, proje yönetimi ve kalite güvence tarafından gözden geçirilir.



(2) Test Tasarımı

- Seçilen test tekniklerinin uygulanmasına yönelik çalışmalar yapılır.
- Testler nasıl yapılacak belirlenir ve tanımlanır.
 - Prosedürler,
 - Test senaryoları,
 - Test verileri.



Test Tasarım Dokümanı: Örnek

- Test Tasarım Kimliği:** Test Planında belirtilen kimlik (örn: OM-LG)
- Test Edilecek Alt Fonksiyonlar:** Alt fonksiyonları sıralar ve kısaltmaları belirler (örn: OM-LG-ga)
- Test Edilmeyecek Alt Fonksiyonlar:** Alt fonksiyonları sıralar ve neden test edilmeyeceklerini açıklar
- Yaklaşım:** Test planında belirtilen yaklaşımın açılımı veya değişik bir yaklaşım kullanılacaksa açıklaması.
- Geçme/Kalma Kriterleri:** Test planında belirtilen kriterlerin açılımı veya değişik kriterler kullanılacaksa açıklaması.
- Ortam Gereksinimleri:** Aşağıdaki testlere özel ortam (donanım, araçlar, veriler, vb.) gereksinimleri
- Test Senaryoları:**

Test No	Özellik No	Senaryo Açıklaması
OM-LG-ga-01	1.1.1	Geçerli kullanıcı ad ve şifre ile giriş yap
OM-LG-ga-02	1.1.1	Geçerli kullanıcı ad ve geçersiz şifre ile giriş yap
OM-LG-ga-03	1.1.1	Geçersiz kullanıcı ad ve geçerli şifre ile giriş yap



Test Senaryosu: Örnek

Test No	:	MO-LG-ga-01
Senaryo	:	Geçerli kullanıcı ad ve şifre ile giriş yap
Özellik No	:	1.1.1
Önem	:	Yüksek (<i>Yüksek, Orta, Düşük</i>)
Kategori	:	Müşteri kabul (<i>Teste kabul, Müşteri kabul, Bağlanım, Tümleştirme, ...</i>)
Tahmini Süre	:	5 Dakika (<i>Tahmini testi koşturma süresi</i>)
Bağımlılık	:	Yok (<i>Bu testten evvel koşturulması gereken testleri sırala</i>)
Kurulum	:	Netscape Web Browser'ını çalıştır (<i>Prosedürden evvel yapılması gerekenler</i>)
Prosedür	:	<ol style="list-style-type: none">1. http://enstitu.hacettepe.edu.tr/akademik/ web sitesine git2. Geçerli kullanıcı ad ve şifreyi gir3. “Tamam” düğmesine bas4. “HACETTEPE ÜNİVERSİTESİ - Lisansüstü Öğrenci İşleri Sistemi” ana sayfasının geldiğini doğrula
Temizlik	:	<i>Çıkış linkine bas (Ortamı bulduğum gibi bırakmak için yapılacaklar)</i>



(3) Test Ortamının ve Test Verilerinin Hazırlanması

- Sistemin test edileceği ortam ve test verileri, işletme ortamını yansıtacak şekilde hazırlanır.
 - Test ortamı: Sunucu, ağ, işletim sistemi, diğer sistemler ile bağlantılar, vb.
 - Test verileri: Müşteri veritabanları, ek tablolar, vb.
 - Test verileri, işin gerçek boyutunu yansıtmalıdır.



(4,5) Testlerin Yapılması ve Sonuçların Raporlanması

- Testler tanımlanan test senaryolarına göre gerçekleştirilir.
- Sistem testleri geliştirme ekibinden farklı kişiler tarafından yapılır.
- Tasarlanan test durumları işletilir ve gözlemlenen hatalar, istenmeyen durumlar kaydedilir.
 - “Test log”
- Tanımlı test durumları işletildikten sonra test tamamlanmış sayılır.
- “Test log”lardaki bilgiler, test stratejisi ve diğer gözlemler, test raporu olarak belgelendirilir.
 - Kaç durum işletilebildi, ne kadar zaman harcandı, kaç durumda hata oluştu, kodun ne kadarı test edilebildi, vb.
- Test raporu kalite güvence ekibine iletilir.

Test Tasarım Raporu: Use-Case Esaslı Test İçin Örnek



USE CASE TEST CHECKLIST

Project Name:	Guidance:
Use Case Name:	Activity Path: Includes list of uniquely numbered activity nodes that make up the activity path.
Implemented By:	Each activity path is tested individually.
Tested By:	Inputs: Test inputs for related activity path. Each input can be written at different rows.
Test Date/Time:	Expected Results: Expected test results with respect to given inputs.
Coverage:	Outputs: Actual test outputs. If output matches exp. results then only a confirmation message is written
	Problem Description: Problem is described if output and exp. result are not consistent.



Test Tasarım Raporu:

Kullanıcı Arayüzü İçin Örnek

GUI Design/Test Checklist

Project Name:

Tester :

Combo Boxes

Date/Time:

	Property	Value	Result	Comment
6.1	name			
6.2	data content			
6.3	visible row count			
6.4	visible			
6.5	enabled			
6.6	editable			
6.7	font (type,size,color)			
6.8	selected index			
6.9	click action			

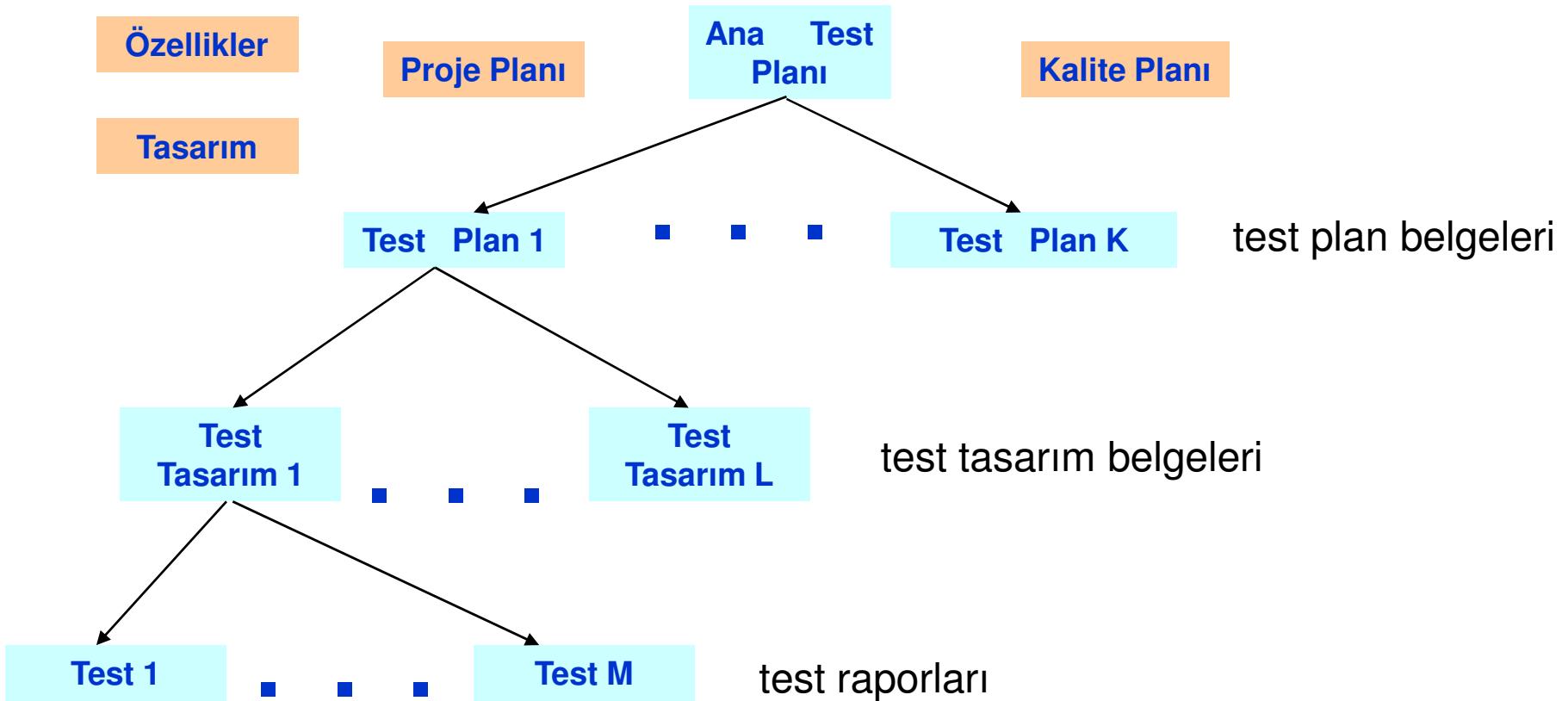


(6) Testlerin Doğrulanması

- Kalite güvence ekibi, test aktivitelerinin planlandığı şekilde yapıldığını güvence altına alır.
- Kalite güvence sorumlusu tüm test kayıtlarını gözden geçirir.
 - Test kapsamı ve kullanılan teknikler test planı ile uyumlu mu?
 - Test ortamı ve verileri tanımlandığı şekilde oluşturulmuş mu?
 - “Test log”ları düzenli tutulmuş mu?
 - ...



Test Belgeleri





BM306

Yazılım Mühendisliği



DERS 12
Yazılım Proje Yönetimi

Dr. Öğr. Üyesi Hasan BADEM
hbadem@ksu.edu.tr



■ Yazılım proje yönetimi

- ▶ “Project Management Body of Knowledge (PMBOK)”a göre proje yönetim kapsamı
- ▶ Yazılım proje planlama döngüsü
- ▶ CMMI modeline göre, proje yönetimi ile ilgili süreç alanları ve gerekleri
- ▶ Proje yönetimi ve ölçme
- ▶ MS Project ile proje planlama: ATM örneği

Yazılım Projesi

■ Tüm mühendislik ürünleri gibi yazılım ürünü de sınırları tanımlanmış bir proje kapsamında geliştirilir.

- ▶ “*A project is a temporary endeavour undertaken to create a unique product or service*” [Project Management Body of Knowledge, 1996]
 - ◆ “Temporary”: Her projenin bir başı ve sonu var
 - Proje sonlandığında; müşteri ihtiyaçları, takvim, maliyet, kalite, vb. açılarından, projenin hedeflerine ulaşması beklenir.
 - ◆ “Unique”: Ürün ya da servisi diğerlerinden ayıran belirli bir özelliği var
 - Her ürün müşterisine özel ihtiyaçları karşılar.

■ Yazılım projesi örnekleri:

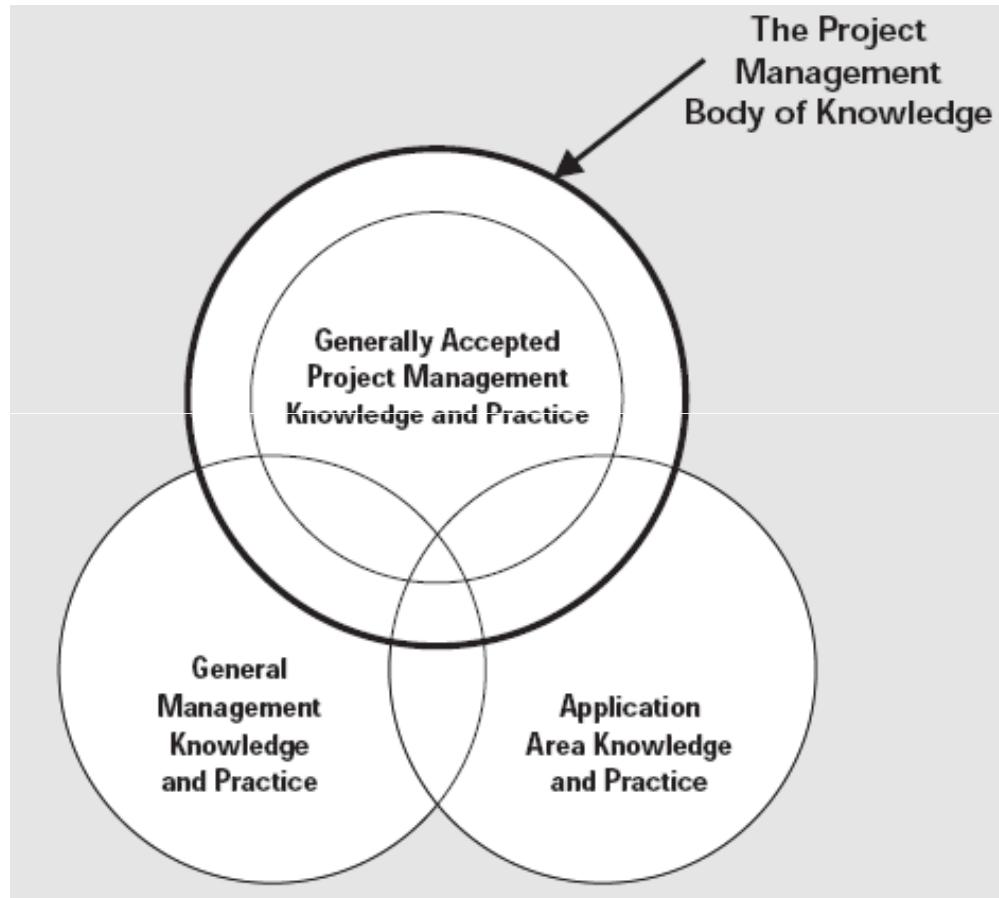
- ▶ Yeni bir işletim sistemi geliştirmek (örn: MS Vista)
 - ◆ Paket yazılım (“commercial-off-the-shelf (COTS) software”)
- ▶ ABC kuruluşu için muhasebe yazılımı geliştirmek
 - ◆ İhtiyaca özel yazılım (“custom software”)

Proje Yönetimi Nedir?

- *“Project management is the application of knowledge, skills, tools, and techniques to project activities in order to meet project requirements”*
[PMBOK, 1996]

- ▶ Kapsam, zaman, maliyet ve kalite parametreleri arasında denge gerektirir
- ▶ Farklı ihtiyaç ve beklenilere sahip paydaşlar arasında uzlaşma gerektirir

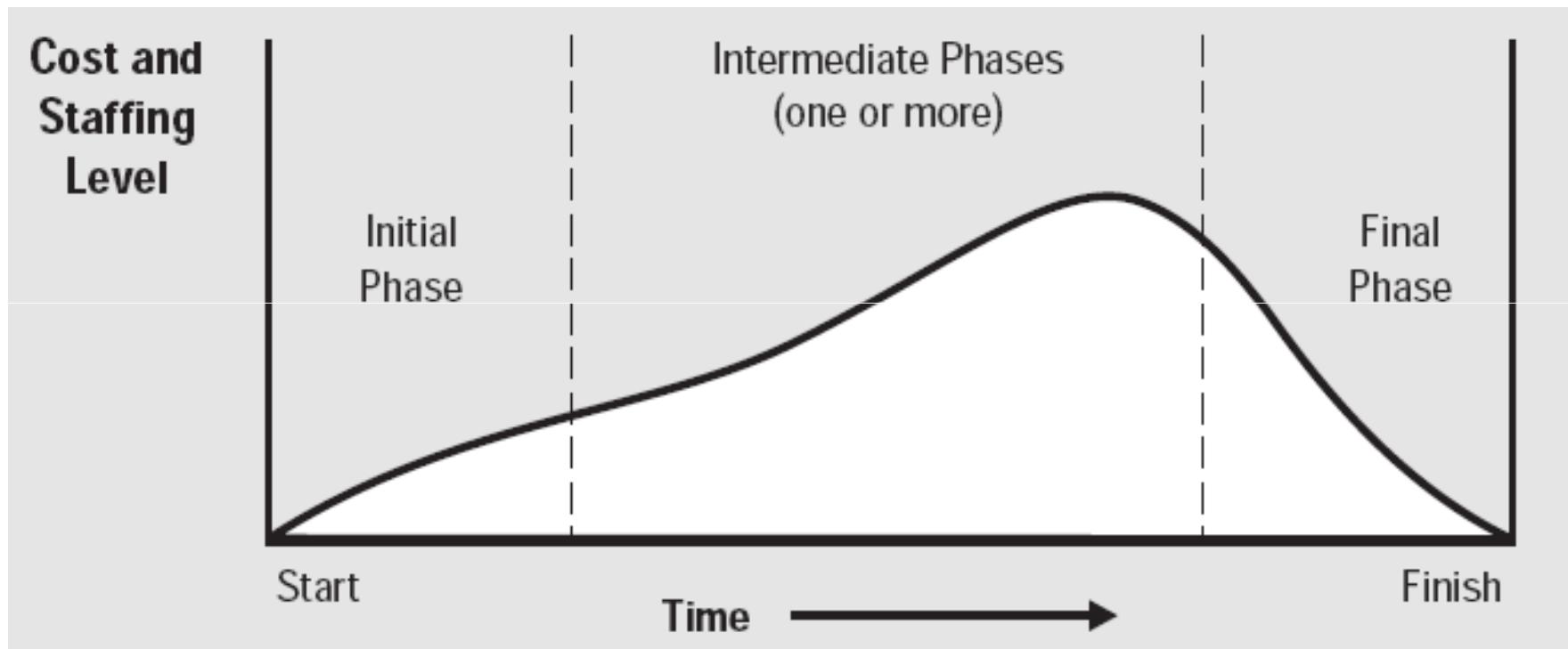
Proje Yönetiminin Diğer Yönetim Disiplinleriyle İlişkisi



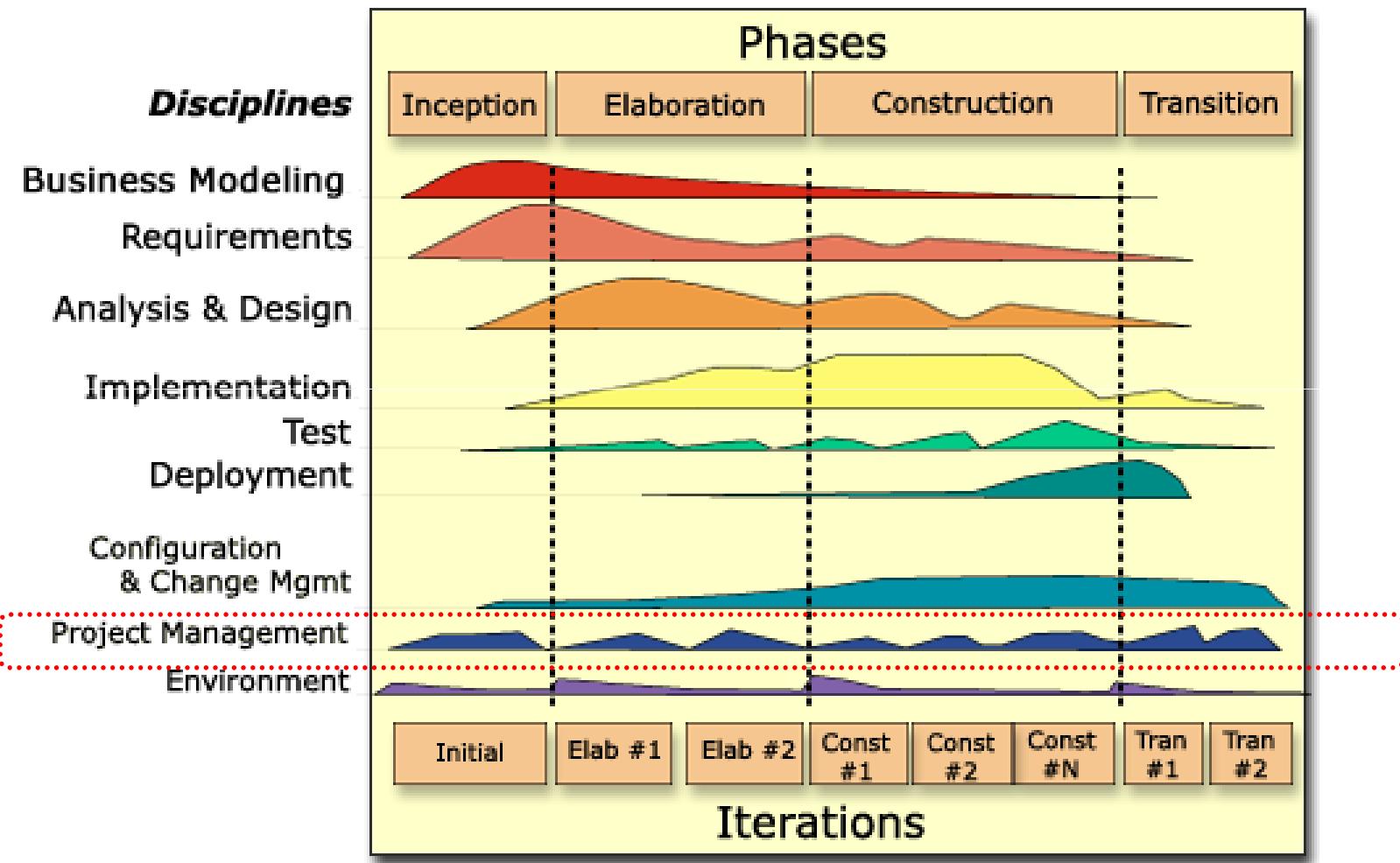
Proje Yönetimi Bilgi Alanları ("Knowledge Areas")

- Project Integration Management
- Project Scope Management
- Project Time Management
- Project Cost Management
- Project Quality Management
- Project Human Resource Management
- Project Communication Management
- Project Risk Management
- Project Procurement Management

Örnek Proje Yaşam Döngüsü

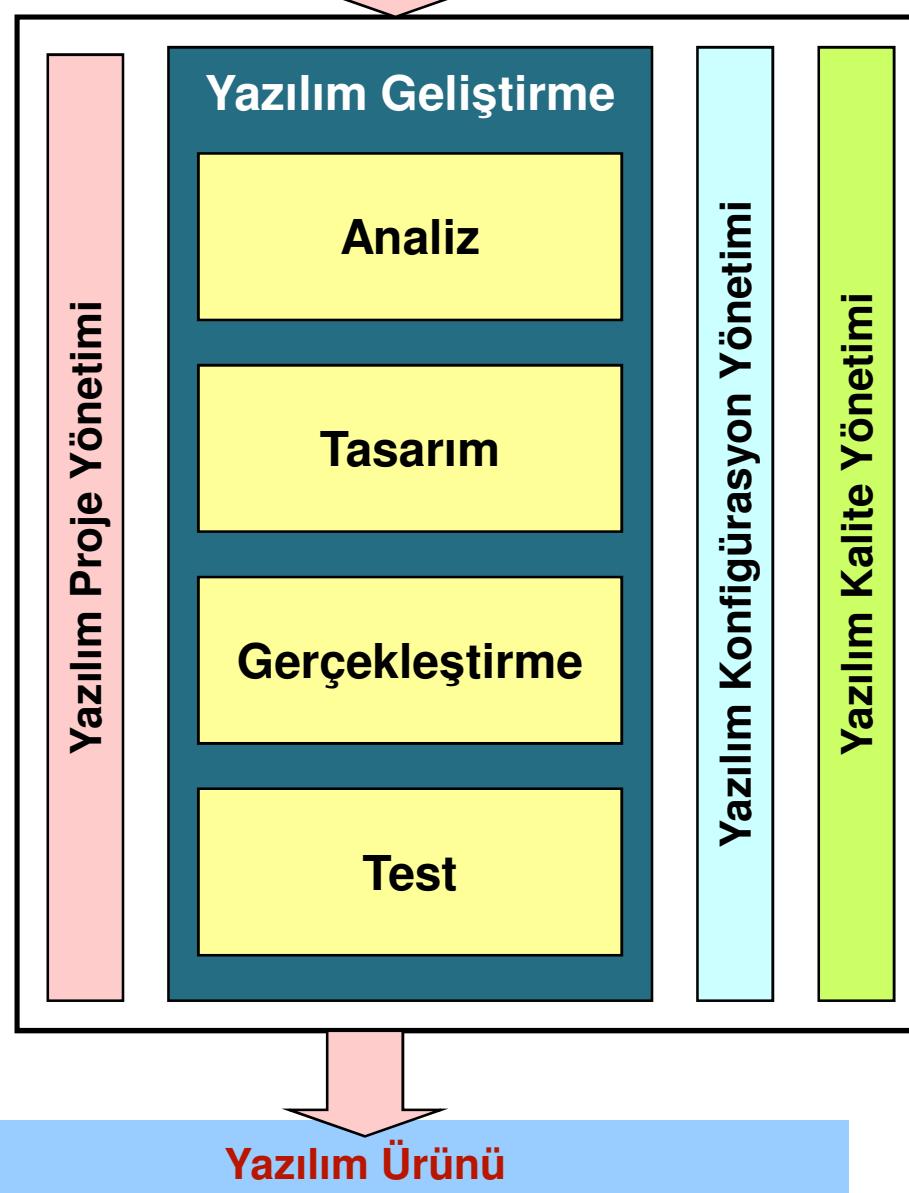


“The Rational Unified Process” (Rational Software'in Tümleşik Süreci)



Müşteri İhtiyaçları

Yazılım Yaşam Döngüsü



- Yazılım ürününün geliştirilmesi ve yönetilmesi için izlenmesi gereken adımların bütünüdür.
 - ▶ Analiz, tasarım, kodlama, test,
 - ▶ Proje yönetimi, kalite yönetimi, konfigürasyon yönetimi
- Yazılım yaşam döngüsü, yazılım ürününe mühendislik etkinliklerinin uygulanmasını sağlar.

*Yönetim etkinlikleri, yazılım geliştirme aşamaları boyunca uygulanır.
→ Bütünleşik yönetim*

Yazılım Proje Ekibi

- Her yazılım projesi, o projeye özel olarak kurulan bir **proje ekibi** tarafından gerçekleştirilir.
 - ▶ **Proje yöneticisi:** Yazılım projesini planlamaktan ve proje gelişimini izlemekten sorumludur.
 - ▶ **Geliştirme ekibi:** Yazılım geliştirme aşamalarını uygulamaktan sorumludur.
 - ◆ Analist, tasarımcı, programcı, testçi
 - ◆ Kapsamlı büyük projelerde bu roller için ayrı alt-ekipler kurulabilir.
 - ▶ **Konfigürasyon yöneticisi:** Yazılım ürününün geliştirilmesi boyunca ortaya çıkan ara ürünler ve son ürünü kontrol altına almaktan sorumludur.
 - ▶ **Kalite yöneticisi:** Yazılım ürününün geliştirilmesi boyunca ortaya çıkan ara ürünlerin ve son ürünün kalitesini kontrol etmekten ve uygunsuzlukları raporlamaktan sorumludur.
- Her projenin, o projeye kaynak sağlayan bir **sponsoru** vardır.
- Her projenin ürettiği ürünü satın alacak bir **müşterisi** vardır.

Yazılım Proje Planlama Döngüsü

Referans: Dr.Çiğdem Gencel, "Yazılım Büyüklük Ölçme ve COSMIC FFN Yöntemi" Semineri, Bilgi Grubu Ltd., 2007.



İş Dağılım Yapısı ("Work Breakdown Structure")

ID	O	WBS	Task Name
1			1 ATM SİSTEMİ GELİŞTİRME PROJESİ
2		1.1	PROJE YÖNETİMİ
3		1.1.1	Proje planlama
4		1.1.1.1	Kestirimlerin yapılması
5		1.1.1.2	Proje planının geliştirilmesi
6		1.1.1.3	Proje planının gözden geçirilmesi
7		1.1.1.4	Proje planının günlenmesi
8		1.1.1.5	Proje planının yapılandırma kontrolü altına alınması
9		1.1.1.6	Proje planı
10		1.1.2	Proje izleme ve kontrol
11		1.1.2.1	Başlangıç toplantısı
12	<input checked="" type="checkbox"/>	1.1.2.2	Izleme toplantısı-1
13		1.1.2.3	Gelisme raporu-1
14	<input checked="" type="checkbox"/>	1.1.2.4	Izleme toplantısı-2
15		1.1.2.5	Gelisme raporu-2
16	<input checked="" type="checkbox"/>	1.1.2.6	...
17		1.1.2.7	...
18	<input checked="" type="checkbox"/>	1.1.2.8	Izleme toplantısı-18
19		1.1.2.9	Gelisme raporu-18
20	<input checked="" type="checkbox"/>	1.1.2.10	Kapanış toplantısı
21		1.1.2.11	Kapanış raporu
22		1.2	İŞ GEREKSİNİMLERİNİN BELİRLENMESİ
23		1.2.1	İş gereksinimlerinin incelenmesi
24		1.2.2	İş gereksinimleri belgesinin geliştirilmesi
25		1.2.3	İş gereksinimleri belgesinin gözden geçirilmesi
26		1.2.4	İş gereksinimleri belgesinin günlenmesi
27		1.2.5	İş gereksinimleri belgesinin yapılandırma kontrolü altına alınması
28		1.2.6	İş gereksinimleri belgesi
29		1.3	SİSTEM GEREKSİNİMLERİNİN VE MİMARISİNİN BELİRLENMESİ
30		1.3.1	Sistem gereksinimleri analizi
32		1.3.2	Sistem gereksinimleri belgesi
33		1.3.3	Sistem mimarisinin belirlenmesi
35		1.3.4	Sistem mimari dokumani
36		1.4	DONANIM GELİŞTİRME
38		1.5	YAZILIM GELİŞTİRME
39		1.5.1	Çevrim-1: ATM Kart Yazılım Paketi Geliştirme
40		1.5.1.1	Çevrim-1: Yazılım gereksinimlerinin belirlenmesi
41		1.5.1.1.1	Yazılım gereksinimleri analizi
42		1.5.1.1.2	Yazılım gereksinimleri belgesinin hazırlanması
43		1.5.1.1.3	Yazılım gereksinimleri belgesinin gözden geçirilmesi
44		1.5.1.1.4	Yazılım gereksinimleri belgesinin günlenmesi
45		1.5.1.1.5	Yazılım gereksinimleri belgesinin yapılandırma kontrolü altına alınması
46		1.5.1.1.6	Yazılım gereksinimleri belgesi

Büyüklük

■ Yazılım ürününün kestirilen büyüklüğü

- ▶ İşlev puan (“function points”)
- ▶ Özellik puan (“feature points”)
- ▶ Kod satır sayısı (“lines of code”)

■ Büyüklük dönüşüm sabitleri

- ▶ Örnek: 1 IFPUG İP = 46 Java KSS

■ Büyüklükten işgücüne geçiş için sabitler

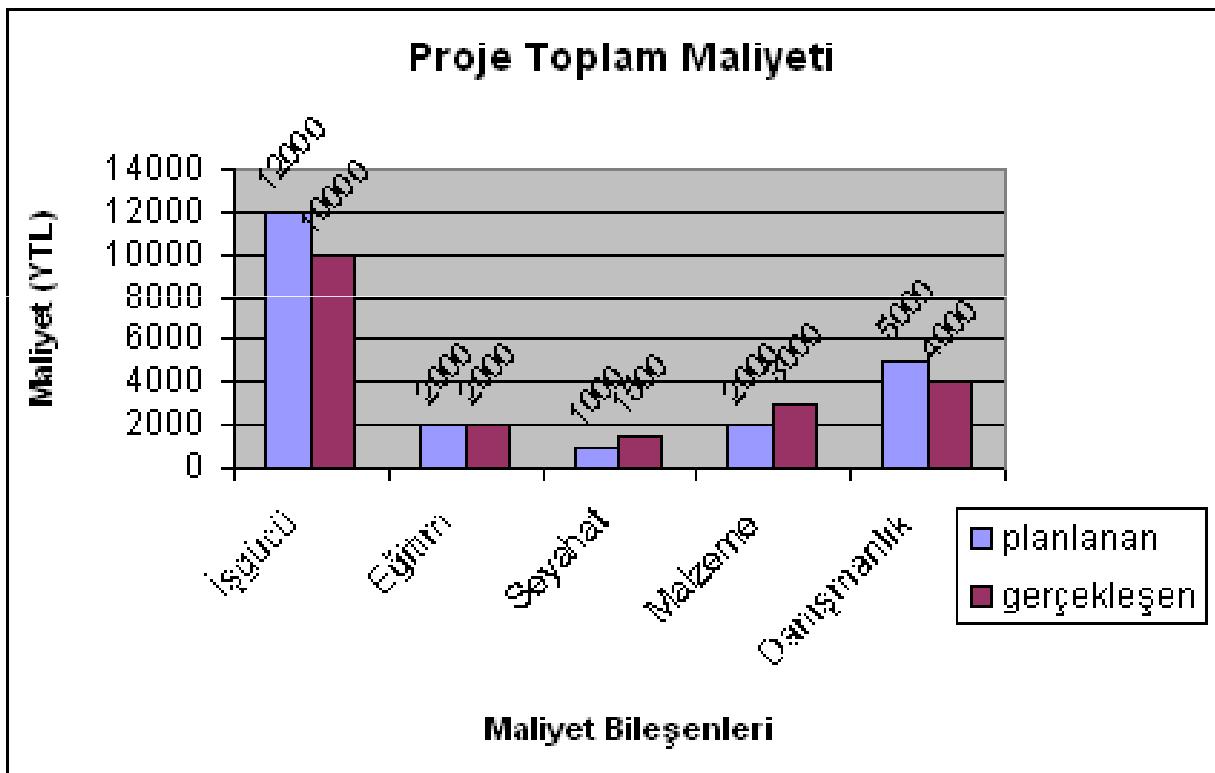
- ▶ Örnek:
 - ◆ 1 FP → 8 kişi-saat (1 kişi-gün)
 - ◆ 1 Java KSS → 0.15 kişi-saat

İşgücü

- İşgücü: Belli bir işi yapabilmek için gereken kişi-zaman miktarı
 - ▶ Birimler: Kişi-saat, kişi-gün, kişi/ay, vb.
 - ▶ Örnek: X projesi için gereken iş gücü: 12 kişi-ay
 - ◆ 12 kişi-ay:
 - 12 kişi 1 ay?
 - 1 kişi 12 ay?
 - 3 kişi 4 ay?
- İşgücü kestirimi İş Dağılım Ağacı üzerinden, yukarıdan aşağıya (“top-down”) veya aşağıdan yukarıya (“bottom-up”) yapılabilir.

Maliyet

İŞGÜCÜ MALİYETİ = İşgücü (kişi-saat) x Kaynak maliyeti (YTL/kişi-saat)



İşgücü dışındaki maliyet bileşenleri de kestirilerek toplam maliyet öngörülür.

Zaman Planı (Takvim)

ID	WBS	Task Name	Duration	Start	Finish	Work
1		1 ATM SİSTEMİ GELİŞTİRMЕ PROJESİ	427 days	Mon 15.12.08	Tue 03.08.10	8.676 hrs
2		1.1 PROJE YÖNETİMİ	426 days	Mon 15.12.08	Mon 02.08.10	260 hrs
3		1.1.1 Proje planlama	10 days	Mon 15.12.08	Mon 29.12.08	164 hrs
4		Kestirimlerin yapılması	5 days	Mon 15.12.08	Mon 22.12.08	100 hrs
5		Proje planının geliştirilmesi	2 days	Mon 22.12.08	Wed 24.12.08	16 hrs
6		Proje planının gözden geçirilmesi	1 day	Wed 24.12.08	Thu 25.12.08	32 hrs
7		Proje planının günlenmesi	1 day	Thu 25.12.08	Fri 26.12.08	8 hrs
8		Proje planının yapılandırma kontrolü altına alınması	1 day	Fri 26.12.08	Mon 29.12.08	8 hrs
9		Proje planı	0 days	Mon 29.12.08	Mon 29.12.08	0 hrs
10		1.1.2 Proje izleme ve kontrol	416 days	Mon 29.12.08	Mon 02.08.10	96 hrs
11		Başlangıç toplantısı	1 day	Mon 29.12.08	Tue 30.12.08	16 hrs
12		Izleme toplantısı-1	1 day	Mon 02.02.09	Mon 02.02.09	16 hrs
13		Gelisme raporu-1	0 days	Mon 02.02.09	Mon 02.02.09	0 hrs
14		Izleme toplantısı-2	1 day	Mon 02.03.09	Mon 02.03.09	16 hrs
15		Gelisme raporu-2	0 days	Mon 02.03.09	Mon 02.03.09	0 hrs
16		...	1 day	Mon 06.04.09	Mon 06.04.09	16 hrs
17		...	0 days	Mon 06.04.09	Mon 06.04.09	0 hrs
18		Izleme toplantısı-18	1 day	Mon 05.07.10	Mon 05.07.10	16 hrs
19		Gelisme raporu-18	0 days	Mon 05.07.10	Mon 05.07.10	0 hrs
20		Kapanış toplantısı	1 day	Mon 02.08.10	Mon 02.08.10	16 hrs
21		Kapanış raporu	0 days	Mon 02.08.10	Mon 02.08.10	0 hrs
22		1.2 İŞ GEREKSİNİMLERİNİN BELİRLENMESİ	40 days	Mon 15.12.08	Fri 06.02.09	632 hrs
23		İş gereksinimlerinin incelenmesi	30 days	Mon 15.12.08	Fri 23.01.09	480 hrs
24		İş gereksinimleri belgesinin geliştirilmesi	5 days	Mon 26.01.09	Fri 30.01.09	80 hrs
25		İş gereksinimleri belgesinin gözden geçirilmesi	1 day	Mon 02.02.09	Mon 02.02.09	16 hrs
26		İş gereksinimleri belgesinin günlenmesi	3 days	Tue 03.02.09	Thu 05.02.09	48 hrs
27		İş gereksinimleri belgesinin yapılandırma kontrolü altına alınması	1 day	Fri 06.02.09	Fri 06.02.09	8 hrs
28		İş gereksinimleri belgesi	0 days	Fri 06.02.09	Fri 06.02.09	0 hrs
29		1.3 SİSTEM GEREKSİNİMLERİNİN VE MİMARİSİNİN BELİRLENMESİ	60 days	Mon 09.02.09	Fri 01.05.09	960 hrs
30		1.3.1 Sistem gereksinimleri analizi	40 days	Mon 09.02.09	Fri 03.04.09	640 hrs
32		Sistem gereksinimleri belgesi	0 days	Fri 03.04.09	Fri 03.04.09	0 hrs
33		1.3.3 Sistem mimarisinin belirlenmesi	20 days	Mon 06.04.09	Fri 01.05.09	320 hrs
35		Sistem mimari dokumani	0 days	Fri 01.05.09	Fri 01.05.09	0 hrs
36		1.4 DONANIM GELİŞTİRMЕ	40 days	Mon 04.05.09	Fri 26.06.09	800 hrs
38		1.5 YAZILIM GELİŞTİRMЕ	302 days	Mon 04.05.09	Tue 29.06.10	5.304 hrs
39		1.5.1 Çevrim-1: ATM Kart Yazılım Paketi Geliştirme	245 days	Mon 04.05.09	Fri 09.04.10	2.344 hrs
40		1.5.1.1 Çevrim-1: Yazılım gereksinimlerinin belirlenmesi	30 days	Mon 04.05.09	Fri 12.06.09	320 hrs
41		Yazılım gereksinimleri analizi	20 days	Mon 04.05.09	Fri 29.05.09	160 hrs
42		Yazılım gereksinimleri belgesinin hazırlanması	5 days	Mon 01.06.09	Fri 05.06.09	80 hrs
43		Yazılım gereksinimleri belgesinin gözden geçirilmesi	1 day	Mon 08.06.09	Mon 08.06.09	24 hrs
44		Yazılım gereksinimleri belgesinin günlenmesi	3 days	Tue 09.06.09	Thu 11.06.09	48 hrs
45		Yazılım gereksinimleri belgesinin yapılandırma kontrolü altına alınması	1 day	Fri 12.06.09	Fri 12.06.09	8 hrs
46		Yazılım gereksinimleri belgesi	0 days	Fri 12.06.09	Fri 12.06.09	0 hrs

CMMI'da Proje Yönetimi (Kurumsal Seviye)

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
- Geçerli kılma
- Kurumsal süreç odağı
- Kurumsal süreç tanımı
- Kurumsal eğitim
- *Bütünleşik proje yönetimi*
- Risk yönetimi
- Karar analiz ve çözümleme

Proje Planlama Süreç Alanı ("Process Area")

Amaç: Proje etkinliklerini tanımlayan planları oluşturmak ve güncellemek

- SG1. Tahminleri oluştur
- SG2. Proje planı geliştir
- SG3. Plana taahhütleri sağla

Referans:

CMMI-DEV-V1.3, "Project Planning" Process Area

SG1. Tahminleri Oluştur

- Projenin kapsamının belirlenmesi
 - ▶ Görev tanımları
 - ▶ İş paketinin tanımları
 - ▶ İş Dağılım Yapısı (WBS)
- İş ürünü ve görev özelliklerinin kestirilmesi
 - ▶ Görevlerin ve iş ürünlerinin büyülüğu ve karmaşıklığı
 - ▶ Büyüklük: Ne kadar büyük bir sistem yapılacak?
 - ▶ Kestirim modelleri
 - ▶ Özellik kestirimleri
 - ▶ Projenin teknik yaklaşımı
- Projenin yaşam döngüsü aşamalarının belirlenmesi
 - ▶ Proje yaşam döngüsü adımları
- İşgücü ve maliyet kestirimlerinin yapılması
 - ▶ İşgücü: Ne kadar kişi/gün harcanacak?
 - ▶ Maliyet: Ne kadar YTL harcanacak?
 - ▶ Kestirimlerin geçmiş verilere dayalı olarak yapılması

Cıktı:
Planlama verileri

SG2. Proje Planı Geliştir

■ Planlama sırasında ele alınması gereken konular:

- ▶ Takvim ve kritik bağımlılıklar
- ▶ Bütçe
- ▶ Olası riskler ve öncelikleri
- ▶ Veri yönetim yaklaşımı
- ▶ Kaynaklar
- ▶ Proje ekibi için gereken bilgi ve yetkinlikler
- ▶ Proje kapsamındaki eğitimler
- ▶ Paydaşların katılımı

Cıktı:
Proje Planı

SG3. Plana Taahhütleri Sağla

- Oluşturulan proje yönetim planı hakkında geniş geribildirim almak ve planı bu çerçevede düzenlemek
 - ▶ İlişkili tüm planların gözden geçirilmesi
 - ▶ İş ve kaynak seviyelerinin eşlenmesi
 - ▶ Plana taahhütlerin alınması
- Kuruma özel yöntem tanımlanmalıdır.

Cıktı:
Düzeltilmiş Proje Planı

Proje Planlama – Özeti (1)

- Proje kapsamını belirle
- İş ürünlerinin ve iş adımlarının özelliklerini, işgücü ve maliyeti tahminle
- Proje yaşam döngüsünü tanımla
- Bütçe ve takvimi belirle
- Proje risklerini belirle
- Veri yönetimi, kaynak yönetimi, gerekli bilgi ve yetkinlik yönetimi için planla
- Belirlenen paydaşların katılımını planla
- Proje planının oluştur
- Planları gözden geçir ve iş seviyelerini uzlaştır
- Plana taahhütleri sağla

Proje Planlama – Özeti (2)

- Oluşturulan proje yönetim planı ve diğer belgeler
 - ▶ Gözden geçirilir
 - ▶ Yapılandırma kontrolü altına alınır
- Eğer planlama aşaması uzun sürecekse (örneğin birkaç ay);
 - ▶ Planlama aşaması da planlanır
- Proje gelişimi için temel alınacak ölçevler de tanımlanır

Proje İzleme ve Kontrol Süreç Alanı

Amaç: Proje performansı planlanandan farklılık gösterdiğinde, düzeltici faaliyetlerin uygulanabilmesi için projenin ilerlemesi hakkında bilgi oluşturmak

- SG1. Projeyi plana göre izle
- SG2. Düzeltici faaliyetleri kapanması için yönet

Referans:

CMMI-DEV-V1.3, “Project Monitoring and Control” Process Area

SG1. Plana Göre Projeyi İzle

■ Projenin planlara göre izlenmesi için yöntemlerin oluşturulması

- ▶ Proje yöneticisi tarafından izleme
- ▶ Proje ekibinin izlemesi
- ▶ Yönetim ve diğer paydaşların izlemesi

Girdi:
Proje Planı

■ İzleme

- ▶ Proje planlama parametreleri (büyüklük, işgücü, takvim, maliyet, kaynak vb.)
- ▶ Taahhütler, riskler, veri yönetimi, paydaş katılımı
- ▶ Durum değerlendirme
- ▶ Ölçevlere ilişkin verilerin incelenmesi
- ▶ Raporlandırma
- ▶ Periyodik gelişim toplantıları
- ▶ Kilometre-taşı toplantıları

Cıktı:
İzleme raporları
(*Toplantı tutanakları, ölçev analizleri vb.*)

SG2. Düzeltici Faaliyetleri Kapanması İçin Yönet

- Projelerde sorun olabilir, ancak sorunlar izleme sırasında belirlenir ve düzeltici faaliyetler ile giderilir
 - ▶ Sorunların analiz edilmesi
 - ▶ Düzeltici faaliyetlerin belirlenmesi
 - ▶ Düzeltici faaliyetlerin yönetilmesi
- Düzeltici faaliyetlerin açılması, değerlendirilmesi, ve gerçekleştirilmesi için kuruma özel yöntem oluşturulmalıdır

Cıktı:

Düzeltici faaliyet kayıtları

Proje İzleme ve Kontrol - Özeti

- Proje planlama parametrelerini izle
- Taahhütleri ve riskleri izle
- Veri yönetimini ve paydaş katılımlını izle
- Gelişme ve kilometre-taşı gözden geçirmeleri uygula
- Düzeltici faaliyetlerde bulun ve yönet

Bütünleşik Proje Yönetimi

Amaç: Projeyi ve ilgili paydaşların katılımını standart süreçlerden uyarlanan bütünlük ve tanımlı bir süreç ile yönetmek

- SG1. Projenin tanımlı sürecini kullan
- SG2. İlgili paydaşlar ile koordinasyon ve işbirliği yap

Referans:

CMMI-DEV-V1.3, “Integrated Project Management” Process Area

SG1. Projenin Tanımlı Sürecini Kullan

- Projenin tanımlı sürecinin oluşturulması
 - ▶ Projenin tanımlı süreci
- Proje etkinliklerinin planlaması için kurumsal süreç varlıklarının kullanılması
 - ▶ Proje kestirimleri ve proje planları
- Projenin çalışma ortamının kurulması
 - ▶ Proje için teçhizat ve araçlar, kurulum için kılavuzlar, destek servisler
- Proje planlarının tümleştirilmesi
 - ▶ Bütünleşik planlar
- Projenin bütünleşik planlara göre yönetilmesi
 - ▶ Bütünleşik adımların iş ürünleri, ilişkili ölçevler ve durum raporları
- Takımların kurulması
 - ▶ Takımlar, takım yapıları ve elemanları, takım durum raporları
- Kurumsal süreç varlıklarına katkıda bulunulması
 - ▶ Kurumsal varlıklara ilişkin iyileştirme önerileri

Cıktılar:
*Bütünleşik planlar
Çalışma ortamı*

SG2. İlgili Paydaşlar ile Koordinasyon ve İşbirliği Yap

■ Paydaşların katılımının yönetilmesi

- ▶ Ortak etkinlikler için gündem ve takvim
- ▶ Paydaş sorunlarının çözümü için öneriler
- ▶ Sorun tanımları

Çıktılar:

Paydaş sorunları

Kritik bağımlılıklar

Koordinasyon sorunları

■ Bağımlılıkların yönetilmesi

- ▶ Paydaş görüşmelerinde tespit edilen hatalar, sorunlar ve faaliyet maddeleri
- ▶ Kritik bağımlılıklar, bunların çözümlerine ilişkin taahhütler, gelişim durumları

■ Koordinasyon sorunlarının çözümlenmesi

- ▶ Paydaşlara ilişkin koordinasyon sorunları
- ▶ Koordinasyon sorunlarının gelişim durumları

Bütünleşik Proje Yönetimi - Özeti

- Projenin tanımlı süreçlerini oluştur
- Kurumsal süreç varlıklarını kullan
- Projenin çalışma ortamını hazırla
- Planları bütünlleştir
- Projeyi bütünleşik planlar ile yönet
- Deneyimleri kurumsal süreç varlıklarına kat
- Paydaşların katılımını ve bağımlılıklarını yönet
- Koordinasyon hususlarını çöz

MS Project ile Proje Planlama: ATM Örneği

Proje Yönetimi ve Ölçme

Amaç

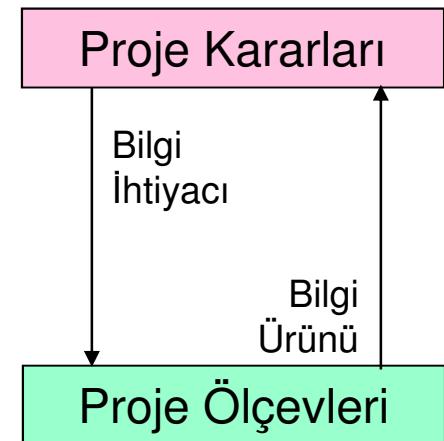
- ▶ Projelerde ölçme ve analiz etkinliklerini tanımlamak ve uygulamak

Prensipler

- ▶ Hedefe yönelik veri toplamak
- ▶ Verinin tutarlı şekilde toplanmasını ve analiz edilmesini sağlamak
- ▶ Analiz sonuçlarını karar almada kullanmak

Uygulama

- ▶ Proje Yöneticileri, Kalite Güvence Sorumluları, Geliştiriciler, vb.
 - ◆ Proje hedeflerinin koyulması ve değerlendirilmesi
 - ◆ Proje gelişiminin (sapmalar, hatalar, teslimatlar vs.) detaylı takibinin yapılması
- ▶ Üst Yönetim
 - ◆ Proje performanslarının (kar/zarar, vb.) değerlendirilmesi



Bilgi İhtiyacı

■ Hedefe yönelik, karar almaya esas

- ▶ Bilgi ihtiyacı ve ilişkili indikatör (tablo, grafik, vs.)
 - ◆ Ölçme kullanıcısı ne görmek istiyor?

Örnek:

Proje Yöneticisi harcanan işgücünü takip etmek istiyor (“bilgi ihtiyacı”)

İndikatör:

“Proje İşgücü Gelişimi” grafiği

Karar alma kriteri:

Planlanan ve gerçekleşen işgückenin kümülatif farkı %10'u aşarsa, düzeltici faaliyet al

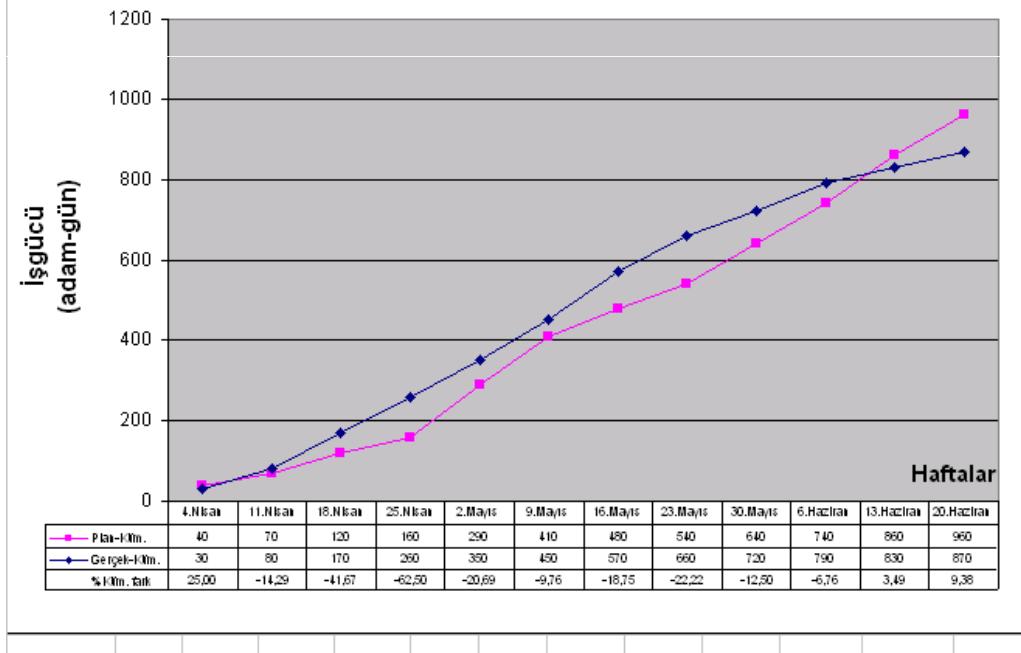
(yeniden planla, personel al/kaydır, tahminleme yöntemini iyileştir, vs.)

Proje İşgücü Gelişimi

	4.Nisan	11.Nisan	18.Nisan	25.Nisan	2.Mayıs	9.Mayıs	16.Mayıs	23.Mayıs	30.Mayıs	6.Haziran	13.Haziran	20.Haziran
Gerçekleşen	30	50	90	90	90	100	120	90	60	70	40	40
Planlanan	40	30	50	40	130	120	70	60	100	100	120	100
Gerçek-Küm.	30	80	170	260	350	450	570	660	720	790	830	870
Plan-Küm.	40	70	120	160	290	410	480	540	640	740	860	960
%Küm. fark	25,00	-14,29	-41,67	-62,50	-20,69	-9,76	-18,75	-22,22	-12,50	-6,76	3,49	9,38
Ort. sapma	20,58											

Proje İşgücü Gelişimi

(Hedef: Ortalama İşgücü Sapma $\geq -10\%$)



Ölçme Yapısı

Ölçme Yapısı

	ÖY #:	1
	İlişkili BİF #:	1
Bilgi İhtiyacı	Projenin İşgücü gelişimini değerlendirmek	
Ölçülecek Kavram	Proje İşgücü performansı	
Uygulanan Seviye	[] Kurum [x] Proje	
Ölçürülecek Nesne	[] Süreç _____ [] Ürün _____ [x] Kaynak _____	
İlişkili Varlıklar	1. Proje Yönetim Planı 2. İşgücü Veri Dosyası	
Ölçülecek Özellikler	1. Planlanan İşgücü 2. Gerçekleşen İşgücü	
Temel Ölçüler	1. Haftalık planlanan İşgücü 2. Haftalık gerçekleşen İşgücü	
Ölçme Yöntemi	1. Haftalık planlanan İşgütünü, Proje Yönetim Planı'ndan al 2. Haftalık gerçekleşen İşgütünü, İşgücü Veri Dosyasından al	
Ölçme Yöntemi Türü	1. Objektif 2. Objektif	
Veri Aralığı	1. 0'dan sonsuza 2. 0'dan sonsuza	
Ölçek	1. Oransal 2. Oransal	
Ölçme Birimi	1. Adam-gün 2. Adam-gün	
Türetilmiş Ölçüler	1. Kümülatif planlanan İşgücü 2. Kümülatif gerçekleşen İşgücü 3. % Kümülatif fark	
Ölçme Fonksiyonu	1. Girilen tarihe kadar (Σ haftalık planlanan İşgücü) 2. Girilen tarihe kadar (Σ haftalık gerçekleşen İşgücü) 3. (Kümülatif planlanan İşgücü – Kümülatif gerçekleşen İşgücü) / Kümülatif planlanan İşgücü * 100	
İndikatör	Projenin toplam planlanan ve gerçekleşen İşgücü değerlerinin (ve aradaki sapmanın) haftalık olarak seyri	
Model	Toplam planlanan ve gerçekleşen İşgücü arasındaki % Kümülatif fark, 0'a yakın olmalı	
Karar Kriteri	Eğer planlanan ve gerçekleşen İşgücü arasındaki fark %10'un altındaysa düzeltici faaliyet al (tekrar planla, personel kaydır, yeni personel al, tahminleme yöntemini iyileştir, vs.)	

■ Tutarlı veri toplama ve analiz için

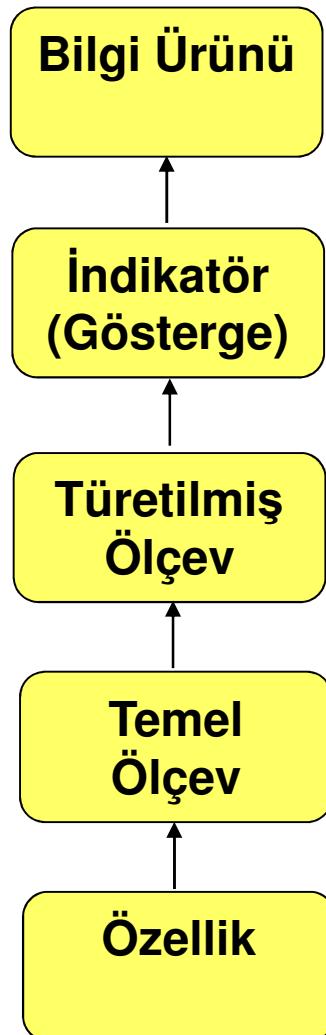
► Ölçme yapısı

◆ İndikatörün tasarımı

Bilgi İhtiyacı (“Information Need”)

- Bilgi ihtiyaçları direkt olarak ölçme hedefleriyle ve bu hedeflerin başarılmasını etkileyebilecek konularla ilişkilidir
 - ▶ Amaçları, hedefleri, riskleri ve problemleri yönetmek için gerekli öngörüyü tanımlar
 - ▶ Proje özelinde proje boyunca kullanılacak her türlü bilgi isteği
 - ◆ Örnek: Biten projelerin üretkenlik değerlerine göre yeni projenin üretkenliğini kestirmek

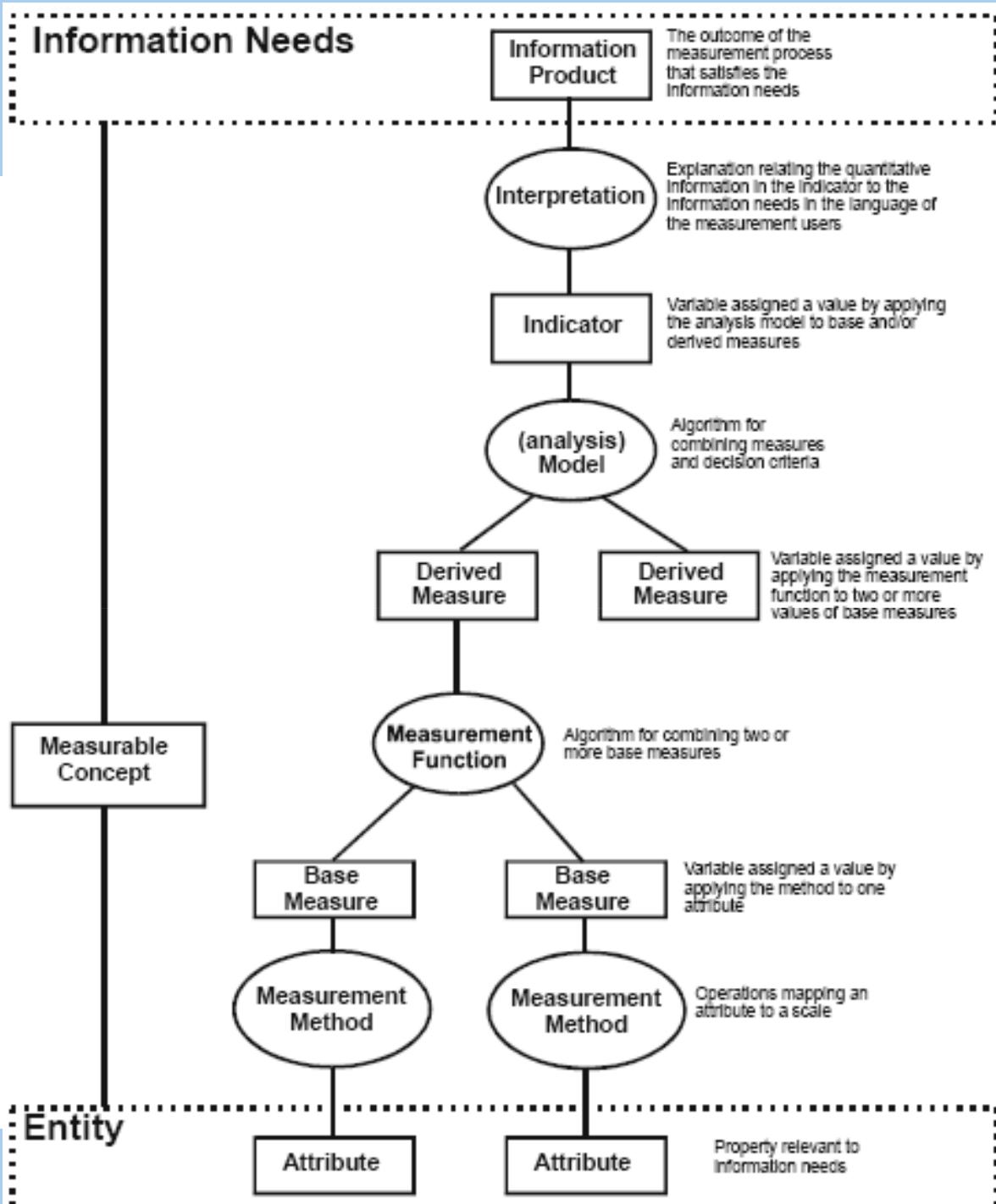
Ölçme Yapısı (“Measurement Construct”)



- Bilgi ürünü
 - ▶ Bilgi ihtiyacını karşılayan indikatör ve yorumu
- İndikatör
 - ▶ Türetilmiş ölçev sonuçlarını belirli bir modele göre ifade etmeyi sağlayan görsel öge (tablo, grafik, vb.)
- Türetilmiş ölçev
 - ▶ İki veya daha çok temel ölçev değerinin fonksiyonu olarak tanımlanmış ölçev
- Temel ölçev
 - ▶ Belirli bir özelliğin ve onu nicelemek için kullanılan yöntemin cinsinden tanımlanmış ölçev
- Özellik
 - ▶ Bir varlığın, insan veya araçla nitel ya da nicel olarak ayırt edilebilen karakteristiği

Bilgi İhtiyacı -

Ölçme Yapısı İlişkisi



Referans:

ISO Std. 15939:2002 –
Software Measurement Process

Ölçme Yapısı: Örnek

INDICATORS

Adjusted mean, two standard deviation confidence limits, and projected new product capability

Productivity Estimate

Analysis Model

Compute mean and standard deviation; multiply mean by new project capability

DERIVED MEASURES

Repeat for each historical project

Project X Productivity

Value for Project X

New Project Capability

Measurement Method

Rate subjectively as 0.9, 1.0, 1.1, ...

BASE MEASURES

Value for Project X

Effort Project X

Size Project X

Value for Project X

Experience

Planned Personnel

Count total hours

Measurement method

Count semicolons

Measurement method

ATTRIBUTES

Timecards

Hours

Semicolons

Ada Source Statements