Hello World!

- All code must be contained in a class 1
  - Class: piece of code containing variables and methods
  - Object: Instance of a class
    - Created and destroyed while the code runs
  - Static methods belong the class. Non-static methods belong to the object
- Entry point: public static void main(String[ ] args){...} 2
- Private variables/functions: Only the class can see it
- Public variables/functions: Everyone can see it
- Protected variables: Can be seen by the class, subclasses, and package classes 3

Objects

4

- Everything that isn't a primitive!
- Strings are built in and are special - don't have to be created with "new"
- Many other objects that be accessed by importing (List, Set, Map, etc…)

5

- May contain a constructor: called when the object is created.
- Or you can create your own objects!
- Classes can be subclassed to create a hierarchy
- Abstract Objects
  - Cannot be instantiated. Have some implemented methods and some unimplemented abstract methods
- Interfaces
  - Cannot be instantiated. Have only abstract methods

Pointers (References!)

- In Java, pointers are known as references.
- Methods pass by value, not pointer.
  - Buuuut they pass by the value of the pointer!!! Leads to much confusion
- Use .equals() instead of == to compare Objects or else you are just comparing the references!

GitLab

- TC Dong

Which command is used to pull an entire library down?

Software Engineering

- Because of their abstract nature, Software Systems can be:
  - Complex
  - Difficult to understand
  - Expensive to change
- Due to the fact that almost everything requires software these days, there is no one process that covers all types of software.
- But, there is a general description of the processes that need to be followed and tailored for the specific project through all phases
- The *software lifecycle* is the progression of software through these phases
- *Software Engineering is an engineering discipline that is concerned with all aspects of software production from initial conception to operation and maintenance*.

The Software Lifecycle

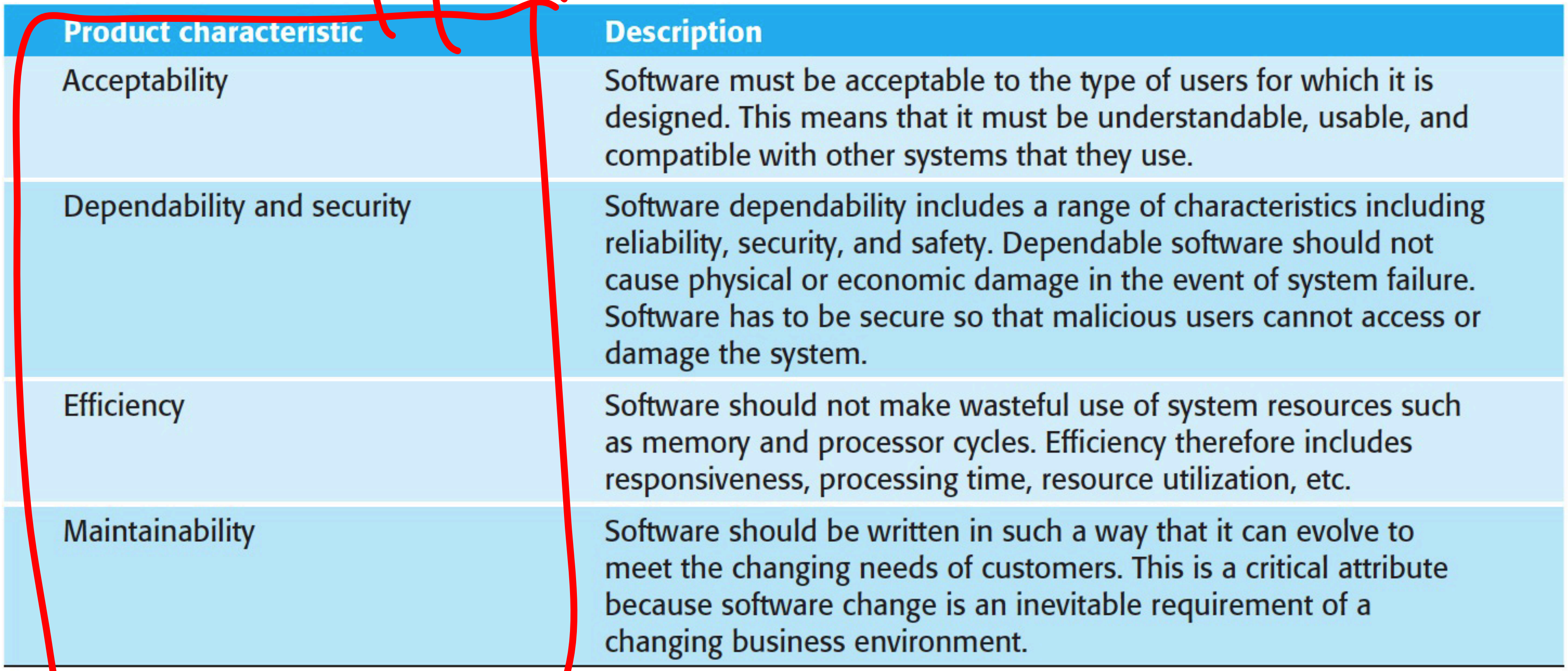
- There are *4 fundamental activities* in the software lifecycle:
  - Specification – definition of what is to be created and how deployed
  - Development – creation of the software – design and code
  - Validation – testing, quality control, verification that it meets rqmts
  - Evolution – maintenance, enhancements due to changing requirements
- Software engineering helps define processes for each activity
- In addition to having defined processes, any type of software (standalone, embedded, web, mobile, etc) needs to:
  - Have very clear requirements
  - Be dependable and perform
  - Reuse when possible

Development

- The process of developing an executable system for delivery to the customer.
  - Design – description of the structure of the software to be implemented, the data models, the interfaces and algorithms.
  - Implementation – the actual programming / coding of the system.
- Design generally follows a structured input-> activity-> output model
- The Design outputs are the inputs to Implementation
- Programmers generally do their own “unit test” before software validation phase

10

Figure 1.2 Essential attributes of good software



Product characteristic	Description
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.
Dependability and security	Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Software has to be secure so that malicious users cannot access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, resource utilization, etc.
Maintainability	Software should be written in such a way that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.

Software Process Models

- A Software Process Model is a simplified representation of a software process.
- Each process model represents a process from a particular perspective
- There are 2 major types of general process models:
  - Waterfall model – Treats the 4 major process activities as separate, serial phases, each with well defined sub-processes
  - Incremental development model – various ways to interleave the 4 major process activities in order to create a more efficient overall process

12

The waterfall model

- The first published model for software development
- Represents software development as a cascading set of 5 stages: 13
  - Requirements definition – system's services, constraints and goals are established by consultation with system users.
  - System and software design – hardware/software requirements, system architecture, components and relationships
  - Implementation and unit testing – instantiate the design in code units. Test each unit to ensure it meets specification
  - Integration and system testing – Integration of program units into a system and tested vs. requirements
  - Operation and maintenance – on-going support and enhancements
- One phase does not start until exit criteria of previous phase are met

Incremental development model

- Based on the idea of developing an initial implementation, getting feedback, and evolving software through iteration.
- Specification, development and validation activities are interleaved
- Best when requirements are likely to change and the team is able to interact easily across all phases
- “Agile Development” is the preeminent example of an incremental development approach

14

Advantages	Disadvantages
Cost of implementing requirement changes is lowered	System structure tends to degrade as new increments are added.
Easier to get feedback on running code	The process is not visible
Early delivery of working code is possible	Very difficult for large complex systems

Requirements Engineering Overview

15

- *The process of establishing the services that a customer requires from a system and the constraints under which it operates and is developed.*
- The system requirements are the descriptions of the system services and constraints that are generated during the requirements engineering process.

Functional and non-functional requirements

- Functional requirements
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
  - May state what the system should not do.

16

- Non-functional requirements
  - Not directly concerned with the specific services delivered by the system to its users
  - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
  - Often apply to the system as a whole rather than individual features or services.
  - Examples include Reliability, Response Time and Memory Use

Non-functional requirements

- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular IDE, programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

Non-functional classifications

14

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organizational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

Metrics for specifying nonfunctional requirements

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Requirements engineering processes

- The processes used for Requirements Engineering (RE) vary widely depending on the application domain, the people involved and the organization developing the requirements.
- However, there are a number of generic activities common to all processes
  - Requirements elicitation;
  - Requirements analysis;
  - Requirements validation;
  - Requirements management.
- In practice, RE is an iterative activity in which these processes are interleaved.

20

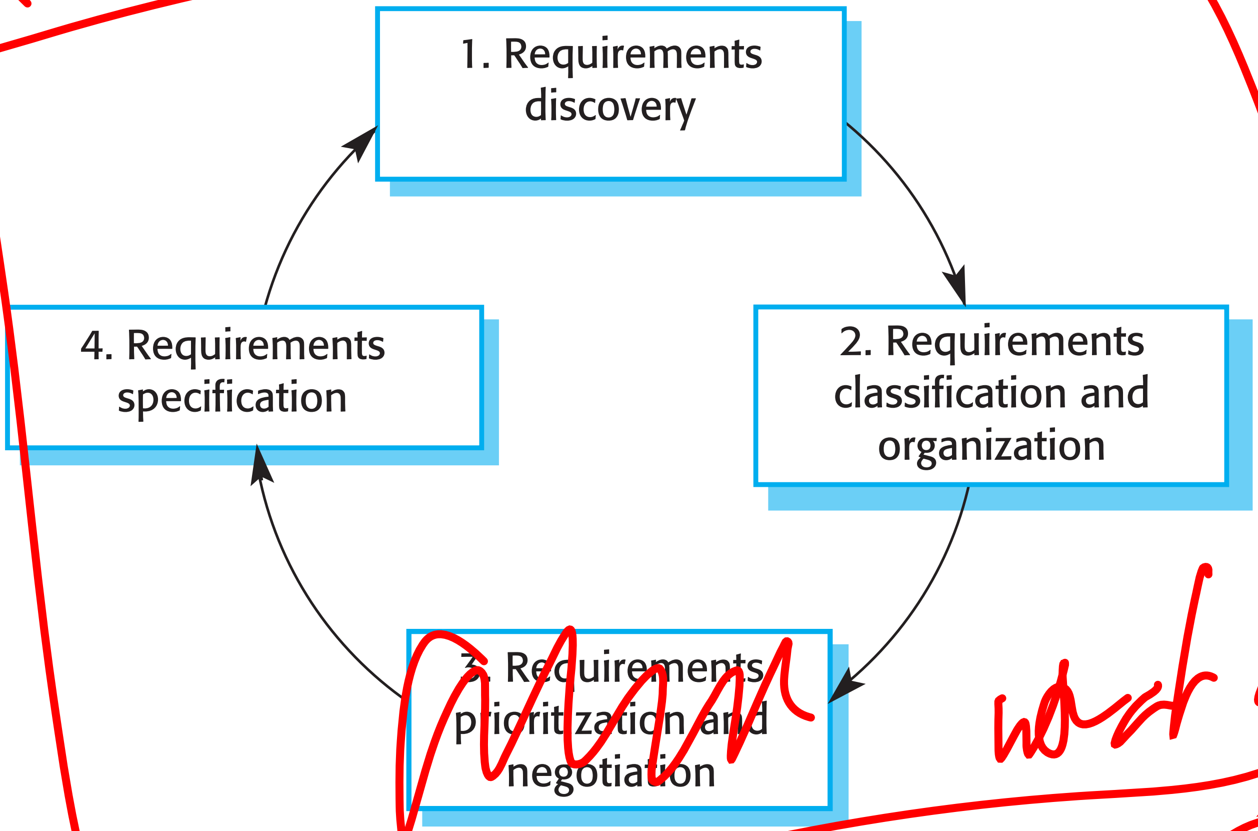
Requirements elicitation and analysis

21

- Sometimes called requirements elicitation or requirements discovery.
- Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system’s operational constraints.
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders.*

The requirements elicitation and analysis process

22


1. Requirements discovery
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements specification

what is missing?

Requirements and design

23

- In principle, **requirements** should state what the system should do and the **design** should describe how it does this.
- In practice, *requirements and design are inseparable*
  - A system architecture may be designed to structure the requirements;
  - The system may inter-operate with other systems that generate design requirements;
  - The use of a specific architecture to satisfy non-functional requirements may be a domain requirement.
  - This may be the consequence of a regulatory requirement.

Fitb

Use cases

2) which is not TRUE

- Use-cases are scenarios of how users interact with the system
- Use cases identify the actors in an interaction and which describe the interaction itself.
- A set of use cases should describe all possible interactions with the system.
- High-level graphical model supplemented by more detailed tabular description (see Chapter 5).
- We will discuss more when we get to Unified Modeling Language (UML) sequence diagrams, which may be used to add detail to use-cases by showing the sequence of event processing in the system.

24

The software requirements document

25

- The software requirements document is the official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

Agile development

26

- Program specification, design and implementation are inter-leaved
- The system is developed as a series of versions or increments with stakeholders involved in version specification and evaluation
- Frequent delivery of new versions for evaluation
- Extensive tool support (e.g. automated testing tools) used to support development.
- Minimal documentation – focus on working code

Which is not in [illegible]

Agile methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.

27

- The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

F, tb

The principles of agile methods

2d

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Extreme programming

- A very influential agile method, developed in the late 1990s, that introduced a range of agile development techniques.
- Extreme Programming (XP) takes an 'extreme' approach to iterative development.
  - New versions may be built several times per day;
  - Increments are delivered to customers every 2 weeks;
  - All tests must be run for every build and the build is only accepted if tests run successfully.

27

Examples of refactoring

- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

30 which are examples of Refectring?

Scrum

- Scrum is an agile method that focuses on managing iterative development rather than specific agile practices.
- There are three phases in Scrum.

Fitb

  - The initial phase is an outline planning phase where you establish the general objectives for the project and design the software architecture.

31

  - This is followed by a series of sprint cycles, where each cycle develops an increment of the system.
  - The project closure phase wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.

Scrum terminology (b)

Scrum term	Definition
Scrum	A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team.
ScrumMaster 32 f:tb	The ScrumMaster is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference. The Scrum developers are adamant that the ScrumMaster should not be thought of as a project manager. Others, however, may not always find it easy to see the difference.
Sprint	A development iteration. Sprints are usually 2-4 weeks long.
Velocity	An estimate of how much product backlog effort that a team can cover in a single sprint. Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.

Scrum benefits

- The product is broken down into a set of manageable and understandable chunks.
- Unstable requirements do not hold up progress.
- The whole team have visibility of everything and consequently team communication is improved.
- Customers see on-time delivery of increments and gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

33 what is a benefit of Scrum

Recap of Week 3

- Agile methods are incremental development methods that focus on rapid software development, frequent releases of the software, reducing process overheads by minimizing documentation and producing high-quality code.
- Agile development practices include
  - User stories for system specification
  - Frequent releases of the software,
  - Continuous software improvement
  - Test-first development
  - Customer participation in the development team.

34

- Scrum is an agile method that provides a project management framework.
  - It is centred round a set of sprints, which are fixed time periods when a system increment is developed.
- Many practical development methods are a mixture of plan-based and agile development.
- Scaling agile methods for large systems is difficult.
  - Large systems need up-front design and some documentation and organizational practice may conflict with the informality of agile approaches.

Announcements

- Upcoming Java Dev lectures
  - Today - UI Development for Android
  - Today - Team-based source control in Git
  - Feb 17th - Java Web UI Development
  - Feb 24th - AWT UI Development
- Mid Term will be March 2nd
  - Covers all material covered to date
  - Focus will be on topics described in “Key Points”, “Recap”, italicized text, bolded text, homework assignment topics.
  - Also one or two high level questions from the Agile Library, a few simple ones on Git and Java
  - Make sure you read the book to reinforce
  - A few questions will be straight from book (maybe not in a lecture), others straight from lectures (maybe not in the book)
  - Multiple Choice, some Fill in the Blank, maybe some “Other”

System modeling

- System modeling is the process (WHAT) of developing **abstract** models of a system, with each model presenting a different view or perspective of that system.
- System modeling has now come to mean representing a system using some kind of graphical notation (HOW), which is ~~now almost always based on~~ notations in the **Unified Modeling Language** (UML). the Stree
- System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers. 35

System perspectives

36

- An *external perspective*, where you model the context or environment of the system.
- An *interaction perspective*, where you model the interactions between a system and its environment, or between the components of a system.
- A *structural perspective*, where you model the organization of a system or the structure of the data that is processed by the system.
- A *behavioral perspective*, where you model the dynamic behavior of the system and how it responds to events.

Topics covered

- Context models
- Interaction models
- Structural models
- Behavioral models
- Model-driven engineering

37

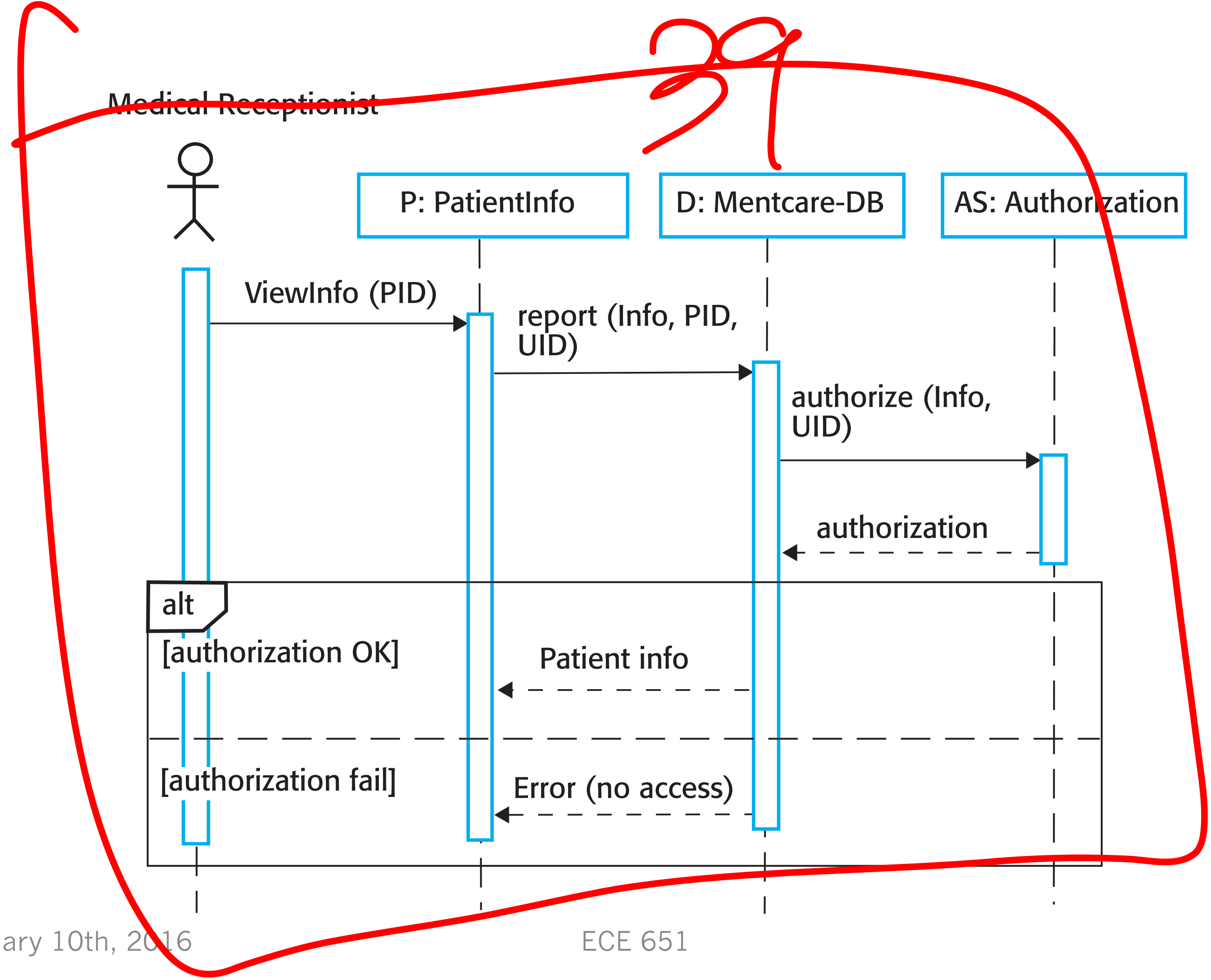
Use case modeling

- Use cases were developed originally to support requirements elicitation and now incorporated into the UML.

38

- Each use case represents a **discrete task** that involves external interaction with a system.
- *Actors* in a use case may be people or other systems.
- Represented diagrammatically to provide an overview of the use case and in a more detailed textual form.

Sequence diagram for View patient information



Generalization

37

- In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization. If changes are proposed, then you do not have to look at all classes in the system to see if they are affected by the change.
- In object-oriented languages, such as Java, **generalization is implemented using the class inheritance mechanisms** built into the language.
- In a generalization, the attributes and operations associated with higher-level classes are also associated with the lower-level classes.
- The lower-level classes are subclasses inherit the attributes and operations from their superclasses. These lower-level classes then add more specific attributes and operations.

Key points

- A model is an abstract view of a system that ignores system details. Complementary system models can be developed to show the system's context, interactions, structure and behavior.
- Context models show how a system that is being modeled is positioned in an environment with other systems and processes.
- Use case diagrams and sequence diagrams are used to describe the interactions between users and systems in the system being designed. Use cases describe interactions between a system and external actors; sequence diagrams add more information to these by showing interactions between system objects.
- Structural models show the organization and architecture of a system. Class diagrams are used to define the static structure of classes in a system and their associations.

40 which are the 2 best?

Architectural design

- *Architectural design is concerned with understanding how a software system should be organized and designing the overall structure of that system.*

41

- Architectural design is the **critical link between design and requirements engineering**, as it identifies the main structural components in a system and the relationships between them.

OR

- The output of the architectural design process is an **architectural model** that describes how the system is organized as a set of communicating components.

Architectural representations

- **Simple, informal block diagrams** showing *entities* and *relationships* are the most frequently used method for documenting software architectures.
- But these have been criticized because they lack semantics, do not show the types of relationships between entities nor the visible properties of entities in the architecture.

42

- Depends on the use of architectural models. The requirements for model semantics depends on how the models are used.

Architectural patterns

- **Patterns are a means of representing, sharing and reusing knowledge.**
- *An architectural pattern is a stylized description of good design practice, which has been tried and tested in different environments.*
- Patterns should include information about when they are and when the are not useful.
- Patterns may be represented using tabular and graphical descriptions.

43 –

list of patterns

Build or buy

- In a wide range of domains, it is now possible to buy off-the-shelf systems (COTS) that can be adapted and tailored to the users' requirements.
  - For example, if you want to implement a medical records system, you can buy a package that is already used in hospitals. It can be cheaper and faster to use this approach rather than developing a system in a conventional programming language.
- When you develop an application in this way, the design process becomes concerned with how to use the configuration features of that system to deliver the system requirements.

Examples of design models

- Subsystem models that show logical groupings of objects into coherent subsystems. Which is NOT.
- Sequence models that show the sequence of object interactions.
- State machine models that show how individual objects change their state in response to events.
- Other models include use-case models, aggregation models, generalisation models, etc.

45

The Observer pattern

- Name
  - Observer.
- Description
  - Separates the display of object state from the object itself.
- Problem description
  - Used when multiple displays of state are needed.
- Solution description
  - See slide with UML description.
- Consequences
  - Optimisations to enhance display performance are impractical.

Which of these was given as an example of a design pattern?

47

What are employers looking for?

- We had a discussion with representatives from various tech companies, including IBM, Google, and NetApp. This is a composite of their feedback.
  - Although language needs varied, the most common request was for C and C++, with good understanding of **OO programming.**
  - Solid in fundamentals, basic algorithms, data structures
  - One company mentioned that the best prep is to enter a **coding contest** / hack-a-thon
  - "Soft skills" were emphasized – **self-directed**, working in teams (including cross-functional), f2f as well as remote/conf call, ability to triage, distill, **prioritize**. Have a "**learning personality**." Ability to **focus** and block out distractions.
  - Project management was mentioned – can you do accurate **estimation** of how long it will take you to complete a task? Understanding the concept of "**MVP**"
  - Software Engineering – code health, code hygiene, version control, documentation, brownfield development, test, open source integration, "**write it for the next guy**."
  - A graduate level class on **Security** was suggested.
  - Know when to step back and take a look at the big picture.
  - Know when to **take a break**!
  - Know Dev/Ops and Agile
  - Be customer-driven – know what they want and don't do stuff they didn't ask for!
- Let me know if you need help developing in any of these areas!

50

Program testing

- Testing is intended to *show that a program does what it is intended to do and to discover program defects before it is put into use.*
- When you test software, you execute a program using artificial data.

51 F: 7b of M2

- You check the results of the test run for errors, anomalies or information about the program’s non-functional attributes.
- Can reveal the presence of errors NOT their absence.
- Testing is part of a more general verification and validation process, which also includes static validation techniques.

Validation and defect testing

- The first goal leads to *validation testing*
  - You expect the system to perform correctly using a given set of test cases that reflect the system's expected use.
- The second goal leads to *defect testing*
  - The test cases are designed to expose defects. The test cases in defect testing can be deliberately obscure and need not reflect how the system is normally used.

Testing process goals

- Validation testing
  - To demonstrate to the developer and the system customer that the software meets its requirements
  - A successful test shows that the system operates as intended.
- Defect testing
  - To discover faults or defects in the software where its behaviour is incorrect or not in conformance with its specification
  - A successful test is a test that makes the system perform incorrectly and so exposes a defect in the system.

Also

so

Verification vs validation

- *Verification:*
  "Are we building the product right".
- The software should conform to its specification.
- *Validation:*
  "Are we building the right product".
- The software should do what the user really requires.

53

Inspections and testing

- Software inspections. Concerned with analysis of the static system representation to discover problems (static verification)
  - May be supplement by tool-based document and code analysis.
  - Discussed in Chapter 15.

S4

- Software testing. Concerned with exercising and observing product behaviour (dynamic verification)
  - The system is executed with test data and its operational behaviour is observed.

Stages of testing

- *Development testing*, where the system is tested during development to discover bugs and defects.
- *Release testing*, where a separate testing team test a complete version of the system before it is released to users.
- *User testing*, where users or potential users of a system test the system in their own environment.

55

Development testing

- Development testing includes all testing activities that are carried out by the team developing the system.
  - Unit testing, where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.
  - Component testing, where several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.
  - System testing, where some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing component interactions.

56

Regression testing

- Regression testing is testing the system to check that changes have not ‘broken’ previously working code.
- In a manual testing process, regression testing is expensive but, with automated testing, it is simple and straightforward. All tests are rerun every time a change is made to the program.

57

- Tests must run ‘successfully’ before the change is committed.

Release testing

- Release testing is the process of testing a particular release of a system that is intended for use outside of the development team.
- The primary goal of the release testing process is to convince the supplier of the system that it is good enough for use.
    - Release testing, therefore, has to show that the system delivers its specified functionality, performance and dependability, and that it does not fail during normal use.

56

- Release testing is usually a black-box testing process where tests are only derived from the system specification.

Release testing and system testing

- *Release testing is a form of system testing.*
- Important differences:
  - A separate team that has not been involved in the system development, should be responsible for release testing.
  - System testing by the development team should focus on discovering bugs in the system (defect testing). The objective of release testing is to check that the system meets its requirements and is good enough for external use (validation testing).

59

Performance testing

- Part of release testing may involve testing the emergent properties of a system, such as performance and reliability.
- Tests should reflect the profile of use of the system.

59

- Performance tests usually involve planning a series of tests where the load is steadily increased until the system performance becomes unacceptable.
- Stress testing is a form of performance testing where the system is deliberately overloaded to test its failure behaviour.

Types of user testing

- Alpha testing
  - Users of the software work with the development team to test the software at the developer's site.

Go

- Beta testing
  - A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.
- Acceptance testing
  - Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom systems.

The Project Plan, continued

- WHAT – task definitions should be small enough that you can do accurate duration estimation, but large enough to be meaningful.
  - Too granular a task makes the overhead of project management unnecessarily complex
  - It is a good practice to have a short Task ID as well as a longer task description
  - Tasks may have a 1:1 map a requirement, but generally not always the case.
  - If you have user stories or use cases, they generally break down into a set of tasks that are on a project plan.
- WHO – generally one team member for our scale projects, but that is dependent on the task.
- WHEN – estimated start and finish dates. If the start date is gated by another task before it can start, that should be noted somewhere (Dependency).

60

- Based on dates and dependencies, you should be able to see your "*critical path*"

61