Université Mohammed V
Faculté des Sciences
Rabat

Software Engineering

Software Engineering

Software Specification
Software Life Cycle
Software Design
Software Metrics
Software Testing
Software Quality
Formal Methods for Software Engineering,
Search-based Software Engineering
Evolutionary Software Engineering

Author: Y. EL AMRANI.

---

foreword: this course has been taught in french from 2011 to 2014, then in English from 2015 up to now ; all at the University Mohammed V Rabat

Université Mohammed V
Faculté des Sciences
Rabat

Université Mohammed V
Faculté des Sciences
Rabat

Université Mohammed V
Faculté des Sciences
Rabat

Université Mohammed V
Faculté des Sciences
Rabat

All Thanks go especially to the
Faculty of Sciences Of Rabat

Université Mohammed V
Faculté des Sciences
Rabat

Université Mohammed V
Faculté des Sciences
Rabat

Introduction Chapter 1

- In the past 50 years, starting in the NATO conference in 1969, the term “Software Engineering” has been coined by FRIEDRICH LUDWIG BAUER (born in 1924: 92 years old in 2016) and emerged as an unavoidable field of computer science in software production.

Friedrich Ludwig Bauer, NATO, 1969

What is Software

- Software is a computer program that can be decomposed in two parts and sometimes three...
- ...the first part, at the heart of the software's entity, one finds a model, used for representing (for modeling) a problem to solve in the physical world or in the cybernetics (science of communication) or in internet itself (searching robot, software fighting viruses, etc.)
- … the software's second part, is a set of many algorithms (or just one) used to compute, optimize the solutions, calculate the problem's solution, learn about alternatives to the problem modeled in the first part!
- …the software's third part is generally the biggest part in term of code, used to manage the interaction between the user of the program and the program's functionalities, lying behind the complexity of its algorithms

What is Software Engineering

- Software Engineering is: all the methods, techniques, environment, requirements engineering, planning and every engineering principles and practices that lead to sound software development and…

Software versus hardware

- The frontier between software and hardware is a moving frontier, the too of them leave in perfect harmony: the software generally drive the hardware, but both of them realize computations; hardware is always much faster at computing, however, software is a driving force for complex computation that have no hardware counterpart

Software Engineering Domains

There are many domains that can be directly or indirectly related to software engineering, here are some of them:

1) **Software requirements engineering**
2) **Software architecture and design**
3) Patterns and frameworks
4) Software components and reuse
5) **Software testing and analysis**
6) **Theory and formal methods**
7) Computer supported cooperative work
8) **Human-Computer Interaction**
9) Software processes and workflows
10) Engineering secure software
11) **Software dependability, safety and reliability**
12) **Reverse engineering and maintenance**
13) Program comprehension and visualization
14) **Software economics and metrics**
15) Empirical software engineering
16) End user software engineering

1) Aspect-orientation and feature interaction
2) **Engineering of distributed/parallel SW systems**
3) **Engineering of embedded and real-time software**
4) Software engineering for mobile, ubiquitous and pervasive systems
5) Software tools and development environments
6) SW Configuration management and deployment
7) Software policy and ethics
8) **Programming languages**
9) AI and Knowledge based software engineering
10) Internet and information systems development
11) End user software engineering

Chapter 1: Perspectives

In perspective, there are formal methods that can be used for modeling software, verifying if before production and… generating automatically code that would be correct by construction.

Chapter 1: Conclusion

- Software engineering is large and vast domain, many computer science activities could be viewed as a particular sub field as to all of them relate to a specific type of software to produce.

Chapter 2: Software Engineering
Software Production Process Life Cycles
Development Models

Chapter 2 : Software Production Life Cycles

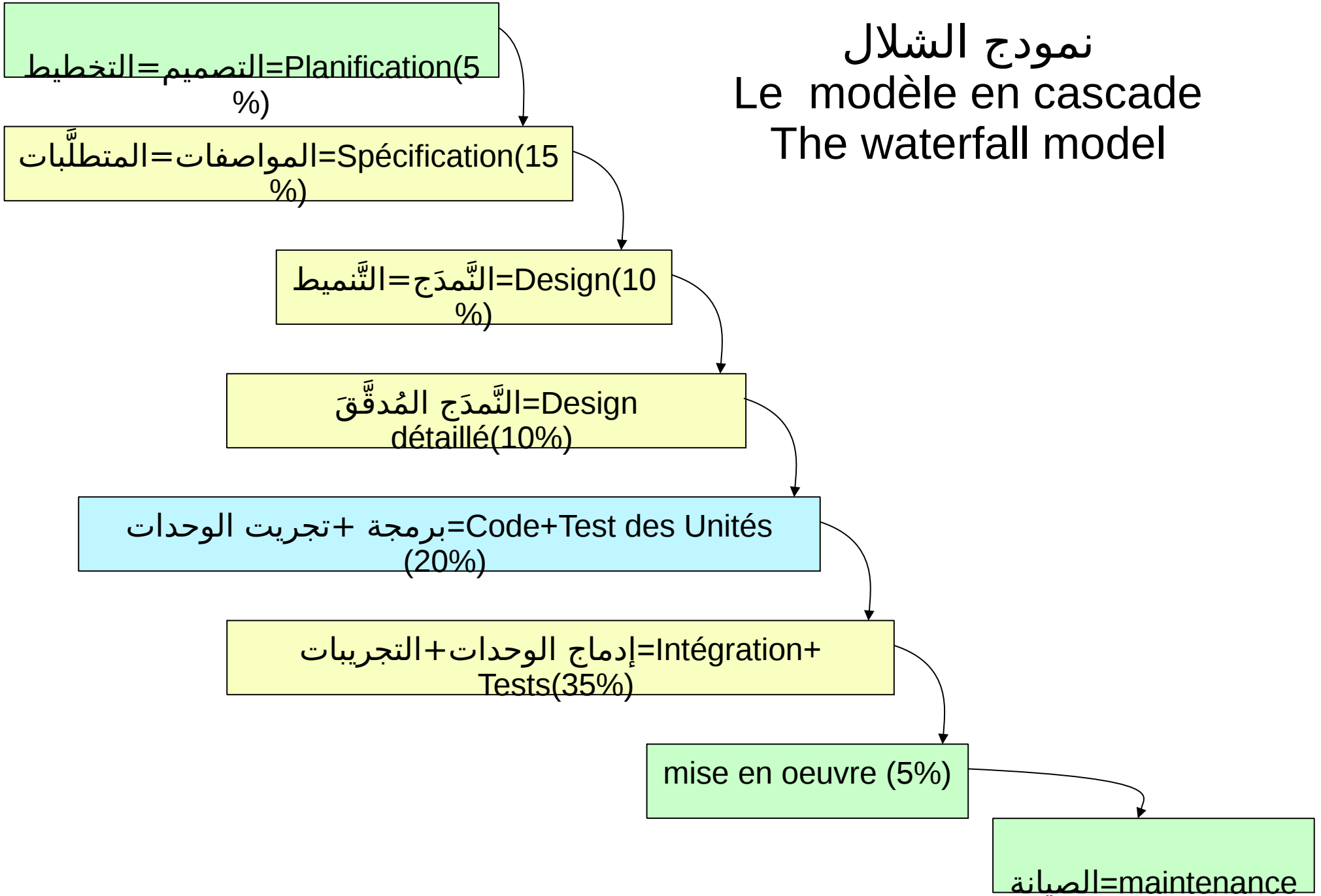
Author : Younès El Amrani

2011-2019

Introduction to chapter 2

- ✔ There are several life-cycle that have been proposed to model the software life cycle, the remaining of this chapter presents some of these

نمودج الشلال
Le modèle en cascade
The waterfall model

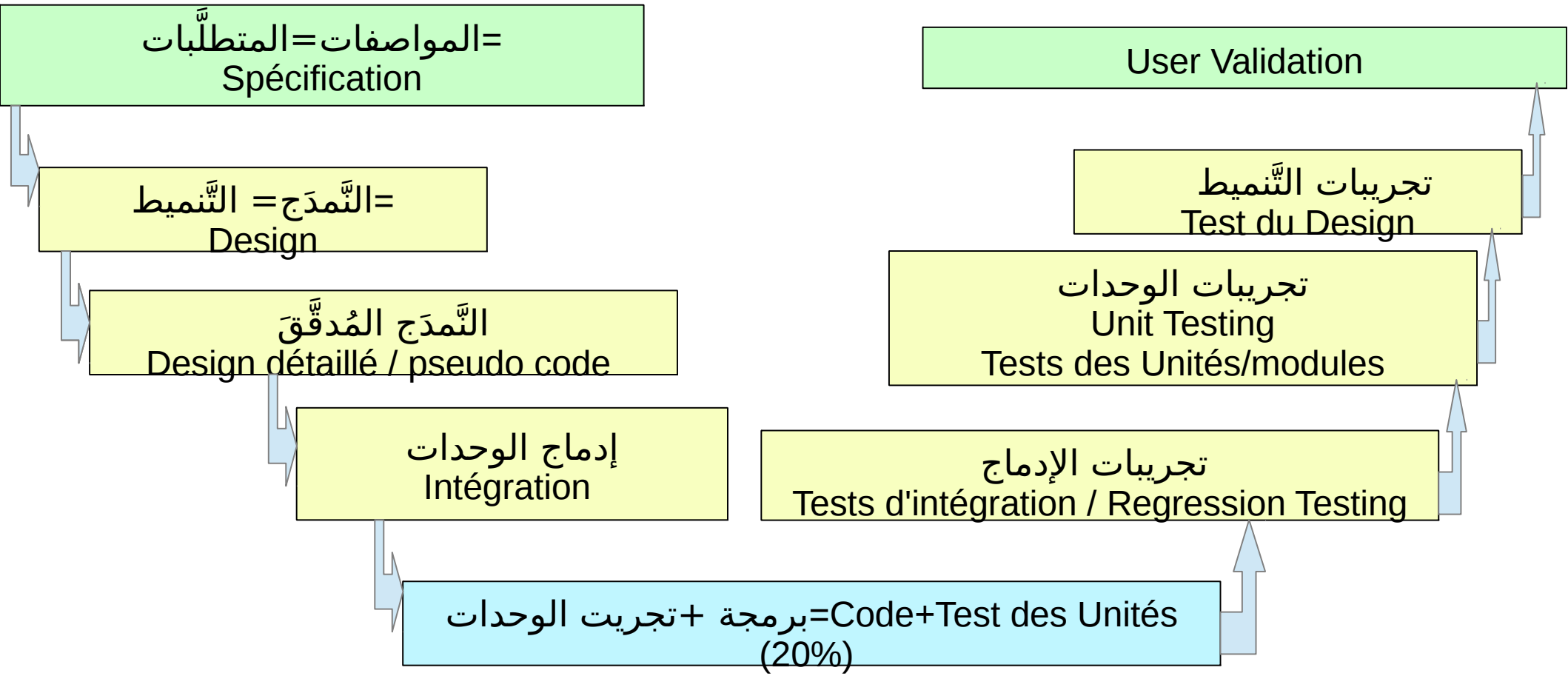


نمودج V
Modèle en V
The V model

Traditionaly, maintenance and operation phase are Not represented in the V Model

الصيانة=maintenance

mise en oeuvre (5%)



Perspectives of chapter 2

- ✔ The future of software production does include a huge part of software production in an automated fashion and automated reuse.
- ✔ Most of the current life-cycles do not cope with software reuse or software automated production and testing.

Conclusion of Chapter 2

- ✔ The immaterial nature of software does necessitate some specific improvement to traditional Life cycle, notably, the introduction and re-introduction of testing nearly in all software production stages!

Chapter 3: Software Effort Estimation with COCOMO

Software Effort Estimation with COCOMO II

---

author: Younès EL AMRANI

2011-2016

Introduction to chapter 3

- ✔ Without metrics and measure, there is no possibility for speaking of any engineering, there is no science at all without measurement !

COCOMO
Constructive Cost Model

- COCOMO : introduction to the effort measurementl

COCOMO's output is Effort and COCOMO's Input are Lines of Code

$$Effort = a * \left(KLOC \right)^{b}$$

Where

Effort is the Effort in staff months (homme mois)

a and b are coefficients depending on the type of project

KLOC is thousands of lines of code

COCOMO's constants, a and b, depend on the nature of the project

The constants a and b depend on the nature of the project, COCOMO proposes a classification of projects in three categories: (1) organic (2) semi-detached (3) embedded.

The Constants

Mode	a	b
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

COCOMO'S MODES : The ORGANIC MODE

A project would be said organic if it is relatively small, and there is little innovation in it.

The Constants

Mode	a	b
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

COCOMO'S MODES : The EMBEDDED MODE

A project would be said embedded if it is relatively large, and there is innovation in it.

The Constants

Mode	a	b
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

COCOMO'S MODES : The SEMI-DETACHED MODE

A project is said to be semi-detached if it is medium sized and somewhere in between organic and embedded

The Constants

Mode	a	b
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

Example with 100 KLOC

With a project of 100 KLOC, we have the following values for the effort:

Organic = 2.4*pow(100,1.05)=302 man/month
Semi-Detached=3.0*pow(100,1.12)=521 man/month
Embedded=3.6*pow(100,1.2)= 904 man/month

The Constants

Mode	a	b
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

COCOMO's unit of measurement: the (mythical) man-month

The Basic COCOMO formula does express the effort of development in a classic measure: man-month.

Such unit of measure, as pointed by **Frederick P. Brooks, JR.** in is book entitled **The Mythical Man-Month**, would work only if Men and Months are interchangeable commodities.

COCOMO’s unit man-month: is it appropriate for computer systems?

Obviously the unit man-month is NOT appropriate for computers systems! Men and Month are not interchangeable commodities in this case for two reasons:

1) on one hand, it is not obvious for any project involving many communication within the system how to partition it logically and consistently, and on a second hand,

2) software systems do involve many communication between developers at ALL stages of development.

COCOMO misses training of the developers on project's technologies

There is a necessary need for multi-channels communication between all programmers involved in the development of computer systems.

This communication is made of two parts:

(1) the training and

(2) intercommunication.

COCOMO: no intercommunication between developers accounted

(2) the intercommunication:

The intercommunication between programmers does explode if we consider that each part of the project does need to communicate with all other parts: assuming the project is divided in N teams of programmers, then you would need **N*(N-1)/2** distinct channels of intercommunication.

Less programmers, induces less intercommunication induces less effort

In his book,**Frederick P. Brooks, JR.** suggests us the following:

If you have 200-man project, with 25 managers who are the best programmers to manage the 175 programmers, then fire the 175 troops and put the 25 managers back to programming !

Second COCOMO equation for TIME of Development Estimation

(TIME OF DEVELOPMENT) TDEV = c (E)^d

TDEV is the TIME of Development

c and d are constants whose value depends on the type of project -ORGANIC, SEMI-DETACHED, EMBEDDED-

E is the Effort, the unit is man-month.

In JAVA, the equation would be implemented:

TDEV=c*(Math.pow(Effort), d) ;

COCOMO's constant values for TIME of Development

TIME OF DEVELOPMENT = TDEV = c (E)^d

c and d are constants whose value depends on the type of project, the project could be one of three: ORGANIC, SEMI-DETACHED or EMBEDDED. Here are the values of c and d for each case:

For ORGANIC projects: (**c=2.5 ; d=0.38)**

For SEMI-DETACHED projects: (**c=2.5 ; d=0.35)**

For Embedded projects (**c=2.5 ; d=0.32)**

Time of Development picking up from the example with 100 KLOC

With a project of 100 KLOC, we have the following values for the effort:

Organic = 2.4*pow(100,1.05)=302 man/month
TDEV=2.5*(302)^0.38 = 21,9 months

Semi-Detached=3.0*pow(100,1.12)=521 man/month
TDEV=2.5*(521)^0.35 =22,3 months

Embedded=3.6*pow(100,1.2)= 904 man/month
TDEV=2.5*(904)^0.32 = 22,1 months

You can notice that the time of development remains almost the same, this is not by chance. For that purpose the effort has been set in an equation in such a way that, consistently, the duration of the project remains the same, whatever the type of project.

Effort / Time-Of-Development = Average Staff Size

With the Effort in man-month and the Time Of Development, we can compute the Average Staff Size on Average

SSOA=E/TDEV=[men per month] / [months] = [men or staff]

SSOA(100)=302/22=14 (Organic)
SSOA(100)=521/22=24 (semi-detached)
SSOA(100)=904/22=41 (embedded)

Productivity can also be evaluated: = Size / Effort

If we divide the size by the Effort, we obtain the average productivity per month:

PRODUCTIVTY = SIZE / EFFORT (#KLOC per man-month)

What After Basic COCOMO?
Here Comes in Intermediate COCOMO

Another Factor is added, called the **EFFORT ADJUSTEMENT FACTOR** abbreviated by EAF.

The **EAF** is itself the product of **15 Adjustment factors, also called COST DRIVERS.**

All the adjustment factors come out from the software engineer's mind !

EFFORT = a * Math.pow(KLOC , b) * EAF

What are the cost drivers of Intermediate COCOMO ?

All the values of the cost drivers are made into a product name EAF.
EAF abbreviates EFFORT ADJUSTMENT FACTOR !
EFFORT = a * Math.pow(KLOC , b) * EAF

algorithm to compute EAF

```
EAF=1 ;
for(double EAF=1,int i=1:i<=15;i++)
        EAF*=costDrivers[ i ] ;
return EAF ;
```

What are the cost drivers Categories of Intermediate COCOMO ?

There are FOUR categories of Cost Drivers:

1) Personnel Attributes
2) Project Attributes
3) Computer Attributes
4) Product Attributes

Cost Drivers related to Personnel Attributes

There are FIVE Cost Drivers associated with Personnel Attributes

1) **ACAP** Analyst Capability, provide a measure of the analyst capability
2) **AEXP** Application Experience: measures experience on the application
3) **PCAP** Measures the Programming Capability of Programmers
4) **VEXPT** Measures the Virtual Machine Experience: both Software and Hardware are meant
5) **LEXP** Programming Language Experience

Cost Drivers related toProject Attributes

1) **MODP** Measure the capability to use Modern Programming Practices
2) **TOOL** Measures/Evaluates the capability to use software tools as of software to design the program, to automate the GUI programming, to automate part or the whole testing stage...
3) **SCED** This is the Required Development Schedule, it measures the constraint we have on the schedule (tight, large, etc.)
Note that: too large schedule impact negatively the project cost as much as too tight schedule !

Cost Drivers related to Computer Attributes

1) **TIME** Measure the capability to cope/sustain Execution Time Constraint: real time constraint, time limit to achieve computation for divers functionalities, speed of answer...

2) **STOR** Measures/Evaluates the Main Storage Constraint that we have on the project (for BIG DATA as a modern example: how to cope with the growing size of information, what are the constraints on the storage ?

3) **TURN** Measure, indicates, evaluates the Computer Turnaround Time (Evolution, Modification, Hardware Innovation)

Cost Drivers related to Product Attributes

1) **RELY** Measure/evaluates/indicates the required Software Reliability
2) **DATA** Measures, Evaluates and Indicates the Data Base Size
3) **CPLX** Measure, indicates, evaluates the product complexity.

COCOMO I (1981) evolution to COCOMO II

There are Changes in cost drivers, one can thing of these changes as more accurate naming:

Added cost drivers: DOCU, RUSE, PVOL, PLEX, LTEX, PCON, SITE

DOCU	Measures suitability of the doc. with project's life cycle
RUSE	Measures additional effort to develop component for reuse
PVOL	Measures Platform Volatility (it includes assemblers, compilers, any technology volatility and major changes over project time)
PLEX	Measures Platform Experience (previously **TURN in Computer Attributes)**
LTEX	Language and Tool Experience : **LEXP** and **TOOL**
PCON	Personal Continuity : the project's annual personnel turnover
SITE	Measures Site collocation and communication support

Deleted cost drivers: VIRT, TURN, VEXP, LEXP, MODP

COCOMO Advantages 1/2

- ✔ COCOMO is the first model ever to provide an estimate for software development, it was introduced in 1981 and was extensively adopted and used by the software industry

COCOMO Advantages 2/2

- ✔ COCOMO is works quite well on similar projects after some time of experience and it is very well documented
- ✔ COCOMO is quite well documented and has a long history of use in the industry
- ✔ COCOMO adjustment factor provide a unique way to adapt carefully and precisely the model to a peculiar context of and industry: all major factors are represented!
- ✔ Furthermore: COCOMO has proved to have evolutionary capabilities and that it can cope with new standards emerging in the industry

COCOMO LIMITATIONS

- ✔ COCOMO must predict project size
- ✔ COCOMO requires adjustment factors to be accurate and that is very demanding in terms of expertise: it requires a very high level of expertise of the project and of all the stakeholders: programmers and designers: the model cannot

Conclusion

- ✔ Software measurement is fundamental to software engineering as a domain of engineering, COCOMO has brought the necessary breakthrough in the early eighties to allow the field to win its badge of honor in the battlefield of excellency, accuracy, robustness and software quality, from specifications to maintenance throughout the whole process of development.

Agility in Software Engineering

Agility in Software Engineering
The Agile Manifesto

Younès EL AMRANI

2011-2016

February 2001, 17 Software Developers met at a ski resort in Snowbird, Utah

The Agile Manifesto – Where it all began

AGILE ACADEMY

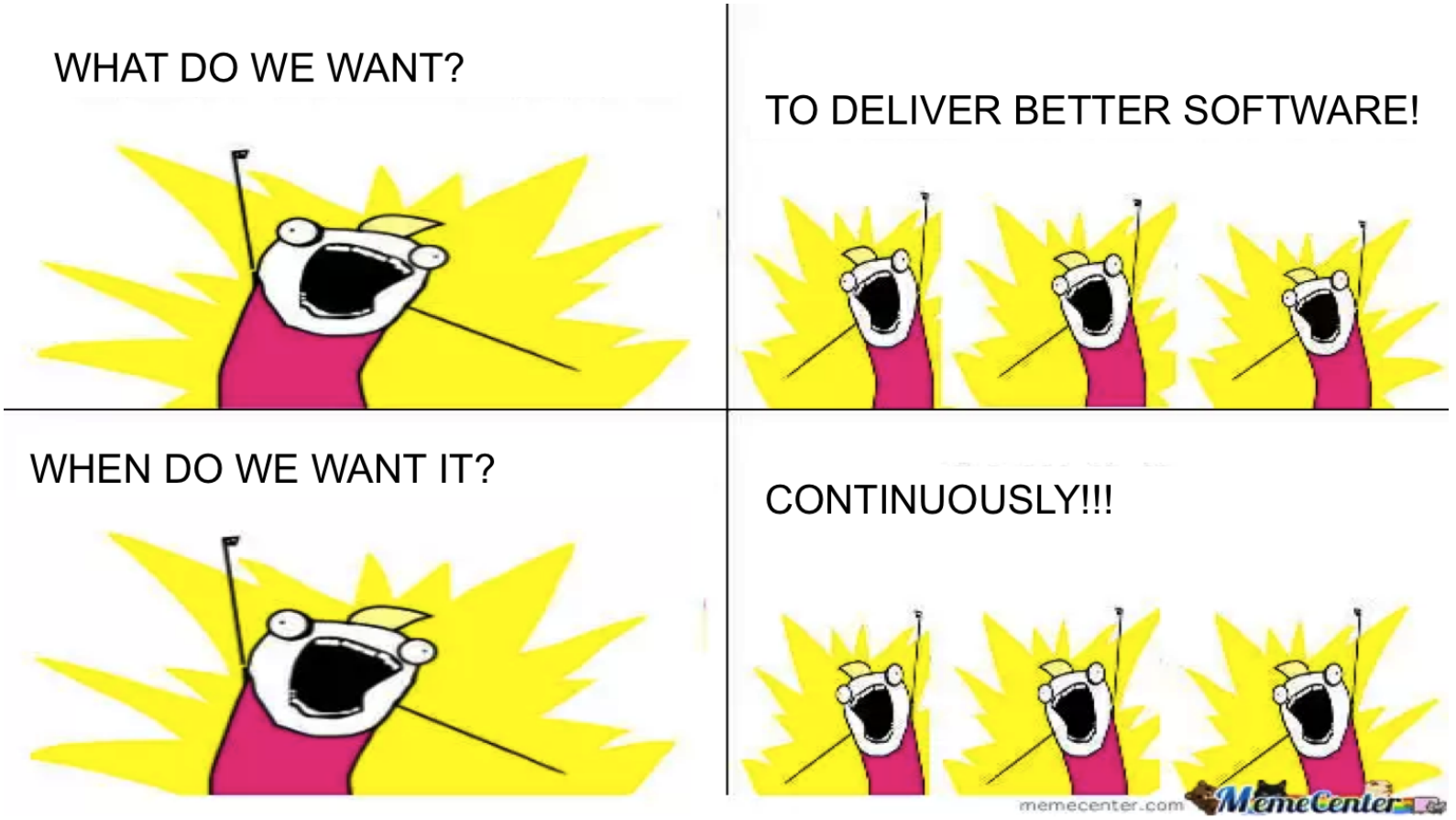
In February 2001, 17 software developers met at a ski resort in Snowbird, Utah, to discuss lightweight development methods. They published the ***Manifesto for Agile Software Development*** to define the approach now known as Agile software development.

Some of the Manifesto's authors formed the Agile Alliance (www.agilealliance.com), a non-profit organisation that promotes software development according to the Manifesto's principles. There is now discussion about changing to focus to solutions delivery for business and IT.

Source: http://en.wikipedia.org/wiki/Agile_Manifesto

The Agile Manifesto reads:

When the 17 developer met, they asked: what do we want?



17 years ago, what is the perspective of the 17’s manifesto

- In February 2001, 17 senior software developers met to discuss lightweight development
- They published the Manifesto for Agile Software Development to define the approach now known as agile software development.

- Some of the manifesto's authors formed the Agile Alliance,
- The Agile Alliance is a nonprofit organization that promotes software development according to the manifesto's principles.

The 17 said: we have come to value:

(1) Individuals and interactions over processes and tools

(2) Working software over comprehensive documentation

(3) Customer collaboration over contract negotiation

(4) Responding to change over following a plan

What the 17 have come to value?

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

Individuals and interactions = individuals co-location + individual organized in pair programming.

- **Working software** = more useful and welcome than just presenting documents to clients
- **Customer collaboration** = continuous customer or stakeholder involvement is very important.
- **Responding to change** = agile development is focused on quick responses to change and continuous development.

The twelve principles

1. Customer satisfaction by **rapid delivery** of useful software
2. Welcome **changing requirements**, even late in development
3. **Working software** is delivered frequently (weeks rather than months)
4. Working software is the principal **measure of progress**
5. **Sustainable development**, able to maintain a constant pace
6. Close, **daily cooperation** between business people and developers
7. **Face-to-face conversation** is the best form of communication (co-location)
8. Projects are built around **motivated individuals**, who should be trusted
9. Continuous attention to **technical excellence** and good design
10. **Simplicity**—the art of maximizing the amount of work not done—is essential
11. **Self-organizing** teams
12. Regular **adaptation to changing** circumstances

Some thirteen methods and... more

1. Agile Modeling
2. Agile Unified Process (AUP)
3. Crystal Clear
4. Crystal Methods
5. Dynamic Systems Development Method (DSDM)
6. **Extreme Programming** (XP)
7. Feature Driven Development (FDD)
8. Graphical System Design (GSD)
9. Kanban
10. Lean software development
11. **Scrum**
12. Velocity tracking
13. Software Development Rhythms

12 principles sprouted from the 4 founding principles

Agility: #1 Customer's satisfaction #2 Welcoming changes

01 Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

02 Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

03 Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

04 Business people and developers must work together daily throughout the project.

05 Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

06 Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

07 Working software is the primary measure of progress.

08 The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

09 Continuous attention to technical excellence and good design enhances agility.

10 Simplicity—the art of maximizing the amount of work not done—is essential.

11 The best architectures, requirements, and designs emerge from self-organizing teams.

12 At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The 12 principles revisited

Agile Manifesto - Principles

1. Our highest priority is to **satisfy the customer** through early and continuous delivery of valuable software.
2. Welcome **changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.
3. **Deliver** working software **frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must **work together** daily throughout the project.
5. Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**.
7. **Working software** is the primary measure of progress.
8. Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to **technical excellence** and **good design** enhances agility.
10. **Simplicity** - the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from **self-organizing teams**.
12. At regular intervals, the **team reflects** on how to become more effective, then tunes and adjusts its behavior accordingly.

The four founding principles in 8 words

CMMI or Agile ?
In 2008 SEI said: take both

- ✔ In 2008 the Software Engineering Institute (SEI, author of CMMI) published a technical report to make clear that Capability Maturity Model Integration and agile can co-exist. The report is entitled:

"CMMI or Agile: Why Not Embrace Both"

- ✔ CMMI Version 1.3 includes tips for implementing Agile and CMMI.

Perspectives to Chapter 04

- Agile methods have been brought to existence in 2001 by 17 re-known software Engineers, almost 17 years later, some 17 agile methods have been proposed, tested and implemented throughout the Software industry.

Conclusion to Chapter 04

- ✔ The Agile methods are now unavoidable in the Software Industry, might be one of the best proposal made in the early twenties to bring new perspectives to the software industry.
- ✔ Agile methods have indeed brought some very efficient processes to produce Software at a right rate and good quality. Methods like SCRUM have found an astounding welcome throughout the industry.
- ✔ Agile Methods have brought nonetheless innovative ideas to the domain, but they also mixed good ideas together and associated inspection to high degrees of communication, based day-to-day, between software project's stakeholders!

Chapter 5: The Agile Method SCRUM

SCRUM
The Agile Method SCRUM

Younès EL AMRANI

2011-2016

Introduction to Chapter 5

The Agile method SCRUM is one of the most successful Agile Methods that has been released after the Agile Manifesto in 2001.

There are grosso modo two types of methodologies:

1) Heavy methodologies following rigid product life cycles like the **waterfall model, the spiral model** and the like and,

2) **Agile methodologies** follow a more flexible, incremental, adaptable, recurrent and innovative product life cycles, like **SCRUM, Extreme Programming (XP) DDSM** and the like

SCRUM come up with its own innovative Software Product Development Life Cycle, bridging incremental development, code inspection and peer to peer communication in a successful mixing successfully several software process development boosters !

.

SCRUM is an answer to the Agile Software Development Manifesto

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right,

we value the items on the left more.”

THE SEVENTEEN SIGNATORIES ARE :

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 ;

SCRUM Terminology Translated in French and Arabic

Scrum flow

Scrum roles

Scrum artifacts

Scrum backlog

Scrum est le nom anglais de la Mêlée en rugby un espèce d'affrontement pour obtenir le ballon dans un effort rapide, énergique!

Artifact: an object made by a human being, typically an item of cultural or historical interest.

Backlog:an accumulation of something, especially uncompleted work or matters that need to be dealt with.

Shippable: deliverable ;

shipping= expedition ;

shipment: boat cargaison

Perspectives to Chapter 5

- Agile methods have been brought to existence in 2001 by 17 re-known software Engineers, almost 17 years later, some 17 agile methods have been proposed, tested and implemented throughout the Software industry.

Conclusion to Chapter 5

- ✔ The Scrum Methods associated two winning attitude of Software engineering field: inspection+communication. The daily scrum meeting are enhancing communication and putting forward the necessity to exchange ideas and motivation between team member on a daily basis.
- ✔ The SCRUM method has also a strong emphasis on code inspection, and more generally all product artifacts inspection. The daily SCRUM meetings are made to present shortly and briefly artifacts and that is a concrete implementation of one of the most efficient toward bugs, design flaws and specification disambiguation !

Conclusion to Chapter 5

- ✔ The Scrum Methods associated two winning attitude of Software engineering field: inspection+communication. The daily scrum meeting are enhancing communication and putting forward the necessity to exchange ideas and motivation between team member on a daily basis.
- ✔ The SCRUM method has also a strong emphasis on code inspection, and more generally all product artifacts inspection. The daily SCRUM meetings are made to present shortly and briefly artifacts and that is a concrete implementation of one of the most efficient toward bugs, design flaws and specification disambiguation !

Metrics for OO Design

Whitmire [WHI97] describes nine distinct and measurable characteristics of an OO design:

1. Size - Size is defined in terms of
   - Volume – number of

     Database references or

     Transactions

     Database updates, etc
   - Length – lines of code, number of classes, number of instances, tec
   - Functionality – using function point analysis or use case point analysis

Metrics for OO Design

Whitmire [WHI97] describes nine distinct and measurable characteristics of an OO design:

2. Complexity

   How classes of an OO design are interrelated to one another

   Halstead Complexity

   McCabes Cyclomatic Complexity

3. Coupling

   The physical connections between elements of the OO design

   - Component coupling (packages), Class coupling, data coupling

Metrics for OO Design

Whitmire [WHI97] describes nine distinct and measurable characteristics of an OO design:

4. Sufficiency

   “the degree to which an abstraction possesses the features required of it, or the degree to which a design component possesses features in its abstraction, from the point of view of the current application.”

Metrics for OO Design

Whitmire [WHI97] describes nine distinct and measurable characteristics of an OO design:

5. Completeness

   An indirect implication about the degree to which the abstraction or design component can be reused

   - Degree of reuse, degree of package, class, method independence.

Metrics for OO Design-II

6. Cohesion

   The degree to which all operations working together to achieve a single, well-defined purpose

7. Primitiveness

   Applied to both operations and classes, the degree to which an operation is atomic

8. Similarity

   The degree to which two or more classes are similar in terms of their structure, function, behavior, or purpose

9. Volatility

   Measures the likelihood that a change will occur

Class-Oriented Metrics

Proposed by Chidamber and Kemerer:

weighted methods per class (WMC)
depth of the inheritance tree (DIT)
number of children (NOC)
coupling between object classes
response for a class (RPC)
lack of cohesion in methods (LCOM)

Class-Oriented Metrics

Proposed by Lorenz and Kidd [LOR94]:

- class size (LOC)
- number of operations overridden by a subclass
- number of operations added by a subclass

Class-Oriented Metrics

The MOOD Metrics Suite

Method inheritance factor

Coupling factor

Polymorphism factor

Operation-Oriented Metrics

Proposed by Lorenz and Kidd [LOR94]:

- average operation size (method LOC)
- operation complexity (method)
- average number of parameters per operation

Component-Level Design Metrics

Cohesion metrics: a function of data objects and the locus of their definition

Coupling metrics: a function of input and output parameters, global variables, and modules called

Complexity metrics: hundreds have been proposed (e.g., cyclomatic complexity)

Code Metrics

Halstead's Software Science: a comprehensive collection of metrics all predicated on the number (count and occurrence) of operators and operands within a component or program

It should be noted that Halstead's "laws" have generated substantial controversy, and many believe that the underlying theory has flaws. However, experimental verification for selected programming languages has been performed (e.g. [FEL89]).

Metrics for Testing

Testing effort can also be estimated using metrics derived from Halstead measures

Binder [BIN94] suggests a broad array of design metrics that have a direct influence on the "testability" of an OO system.

- Lack of cohesion in methods (LCOM).
- Percent public and protected (PAP).
- Public access to data members (PAD).
- Number of root classes (NOR).
- Fan-in (FIN).
- Number of children (NOC) and depth of the inheritance tree (DIT).

Metrics for Design

- McCabe's Cyclomatic Complexity
  - Measures the number of linearly independent paths within code
  - Defined as number of decision points + 1
    - where decision points are conditional statements such as *if/else* or *while*

Metrics for Design

McCabe's Cyclomatic Complexity

```
lettergrade = "F";
if (average >= 90)
        lettergrade = "A";
else if (average >= 80)
        lettergrade = "B";
else if (average >= 70)
        lettergrade = "C";
else
        lettergrade = "D";
```

Metrics for Design

McCabe's Cyclomatic Scale

Number of Paths	Code Complexity	Risk
1-10	Not very complex	Low
11-20	Moderately complex	Moderate
21-50	Highly complex	High
51+	Unstable	Very high

Metrics for Design

- Cohesion and Coupling
  - Widely accepted measures of the quality of the design

Cohesion

- Measure of degree of interaction within a module
- Measure of the strength of association of the elements inside a module
- Functionality inside a module should be so related that anyone can easily see what the module does
- Goal is a highly cohesive module

Cohesion

- For structured design
  - Deals with the cohesion of the actions in a module (unit) to perform one and only one task
- For object-oriented methods
  - Deals with the ability of a module to produce only one output for one module

Levels of Cohesion in Structured Design

Functional cohesion **(Good)**

Sequential cohesion

Communicational cohesion

Procedural cohesion

Temporal cohesion

Logical cohesion

Coincidental cohesion (Bad)

Comparison of Cohesion Levels for Structured Design

Cohesion Level	Cleanliness of Implementation	Reusability	Modifiability	Understand-ability
Functional	Good	Good	Good	Good
Sequential	Good	Medium	Good	Good
Communicational	Good	Poor	Medium	Medium
Procedural	Medium	Poor	Variable	Variable
Temporal	Medium	Bad	Medium	Medium
Logical	Bad	Bad	Bad	Poor
Coincidental	Poor	Bad	Bad	Bad

Levels of Cohesion for Object-oriented Methods

Functional cohesion **(Good)**

Sequential cohesion

Communicational cohesion

Iterative cohesion

Conditional cohesion

Coincidental cohesion (Bad)

Functional Cohesion in Structured Design

IN STRUCTURED DESIGN

A module performs exactly one action or achieves a single goal

IN OO DESIGN

Only one output exists for the module

Ideal for object-oriented paradigm

Functional Cohesion for Object-oriented Methods

```
public void deposit (double amount)
{
    balance = balance + amount;
}
```

Sequential Cohesion in Structured Design

STRUCTURED DESIGN

Outputs of one module serve as input data for the next module

OBJECT ORIENTED DESIGN

One output is dependent on the other output

Modifications result in changing only one instance variable

Sequential Cohesion for Object-oriented Methods

```
public double withdraw (double amount, double fee)
{
    amount = amount + fee;
    if (amount < 0)
     System.out.println (“Error: withdraw amount is invalid.”);
    else if (amount > balance)
     System.out.println (“Error: Insufficient funds.”);
    else
     balance = balance – amount;
    return balance;
}
```

Communicational Cohesion in Structured Design

STRUCTURED DESIGN

Various functions within a module perform activities on the same data

OBJECT ORIENTED DESIGN

Two outputs are iteratively dependent on the same input

Communicational Cohesion for Object-oriented Methods

```
public void addCD (String title, String artist,
                   double cost, int tracks)
{
    if (count = = collection.length)
     increaseSize ( );
    collection [count] = new CD (title, artist, cost, tracks);
    totalCosts = totalCosts + cost;
    count++;
}
```

Iterative Cohesion for Object-oriented Methods

Two outputs are iteratively dependent on the same input

Iterative Cohesion for Object-oriented Methods

```
void formDet (float Equations[2][3], float x[2][2],
                    float y[2][2], float D[2][2])
{
      for (int Row = 0; Row < 2; ++Row)
        for (int Col = 0; Col < 2; ++Col)
        {
              x[Row][Col] = Equations[Row][Col];
              y[Row][Col] = Equations[Row][Col];
              D[Row][Col] = Equations[Row][Col];
        }
      x[0][0] = Equations[0][2];
      x[1][0] = Equations[1][2];
      y[0][1] = Equations[0][2];
      y[1][1] = Equations[1][2];
}
```

Conditional Cohesion for Object-oriented Methods

Two outputs are conditionally dependent on the same input

Conditional Cohesion for Object-oriented Methods

```
public boolean checkBookIn ( )
{
      if (this.isAvailable ( ))
      { //this object cannot be checked out
        System.out.println (“Error: “ + callNumber +
                            “ is not checked out”);
        return false;
      }
      else
      {
        dueDate = null;
        availability = true;
        return true;
      }
}
```

Coincidental Cohesion for Object-oriented Methods

Two outputs have no dependence relationship with each other and no dependence relation on a common input

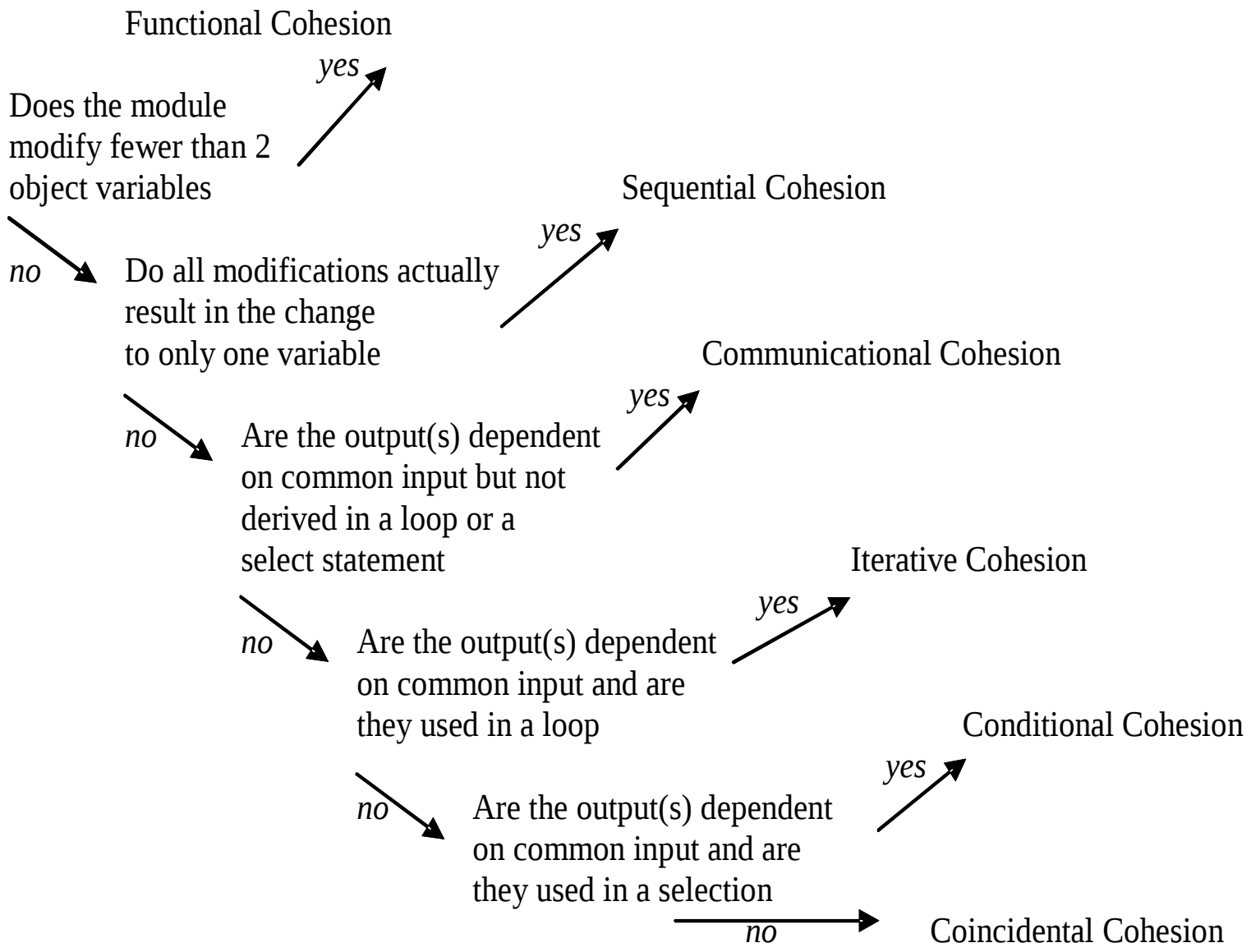
Coincidental Cohesion for Object-oriented Methods

```
public void readInput ( )
{
    System.out.println (“Enter name of item being purchased: “);
    name = MyInput.readLine ( );
    System.out.println (“Enter price of item: “);
    price = MyInput.readLineDouble ( );
    System.out.println (“Enter number of items purchased: “);
    numberBought = MyInput.readLineInt ( );
}
```

Coincidental Cohesion for Object-oriented Methods

```
public String AcceptItemName ( )
{
    System.out.println ("Enter name of item being purchased: ");
    name = MyInput.readLine ( );
    return name;
}
```

Cohesion Decision Tree for Object-oriented Methods



Coupling

Measure of degree of interaction between two modules

Measure of interdependence of modules

Goal is to have so little coupling that changes can be made within one module without disrupting other modules

Levels of Coupling for Object-oriented Methods

No coupling	(Good)
Sequential coupling	
Computational coupling	
Conditional coupling	
Common coupling	
Content coupling	(Bad)

No Coupling for Object-oriented Methods

No global data sharing or functional calls between two modules

Sequential Coupling for Object-oriented Methods

Two modules exist where the outputs of one module are the inputs of another

Computational Coupling for Object-oriented Methods

Two modules exist where one module passes a parameter to another module and the parameter has control or data dependence on the output

Conditional Coupling for Object-oriented Methods

One module passes a parameter to another module and the parameter has control dependence on an output

Common Coupling for Object-oriented Methods

One module writes to the global data and another module reads from the global data

Content Coupling for Object-oriented Methods

One module references the contents of another module

Coupling Decision Tree for Object-oriented Methods