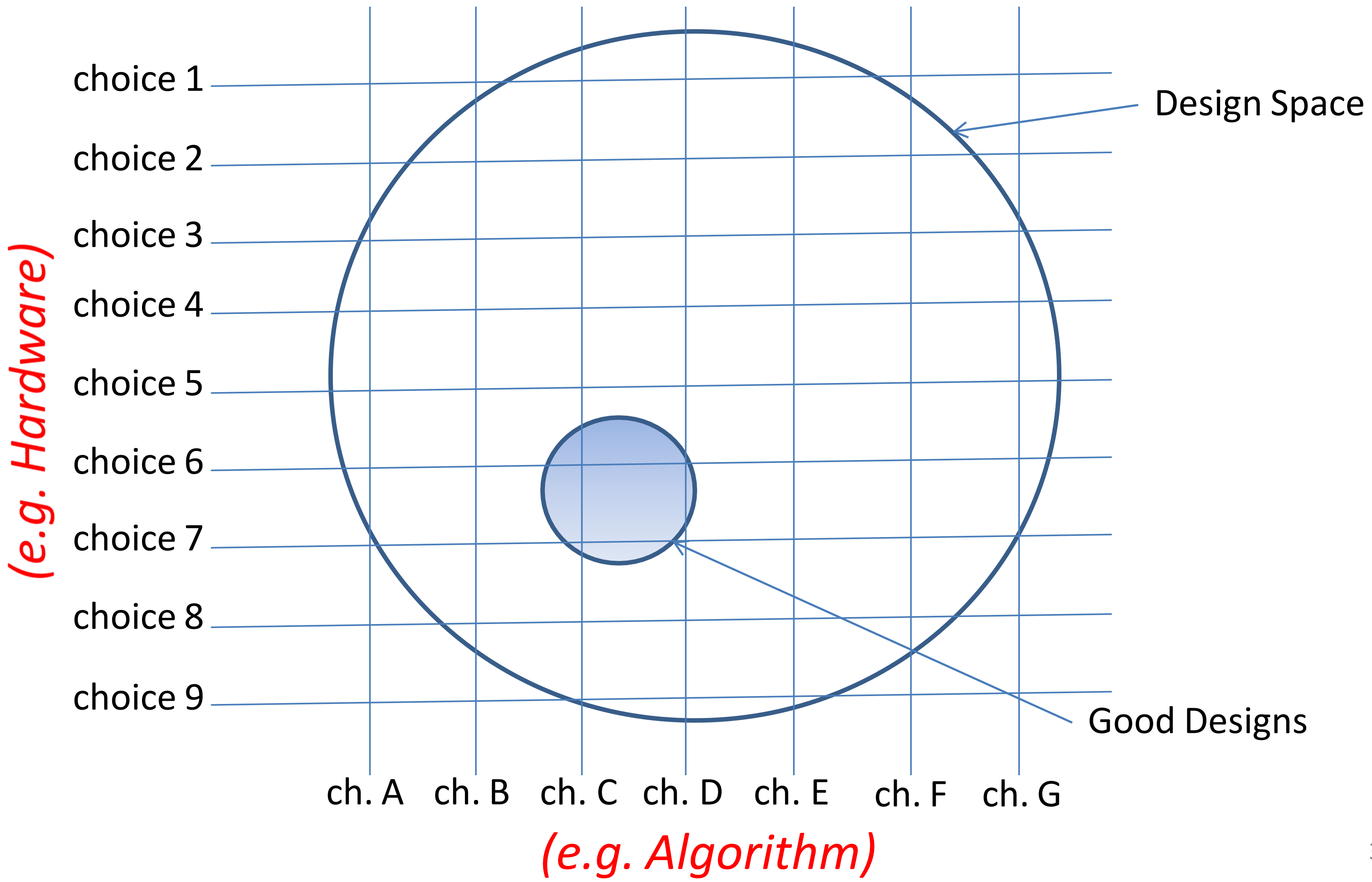
50.003: Elements of Software Construction

Week 2

Introduction to Software Design

What is Design?

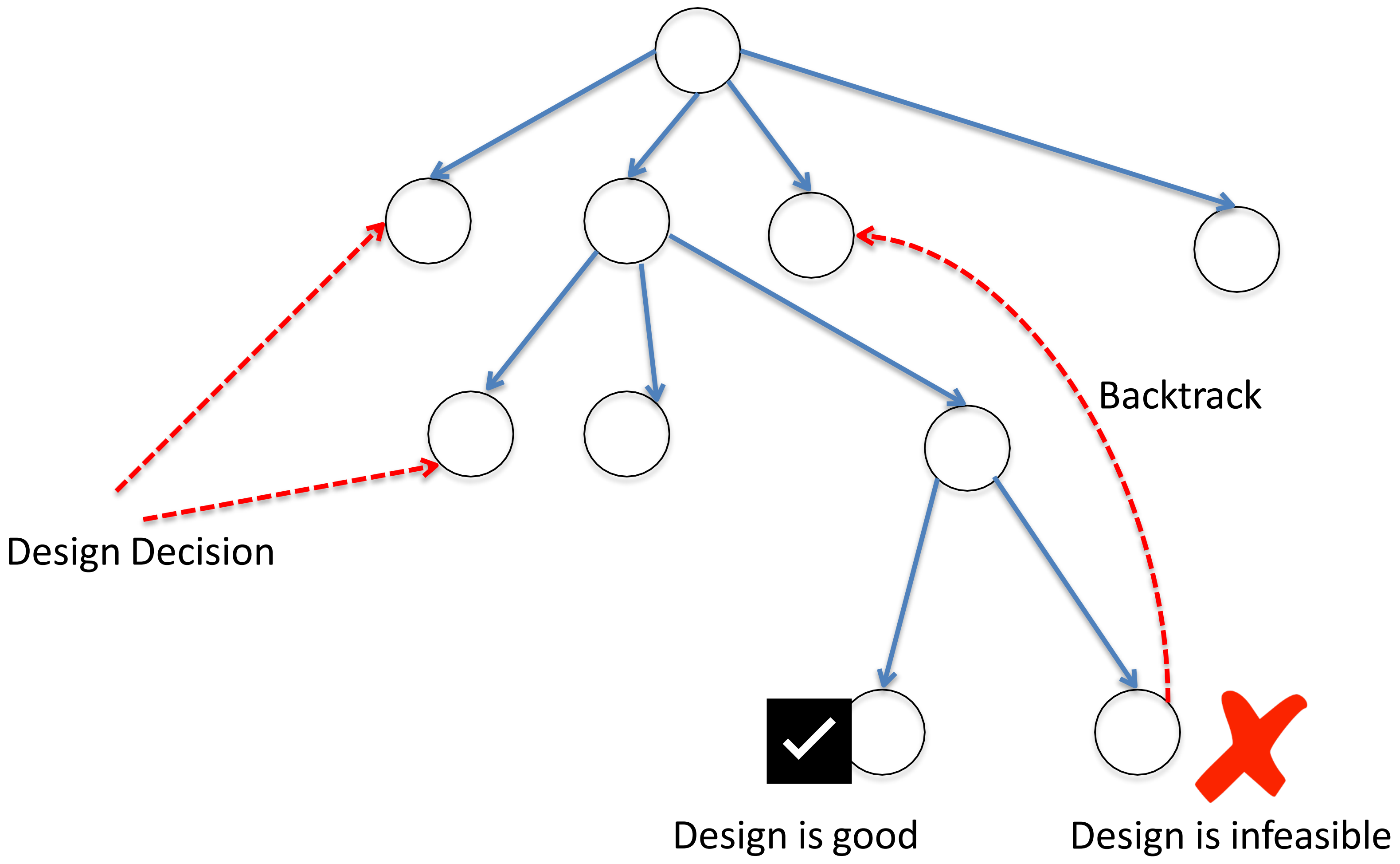


The Rational Model

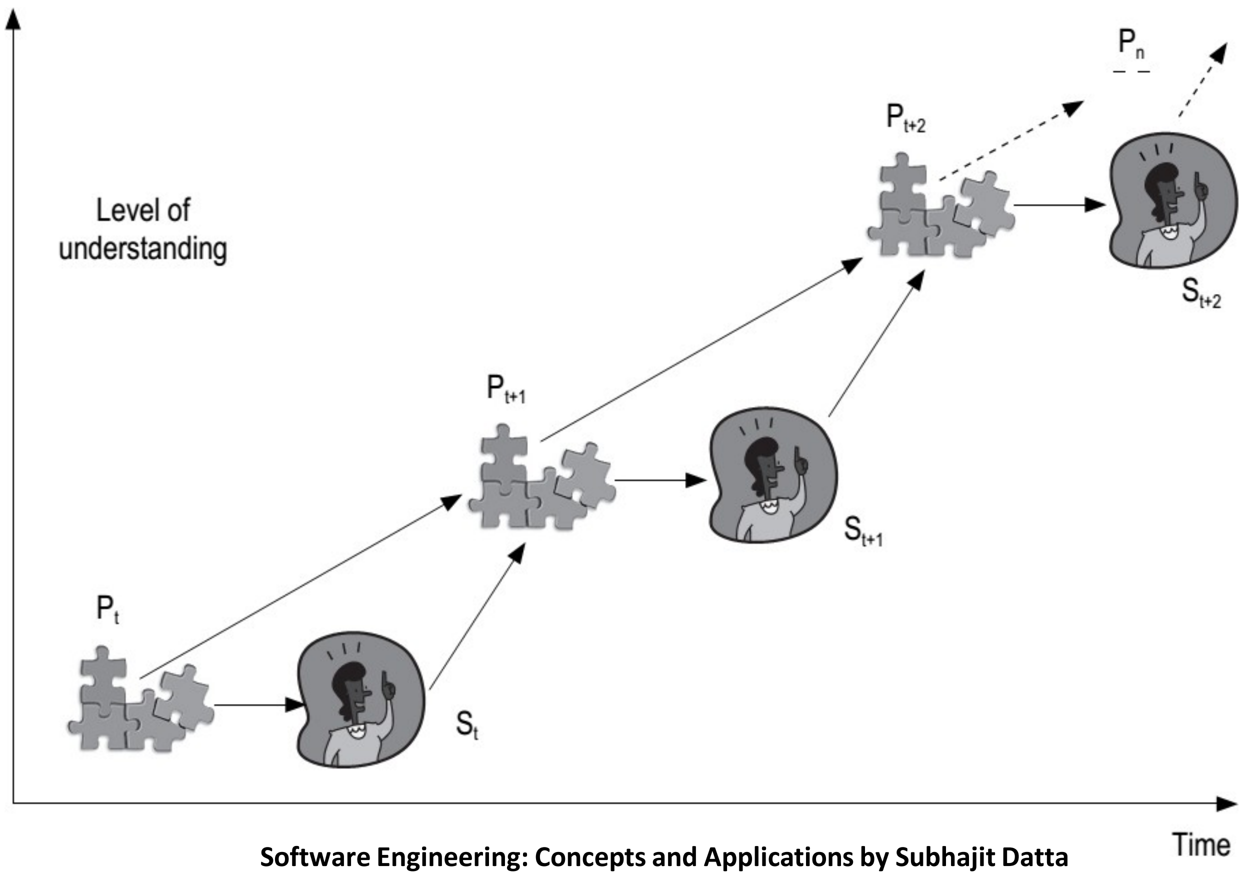
Input: Goal, Desiredata, Utility function, Constraints, Design tree of decisions

```
Until ("good enough") or (time runs out) {
        Do another design (to improve utility function)
                Until design is complete
                        Backtrack up design tree
                        Explore a path not searched before
                                While design remains feasible,
                                make another design decision
                        EndWhile
                EndUntil
        EndDo
        Take best design
EndUntil
```

The Rational Model



Criticizing the Rational Model



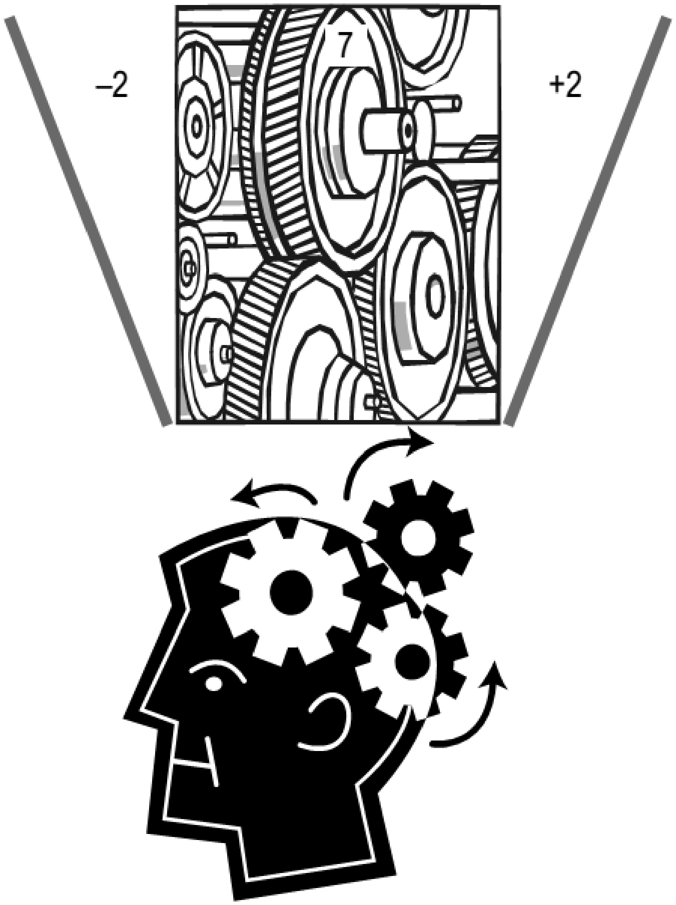
Software Engineering: Concepts and Applications by Subhajit Datta (Oxford University Press, 2010)

Software Design is Hard (cont'd)

- Even if we know all the design choices, the combinations of design choices are exponential and therefore it may not be easy to find a good design.

ADDRESSING COMPLEXITY IN DESIGN

What's the implication in software engineering? We concentrate on the most relevant details at a point and ignore the details. Refine design strategies step by step.



Miller's Law

At any time a human being can concentrate on 7 +- 2 chunks (quota of information)

Strategies for addressing complexity

- Decomposition
- Abstraction
- Hierarchy

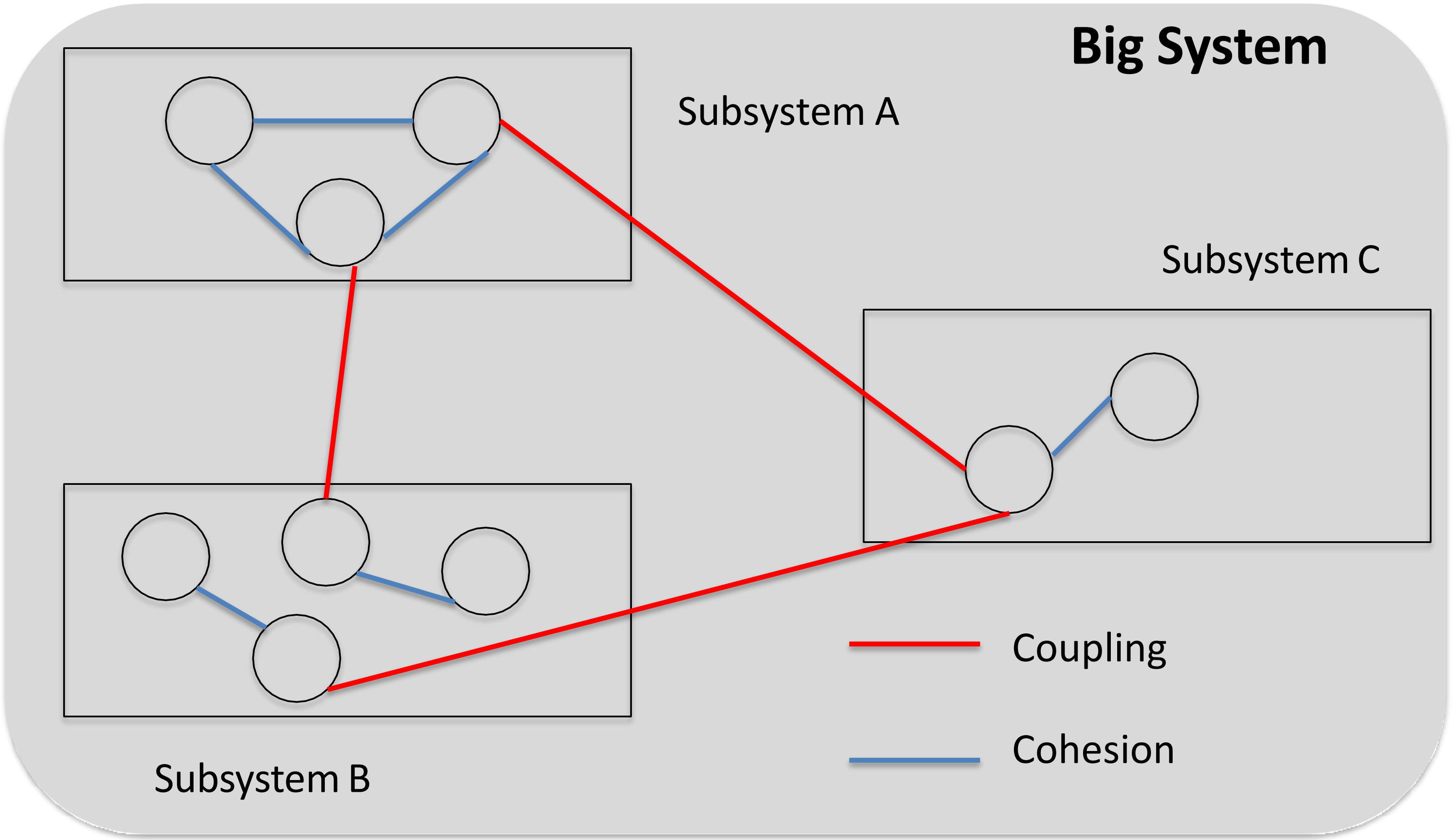
Decomposition

- *Divide et impera* (divide and rule) is a technique of mastering complexity in vogue since ancient times [Dijkstra 1979].
- Key idea:
  - Break down a system into smaller and more manageable parts, so that the channel capacity of our comprehension is not breached.
- Coupling and Cohesion

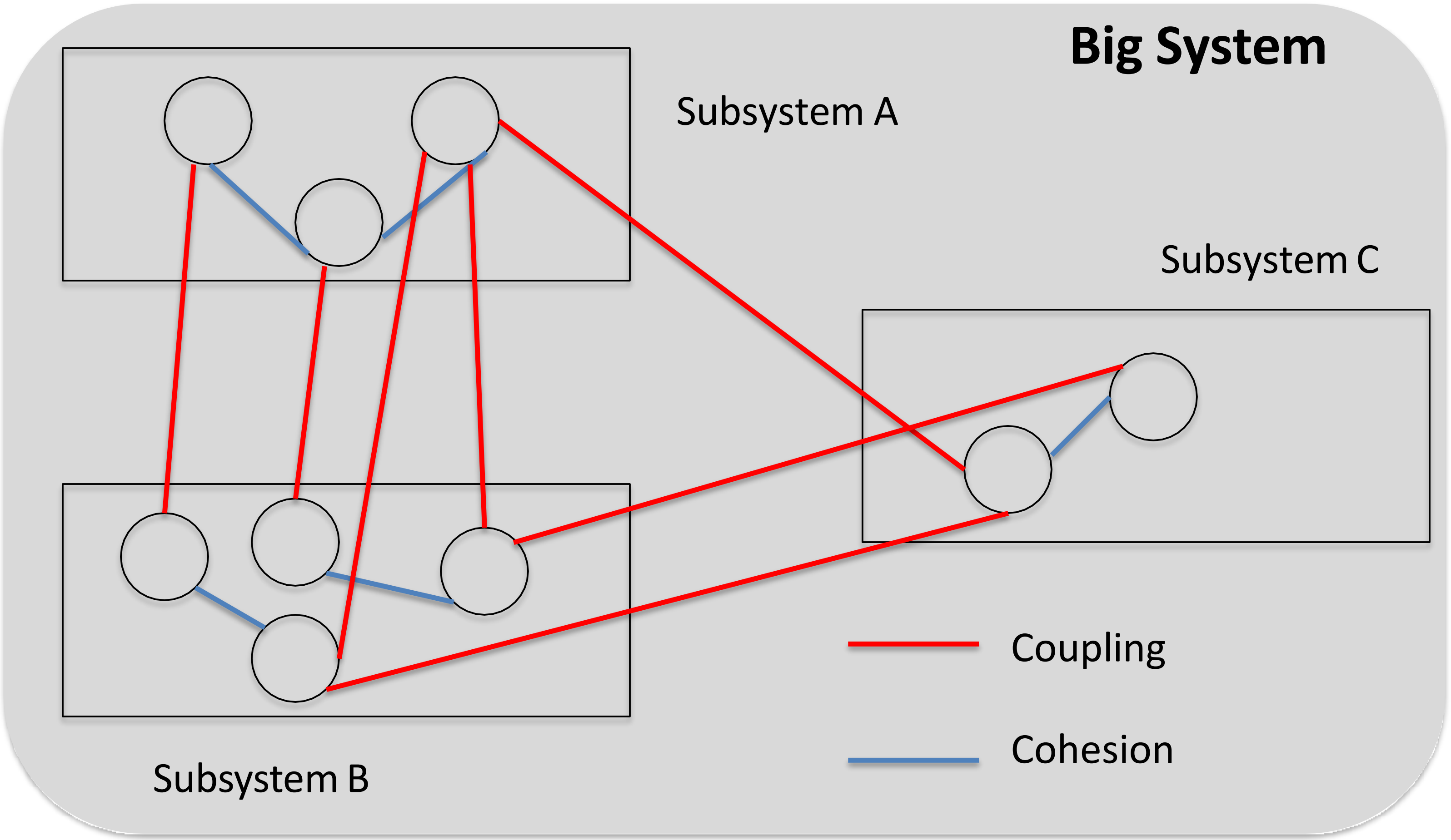
Human mind can handle complexity only up to a certain threshold. Hence, decomposition helps in the design process. For your app searching for restaurants, let us say someone focus on the search algorithm whereas someone focus on the GUI development.

Aim: High Cohesion, Low Coupling

Decomposition

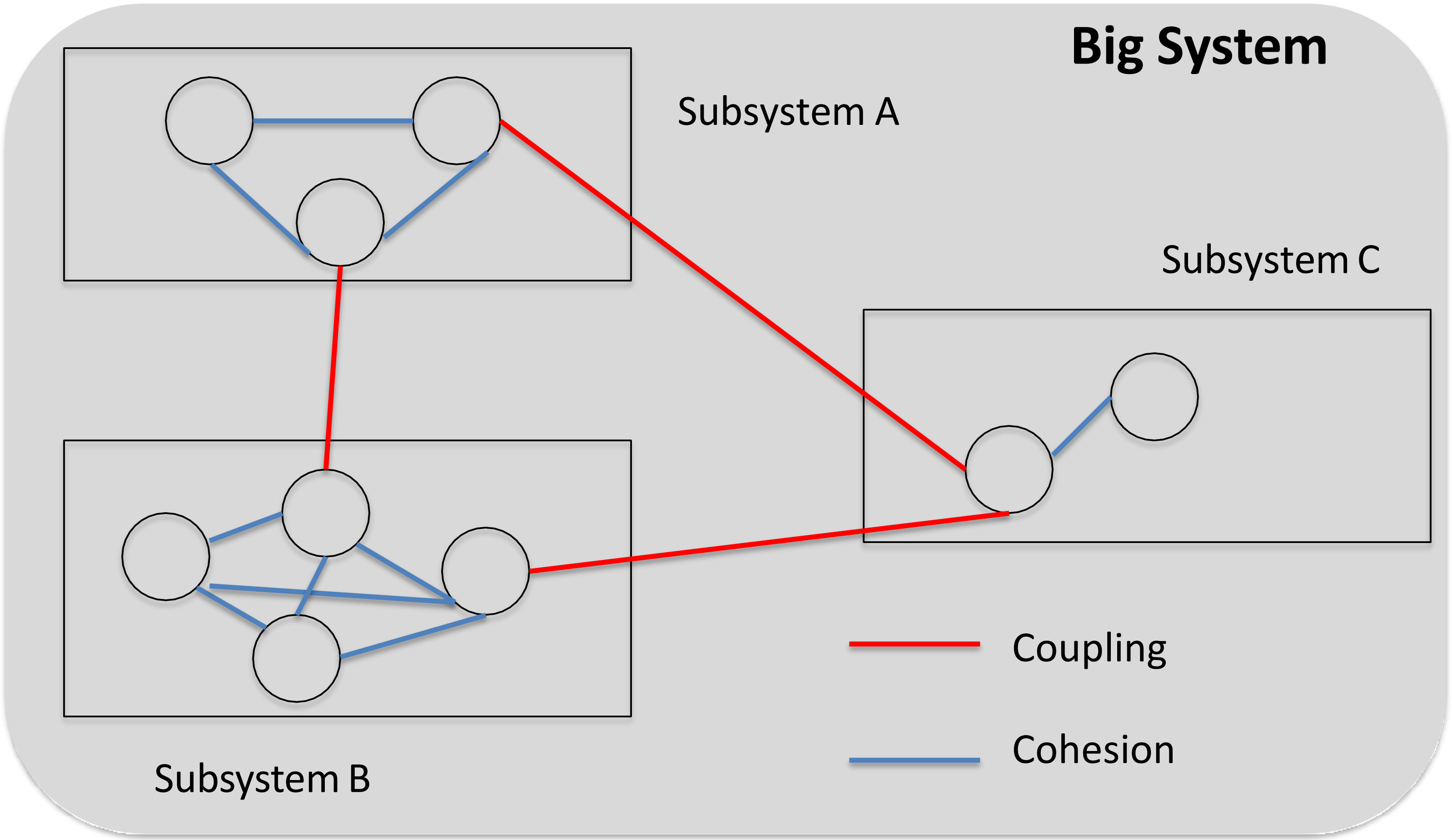


Decomposition



Is it a Good Design?

Decomposition



Is it a Good Design?

Cohort Exercise 1.1

- Assume you are designing a friend-making application which displays potential friends via a configurable distance parameter. This means the user can configure the distance/location parameter, say 20 Km/Singapore, the app will find possible friends within this distance and display it. The user can like or dislike the friends, based on which the app will compute what type of friends to show the user in the future. Decompose the application into (at least) four subsystems. Label the Coupling and Cohesion edges to show interaction.

Abstraction

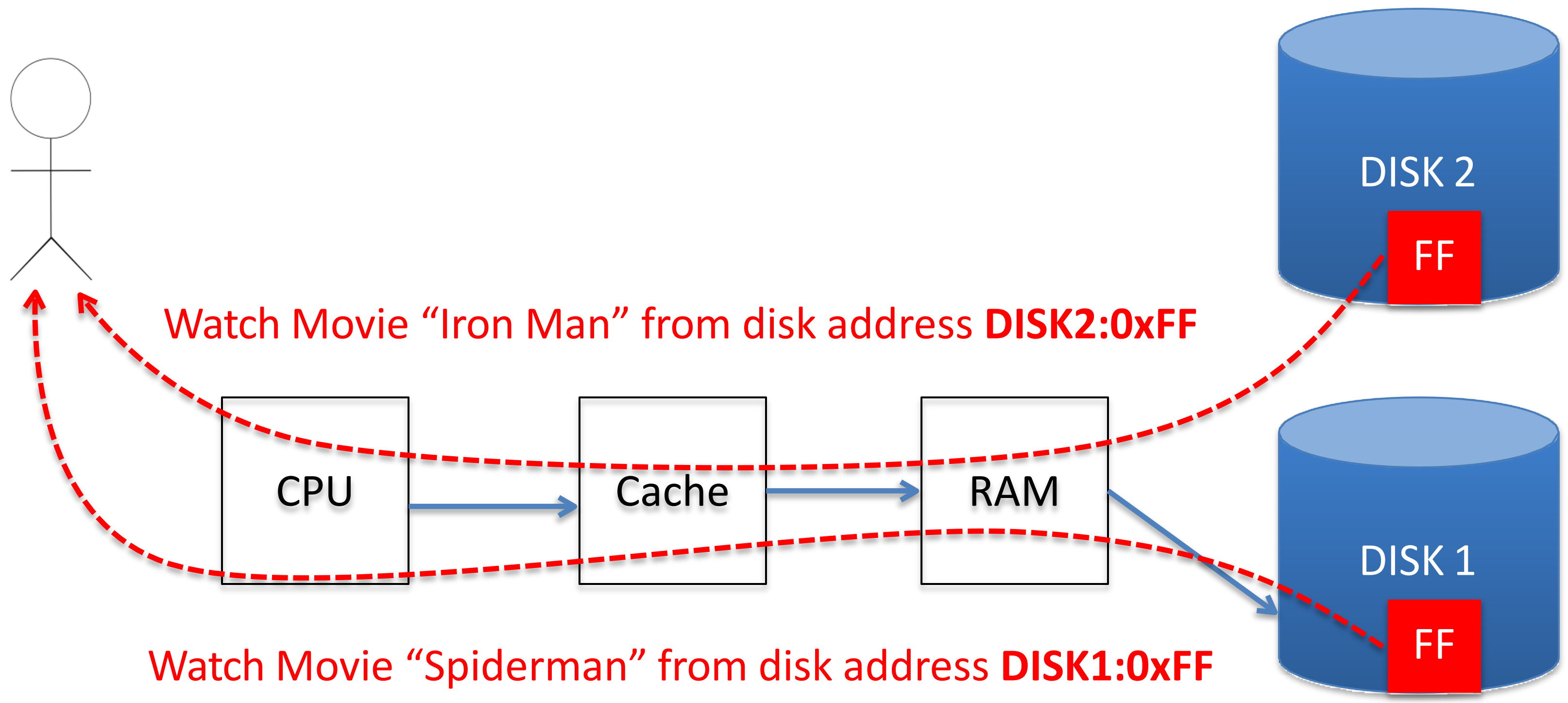
The power of abstraction

Abstraction is at the center of much work in Computer Science. It encompasses finding the right interface for a system as well as finding an effective design for a system implementation.

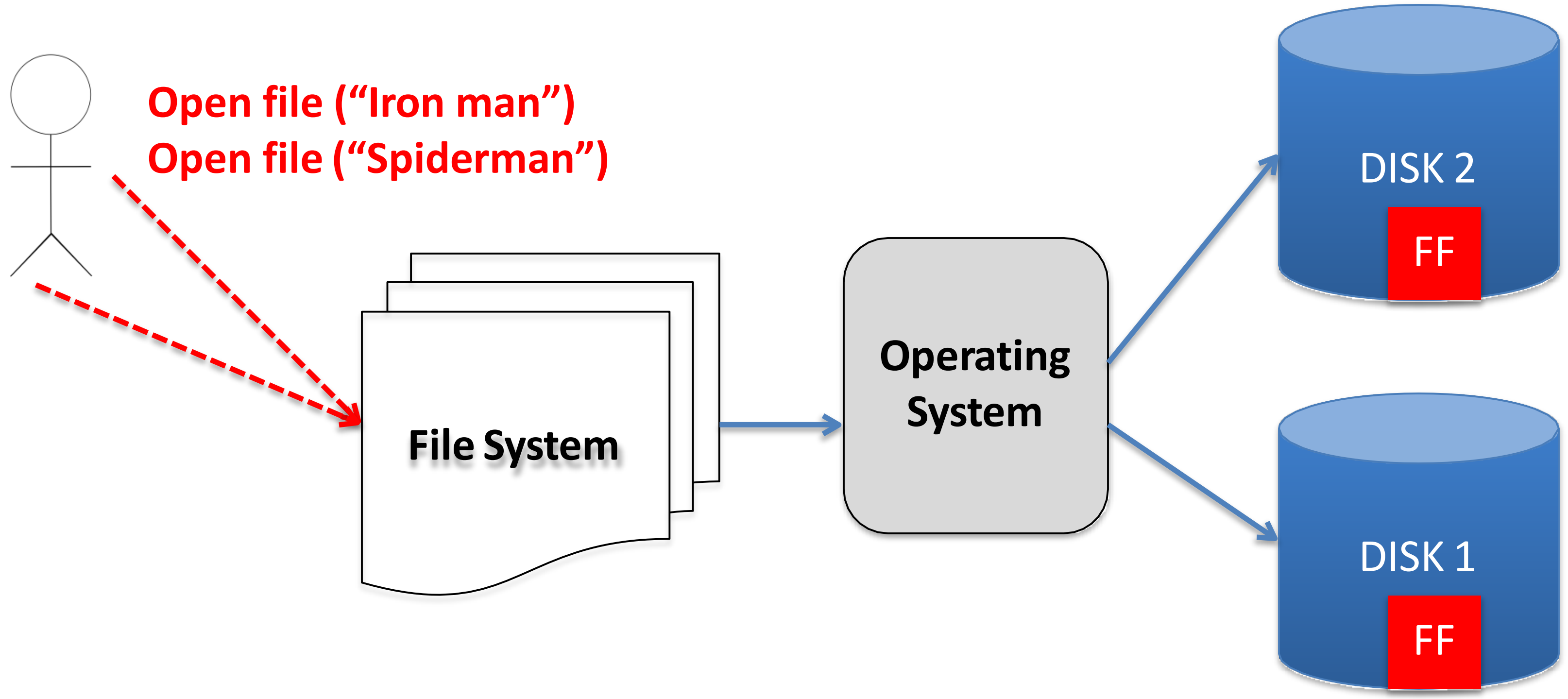
Abstraction

- One of the most frequently used words in the context of software design
- Key idea:
  - Unable to master the entirety of a complex object, we choose to ignore its inessential details, dealing instead with the generalized, idealized model of the object [Booch et al. 2007].

Abstraction

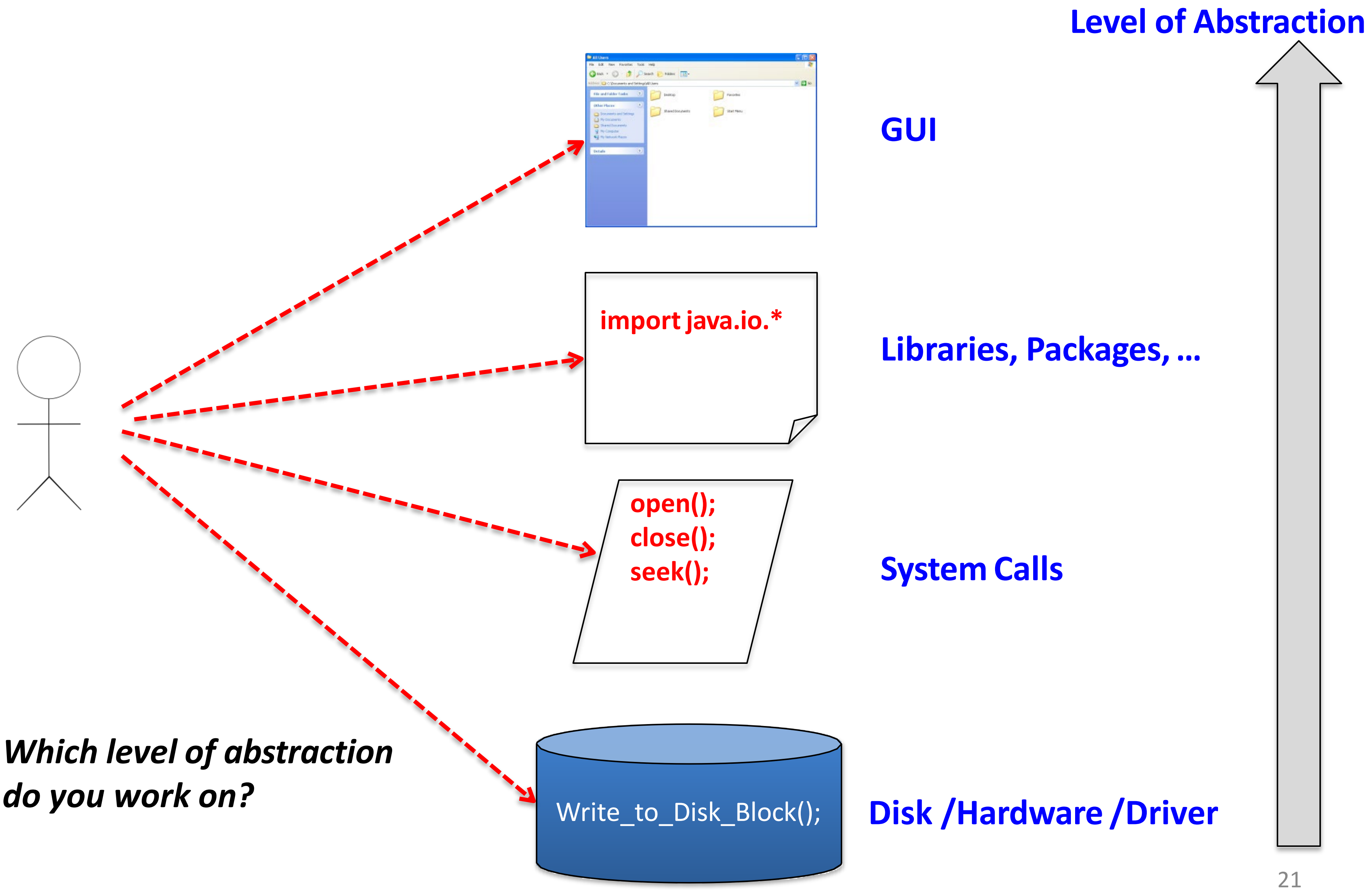

DISK 2
FF
Watch Movie “Iron Man” from disk address **DISK2:0xFF**
CPU
Cache
RAM
DISK 1
FF
Watch Movie “Spiderman” from disk address **DISK1:0xFF**

Abstraction



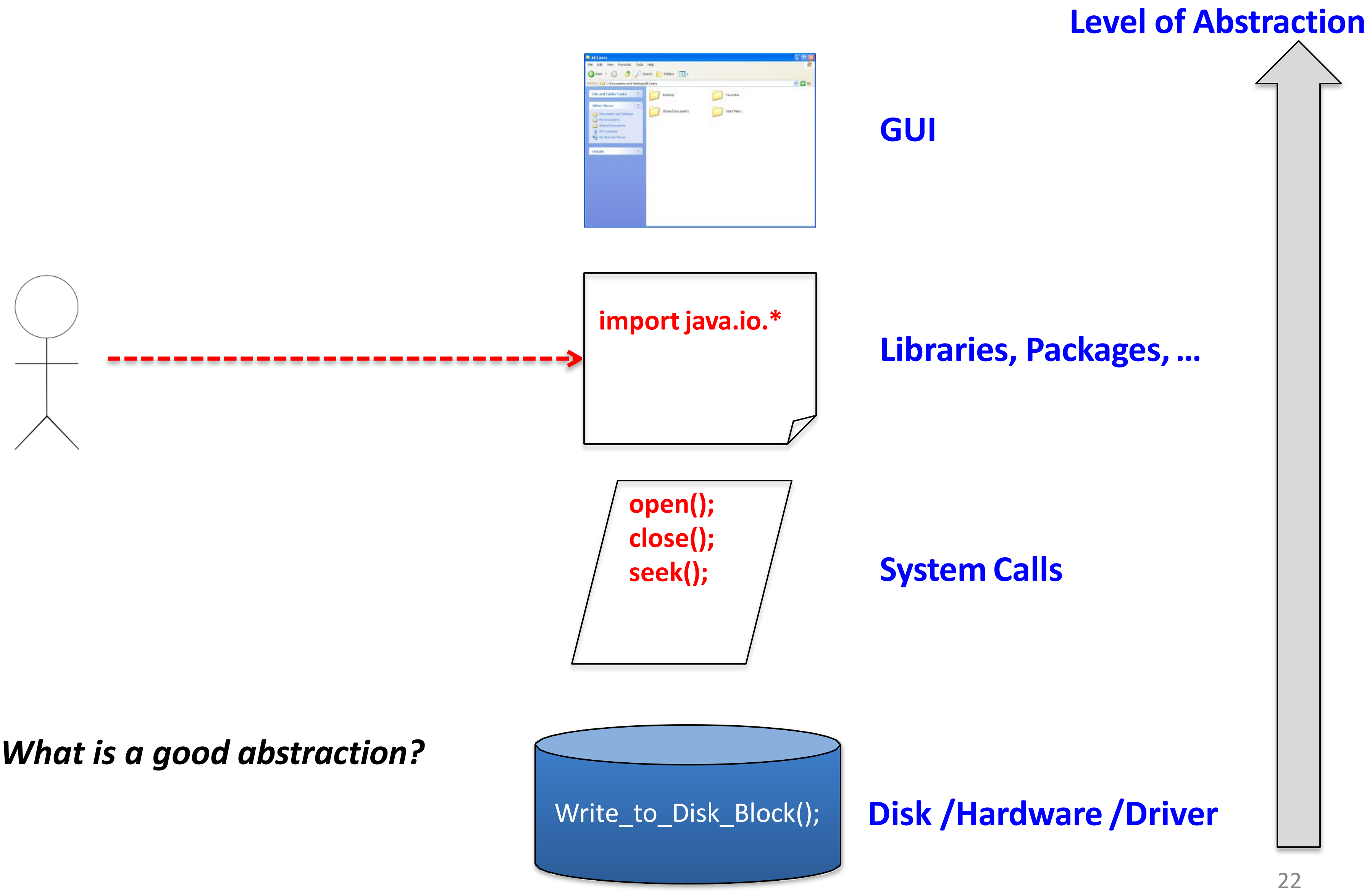
File system is an abstraction we use in daily life. All details on where are how data and retrieved are hidden.
The "open file" call abstracts all details away.

Abstraction



Which level of abstraction do you work on?

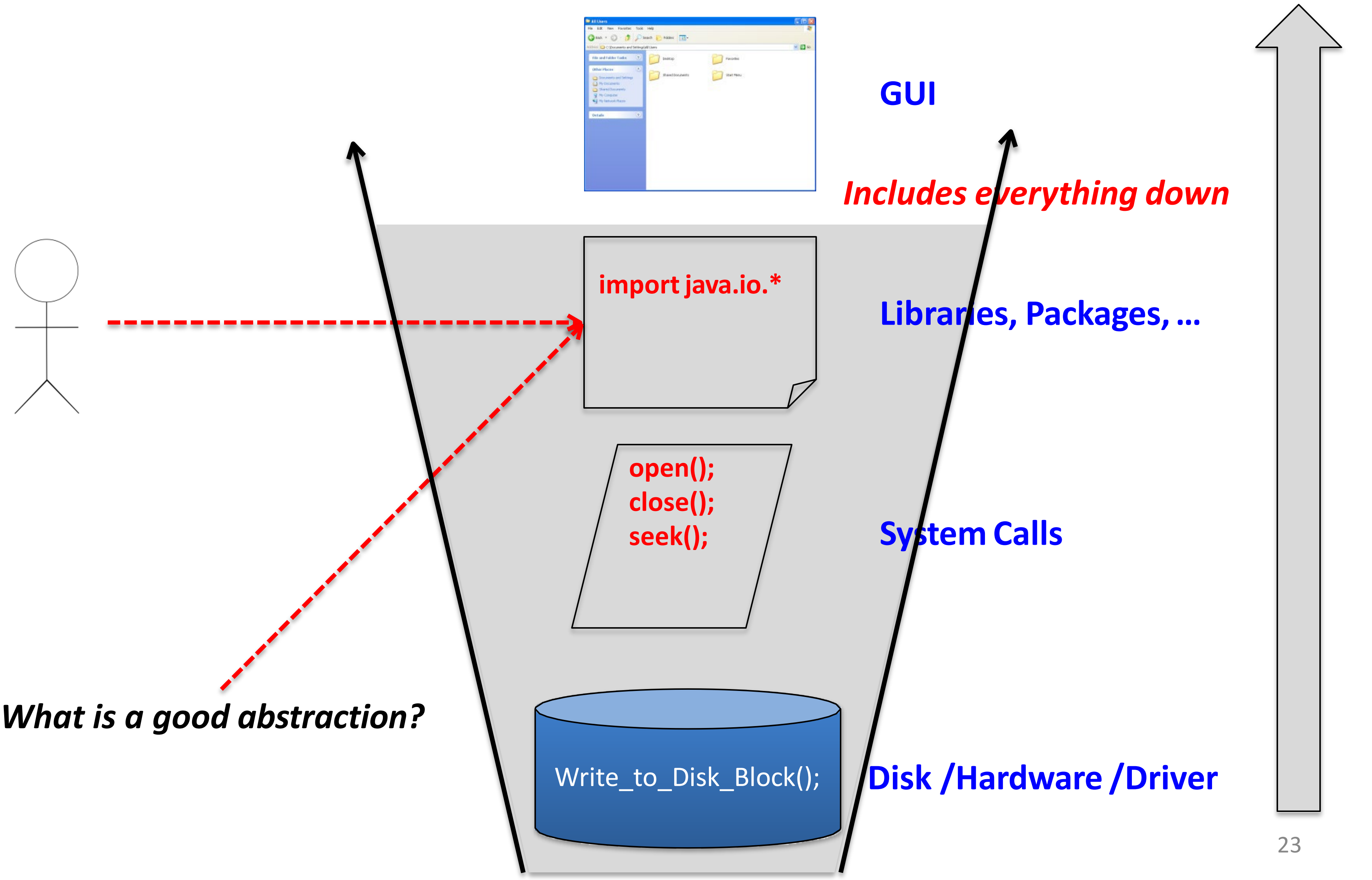
Abstraction



What is a good abstraction?

system calls is a complete abstraction of the disk when:
if you can access any file thru disk, if you can also access through system call then its true.
Any file accessed via system calls can also be access via libraries, then Libraries is a complete abstraction of system calls
(simulate all possible physical behaviour)

Abstraction



What is a good abstraction?

Abstraction

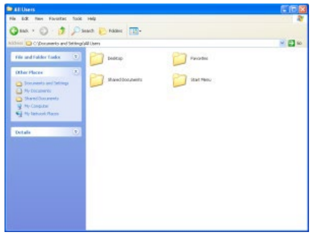
- High-level programming language is another abstraction we use daily
- It abstracts away the low-level implementation details
  - Which processor?
  - What type of Memory?
  - What type of system (PC, supercomputer, embedded systems)?

A Language is **turing complete** if it can be used to simulate any **turing machine**.

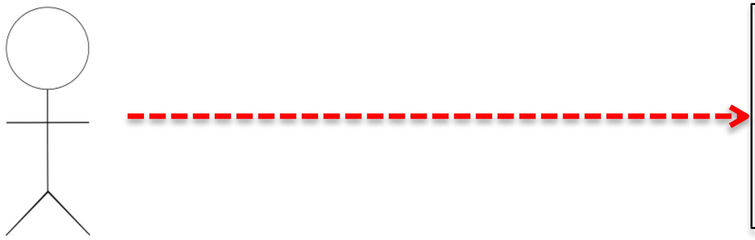
Abstraction

language is good if it can write any program to compute anything that can be done using assembly/ISA => complete abstraction

Level of Abstraction



DSL (domain specific)



```
import java.io.*
```

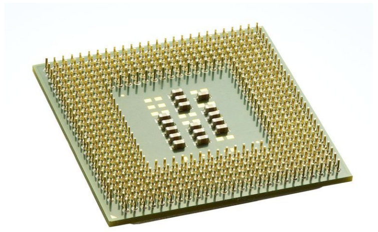
JAVA

```
mov  x, r2
add r1, r2
mov r1, y
```

Assembly code

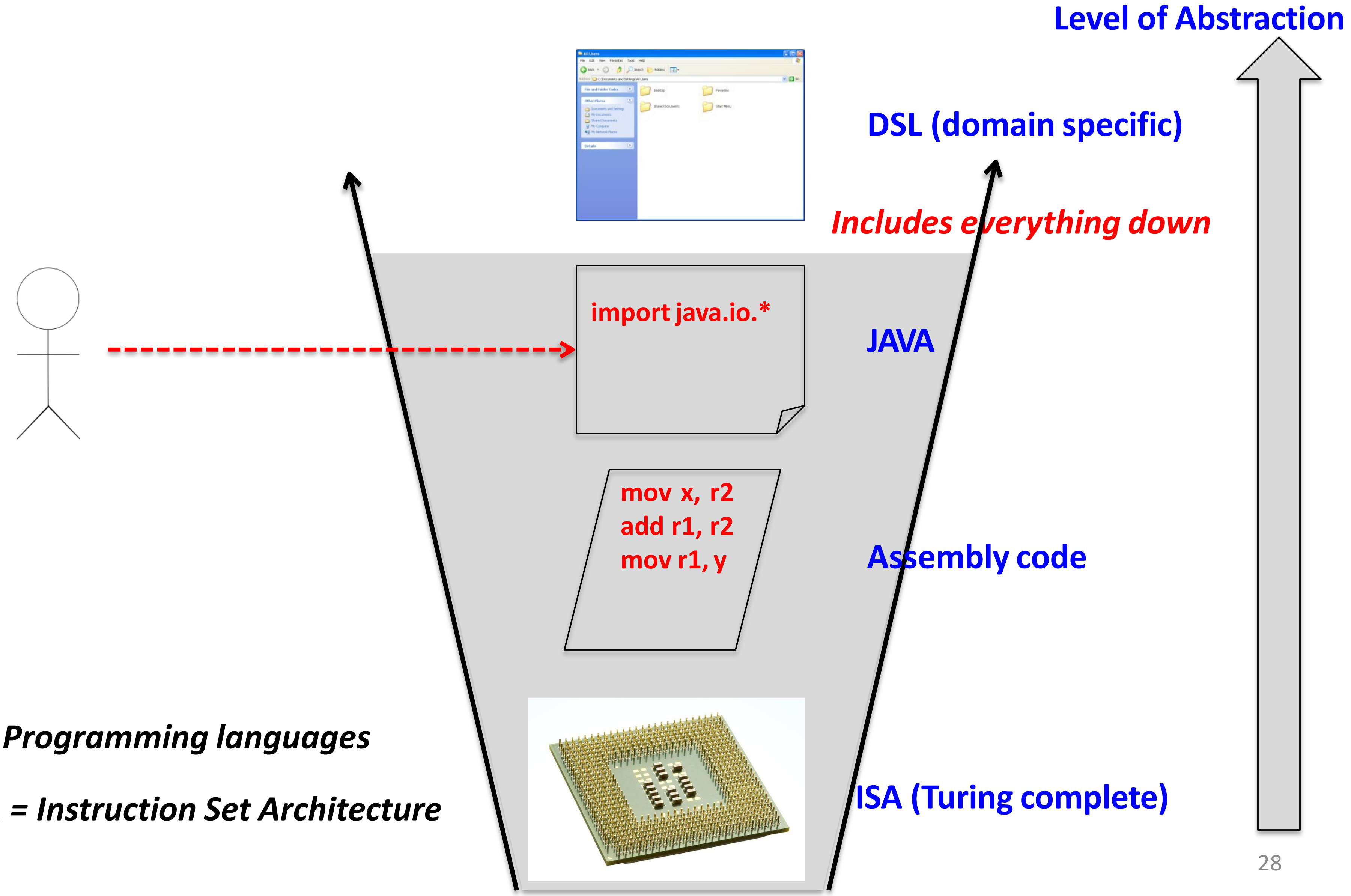
Programming languages

ISA = Instruction Set Architecture



ISA (Turing complete)

Abstraction



```
import java.io.*
```

```
mov  x,  r2
add r1, r2
mov r1, y
```

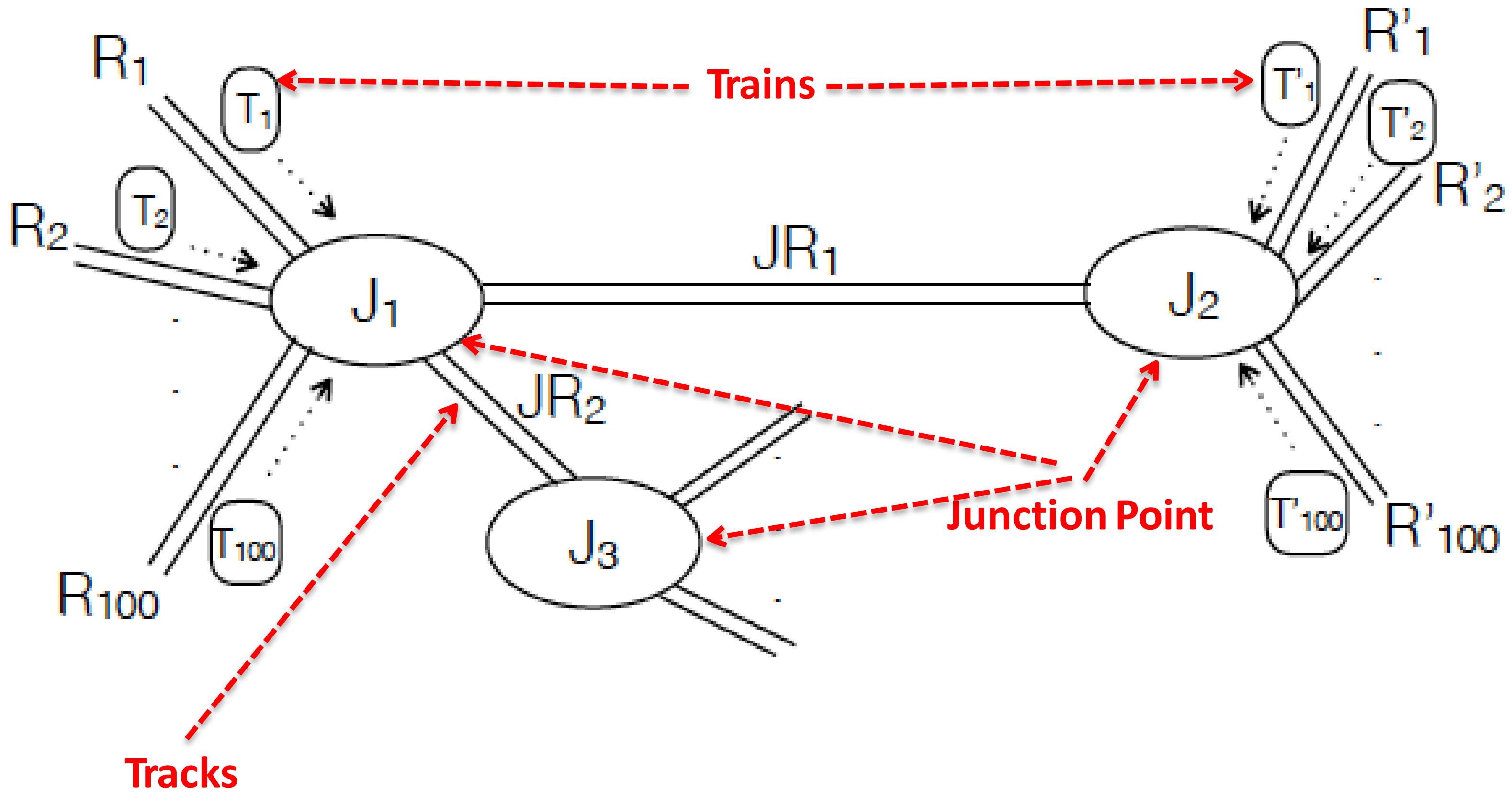
Programming languages

ISA = Instruction Set Architecture

Cohort Exercise 1.2

- A railway network (see next slide) consists of several tracks, junction points and trains. A train moves along the track until it requires to change the track to reach destination. The changes of track occur only through a junction point. Each train is initially positioned at a junction point and its destination is a different junction point. Hence, while a train approaches its destination junction point, it does not need to change tracks any more. Each track can have at most one train at a time to avoid collision, but all tracks in the railway network are bi-directional. Design a set of interfaces (e.g. Java interface) for a safe (i.e. collision free) system. No coding is required, the interfaces (i.e. member variables and implementable functions) and their purpose will suffice.

Cohort Exercise 1.2



Hierarchy

- Whenever we examine something new or something that seems to be complex, we try to see what common characteristic it shares with other things we know, or if it is part of something we have seen earlier.
- Key idea:
  - "Is a" relation: E.g. a car *is a* locomotion device with wheels
  - "Part of" relation: A *part of* a car is the engine

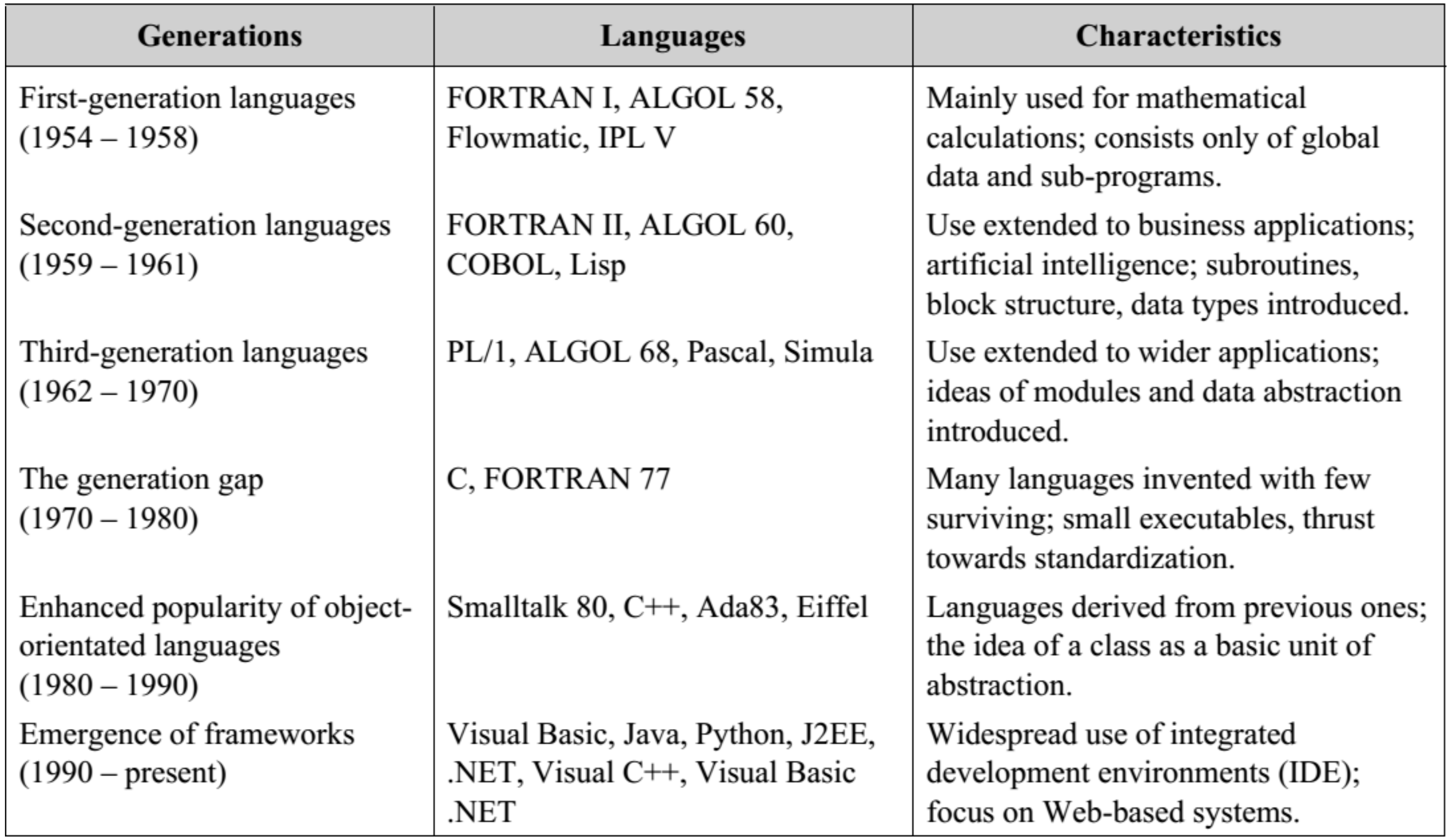
Cohort Exercise 1.3

- Consider the railway network from previous exercise. Assume tracks of different type – broad gauge, meter gauge and narrow gauge. Each junction point is further divided into individual establishments that exclusively handle a specific track (i.e. meter, broad or narrow). Similarly, trains for meter gauge track is different from trains for broad gauge and narrow gauge. Narrow gauge trains are not powerful to run longer than the distance between two junction points. At each junction point, therefore, its engine is changed. Refine your set of interfaces to capture this system.

The Spectrum of Languages

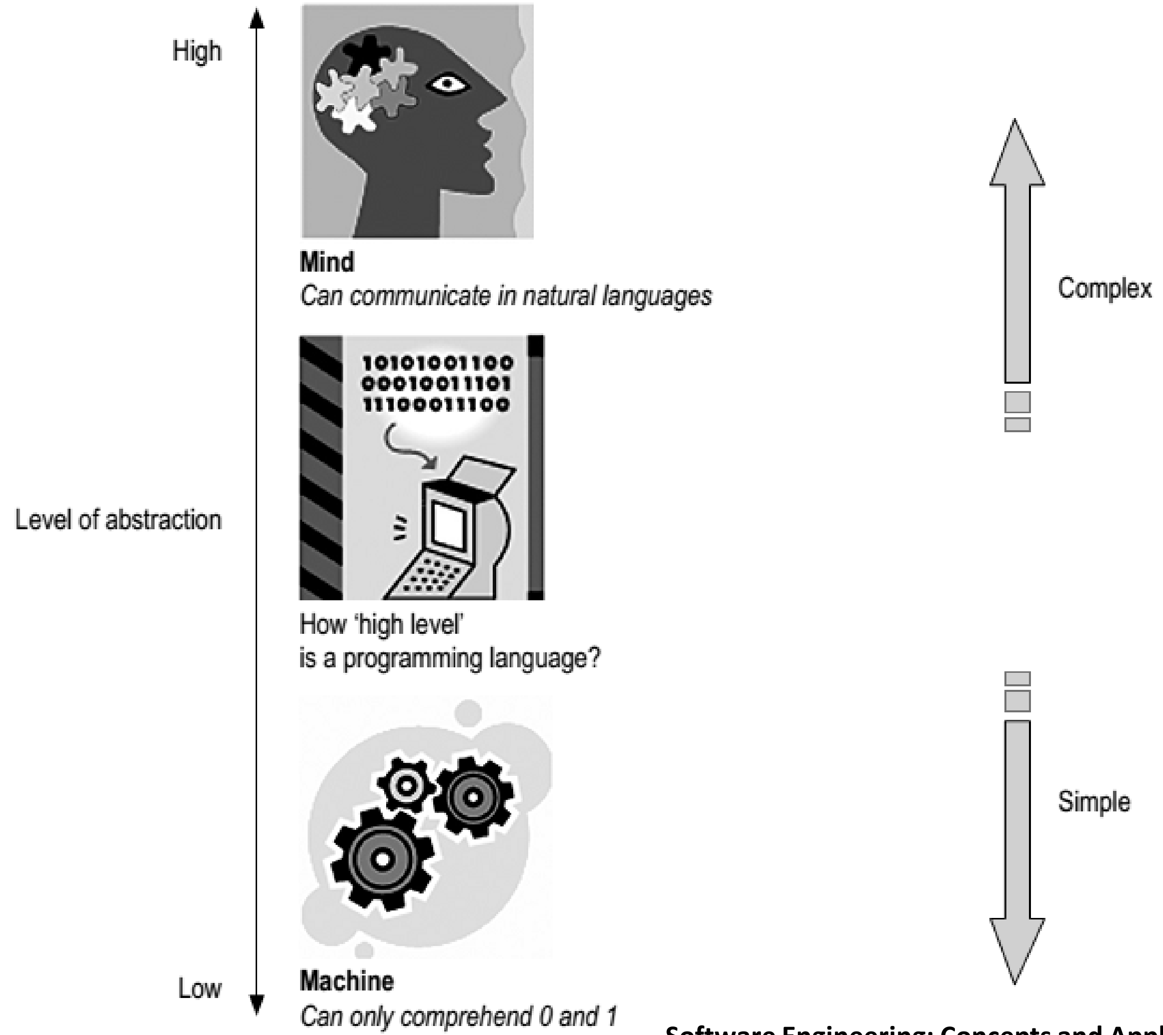
- Languages for implementation:
  - programming languages: precise and executable but perhaps not a good media for communication
- Languages for software development process:
  - modelling languages: often visual, often not executable, high-level, aiming for communication between different parties in software development processes
- Languages for user requirements:
  - Structured English: not visual, not executable, not precise, extremely flexible

Generations of PL



Generations	Languages	Characteristics
First-generation languages (1954 – 1958)	FORTRAN I, ALGOL 58, Flowmatic, IPL V	Mainly used for mathematical calculations; consists only of global data and sub-programs.
Second-generation languages (1959 – 1961)	FORTRAN II, ALGOL 60, COBOL, Lisp	Use extended to business applications; artificial intelligence; subroutines, block structure, data types introduced.
Third-generation languages (1962 – 1970)	PL/1, ALGOL 68, Pascal, Simula	Use extended to wider applications; ideas of modules and data abstraction introduced.
The generation gap (1970 – 1980)	C, FORTRAN 77	Many languages invented with few surviving; small executables, thrust towards standardization.
Enhanced popularity of object-orientated languages (1980 – 1990)	Smalltalk 80, C++, Ada83, Eiffel	Languages derived from previous ones; the idea of a class as a basic unit of abstraction.
Emergence of frameworks (1990 – present)	Visual Basic, Java, Python, J2EE, .NET, Visual C++, Visual Basic .NET	Widespread use of integrated development environments (IDE); focus on Web-based systems.

The High and Low of PL



High
Level of abstraction
Low
Mind
Can communicate in natural languages
10101001100
00010011101
11100011100
How 'high level' is a programming language?
Machine
Can only comprehend 0 and 1
Complex
Simple

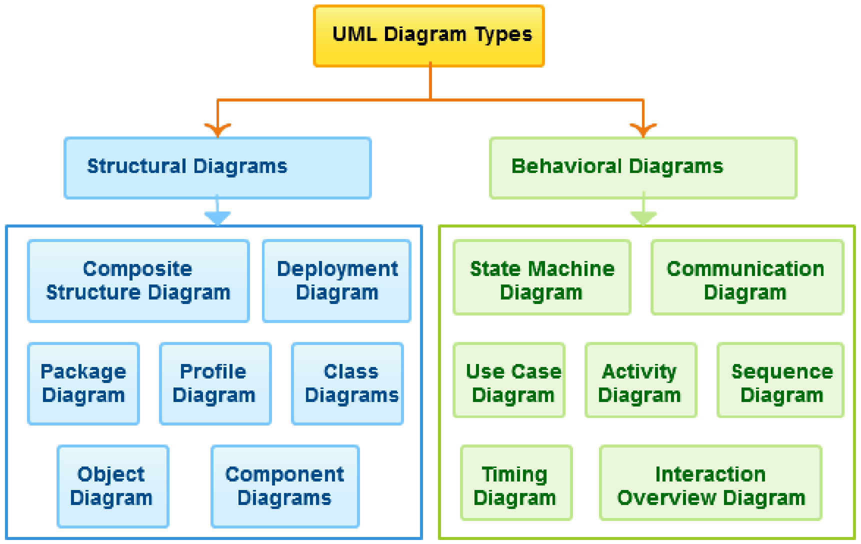
Software Engineering: Concepts and Applications by Subhajit Datta (Oxford University Press, 2010)

Why Modeling Languages?

- Modeling languages are mainly for documentation and communication.
- Documentation is an important part of software engineering.
  - Documentation is perhaps a part of any software development process.
  - It forms the basis of an iterative design process.
  - It forms the basis of effective communication among three parties, through time.
- There are formal modeling languages and informal ones.

Unified Modelling Language

- UML is extensible.
- There are 14 UML diagram types.



- UML is flexible.

User Requirements: a Case Study

Preeti is leading a team of software engineers engaged in building the system. Kuber Bank (KB) has been in the banking business for over thirty years with branches all over India. About 15 years ago, the branches were computerized and they are currently connected through a nation-wide network. To keep up with the competition, KB needs to offer online banking facilities to its customers. Preeti's organization has won the contract to 'Webify' KB. An initial meeting was arranged between KB's senior management and Preeti's team to understand the scope of the projects.

- 'So, as a first step, it would be good for us to know the broad requirements you have in mind', began Preeti.

- 'Well, we want to let the customers do online all they can do at a branch', Sanjeev Kumar, KB's senior vice-president, replied.

Software Engineering: Concepts and Applications by Subhajit Datta (Oxford University Press, 2010)

Roles and Functionality

User can:

- View their profile information
- View list of their accounts with the bank
- View transactions for each of their accounts
- Send messages to the bank
- View history of messages they have sent to the bank and replies from the bank

Administrator can:

- View transactions across all accounts
- Add transactions to any account
- Delete transactions from any account
- Review all messages sent by users
- Reply to messages sent by users

Glossary

- Account: the bank account held by a user with Kuber Bank (KB), identified by a unique account number
- Administrator: A special type of user who has unique privileges in manipulating information relevant to KBO.
- KBO: the Kuber Bank Online system
- User: An account holder of KB who is allowed by KB to use KBO
- …

Can the Administrator be an User too?

Description of Requirements

KBO_Req_01: Recording Transaction Details in KBO by administrator

Administrator shall access transaction information of a particular account on a periodic basis and record the same in the KBO system such that user can view them through the Web interface. The transaction information will be available to be accessed by the administrator in the existing Master Ledger (ML) system. Details for a transaction need to be available at KBO for viewing by a user within 48 hours after the transaction have been initiated by the user.

Is this good enough?

Cohort Exercise (5 minute)

Read the requirement and find out some incompleteness or ambiguity.

Administrator shall access transaction information of a particular account on a periodic basis and record the same in the KBO system such that user can view them through the Web interface. The transaction information will be available to be accessed by the administrator in the existing Master Ledger (ML) system. Details for a transaction need to be available at KBO for viewing by a user within 48 hours after the transaction have been initiated by the user.

Description of Requirements

KBO_Req_01 Expanded

Administrator shall manually access transaction information for a particular account on a periodic basis from the existing Master Ledger system. (Automation of this access mechanism is outside the scope of KBO's current release and may become a Requirement for a future release.) Administrator shall record the transaction information for a particular account accessed from the ML system, in the KBO system. Only successful transactions need to be available at KBO for viewing by users within 48 hours after they have been initiated by the user. Users will be notified of failed transactions through procedures outside the scope of KBO's current release.

Description of Requirements

KBO_Req_01: Non-functional Parts

- Ensuring transactions are recorded in a format that makes it reasonably easy to be understood by users. *(user friendly)*
- Ensuring successful transactions are recorded at KBO within 48 hours of their initiation. *(performance)*
- Ensuring large volumes of transaction for a particular account within a particular period of time can be handled. *(scalability)*

Use Case

“To my knowledge, no other software engineering language construct as significant as use cases has been adopted so quickly and so widely among practitioners. I believe this is because use cases play a role in so many different aspects of software engineering.”

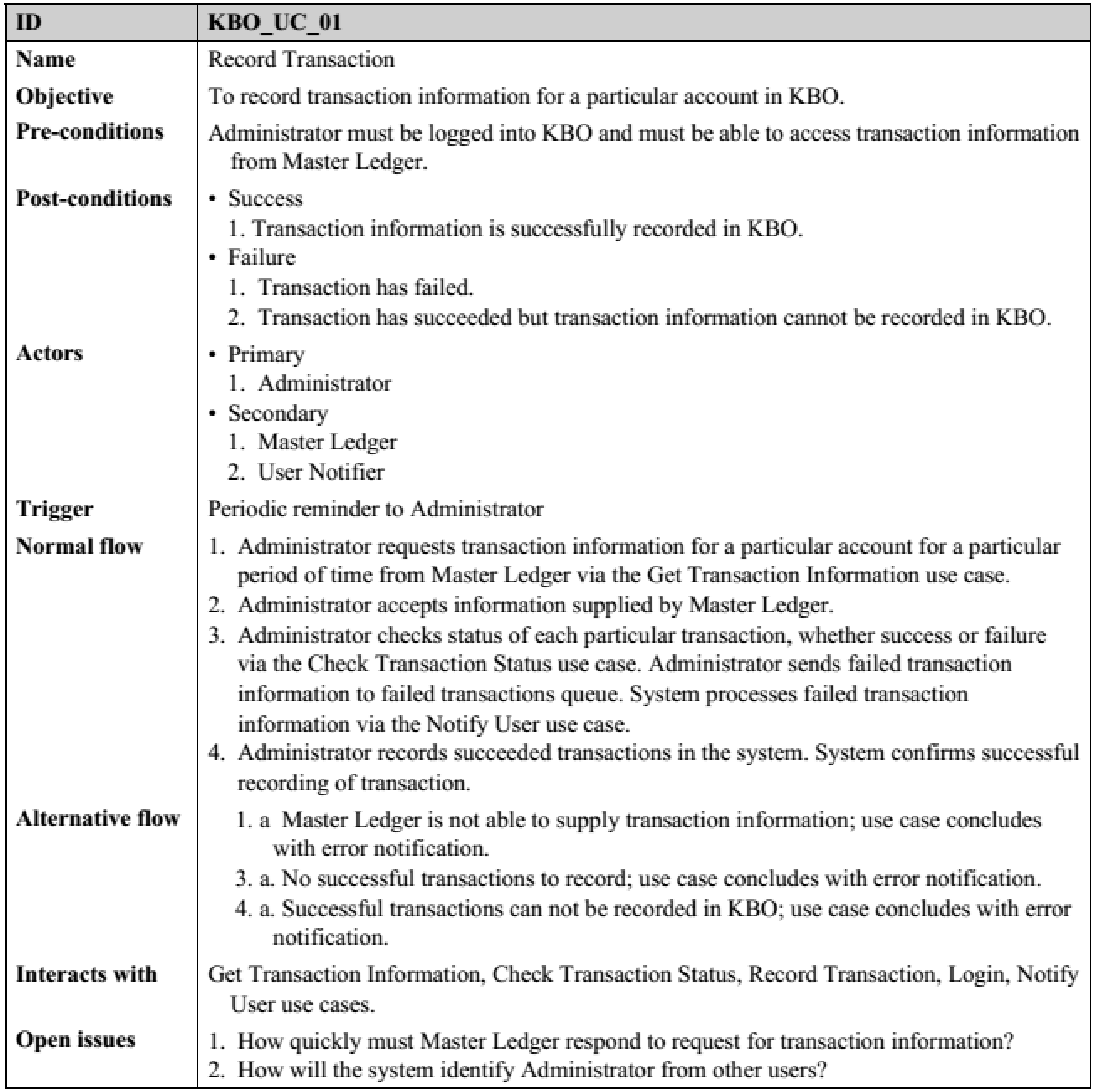
Ivar Jacobson

Founding father of UML

Creator of Use Case Diagram

"Interacts with" captures the set of use case documents "this" use case interacts with.

Table 15.6 Use Case KBO_UC_01: Record Transaction



ID	**KBO_UC_01**
Name	Record Transaction
Objective	To record transaction information for a particular account in KBO.
Pre-conditions	Administrator must be logged into KBO and must be able to access transaction information from Master Ledger.
Post-conditions	• Success 1. Transaction information is successfully recorded in KBO. • Failure 1. Transaction has failed. 2. Transaction has succeeded but transaction information cannot be recorded in KBO.
Actors	• Primary 1. Administrator • Secondary 1. Master Ledger 2. User Notifier
Trigger	Periodic reminder to Administrator
Normal flow	1. Administrator requests transaction information for a particular account for a particular period of time from Master Ledger via the Get Transaction Information use case. 2. Administrator accepts information supplied by Master Ledger. 3. Administrator checks status of each particular transaction, whether success or failure via the Check Transaction Status use case. Administrator sends failed transaction information to failed transactions queue. System processes failed transaction information via the Notify User use case. 4. Administrator records succeeded transactions in the system. System confirms successful recording of transaction.
Alternative flow	1. a Master Ledger is not able to supply transaction information; use case concludes with error notification. 3. a. No successful transactions to record; use case concludes with error notification. 4. a. Successful transactions can not be recorded in KBO; use case concludes with error notification.
Interacts with	Get Transaction Information, Check Transaction Status, Record Transaction, Login, Notify User use cases.
Open issues	1. How quickly must Master Ledger respond to request for transaction information? 2. How will the system identify Administrator from other users?

Every use case must specify: name, objective, actors, normal flow (precondition, post condition)

Software Engineering: Concepts and Applications by Subhajit Datta (Oxford University Press, 2010)

Use Case

- Constraints - The formal rules and limitations a Use Case operates under, defining what can and cannot be done. These include:
  - Pre-conditions that must have already occurred or be in place before the use case is run; for example, <create order> must precede <modify order>
  - Post-conditions that must be true once the Use Case is complete; for example, <order is modified and consistent>
  - Invariants that must always be true throughout the time the Use Case operates; for example, an order must always have a customer number.

Flow of Events

Basic flow of events

- Is the most common pathway. It usually depicts a perfect situation, in which nothing goes wrong.

Alternative flow of events

- Is a pathway that is still considered a good pathway; it’s just not the most heavily travelled one.

Exception Flow of Events

- Things don’t always go as planned. An exception is an error condition that is important enough to the application to capture.