Lecture

Content Keywords

* Traditional software development process methodologies
* Metrics
* Software estimation techniques
* Project management techniques
* Risk management techniques
* Software quality assurance
* Testing techniques
* Agile methodologies
* Case studies

Success Factors

* Scope – meet project specifications – features and functionality
* Quality -Produce the Software with the best Quality
* Time -Do the Software development in time (preferably on time)
* Budget -Do the Software development within the budget

Failed Factors of a Software

* Late Delivery
* Over budget
* Unreliable systems
* Difficult to maintain

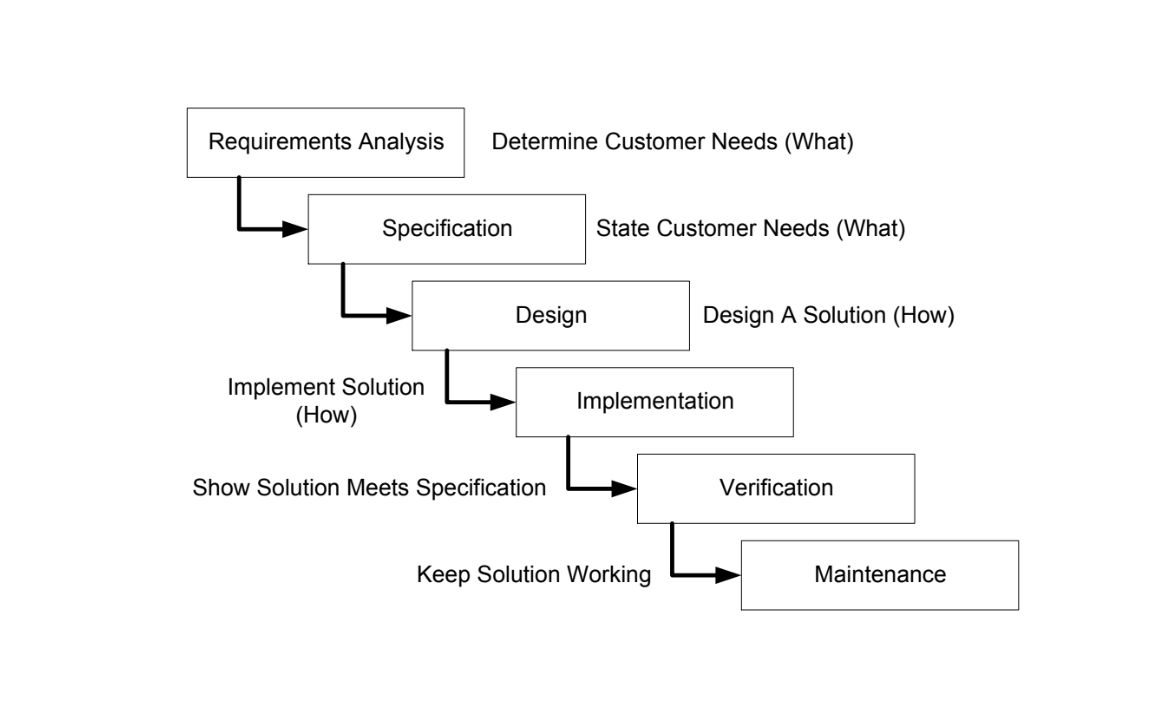
Lecture 2

Software Development Process Models

LOC = Lines of Code

1000 LOCS = 1KLOC

Development Phase of a Software (Possible keywords to ask or find in the document)



Software Development Processes

* Waterfall model
* Iterative models
  + Incremental Model
  + Evolutionary Model
* Unified Process
* Agile methods
  + Scrum
  + Extreme Programming

Waterfall Model

It’s a phase by phase Model. What it means is first you do one step/phase and then proceed to the next one. Also its one of the most popular model in the industry.

Phases of the Model

Requirements -> Design -> Implementation -> Verification -> Maintenance

Advantages of the Model

* Simple and easy
* Useful for small Projects
* Easy to manage
* End Goal is determined early
* It provides structured ways to do things

Problems with waterfall model (Disadvantages)

* Complexity overload
* Delayed Feedback
* Frozen requirements
* Increasing cost of change
* Delayed risk mitigation

One big problem of the waterfall model is the late discovery of design defects and the change of the requirements in the process of it. As a result we will have costly over-runs and in some cases even project cancellation. Moreover you cant really verify whether you have stayed clear of a risk until late in the production of the software due to its phase by phase design.

Iterative Models(Incremental & Evolutionary)

Development is broken down in iterations. Every iteration repeats a “small Waterfall”.

Incremental Model

Development Process:

First an overall architecture of the total system is developed.

* Requirement Analysis
* Architectural Design

Detailed increments and releases are planned and lastly each increment is developed in its own lifecycle:

* Detailed design
* Coding
* Integration
* Testing
* Release

Increments may be built sequentially or in parallel depending on the nature of the dependencies among releases and on availability of resources.

Each increment adds additional or improved functionality to the system.

Each increment produces a deliverable increment of the software.

Advantages

* Increments are easier to design, implement, test and manage
* More adaptable to changing requirements
* Early User functionality and user Feedback
* Customer Satisfaction

Although it requires

* Clear interfaces between components
* Well known and understood user requirements.

Disadvantages

* Management should be in a continuous communication
* The complete requirements of the software should be clear
* Required good planning and designing
* Total cost of the Model is higher

Evolutionary Model

It is based on sequential/incremental development as well but:

* Does not require an architectural design first
* The first release is the architectural prototype
* Previous iterations experience is used to define next iteration requirements
* System evolves over time

Advantages

* All advantages of incremental model
* Works even when requirements are not yet decided or known
* Allows evolution and discovery of requirements

Disadvantages

* Difficult project planning due to uncertainty of user requirements and program features
* First architectural prototype may not be optimal to accommodate unplanned evolutions
* Criticism: similar to “code and fix”

Prototyping Technique of Evolutionary model

Strengths:

* Early functionality allows better requirements elicitation
* Risk control
* Formal specification embodied in an operating replica

Weakness:

* Quick and dirty
* Bad (or no) documentation
* System with poor performance
* Tendency for difficult problems to be pushed to the future

Dilemma: Throw away or keep?

Prototyping should be considered as a technique, NOT a process model

* After prototyping a process model must be followed
* Some process models encompass prototyping

Spiral Model

It has a risk driven approach instead of documents driven or code driven as the previous models.

Prototyping, simulations and benchmarking are used to control cost and risks

Resources are increased when the risk has been minimized

Only ¼ of total effort is spent on software development activities

Each cycle consists of the following high level processes:

* Determine objectives, alternatives and constrains
* Evaluate alternatives, identify and resolve risks
* Develop, verify next-level product
* Plan next phase

Cycles focus on tasks similar to the waterfall model like :

* Concept of operation
* Software requirements
* Product design
* Detailed design, code, test and integration

Advantages

* The advantage of spiral lifecycle model is that it allows elements of the product to be added in when they become available or known. This assures that there is no conflict with previous requirements and design
* This method is consistent with approaches that have multiple software builds and releases which allows making an orderly transition to a maintenance activity. Another positive aspect of this method is that the spiral model forces an early user involvement in the system development effort.
  + Waterfall
  + Incremental
  + Evolutionary

Disadvantages

* It takes a very strict management to complete such products and there is a risk of running the spiral in an indefinite loop. So, the discipline of change and the extent of taking change requests is very important to develop and deploy the product successfully.

Factors to be considered in choosing a process model:

* Risk
  + Technological
  + Business
  + Customer related
* Resources
  + Team size, experience
* Project size
* A process has to be adapted before and after adopting
  + Monitoring
  + Evaluation
  + Measuring

Software Metrics

Lecture 3

Why should we measure a software: To avoid subjective evaluations and empirical preconceptions

Aim is:

* Develop high quality software
  + Within:
    - Time constraint
    - Budget
  + But it meets the Scope
* Many measures have been considered for software quality such as interoperability, functionality and so on.

It has been observed that:

* Reliability
* Correctness
* Maintainability
* Accuracy
* Usability
  + Are most useful as they provide valuable indicators to the project team for product quality.

Reliability:

* Systems/Software should maintain performance under given conditions
* Defined as the ability to perform required functions over a specified time or operations
* Measured using
  + MTBF (Mean time Between Failure): Average time between failures
  + MTTR (Mean Time To Repair)

Correctness:

* A system or software must function correctly
* Correctness can be defined as the degree to which software performs its specified function
* It can be measured in terms of defects per KDLOC
* For quality assessment, defects are counted over a specified period of time

Maintainability:

* Software maintenance is costly and time-consuming
* Defined as the ease of modifying software to fix errors, meet new needs, or adapt to changes
* Measured using Mean Time to Change (MTTC): The time required to analyze design, implement, test and deploy changes.
* Lower MTTC indicates easier maintainability.

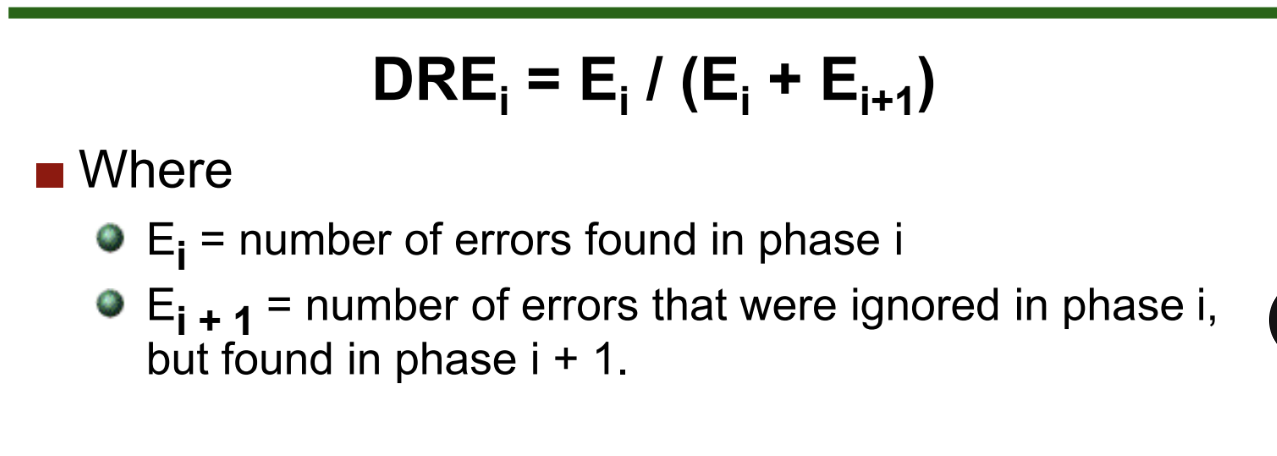
Integrity:

* Critical in countering cyber threats and unauthorized access
* Defined as the ability to control unauthorized access to software components (programs, data, documents)
* Measured using threat (probability of an attack) and security (probability of repelling an attack)
* There is a formula but not needed to know!

Usability:

* Software which is easy to understand and easy to use is always preferred by the user
* Usability can be defined as the capability of the software to be understood, learned and used under specified conditions
* Software which accomplishes all the user requirements but is not easy to use is often destined to fail.

Defect Removal Efficiency (DRE)

* Can be defined: as the quality metrics which is beneficial at both the project level and process level.
* Quality assurance and control activities that are applied throughout software development are responsible for detecting errors introduced at various phases of SDLC
* Formula DRE= E/(E+D)
  + Where
    - E = number of errors found before software is delivered to the user
    - D = number of defects found after software is delivered to the user
* The value of DRE approaches 1, if there are no defects in the software. As the value of E increases for a given value of D, the overall value of DRE starts to approach 1. With an increase in the value of E, the value of E, the value of D decreases as mor errors are discovered before the software is delivered to the user.
* DRE improves the quality of software by establishing methods which detect maximum number of errors before the software is delivered to the user.
* DRE is also used to assess the softwares team’s ability to find errors at each phase before they are passed on to the next development phase.
* When DRE is defined in the context of SDLC phases it can be calculated by the following equation.
* 

What is a metric :

* It is a quantitative (numeric), systematic (objective, repeatable) assessment to gain insight in the efficacy of the software system.
* We measure
  + Product
  + Process
  + Project

Functions Points/Functions Point Analysis (FP/FPA)

A functions point is a unit of measurement to express the amount of business functionality, an information system (as a product) provides to a user.

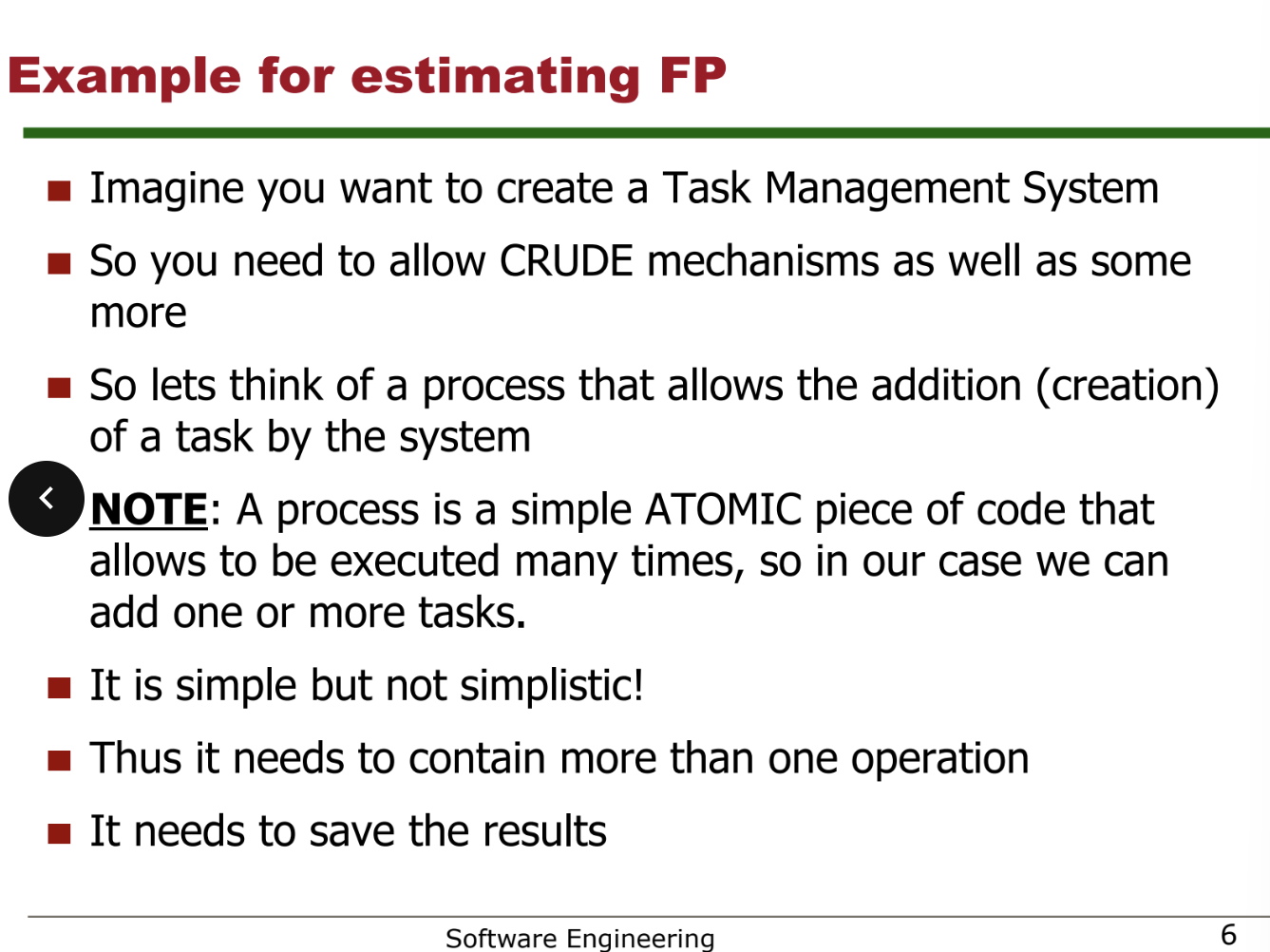
Functions Point Analysis (FPA)

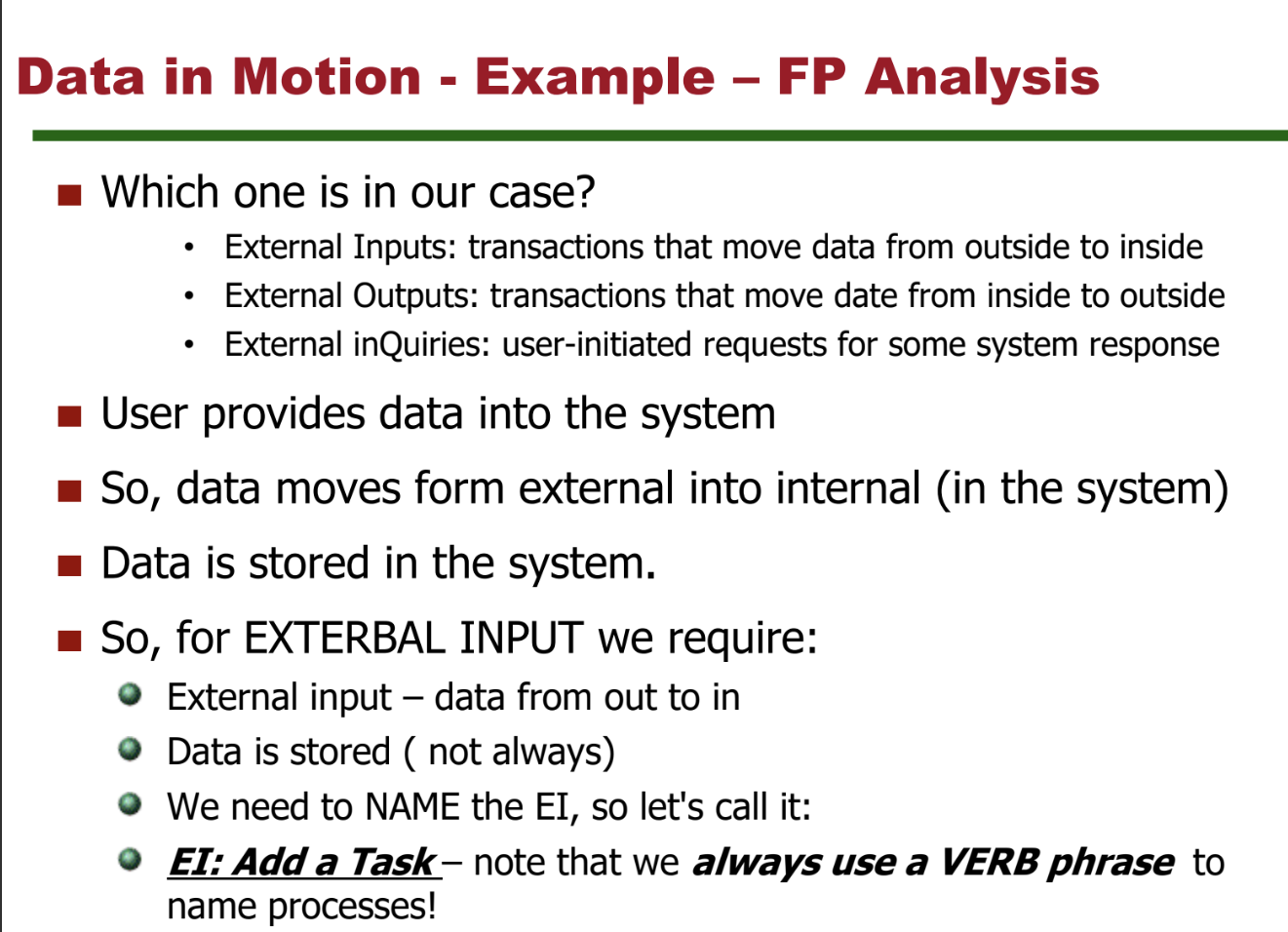
FP analysis counts five Information Domain Values (IDV):

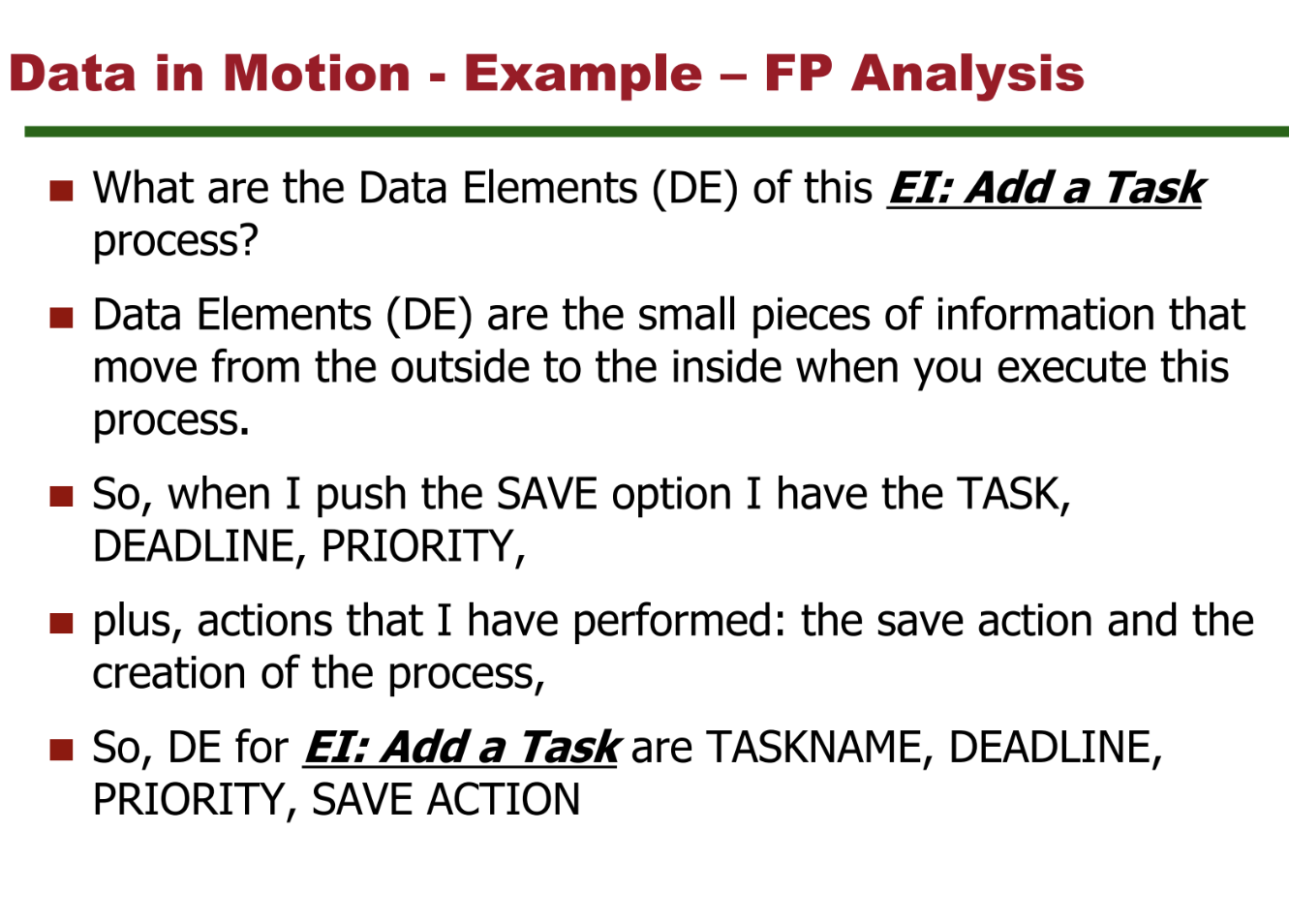
* Data in motion:
  + External Inputs: Transactions that move data from outside to inside
  + External Outputs: Transactions that move data from inside to outside
  + External Inquiries: user-initiated requests for some system response
* Data as rest:
  + Internal Logical Files: data maintained by the applications
  + External Interface Files: data maintained by another application

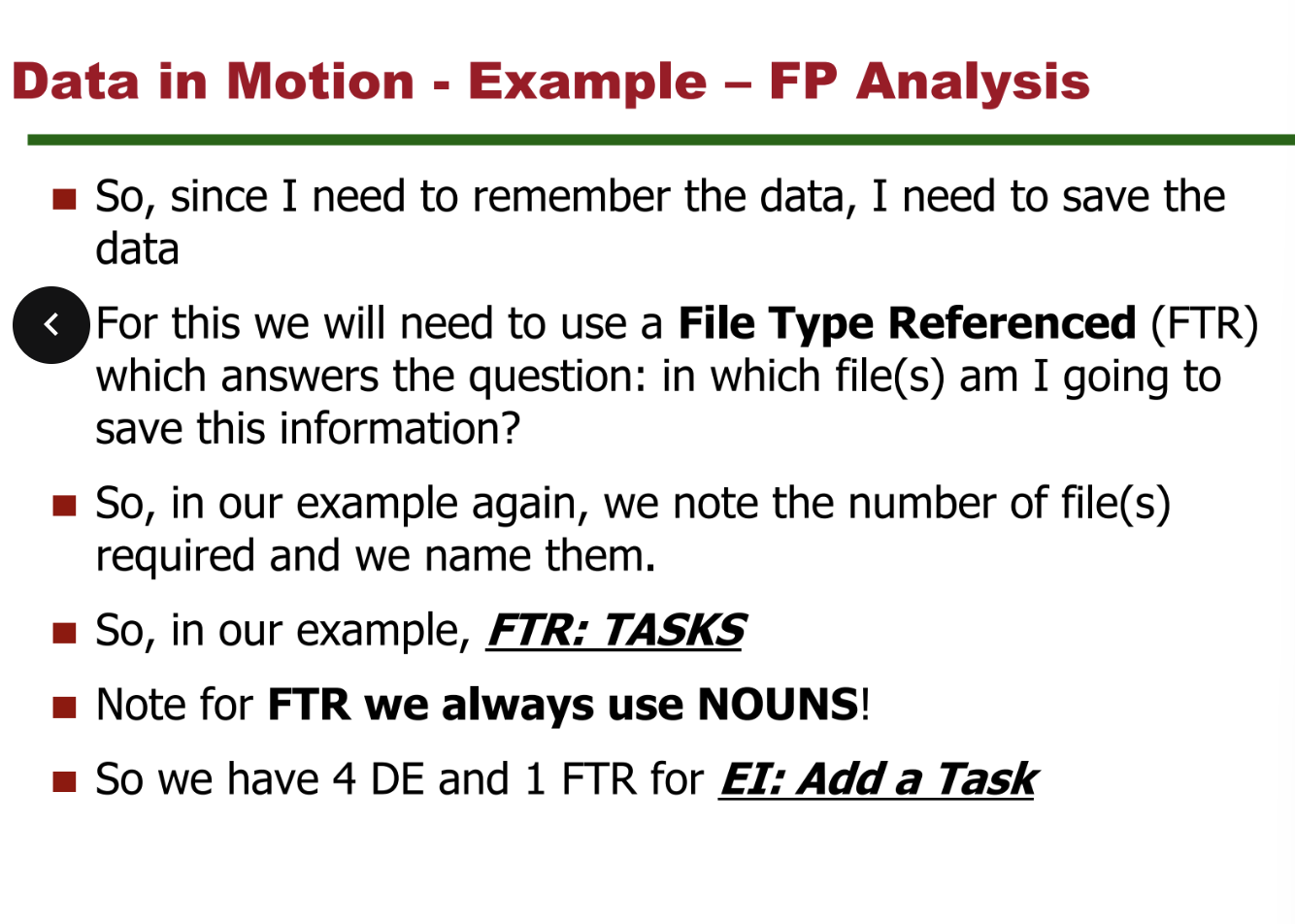
FPA and Project Size

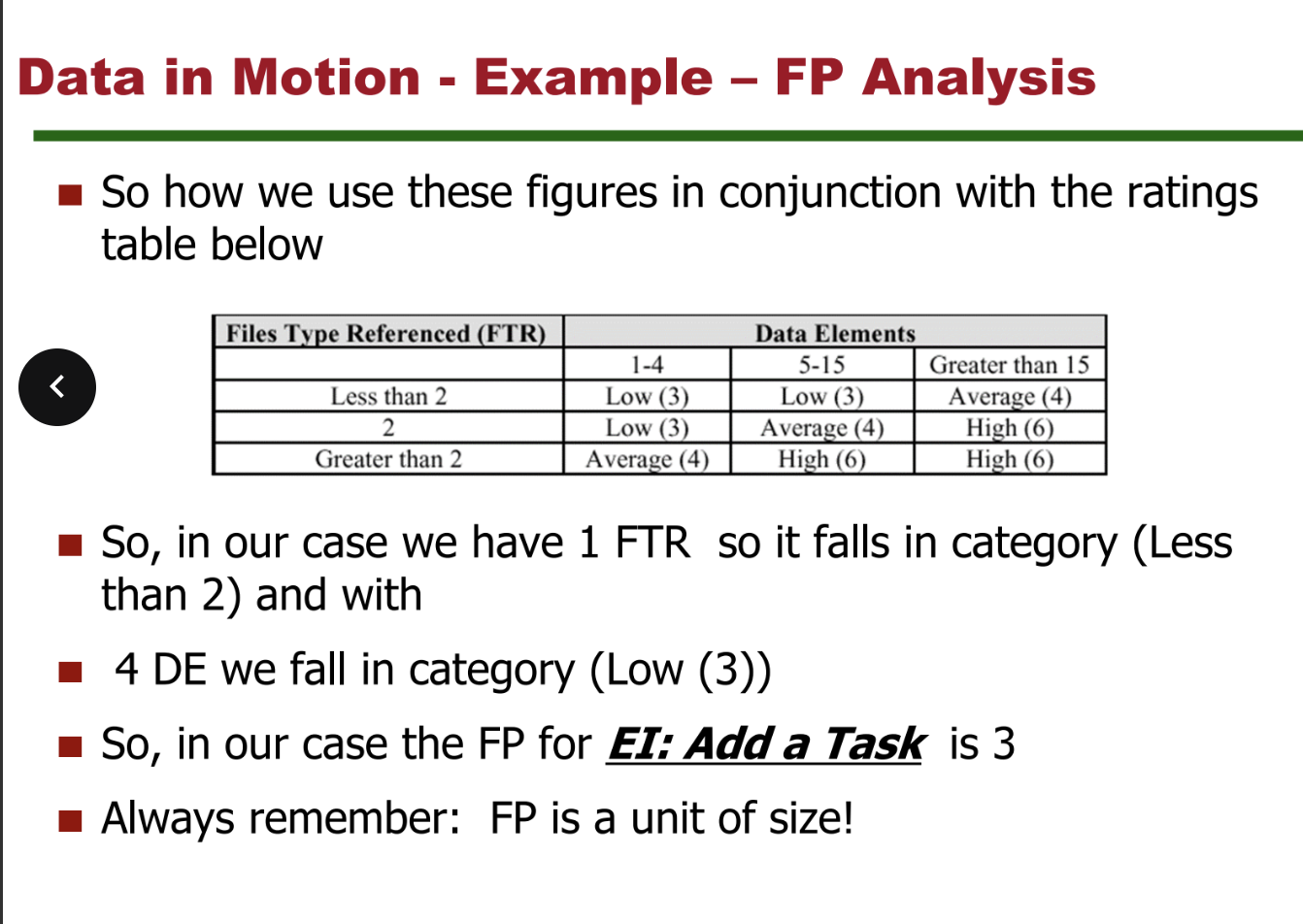
* Scope
* Extract Processes from Scope
* For each process we will find the corresponding FPA Points
* Adding all the FPA points that correspond to all processes we will be able to determine the project size in KLOC











Data elements are the small pieces of information that move from the outside to the inside when you execute the process.

External Input we need to find them

External Output has always 5 DE (data elements) Also external output most of the time has processed data

External Inquiry has always 3 DE while this mostly has unprocessed data

Lecture 4

COCOMO

Basic COCOMO model

E is the effort applied in person-months

D is the Development time in months and KLOC is the estimated number of delivered lines of code for the Project.

Parameters a,b,c and d depend on type of project.

Modes of COCOMO

Organic mode:

* Relatively small, simple software projects in which small teams with good application experience work to a set of less than rigid requirements

Semi-Detached mode:

* An intermediate (in size and complexity) software project in which teams with mixed experience levels must meet a mxi of rigid and less than rigid requirements (e.g a transaction processing system with fixed requirements for terminal hardware and data base software)

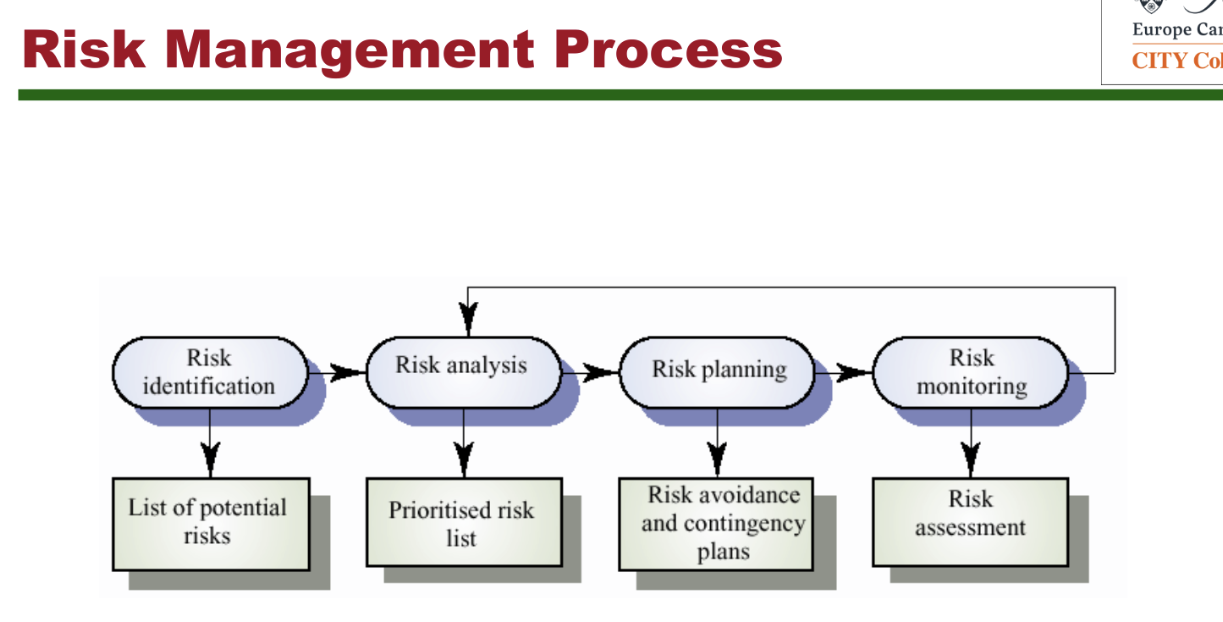
Embedded mode:

* A software project that must be developed within a set of tight hardware software and operational constraints (e.g flight control software for aircraft)

See the Project management slides in Lecture 4 it is shown how to do some exercise stuff that the mock exam did not have

Lecture 5

RISK MANAGEMENT



How to identify Risks

* Start with a typical list (checklist) of software risks
  + Structured tool to learn from past mistakes
* Review development plan
  + Critical paths
  + Critical staff members
  + Critical vendor deliveries
  + Critical milestones
* Review Requirements
* Review technical design
* Review past Projects

Keywords to look for risks:

* Underestimated the time needed to develop the Software
* The rate of defect repair is underestimated
* The size of the software is underestimated
* Changes to requirements which require major design rework are proposed
* Customers fail to understand the impact of requirements changes
* The code generated by CASE tools is inefficient
* Case tools cannot be integrated
* The organization is restructured so that different management are responsible for the project.
* Organizational financial problems force reductions in the project budget
* It is impossible to recruit staff with the skills required
* Key staff are ill and unavalabile at critical times
* Required training for staff is not available
* The database used in the system cannot process as many transactions per second as expected.
* Software components which should be reused contain defects which limit their functionality.

For each risk:

We need to determine:

* What is the likelihood of occurrence
* What is the impact if it occurs
* What will we lose if the risk occurs
* Where should we put our limited resources (prioritize for all risks)

Lecture 6

TESTING

Types of testing

* Black box testing (functional)
  + Test the functional requirements
  + Specification => test cases
* White box testing (structural)
  + Tests the internal logic and flow of a program
  + Program => test cases

White-Box methodologies

* Complete path testing is usually impossible (loops)
* Statement coverage
* Decision coverage
* Condition coverage
* Decision/condition coverage

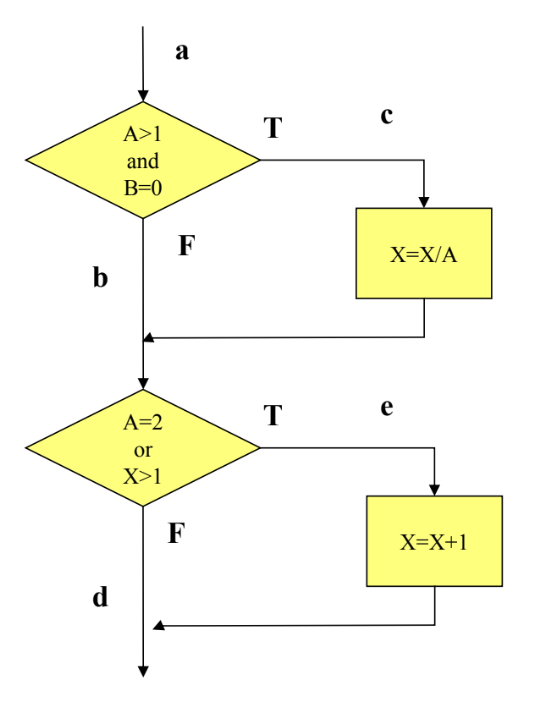
Statement coverage

Coverage creation:

* Every statement in the program is executed at least once

One test is enough

A=2, B=0, X=3



Decision coverage

Coverage criterion:

* Each decision (diamond, compound condition) should have a true and false outcome at least once.

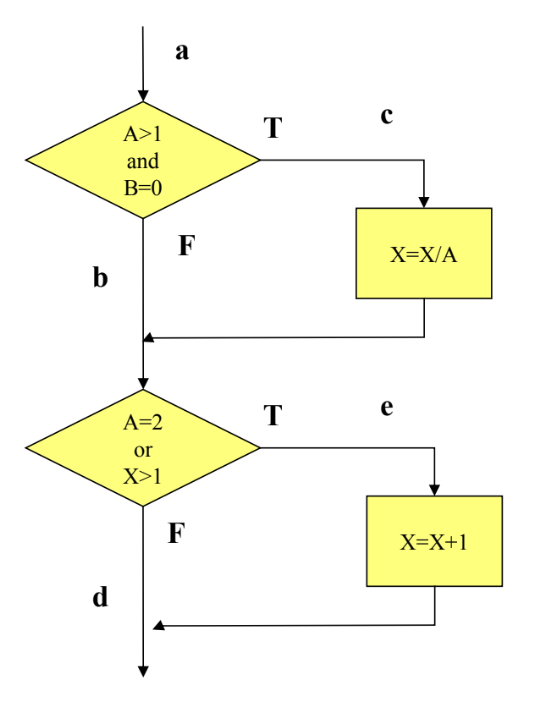
Covers statement coverage

Two test cases

* Ace
* Abd

A=2, B=0, X=2

A=0, B=0, X=1



Condition coverage

Coverage criterion

* Each condition in a decision takes all possible outcomes at least one

Four Conditions

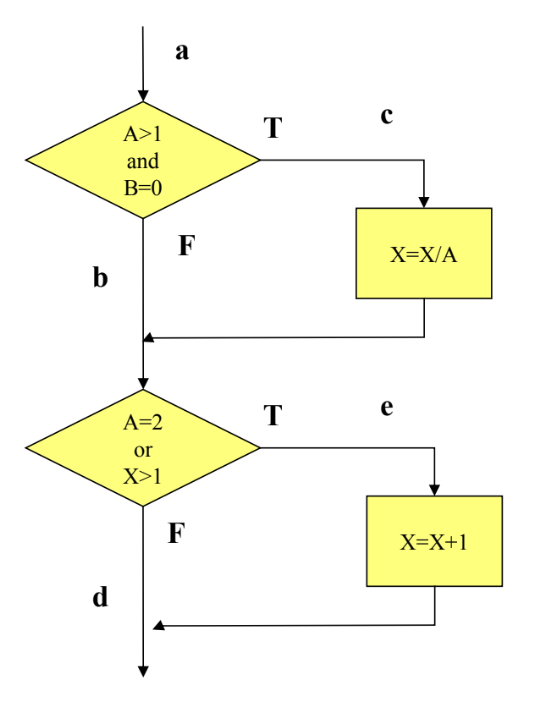
* A>1, A<=1
* B=0, B<>0
* A=2, A<>2
* X>1, X<=1

Two test cases

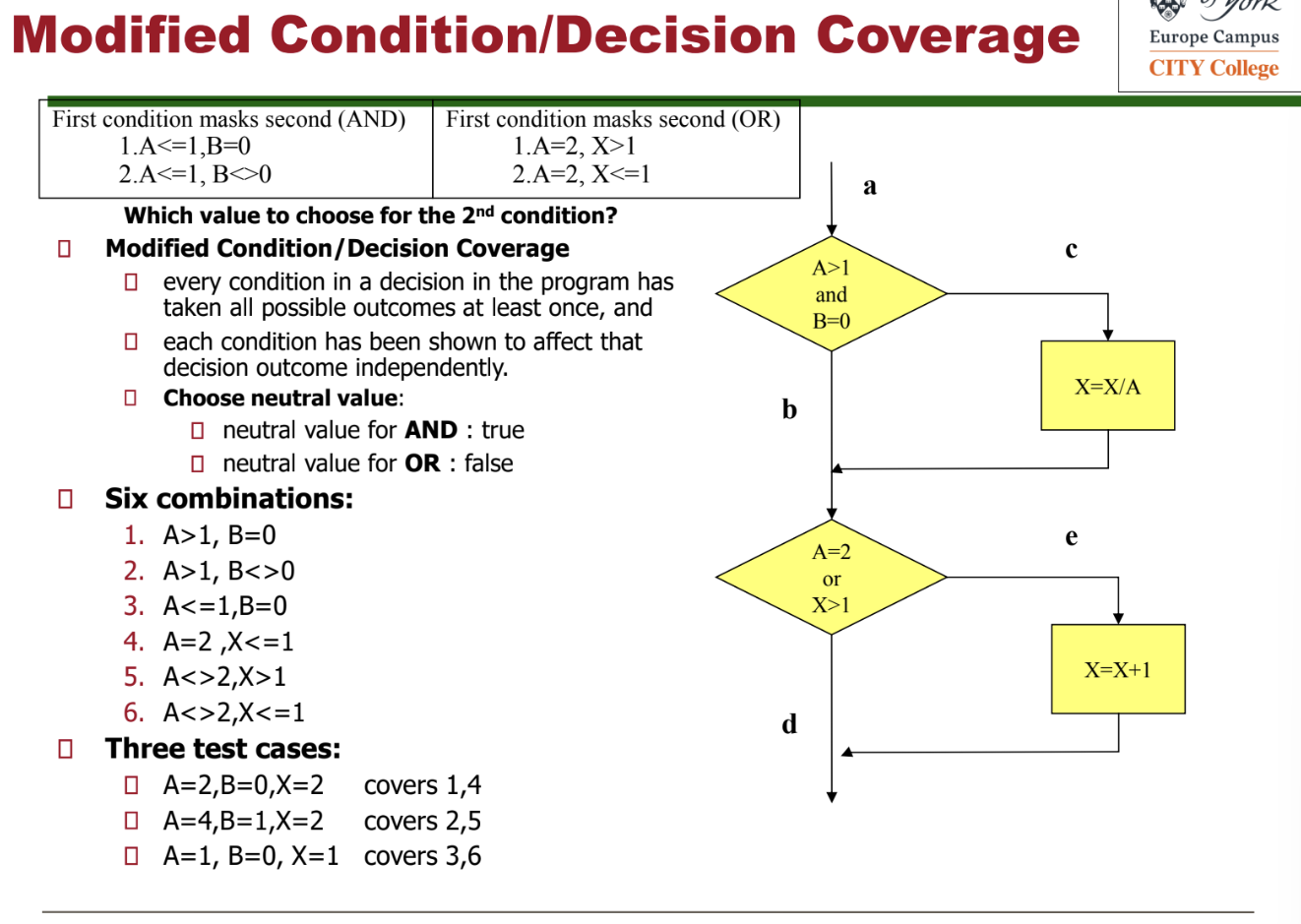
* A=2, B=0, X=4 ace
* A=1, B=1, X=1 abd

Does not always cover decision coverage

* A=1, B=0, X=3 abe
* A=2 , B=1, X=1 abe



Decision/Condition Coverage



Basis path testing

Goal:

* Execute every statement in the program at least once
* Execute every edge in the program flow at least once

Cyclomatic Complexity

Cyclomatic complexity:

* Software metric which measures the logical complexity of a program
* Defines upper bound for the number of test cases that ensure execution of every statement and every edge at least once

3 ways of calculating cyclomatic complexity V(G)

* V(G) = number of regions = 5
* V(G)= number of edges – number of nodes +2 = 11-8+2 = 5
* V(G) = number of predicate nodes + 1 = 4+1 = 5

Lecture 7

SOFTWARE QUALITY ASSURANCE

Goals:

* Improve quality by monitoring software product and development process
* Ensure compliance with established standards and procedures
* Ensure management is informed about inadequacies in product, process or standards so that these can be fixed

SQA tasks

* Not develop
* But
  + Review
  + Audit
  + Inform

Requirements for effective SQA

* Management commitment should enforce SQA suggestions
* SQA should be independent from development teams and management
* SQA should be staffed with technically competent, cooperative people
* Development teams should cooperate with SQA team
  + Same goal: development of high quality software

SQA Group

General Responsibilities

* Quality assurance planning, record keeping, reporting

Activities

* Technical reviews, audits, testing (verification & validation), measuring.

A SQA Group

* Prepares a SQA plan for each new project. The plan is part of the project plan and identifies:
  + Evaluations, audits and reviews standards error reporting procedures and documents
* Participates in the definition of the projects software process. The process is reviewed against standards, policies and other parts of the software project plan.
* Continuously reviews Software Engineering activities to track deviations, identify problems and suggest corrective actions.
* Collects and analyses metrics and reports about changes at the level of quality (maturity) of the organization.

CMM

CMM: Capability Maturity Model

* A model of process maturity for software development -an evolutionary model of the progress of a company’s abilities to develop software.
* Control, Measure and Improve
* Achievement of levels via Key Process Areas consisting of Key Practices
* 2 years from initial to defined

CMM has 5 levels

Initial Level => Repeatable Level => Defined Level => Managed Level => Optimizing Level

Initial Level

Characteristics:

* Processes are unpredictable, poorly controlled and reactive
* Success often depends on individual effort rather than established processes.
* High risk of failure, cost overruns and schedule delays

Repeatable

Characteristics:

* Basic project management processes are established to track cost, schedule and functionality.
* Success can be repeated for similar projects with defined procedures
* Focus on project-specific process improvements

Defined

Characteristics:

* Processes are documented, standardized and integrated into a standard software process fore the organization
* Focus on defining roles, responsibilities and tools for process execution
* Organization-wide process consistency

Managed

Characteristics:

* Quantitative metrics are used to monitor process performance and product quality
* Predictability of process outcomes is improved due to data-driven decision-making
* Focus on reducing variability in processes

Optimizing

Characteristics:

* Continuous process improvement is emphasized through incremental and innovative changes
* Organization is agile, responsive and focused on optimizing performance
* Lessons learned are used to refine processes