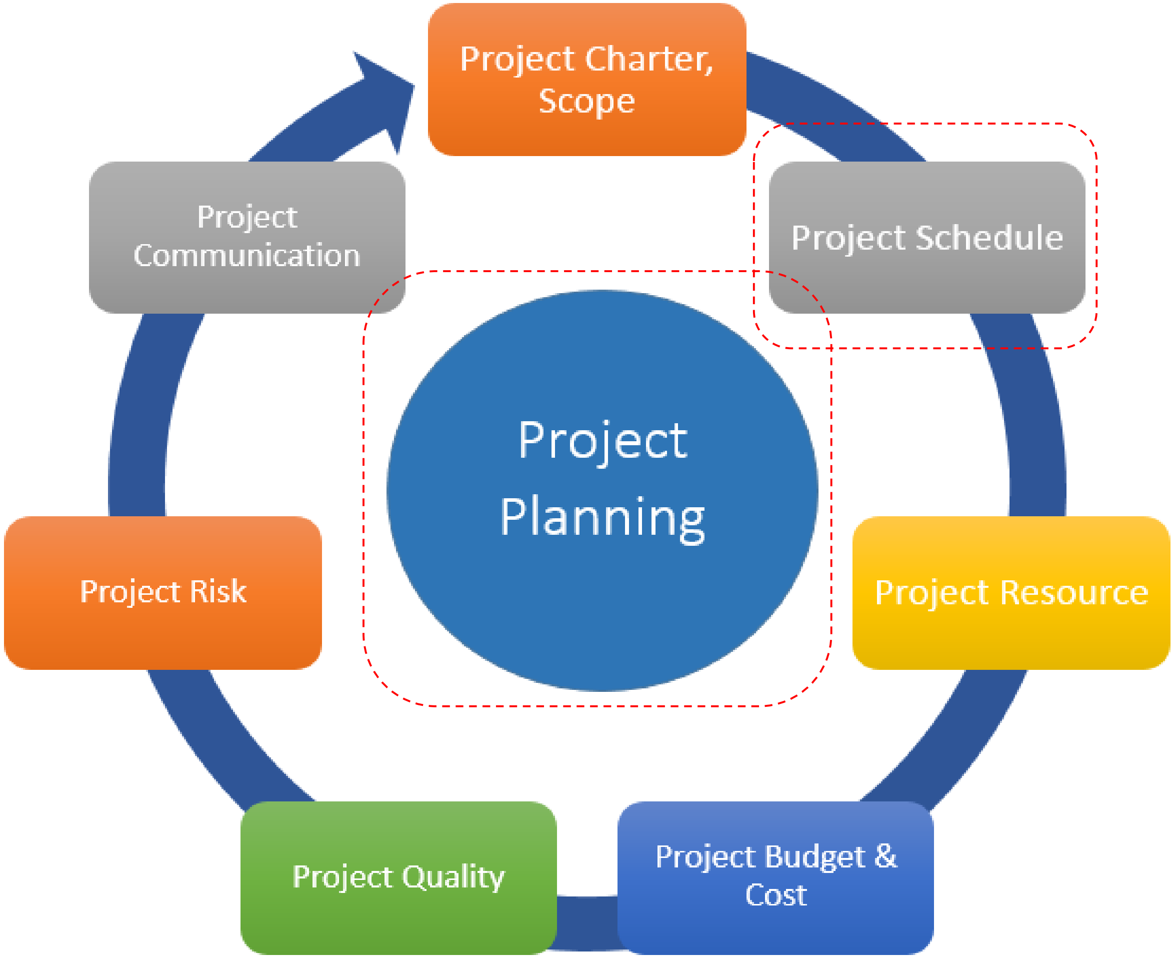
SWEN90016
Software Processes & Project Management

Cost Estimation

Shanika Karunasekera
Department of Computing and Information Systems
The University of Melbourne
karus@unimelb.edu.au

2019– Semester 2
Lecture 6

RECAP – Project Scheduling



RECAP – Project Scheduling

1. Understand the role of a project schedule

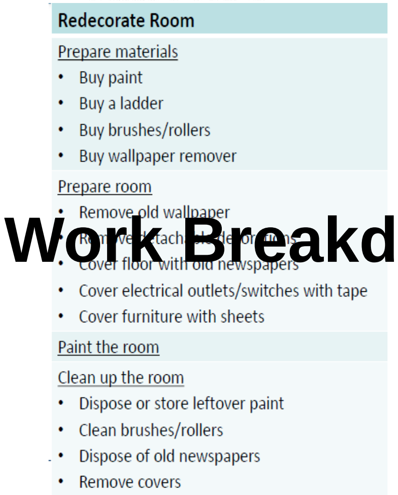
2. Understand how to develop a project schedule

3. Understand how to use a project schedule to monitor and track project progress

4. Understand agile planning principles

RECAP – Project Scheduling

Work Breakdown Structure



Redecorate Room

Prepare materials

- Buy paint
- Buy a ladder
- Buy brushes/rollers
- Buy wallpaper remover

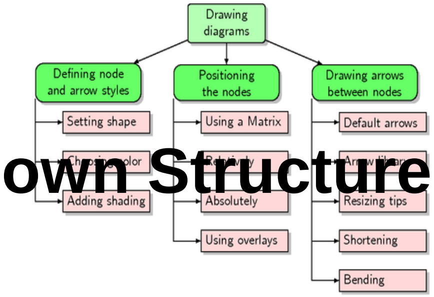
Prepare room

- Remove old wallpaper
- Remove detachable decorations
- Cover floor with old newspapers
- Cover electrical outlets/switches with tape
- Cover furniture with sheets

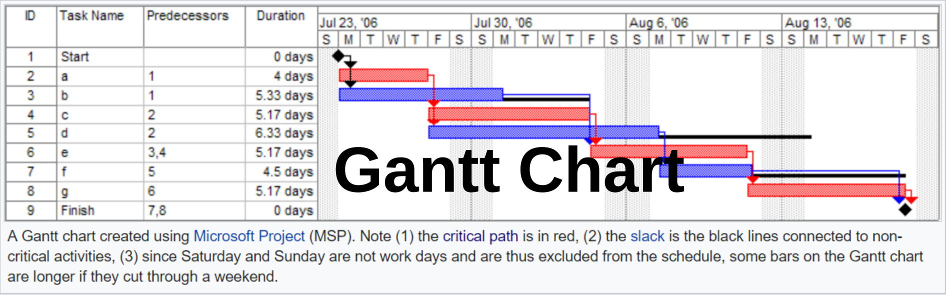
Paint the room

Clean up the room

- Dispose or store leftover paint
- Clean brushes/rollers
- Dispose of old newspapers
- Remove covers



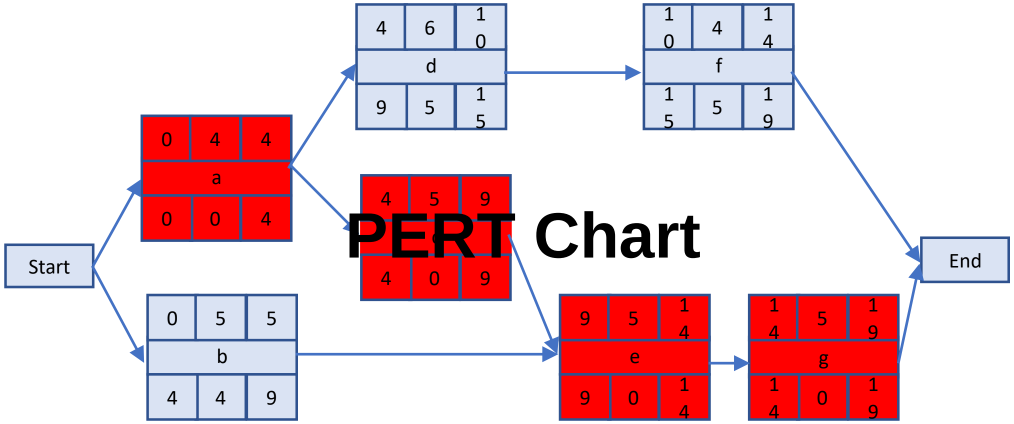
Gantt Chart



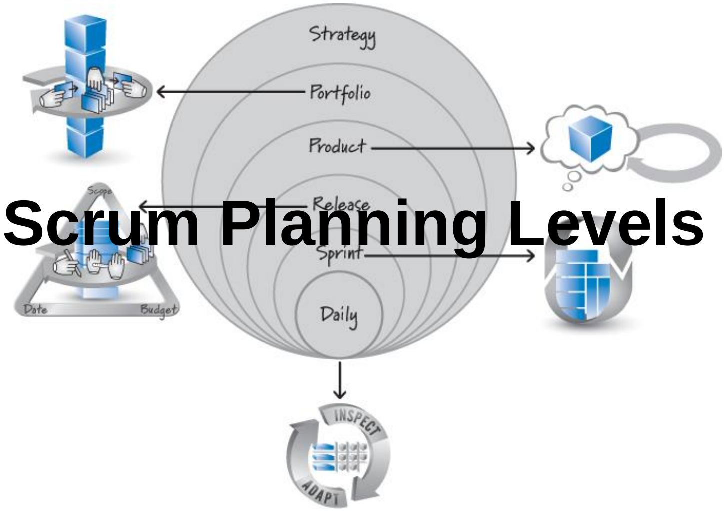
ID	Task Name	Predecessors	Duration
1	Start		0 days
2	a	1	4 days
3	b	1	5.33 days
4	c	2	5.17 days
5	d	2	6.33 days
6	e	3,4	5.17 days
7	f	5	4.5 days
8	g	6	5.17 days
9	Finish	7,8	0 days

A Gantt chart created using Microsoft Project (MSP). Note (1) the critical path is in red, (2) the slack is the black lines connected to non-critical activities, (3) since Saturday and Sunday are not work days and are thus excluded from the schedule, some bars on the Gantt chart are longer if they cut through a weekend.

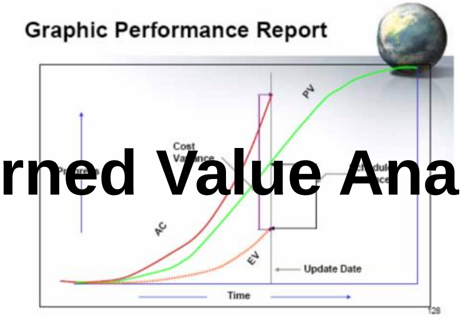
PERT Chart

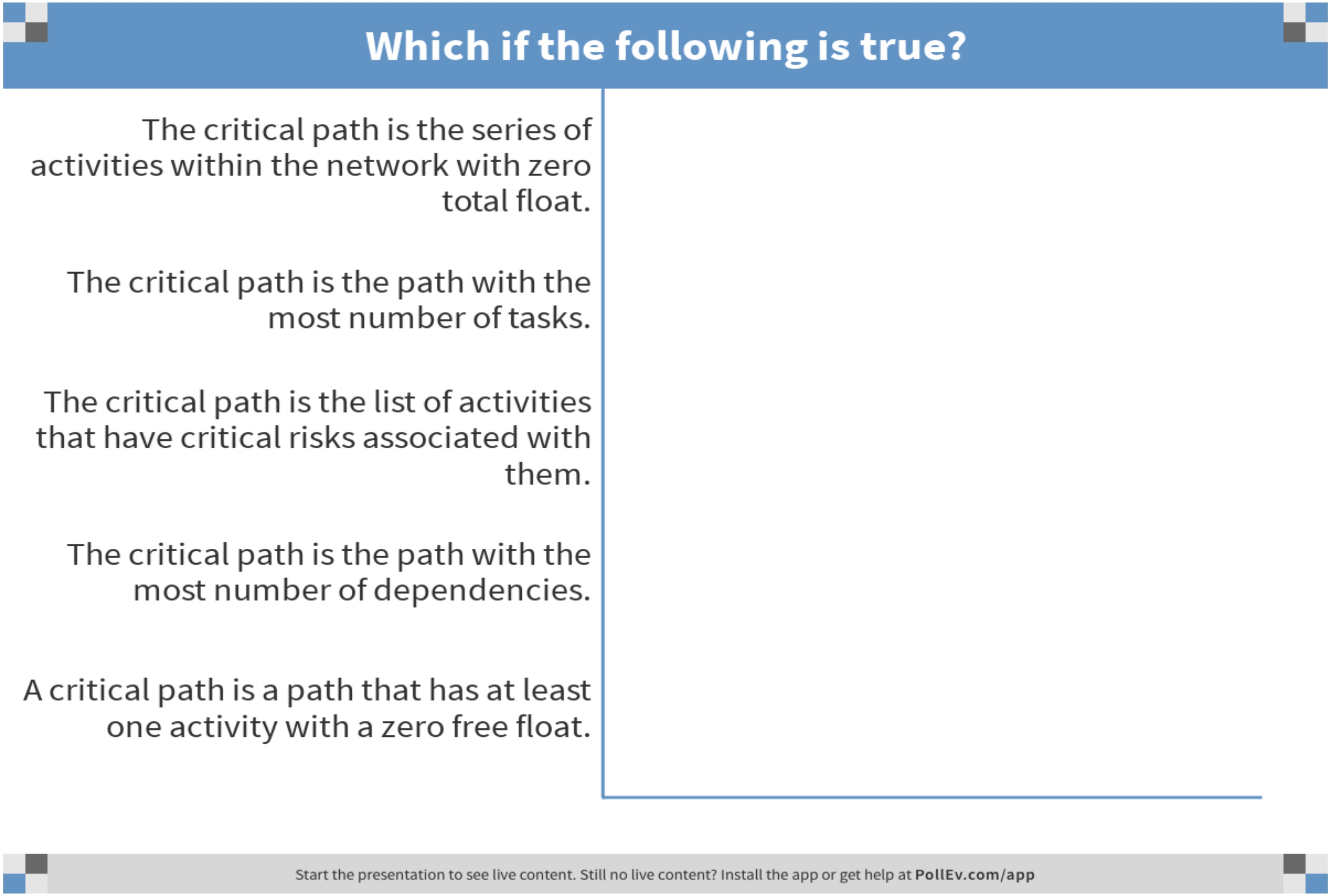


Scrum Planning Levels



Earned Value Analysis





Which if the following is true?

The critical path is the series of activities within the network with zero total float.

The critical path is the path with the most number of tasks.

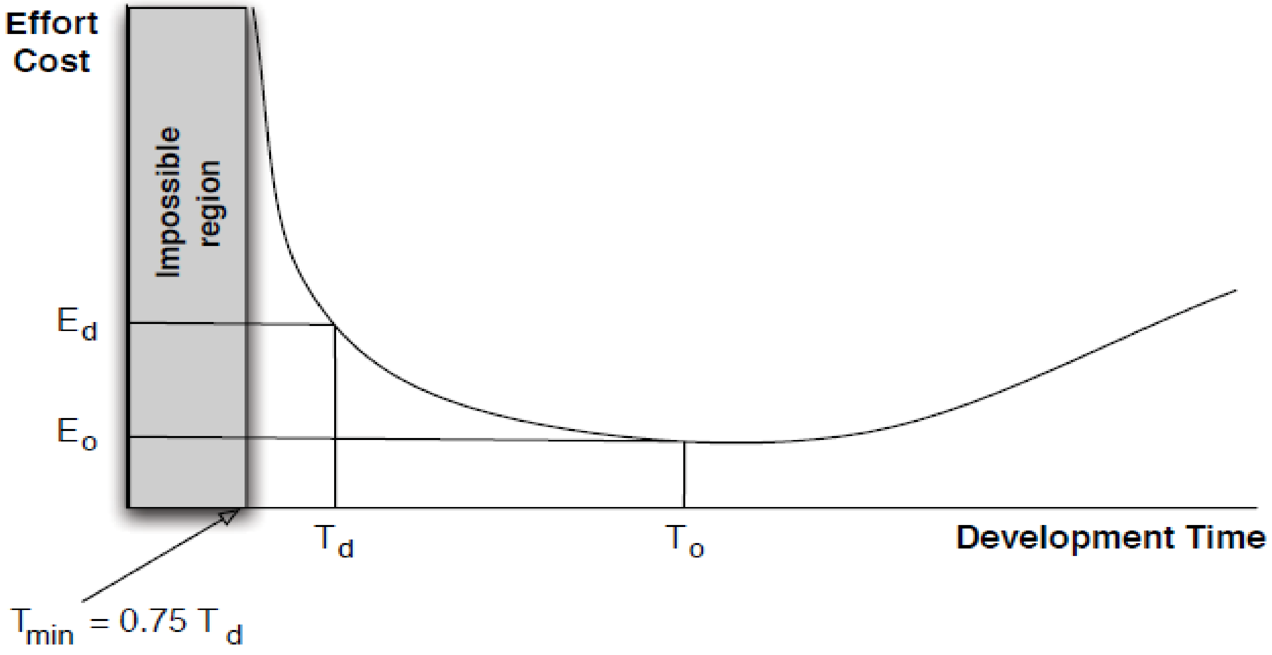
The critical path is the list of activities that have critical risks associated with them.

The critical path is the path with the most number of dependencies.

A critical path is a path that has at least one activity with a zero free float.

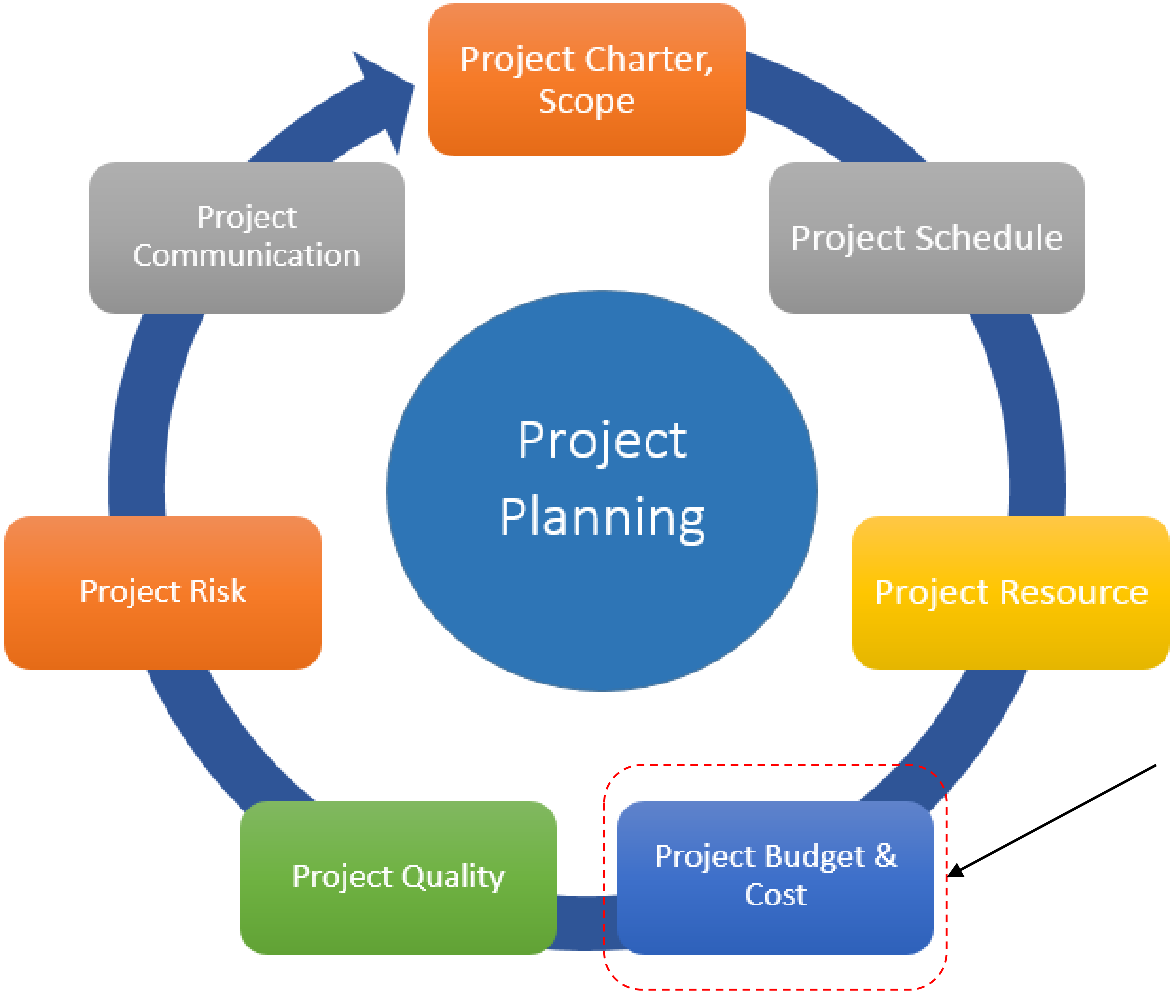
Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**

RECAP – Project Scheduling



Putnam-Norden-Rayleigh curve

Topic this week – Cost Estimation

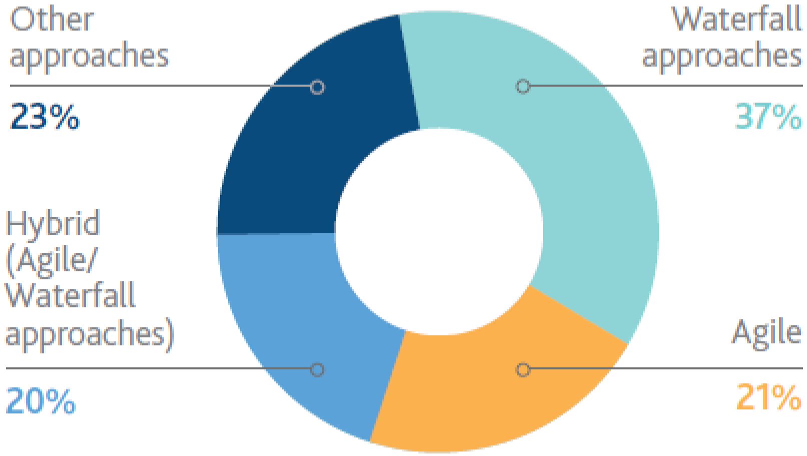


Today's lecture

Semester Structure

Week #	Lecture Week Start	Old Arts Public Lecture Theatre Friday 3.15pm to 5.15pm	Assignment
1	29/07/19	Subject Introduction, Introduction to Projects and Project Management	
2	05/08/19	Project Management Plan & SDLC's	Assignment 1 Spec available on LMS 05/08
3	12/08/19	SDLC - Agile Scrum – continued Individuals, Motivation and Teams	
4	19/08/19	Stakeholder Management Communication Management	Assignment 2 available & Groups created during the workshops / tutorials – attendance mandatory
5	26/08/19	Project Planning and Scheduling	Assignment 1 (Individual) due Fri 31/08 @ 11.59 pm
6	02/09/19	Cost Estimation	
7	09/09/19	Risk Management	
8	16/09/19	Quality Management/Configuration Management	Assignment 2 (Part 1) due Wed 18/09 @ 11.59 pm
9	***23/09/19***	***University Holiday***	
	30/09/19	***Non Teaching Week – Mid semester break***	Assignment 2 (Part 2) due Sat 28/09 @ 11.59 pm
10	07/10/19	Ethics, Outsourcing & Procurement	Assignment 2 (Part 3) due Sat 12/10 @ 11.59 pm
11	14/10/19	Guest Lecture	Assignment 2 (Final) due Sat 19/10 @ 11.59 pm
12	21/10/19	Subject Revision and Exam Prep	Assignment 2 Project Demonstration during tutorials

Q: In your estimation, what percentage of the projects completed within your organization in the past 12 months has used the following types of approaches?



Note: Numbers may not sum to 100% due to rounding

PMI's PULSE of the PROFESSION -https://www.pmi.org/learning/thought-leadership/pulse/pulse-of-the-profession-2017

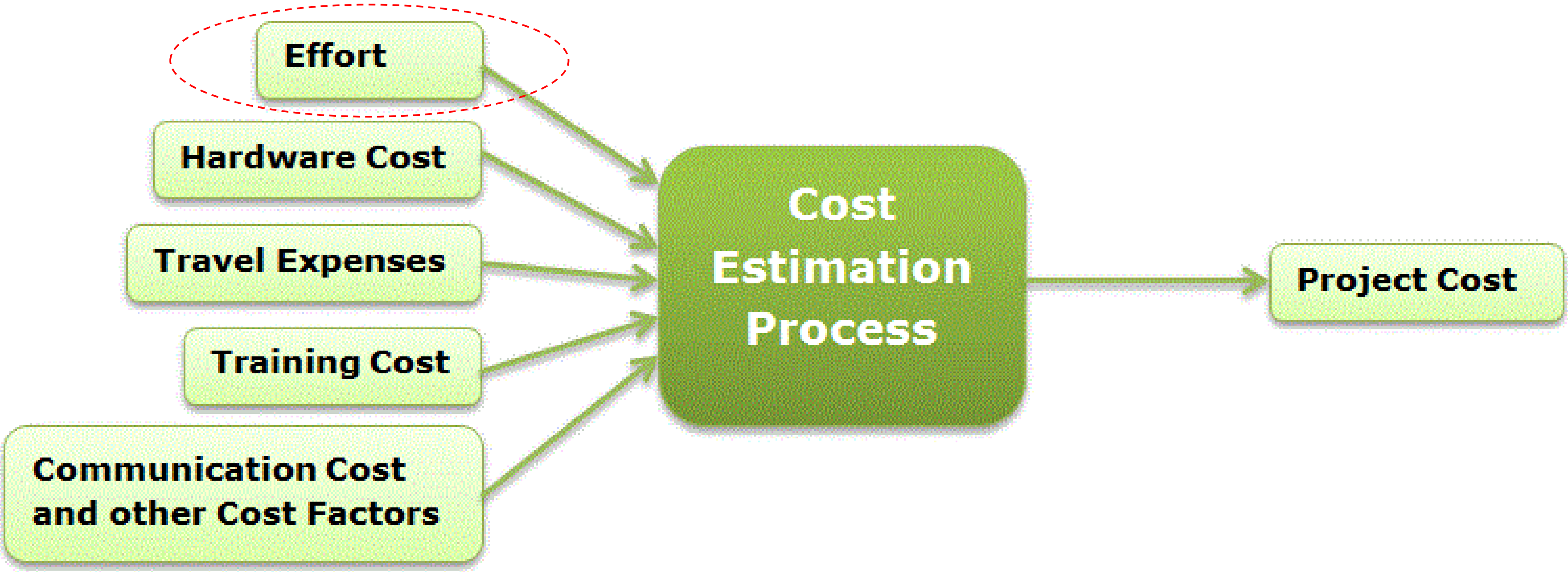
Learning Outcomes

1. Understand the importance of cost estimation and the challenges involved
2. Understand the techniques used for cost estimation
3. Understand software size estimation techniques
4. Understand the principles of the COCOMO II model for algorithmic cost estimation
5. Understand cost estimation techniques used in Agile software development lifecycles

Learning Outcomes

1. Understand the importance of cost estimation and the challenges involved
2. Understand the techniques used for cost estimation
3. Understand software size estimation techniques
4. Understand the principles of the COCOMO II model for algorithmic cost estimation
5. Understand cost estimation techniques used in Agile software development lifecycles

What determines project cost?



http://geethanjalisnsce.blogspot.com.au/2015/04/

What is software cost estimation?

- What is estimation?
  - Is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if *input data may be incomplete, uncertain, or unstable*

- What is *software cost estimation*?
  - Estimation of how much *money, effort, resources, and time* will take to build a specific software based system or product

- Why is it Important?
  - Would you build a house without knowing how much you were about to spend - of course not
  - Since most software systems cost considerably more to build than a large house, it would seem reasonable to develop an estimate before you start creating the software

Challenges in cost estimation

- There is no exact science for cost estimation – it will never be considered as all accurate
- No person can reasonably predict what can go wrong in the project
- Most estimation methods assume things will proceed as expected and simply adds some slack to account for what can go wrong

Possible solutions

1. Delay estimation – 100% accuracy at the end of the project but less useful!

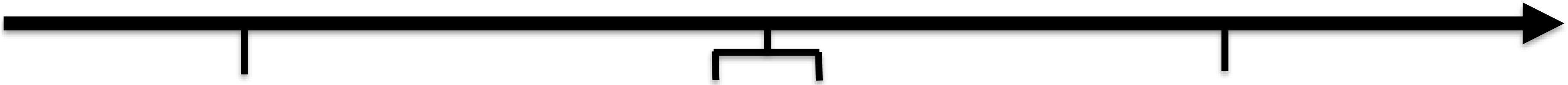
2. Base estimation on data from previous projects that have been completed

3. Break the system to smaller parts and generate the estimates for smaller parts, which is easier

4. Use empirically-based estimation methods

Estimation – current status

Why do we need to improve effort estimation?



2003

Effort estimation by experts' opinion was found as the most used method across the IT industry[4]

Cost and schedule overrun were very common (30-40%), and **estimation accuracy** was perceived as a problem by project managers [4].

2013 2014

Expert-based estimation method still remains popular in Agile methodologies (2013[5], 2014[6])

2017

71% of organizations adopted Agile approaches

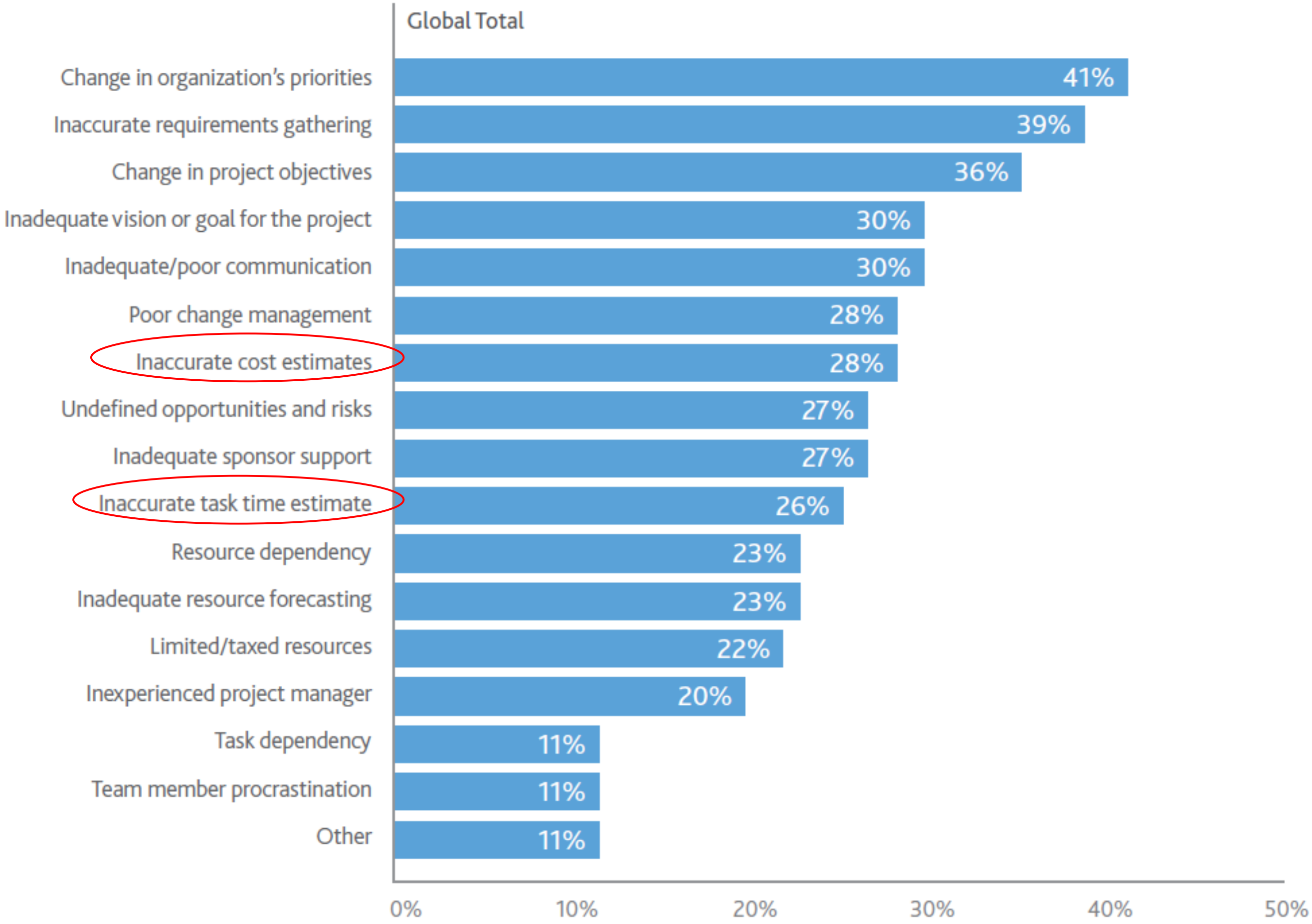
9.7% of investment lost due to poor project performance

Many projects finished beyond their initially scheduled time(49%) and budget(43%)

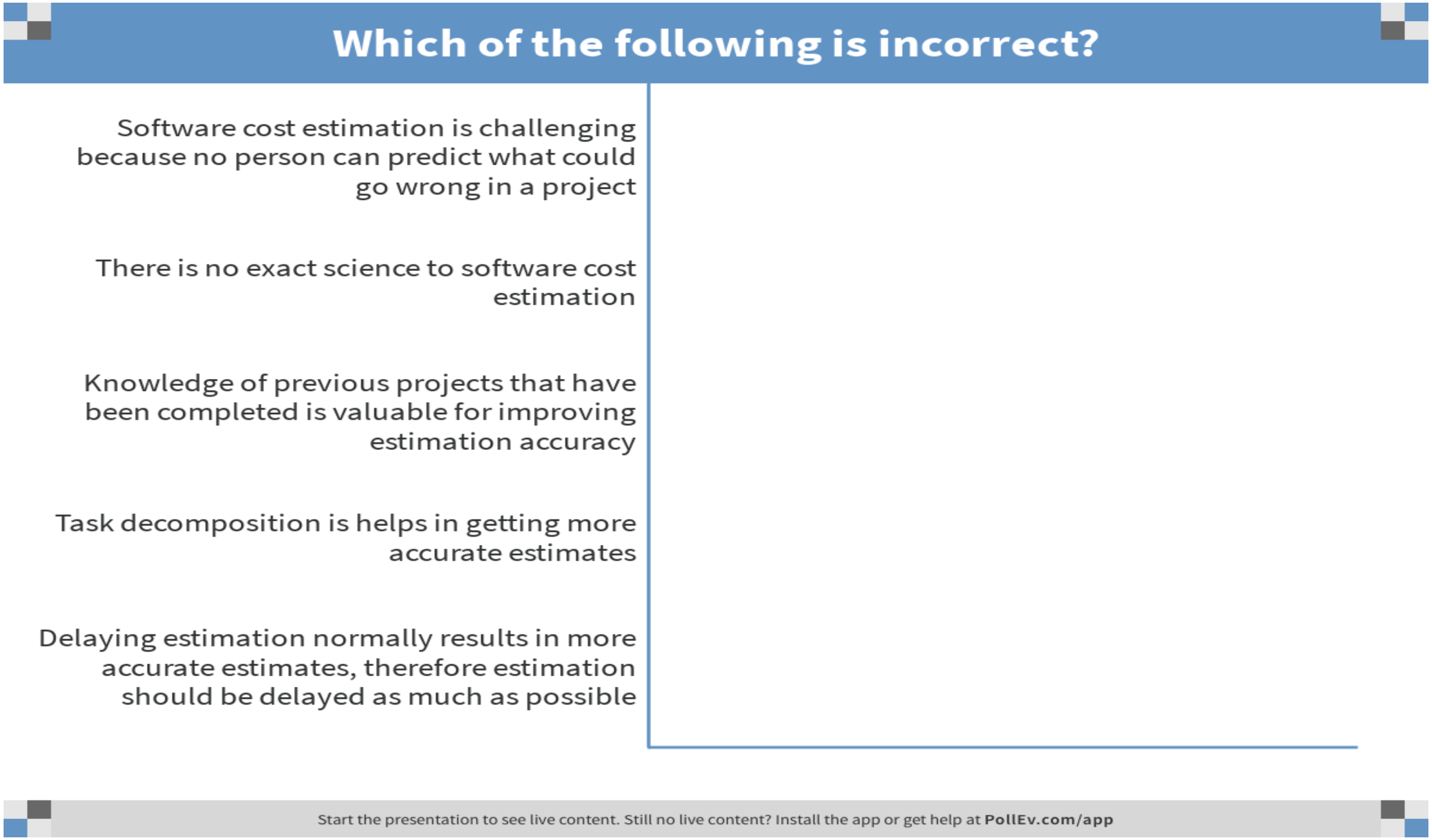
inaccurate cost and task time estimation still highly-ranked among failure reasons (28% and 26%)

Source: Literature Review PhD Student Jirat Pasuksmit

Q: Of the projects started in your organization in the past 12 months that were deemed failures, what were the primary causes of those failures? (Select up to three.)



PMI's PULSE of the PROFESSION -https://www.pmi.org/learning/thought-leadership/pulse/pulse-of-the-profession-2017



Which of the following is incorrect?

Software cost estimation is challenging because no person can predict what could go wrong in a project

There is no exact science to software cost estimation

Knowledge of previous projects that have been completed is valuable for improving estimation accuracy

Task decomposition is helps in getting more accurate estimates

Delaying estimation normally results in more accurate estimates, therefore estimation should be delayed as much as possible

Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**

Learning Outcomes

1. Understand the importance of cost estimation and the challenges involved
2. Understand the techniques used for cost estimation
3. Understand a range of software size estimation techniques
4. Understand the principles of the COCOMO II model for algorithmic cost estimation
5. Understand cost estimation techniques used in Agile software development lifecycles

Techniques for cost estimation

1. Expert judgement
   - Several experts on the proposed software development technique and the application domain estimate project cost. These are then discussed, compared and adjusted until consensus is reached
   - Some expert judgement techniques involve polling each expert independently, in some cases for three estimates, pessimistic estimate (p), optimistic estimate (o) and the most likely estimate (m), and the expert's estimate is computed as the:

$$e = (p + 4m + o)/6$$

   - *Delphi technique:* asks several experts to make an individual judgement of the effort using any method they wish. Then, the average effort is calculated, and presented to all of the experts. Each expert is then given a chance to revise their estimate, in some cases after a discussion between all experts. This continues until no expert wishes to revise their estimate.

Techniques for cost estimation

2. Estimation by Analogy

- The cost of a new project is estimated based on similar projects in the same application domain

3. Parkinson's Law

- This law states that the work will expand to fill the time available
- The cost is determined by available resources rather than by objective assessment
- For example, if the software is to be delivered in 12 months, and 3 people are available, the effort is 36 person months

4. Pricing to win

- The cost is estimated to be whatever the customer has available to spend on the project - cost depends on the budget not on the software functionality

Techniques for cost estimation

5. Algorithmic cost modelling

- A ***model*** is developed using ***historical cost information*** based on some ***software metric (usually its size)*** to the project cost
- When a project effort needs to be estimated, an estimate of the metric is computed
- Using the model, the effort is predicted
- The most general form of an algorithm cost estimate is given by:

$$Effort = A \times Size^{B} \times M$$

A - a constant factor that depends on the organizational practices

$Size$ - size of the software estimated in a metric of choice (e.g. lines of code, function point, use case points)

B - a value between 1 and 1.5 derived experimentally

M − a multiplier made by combining process, product and developmet attributes such as stability of requirement, experience of the team

Techniques for cost estimation

5. Algorithmic cost modelling - cont…

Basic steps in algorithmic cost estimation

1. Estimate the *size* of the development product
2. Estimate the *effort in person-months* or person-hours
3. Estimate the *schedule in calendar months*
4. Estimate the *project cost in agreed currency*

Learning Outcomes

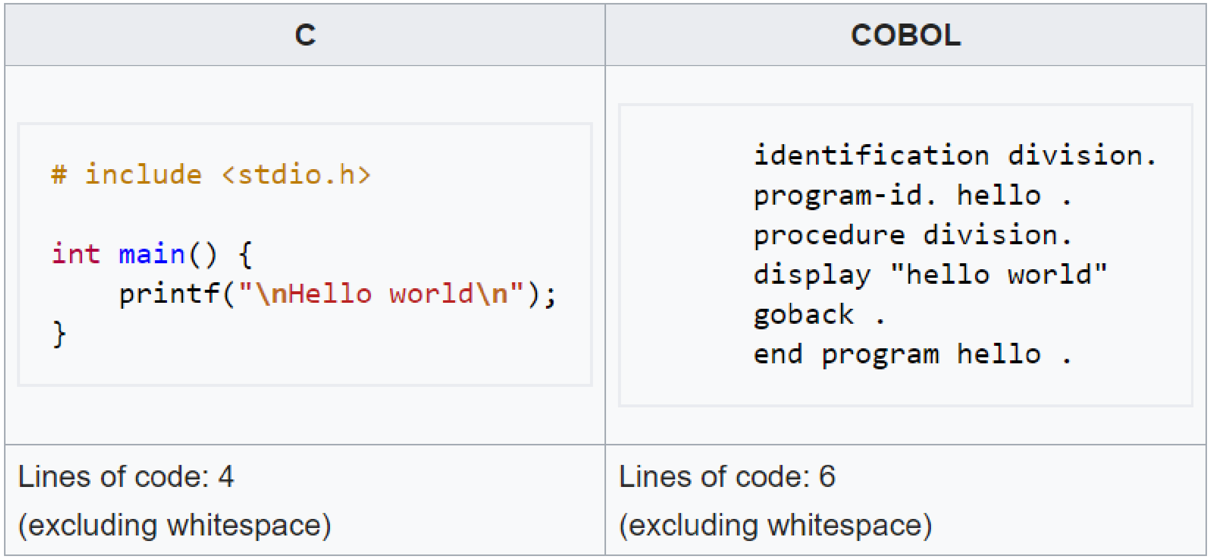
1. Understand the importance of cost estimation and the challenges involved
2. Understand the techniques used for cost estimation
3. Understand software size estimation techniques
4. Understand the principles of the COCOMO II model for algorithmic cost estimation
5. Understand cost estimation techniques used in Agile software development lifecycles

Software Size Estimation

- Commonly used metric for software size estimation
  - Source Lines of Code (SLOC)
    - Based on code
  - Function Points (FP)
    - Based on the Requirements Specification
  - Use-case Points (UCP)
    - Based on Use Cases

Source Lines of Code (SLOC)

- There are two types of SLOC:
  - **Physical SLOC:** Count the number of lines excluding comments and blank lines
  - **Logical SLOC:** Measure the number of executable "statements", but their specific definitions are tied to specific computer languages



C	COBOL

# include <stdio.h> 	
 	
int main() {
    printf("\nHello world\n");
}
	identification division.
	program-id. hello .
	procedure division.
	display "hello world"
	goback .
	end program hello .

Lines of code: 4 (excluding whitespace)	Lines of code: 6 (excluding whitespace)

Source Lines of Code (SLOC)

- Advantages of SLOC:
  - Scope for Automation of Counting: Since Lines of Code is a physical entity it is easy to count and can be automated using a tool
  - An Intuitive Metric: Lines of Code serves as an intuitive metric for measuring the size of software because it can be seen and the effect of it can be visualized

Source Lines of Code (SLOC)

- Disadvantages of SLOC:
  - Variability: Depends on programmer experience, programming language, framework support (auto generated code), reuse, etc.
  - It is difficult to estimate the number of lines of code that will be needed to develop a system from information that is available in analysis and design phases
  - Lack of a universally accepted definition for exactly what a line of code is

Function Points (FP)

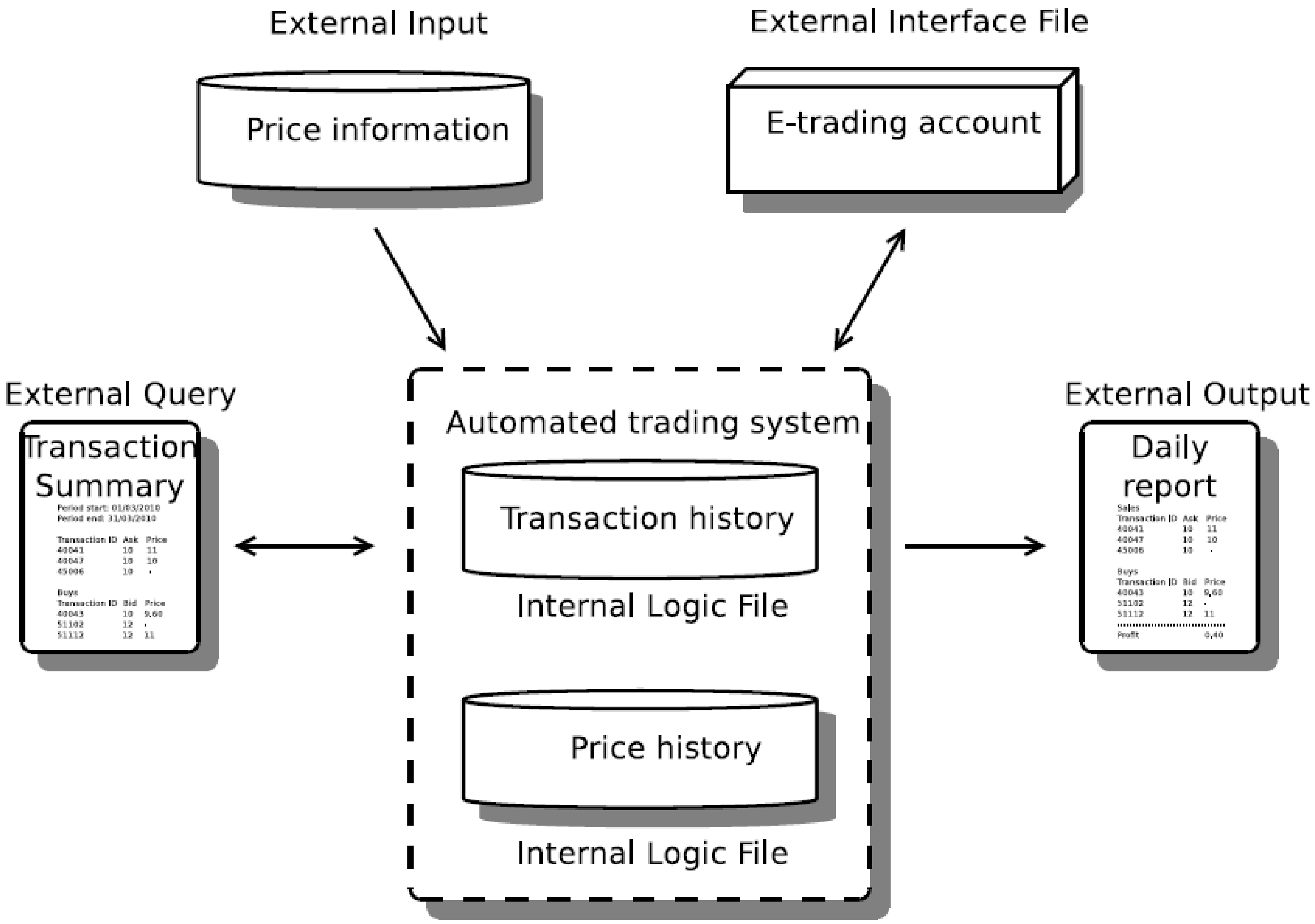
- Is used to express the *amount of functionality* in a software system, as seen by the user
- A *higher number of function points* indicates *more functionality*
  - Empirical evidence demonstrates that there is a *positive correlation between function points and the complexity* of the system
- Typically used to:
  - Estimate the cost and effort required to design, code and test a software system
  - Predict the number of errors
  - Predict the number of components
  - Measure productivity
- Function points are computed from the *Software Requirements Specification (SRS)*

Software Requirements

- **Software Requirements Specification (SRS)**
  - A document that specifies what is expected of a software system; referred to as the requirements of the system
  - It contains:
    - Functional Requirements:
      - Specify the functions that are required in the system
    - Non-functional Requirements:
      - Specify requirements that are not directly functions, such as performance, reliability, scalability etc. (quality requirements)

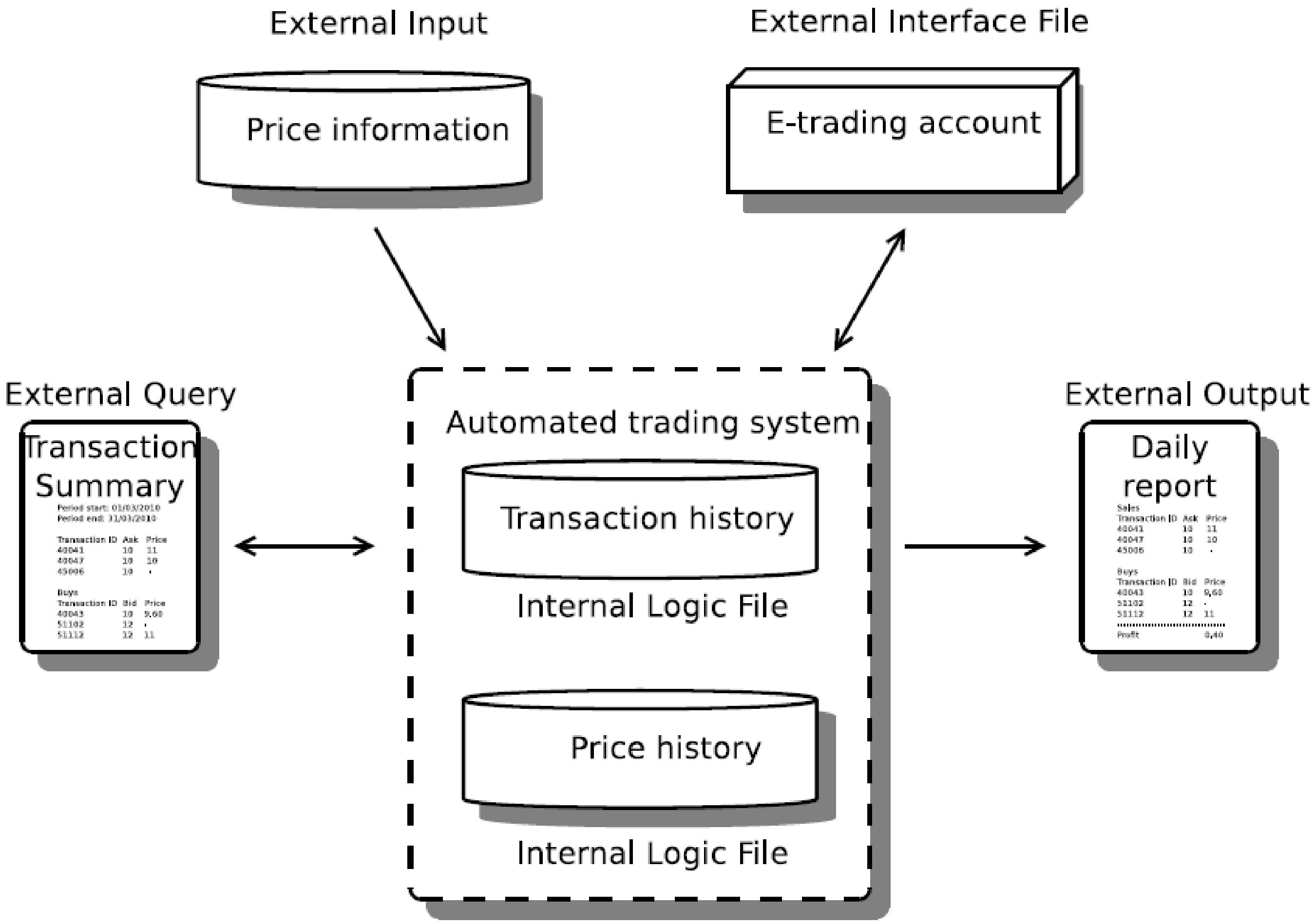
Example: Functional Requirements

Automated Trading System



Example: Functional Requirements

Automated Trading System



Period start: 01/03/2010
Period end: 31/03/2010

Transaction ID	Ask	Price
40041	10	11
40047	10	10
45006	10	.

Buys

Transaction ID	Bid	Price
40043	10	9.60
51102	12	.
51112	12	11

Sales

Transaction ID	Ask	Price
40041	10	11
40047	10	10
45006	10	.

Buys

Transaction ID	Bid	Price
40043	10	9,60
51102	12	.
51112	12	11

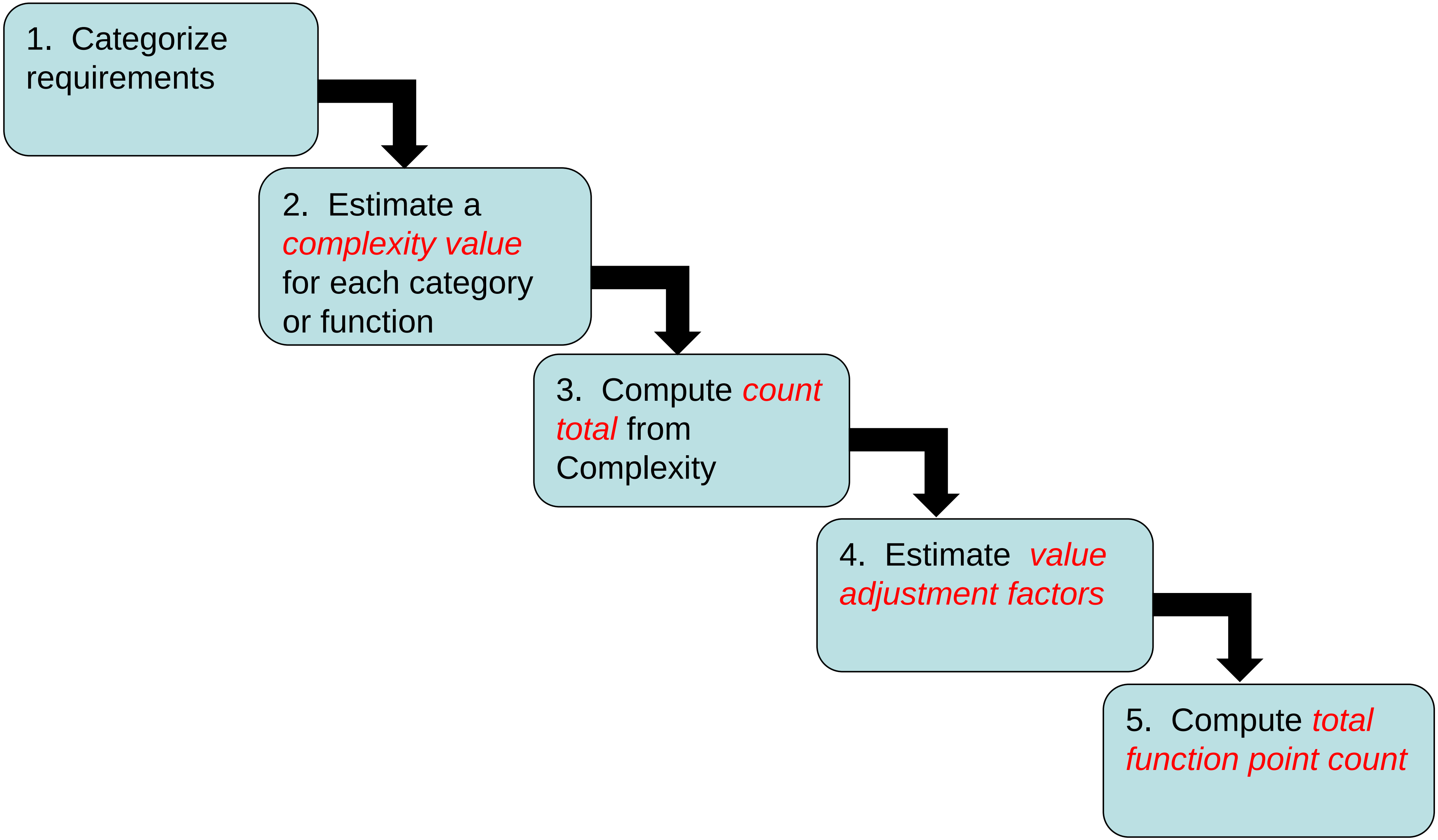
Profit 0,40

Example: Automated Trading System

Automated Trading System: Functional Requirements

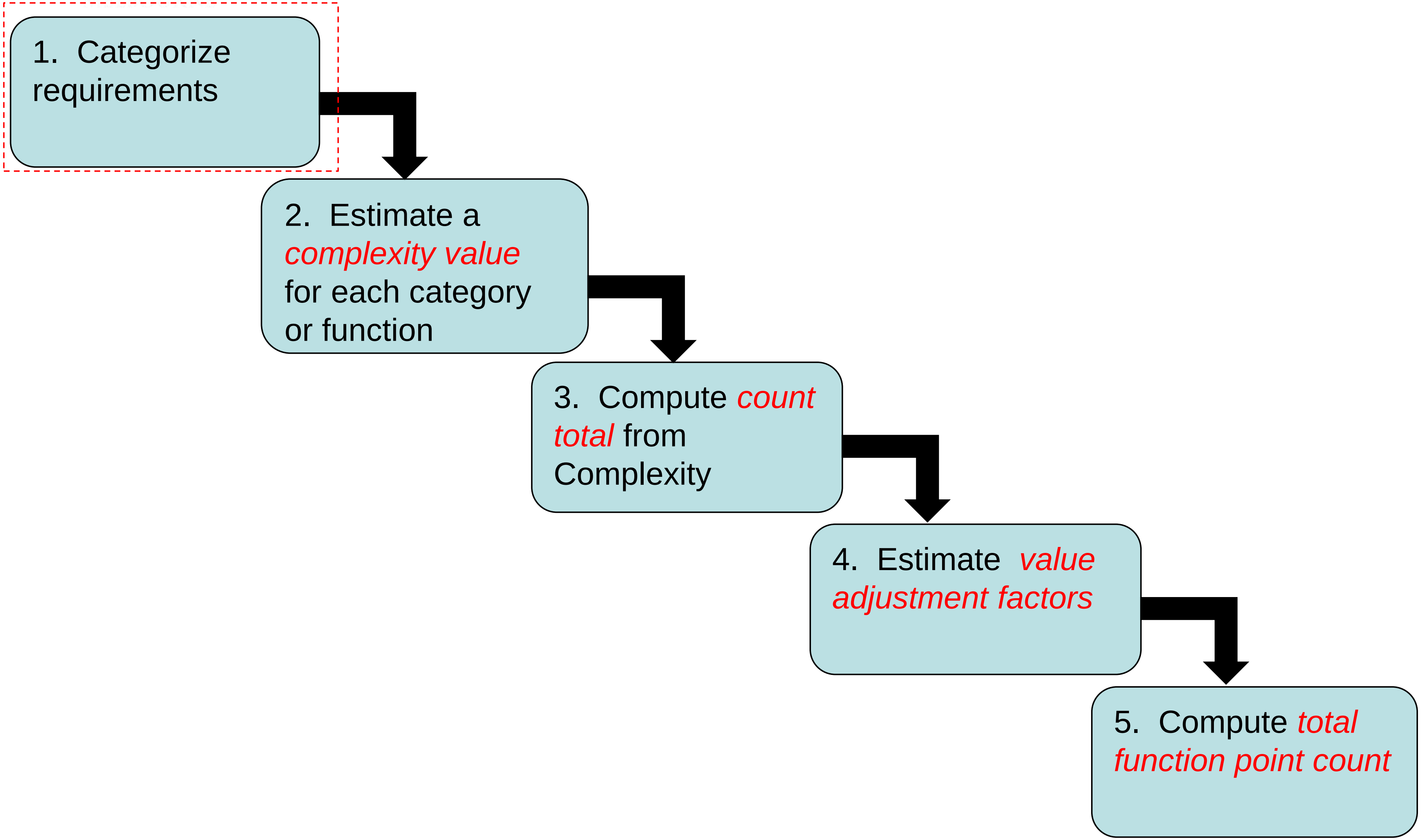
R.1	Read the previous day’s trading information (high price, low price, opening price, closing price) from a third-party-server.
R.2	Save a complete “price history” in a database.
R.3	Based on the trends in prices for commodities, decide which commodities to bid for, and which to try to sell.
R.4	Send information to an external e-trading account which places the bid/ask for each commodity.
R.5	When the bid/ask is either accepted or expires, record the result in a “transaction history” database.
R.6	At the end of the day, produce a report that summaries the transactions of the day along with the following information: profit/loss for the day; number of trades of each commodity; average market price of all commodities; account summary.
R.7	At anytime the user can request a transaction history for a period which gives: transaction IDs; commodity type; bid/ask; price.

FP Computation Steps



1. Categorize requirements
2. Estimate a *complexity value* for each category or function
3. Compute *count total* from Complexity
4. Estimate *value adjustment factors*
5. Compute *total function point count*

FP Computation Steps



1. Categorize requirements

2. Estimate a *complexity value* for each category or function

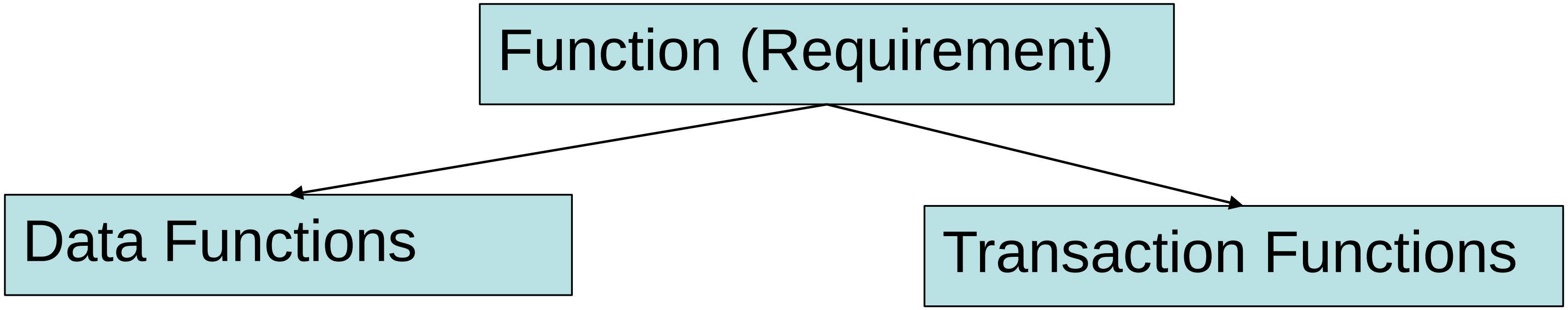
3. Compute *count total* from Complexity

4. Estimate *value adjustment factors*

5. Compute *total function point count*

FP Computation – Step 1

1. Categorize Requirements



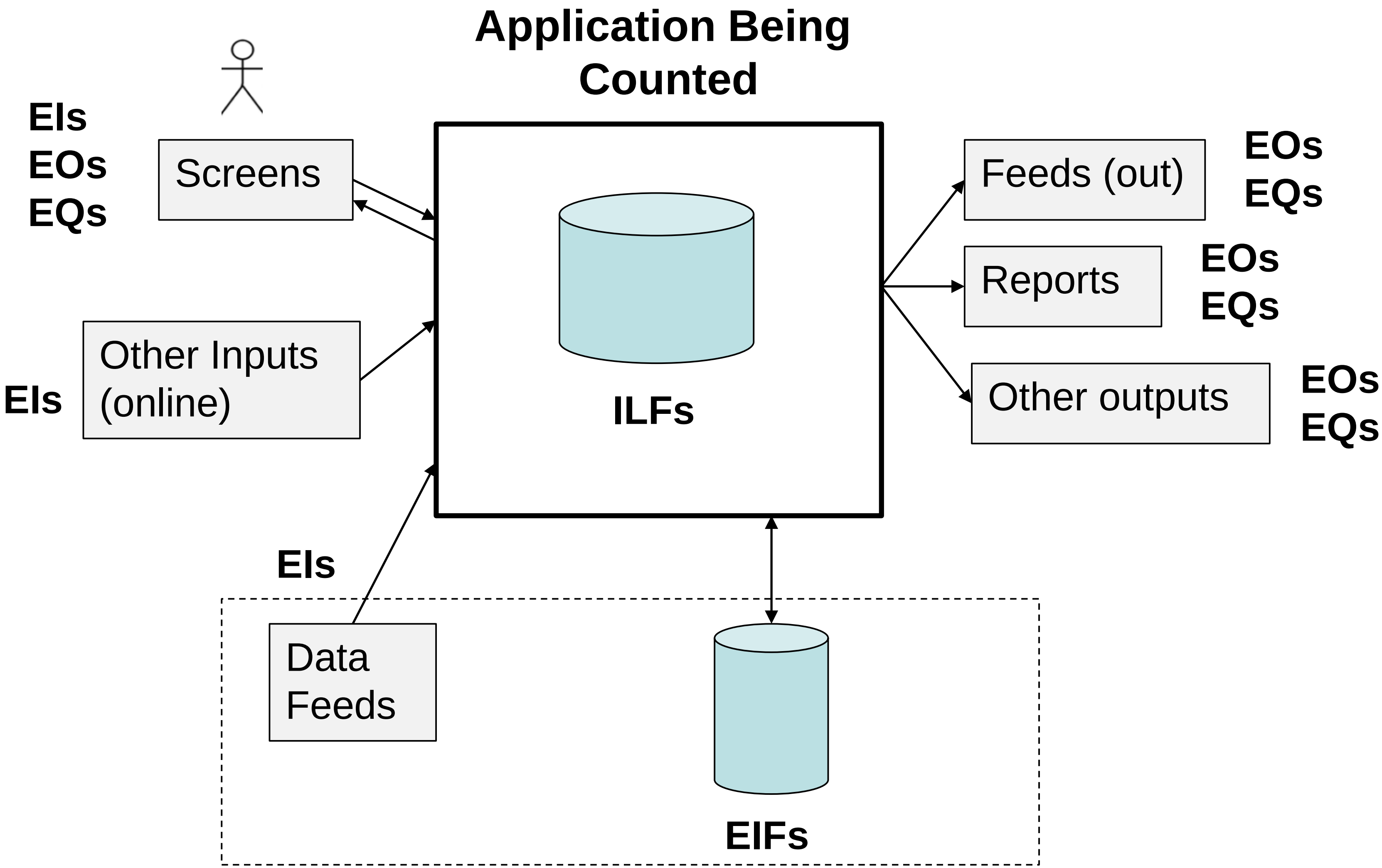
Data Functions:

- Concerned with maintenance of the data for the application – Internal Logical Files (ILF), External Interface Files (EIF)

Transaction Functions:

- Concerned with information being passes to and from the system - External Inputs (EI), External Output (EO), External Inquiries/Queries (EQ)

FP Computation – Step 1



FP Computation – Step 1

1. **Categorize Requirements (five categories)**
   - **Internal Logical File (ILF)**
     - A logical grouping of data that the system maintains over a period of time, and is modified using external inputs

       examples - tables in a relational database, files containing user setting
   - **External Interface File (EIF)**
     - A logical grouping of data that is maintained external to the system, but which may be used by the system.

       examples - are the same as ILFs, except that the data is maintained outside the system, such as data hosted on a third-party servers, or data structures holding information about system state

FP Computation - Step 1

- **External Input (EI)**
  - An input to the system from a user or another application, which is used to control the flow of the system, or provide data. External inputs generally modify internal logic files

    examples - data fields populated by users, inputs files (e.g. program source code to a compiler), and file feeds from an external application.

- **External outputs (EO)**
  - An output to the user that provides information about the state of the system

    examples - screens, error messages, and reports that are shown to the user. Individual data fields in these are grouped as one external output

FP Computation – Step 1

- **External Inquiries/Queries (EQ)**
  - the input is not used to update an internal logic file, but is used to query the internal logic file and provide an output; the output is retrieved directly, with no derived data included

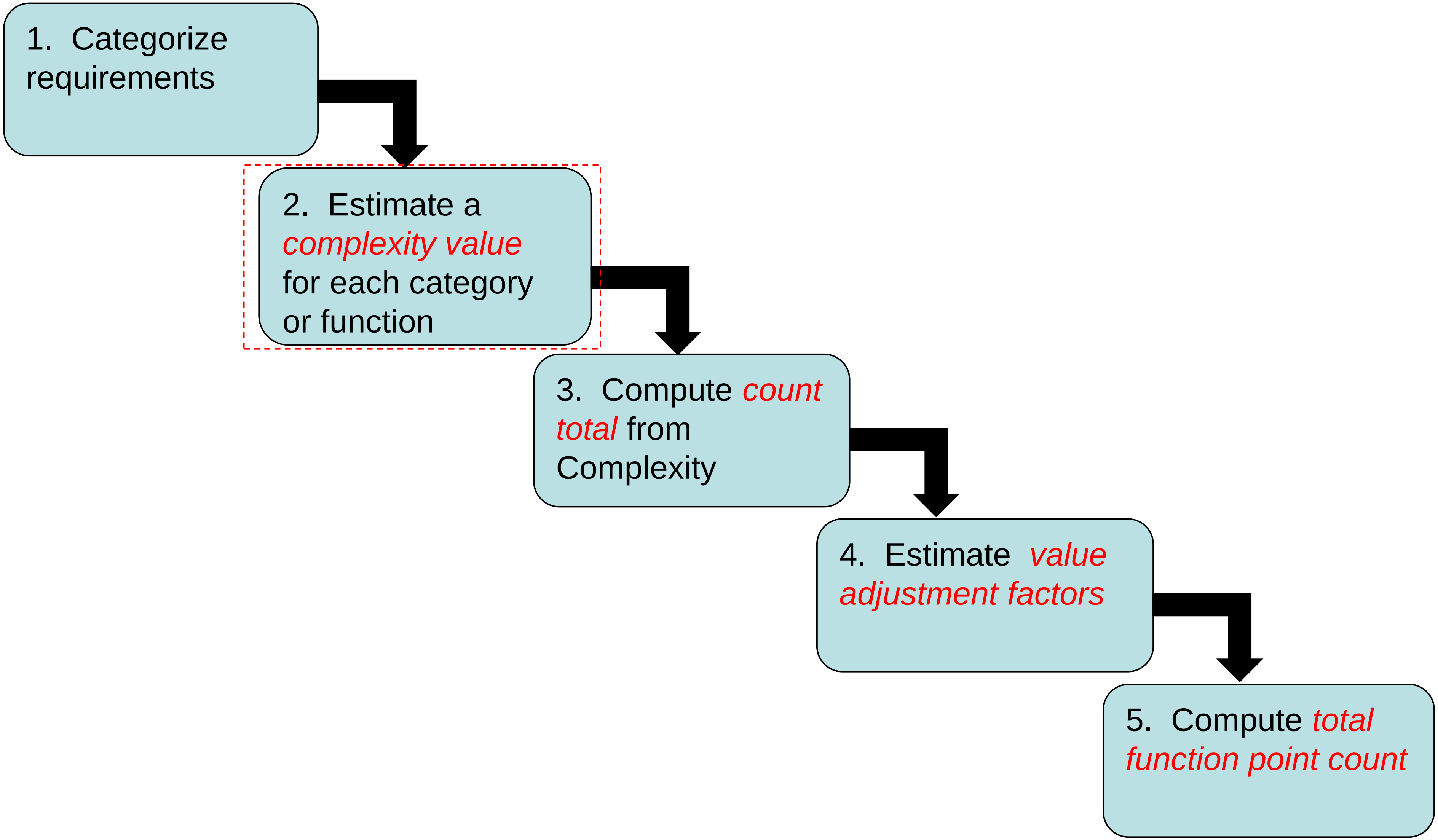
    examples - reading a user setting, or reading a record from a database table

Example: Automated Trading System

Automated Trading System: Function Categories

R.1	Read the previous day’s trading information (high price, low price, opening price, closing price) from a third-party-server.	EI
R.2	Save a complete “price history” in a database.	ILF
R.3	Based on the trends in prices for commodities, decide which commodities to bid for, and which to try to sell.	
R.4	Send information to an external e-trading account which places the bid/ask for each commodity.	EIF
R.5	When the bid/ask is either accepted or expires, record the result in a “transaction history” database.	ILF
R.6	At the end of the day, produce a report that summaries the transactions of the day along with the following information: profit/loss for the day; number of trades of each commodity; average market price of all commodities; account summary.	EO
R.7	At anytime the user can request a transaction history for a period which gives: transaction IDs; commodity type; bid/ask; price.	EQ

FP Computation Steps



1. Categorize requirements

2. Estimate a *complexity value* for each category or function

3. Compute *count total* from Complexity

4. Estimate *value adjustment factors*

5. Compute *total function point count*

FP Computation – Step 2

2. Estimate a *complexity value* for each category or function

- Complexity is ranked either ***simple, average or complex***
- Normally ***assigned for a category*** rather than for each requirement – rather crude
- A technique commonly used is based on Data Element Types (DETs), Record Element Types (RETs), and File Type References (FTRs):

Data Element Types (DETs)	A unique, user-recognizable, non-repeated data field in a system
Record Element Types (RETs)	A user-recognizable subgroup of data elements in an ILF or EIF
File Type References (FTRs)	A file (ILF, EIF) referenced by a transaction

FP Computation – Step 2

Relationship between DETs, RETs, FTRs, and the function categories

Function	DETs	RETs	FTRs
Internal Logical Files (ILFs)	x	x	
External Interface Files (EIFs)	x	x	
External Inputs (EIs)	x		x
External Outputs (EOs)	x		x
External Inquiries (EQs)	x		x

Complexity table for Data Functions

	DETs		
RETs	1-19	20-50	51+
1	Simple	Simple	Average
2-5	Simple	Average	Complex
6+	Average	Complex	Complex

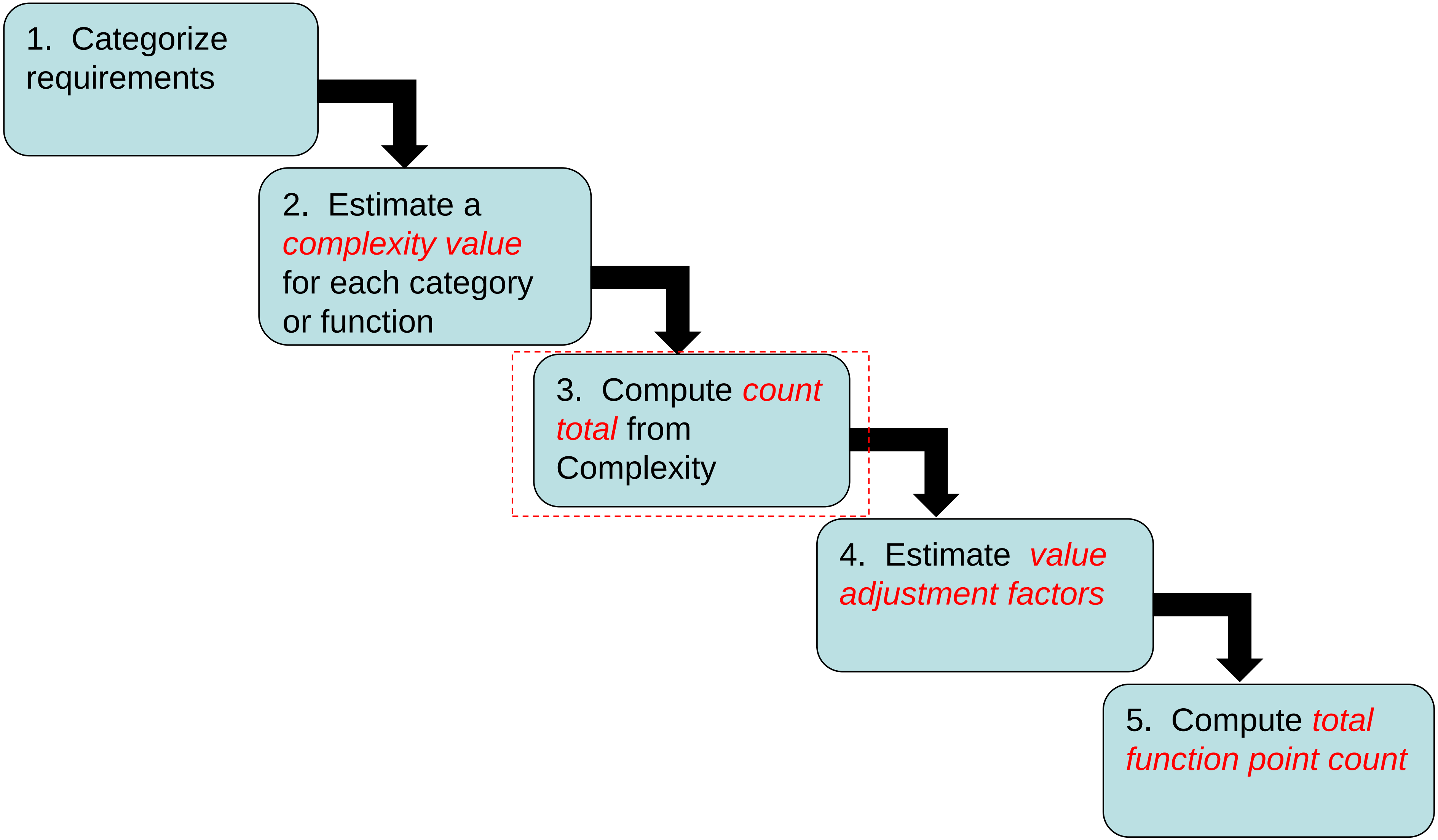
Complexity table for Transaction Functions

	DETs		
FTRs	1-5	6-19	20+
1	Simple	Simple	Average
2-3	Simple	Average	Complex
4+	Average	Complex	Complex

Automated Trading System: Function Categories

			DETs	RETs	FTRs	Complexity
R.1	**EI**	high price, low price, opening price, closing price	5		1	Simple
R.2	**ILF**	commodity name, high price, low price, opening price, closing price	6	1		Simple
R.3						
R.4	**EIF**	Commodity, bid/asking price, buy/sell	3	1		Simple
R.5	**ILF**	…	4	1		Simple
R.6	**EO**	…	5		3	Simple
R.7	**EQ**	…	4		2	Simple

FP Computation Steps



1. Categorize requirements

2. Estimate a *complexity value* for each category or function

3. Compute *count total* from Complexity

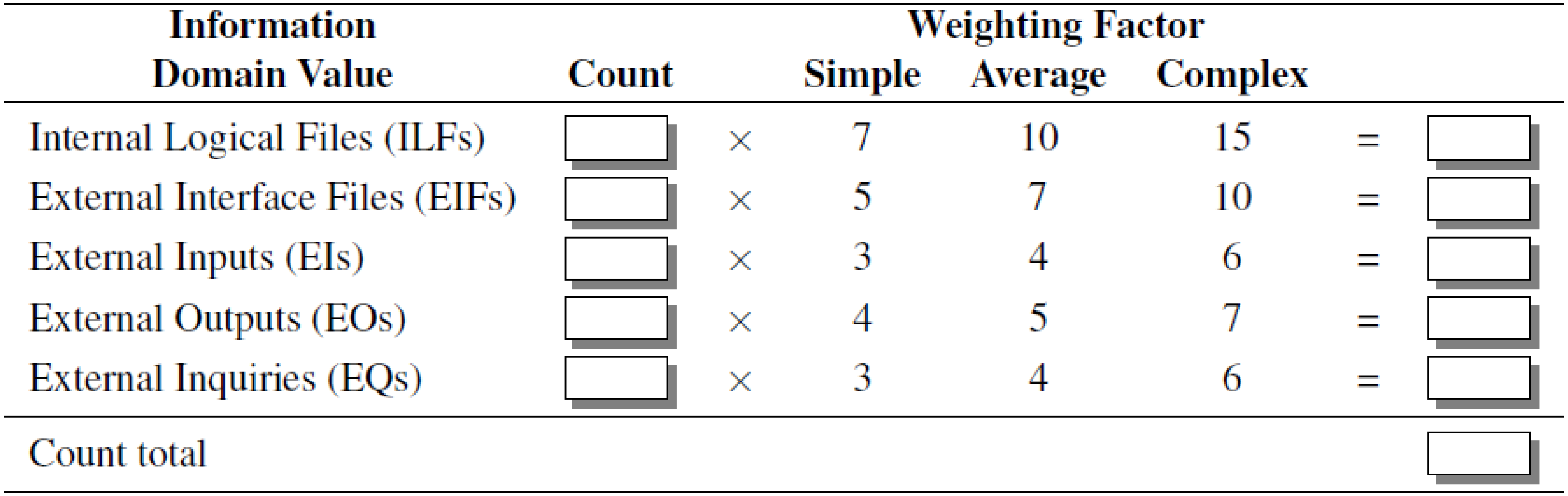
4. Estimate *value adjustment factors*

5. Compute *total function point count*

FP Computation – Step 3

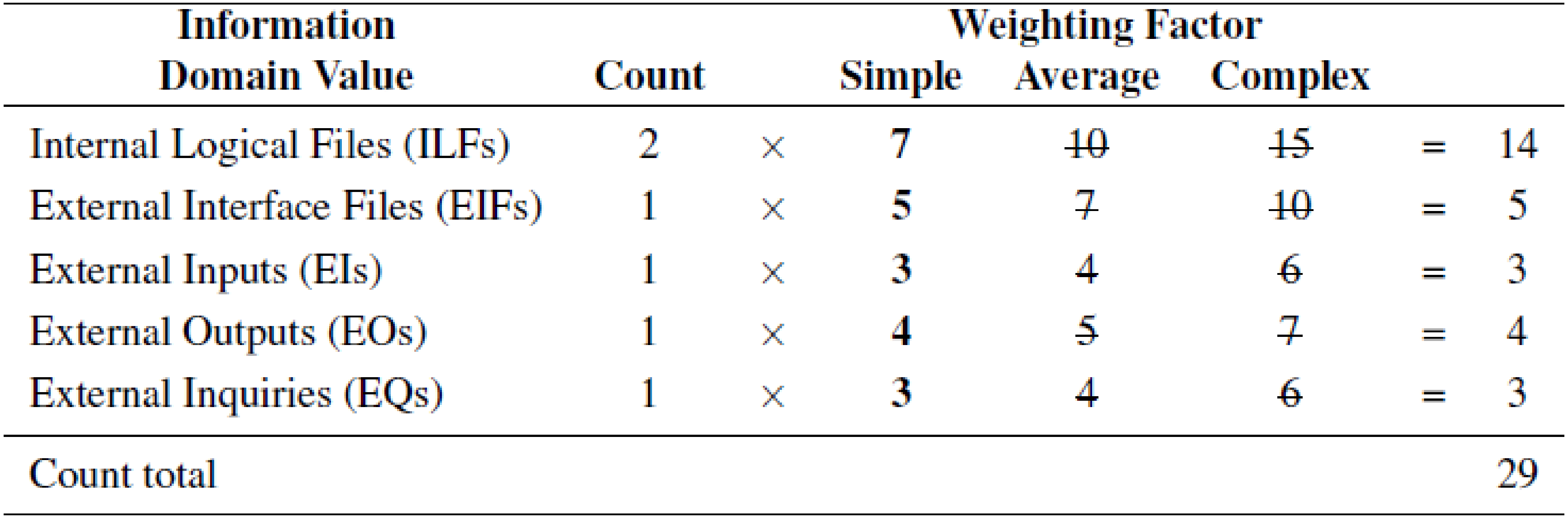
3. Compute *count total* from Complexity

- Using count and complexity estimates compute total count



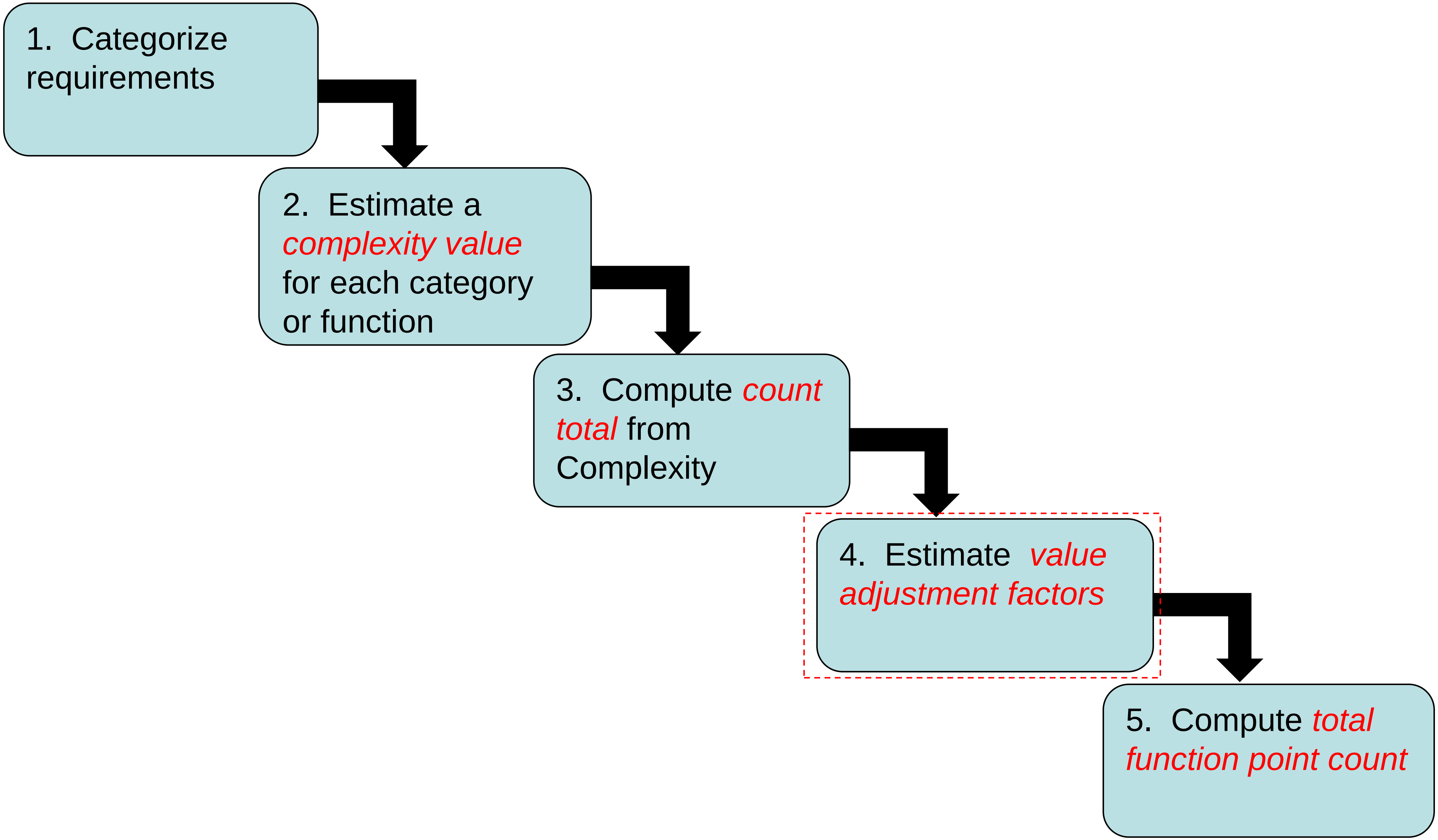
Information			Weighting Factor				
Domain Value	Count		Simple	Average	Complex		
Internal Logical Files (ILFs)		×	7	10	15	=	
External Interface Files (EIFs)		×	5	7	10	=	
External Inputs (EIs)		×	3	4	6	=	
External Outputs (EOs)		×	4	5	7	=	
External Inquiries (EQs)		×	3	4	6	=	
Count total							

Automated Trading System: Computing Count Total



Information			Weighting Factor				
Domain Value	Count		Simple	Average	Complex		
Internal Logical Files (ILFs)	2	×	**7**	~~10~~	~~15~~	=	14
External Interface Files (EIFs)	1	×	**5**	~~7~~	~~10~~	=	5
External Inputs (EIs)	1	×	**3**	~~4~~	~~6~~	=	3
External Outputs (EOs)	1	×	**4**	~~5~~	~~7~~	=	4
External Inquiries (EQs)	1	×	**3**	~~4~~	~~6~~	=	3
Count total							29

FP Computation Steps



1. Categorize requirements

2. Estimate a *complexity value* for each category or function

3. Compute *count total* from Complexity

4. Estimate *value adjustment factors*

5. Compute *total function point count*

FP Computation – Step 4

4. Compute value adjustment factors

- is computed based on 14 characteristics
- each of the characteristics is ranked on a scale of 0-5; 0 not important and 5 critical

- Data Communications
- Distributed Data Processing
- Performance
- Heavily Used Configuration
- Transaction Rate
- Online Data Entry
- End-User Efficiency

- Online Update
- Complex Processing
- Reusability
- Installation Ease
- Operational Ease
- Multiple Sites
- Facilitate Change

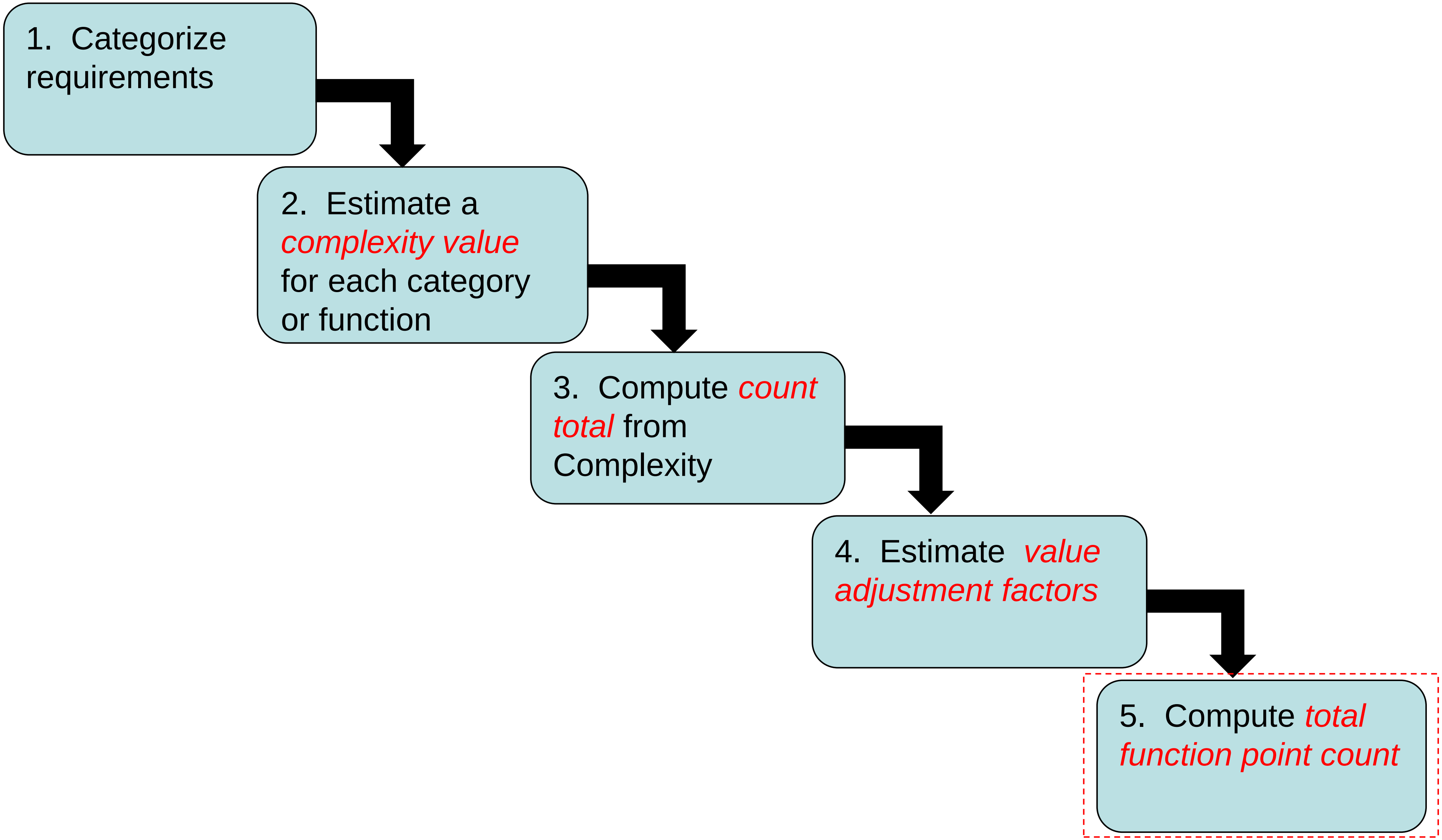
Automated Trading System: Value Adjustment Factors

- Data Communications - 2
- Distributed Data Processing - 4
- Performance - 5
- Heavily Used Configuration - 2
- Transaction Rate - 4
- Online Data Entry - 2
- End-User Efficiency - 5

- Online Update - 2
- Complex Processing - 2
- Reusability - 2
- Installation Ease - 2
- Operational Ease - 2
- Multiple Sites - 2
- Facilitate Change - 2

Assume the rating for all other features is 2
Value Adjustment = 2*5 + 4*2 + 10*2 = 38

FP Computation Steps



1. Categorize requirements
2. Estimate a *complexity value* for each category or function
3. Compute *count total* from Complexity
4. Estimate *value adjustment factors*
5. Compute *total function point count*

FP Computation – Step 5

5. Compute *total function point*

- Count total and value adjustment factors are then plugged-in to the following formula to estimate the total point count

$$FP = count\ total\ \times (0.65\ + 0.01\ \times\ \sum_{i=1}^{14} F_i)$$

F_i - VAF corresponding to the *i-th* VAF question

Example: Automated Trading System

Automated Trading System: Compute total function point count

$$FP = count\ total \times (0.65 + 0.01 \times \sum_{i=1}^{14} F_i)$$

FP = 29 x(0.65 + 0.01*38)
= 29 x1.03
= 29.87

A more complex and comprehensive example of FPs can be found at https://alvinalexander.com/FunctionPoints/

Function Points

- **Advantages of Function Points**
  - Measures the size of the solution instead of the size of the problem
  - Requirements are the only thing needed for function points count
  - Can be estimated early in analysis and design
  - Is independent of technology
  - Is independent of programming languages

Function Points

- **Disadvantages of Function Points**
  - A well defined requirements specification is necessary
  - Gaining proficiency is not easy, the learning curve is quite long
  - Could be quite time-consuming thus could be costly

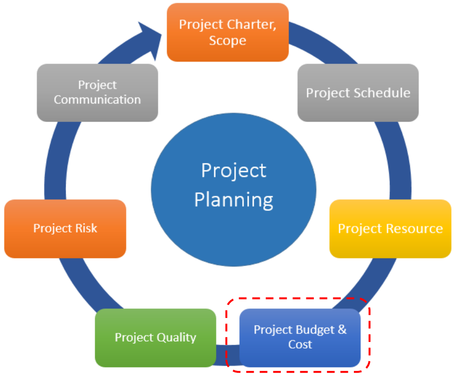
Lecture Break

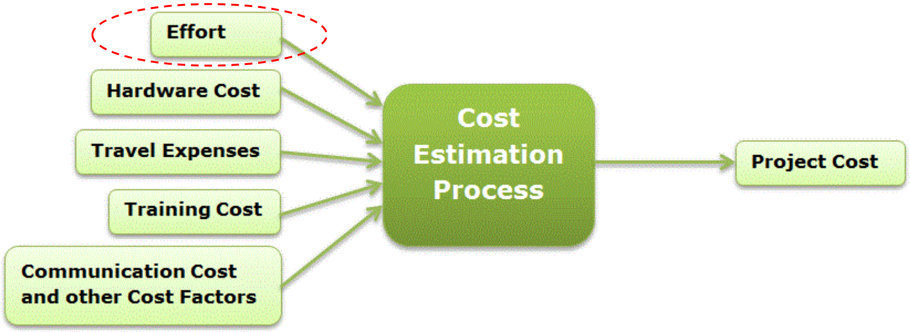
BREAK

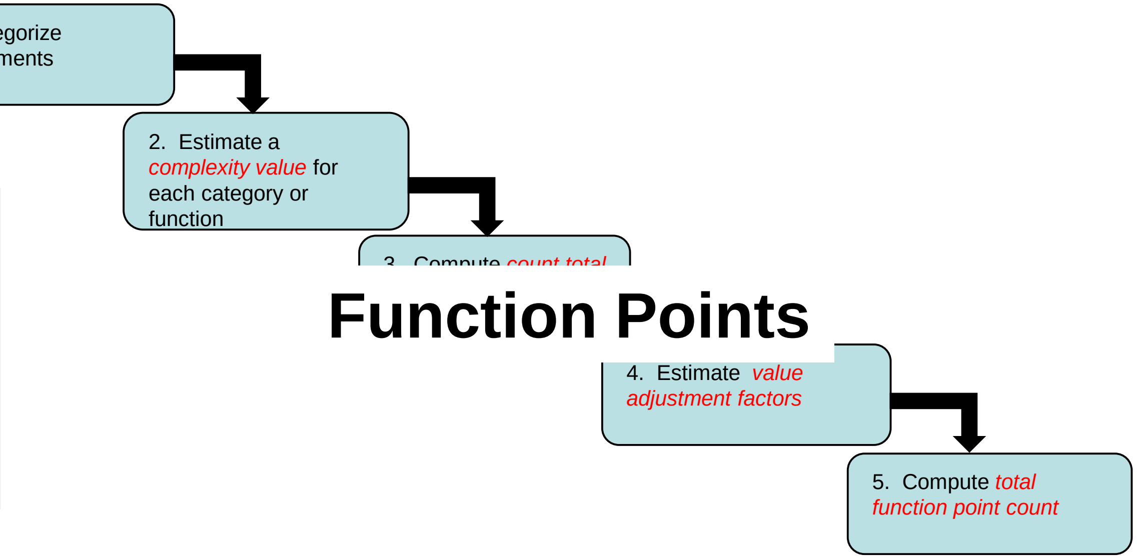
Please return promptly as the

Lecture will re-start in *5 mins*

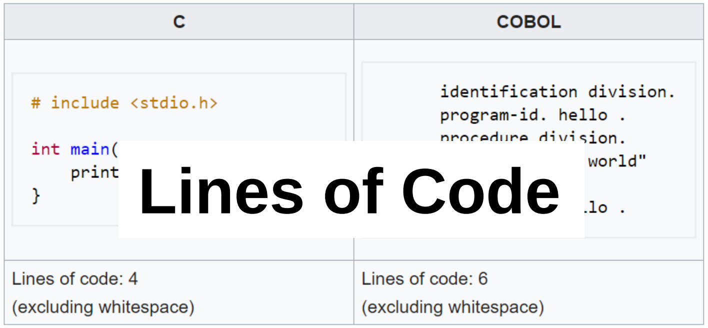
RACAP






1. Categorize requirements
2. Estimate a complexity value for each category or function
3. Compute count total
4. Estimate value adjustment factors
5. Compute total function point count

Function Points



Lines of Code

C	COBOL

`# include <stdio.h>`
`int main(`
`    print`
`}`
	`identification division.`
	`program-id. hello .`
	`procedure division.`
	`world"`
	`lo .`

Lines of code: 4 (excluding whitespace)	Lines of code: 6 (excluding whitespace)

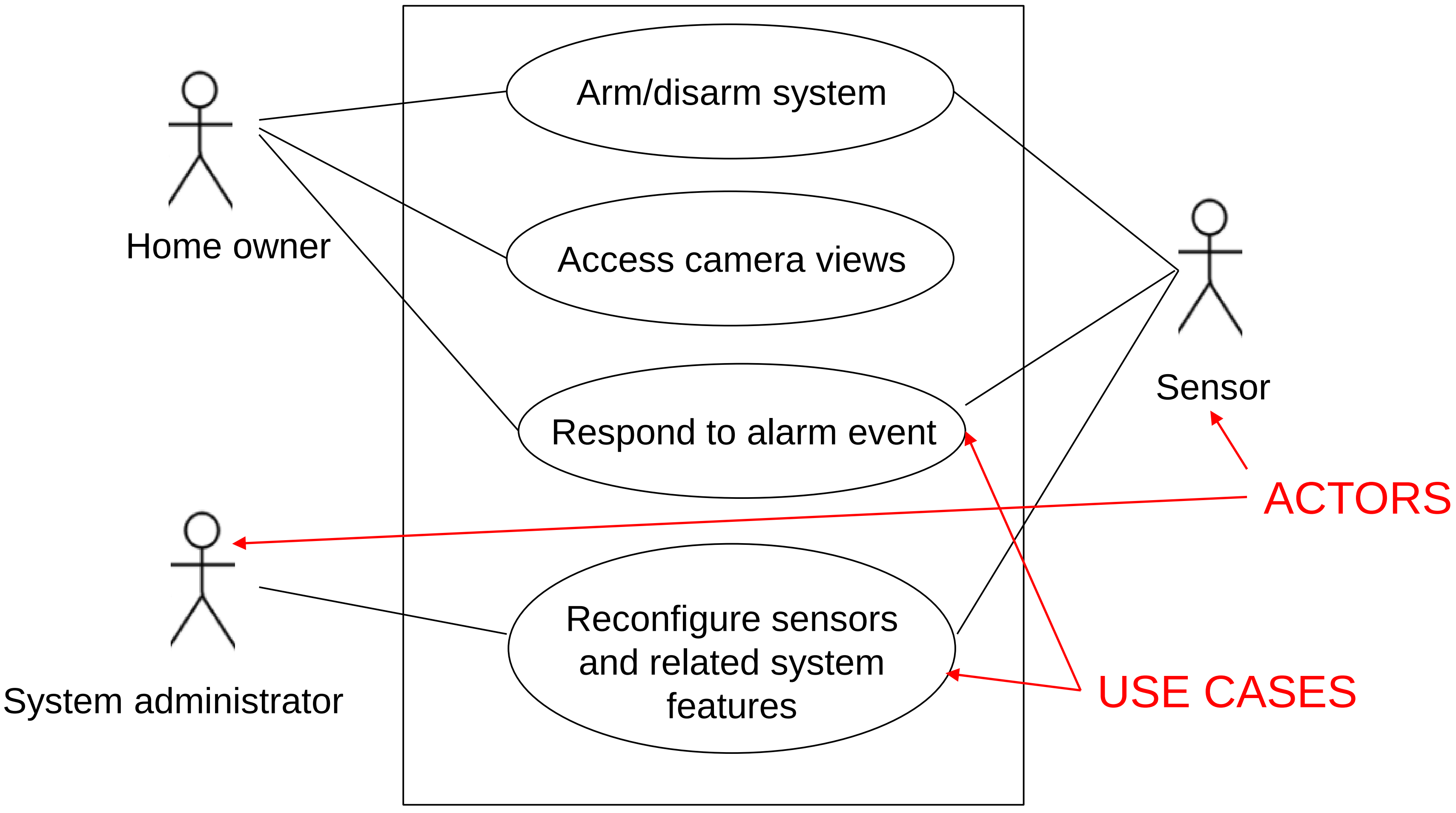
Use Case Points

- Is a software estimation technique used to measure the software size with Use Cases

- Developed in 1993 for sizing and estimating projects using OO methodology
  - developed by Gustav Karner of Objectory (now Rational Software)

- The concept of UCP is similar to FPs

Use Case Diagram



Use Case Diagram for a SafeHome Security System

Use Case Diagram

Use Case: Access camera views

Description: To view output of cameras from any remote location via the Internet

Primary Actor: Home owner

Preconditions: System must be fully configured; appropriate use-id and password must be available

Trigger: Home owner decides to view the camera output while away.

Scenario:

1. The home owner logs onto the SafeHome system via the Internet.
2. The home owner enters the user ID and password.
3. The system display major function buttons.
4. The home owner selects ‘surveillance’ from the major function buttons.
5. The home owner selects ‘pick a camera’ option.
6. The system displays the available cameras on a map.
7. The home owner selects the camera of interest.
8. The home owner selects the ‘view’ button.
9. The system display the video output on the viewing window.

Extensions:

2a. ID or password are incorrect; user is requested to re-enter the password or to validate password.
4a. Surveillance function is not properly configured; system displays an error message and exits.
6a. Map is not properly configured; system request the user to configure the map.
….

Use Case Points

1. Compute Unadjusted Use Case Weight (UUCW)
2. Compute Unadjusted Actor Weight (UAW)
3. Compute Technical Complexity Factor (TCF)
4. Compute Environmental Complexity Factor (ECF)
5. Compute the final size estimate

UCP Computation – Step 1

1. Compute Unadjusted Use Case Weight (UUCW)

Count the number of simple average, complex use cases, N_s, N_A, N_C based on the number of transactions as per table below.

- The number of transactions can be computed by counting the number of steps in the scenario

Use Case Classification	Type of Actor	Weight
Simple	1 to 3 transactions	5
Average	4 to 7 transactions	10
Complex	8 or more transactions	15

$$UUCW = N_s \times 5 + N_A \times 10 + N_c \times 15$$

Use Case Points – Step 2

2. Compute Unadjusted Actor Weight (UAW)

Count the number of simple average, complex actors, N_s,N_A ,N_C as per table below.

Actor Classification	Type of Actor	Weight
Simple	External system interacting using a well defined API	1
Average	External system interacting using a standard protocol (e.g. TCP/IP, FTP, HTTP)	2
Complex	Human actor using a GUI	3

$$UAW = N_s \times 1 + N_A \times 2 + N_c \times 3$$

Use Case Points – Step 3

3. Compute Technical Complexity Factor (TCF)

Factor	Description	Weight
T1	Distributed system	2.0
T2	Response time	1.0
T3	End-user efficiency	1.0
T4	Internal processing complexity	1.0
T5	Code reusability	1.0
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	Portability to other platforms	2.0
T9	System maintenance	1.0
T10	Concurrent/parallel processing	1.0
T11	Security features	1.0
T12	Access for third parties	1.0
T13	End user training	1.0

Score each factor between 0 – 5

- 0 - irrelevant
- 5 - essential

$$TF = \sum_{i=1}^{13} S_i \times W_i$$

S_i - Score for the *i-th* factor

W_i - Weight of the *i-th* factor

$$TCF = 0.6 + TF/100$$

Use Case Points – Step 4

4. Compute Environmental Complexity Factor (ECF)

Factor	Description	Weight
E1	Familiarity with development process used	1.5
E2	Application experience	0.5
E3	Object-oriented experience of team	1.0
E4	Lead analyst capability	0.5
E5	Motivation of the team	1.0
E6	Stability of requirements	2.0
E7	Part-time staff	-1.0
E8	Difficult programming language	-1.0

Score each factor between 0 – 5

- 0 - irrelevant
- 5 - essential

$$EF = \sum_{i=1}^{8} S_i \times W_i$$

S_i - Score for the *i-th* factor
W_i - Weight of the *i-th* factor

$$ECF = 1.4 + (-.03 * EF)$$

UCP Computation – Step 5

5. Compute the final size estimate

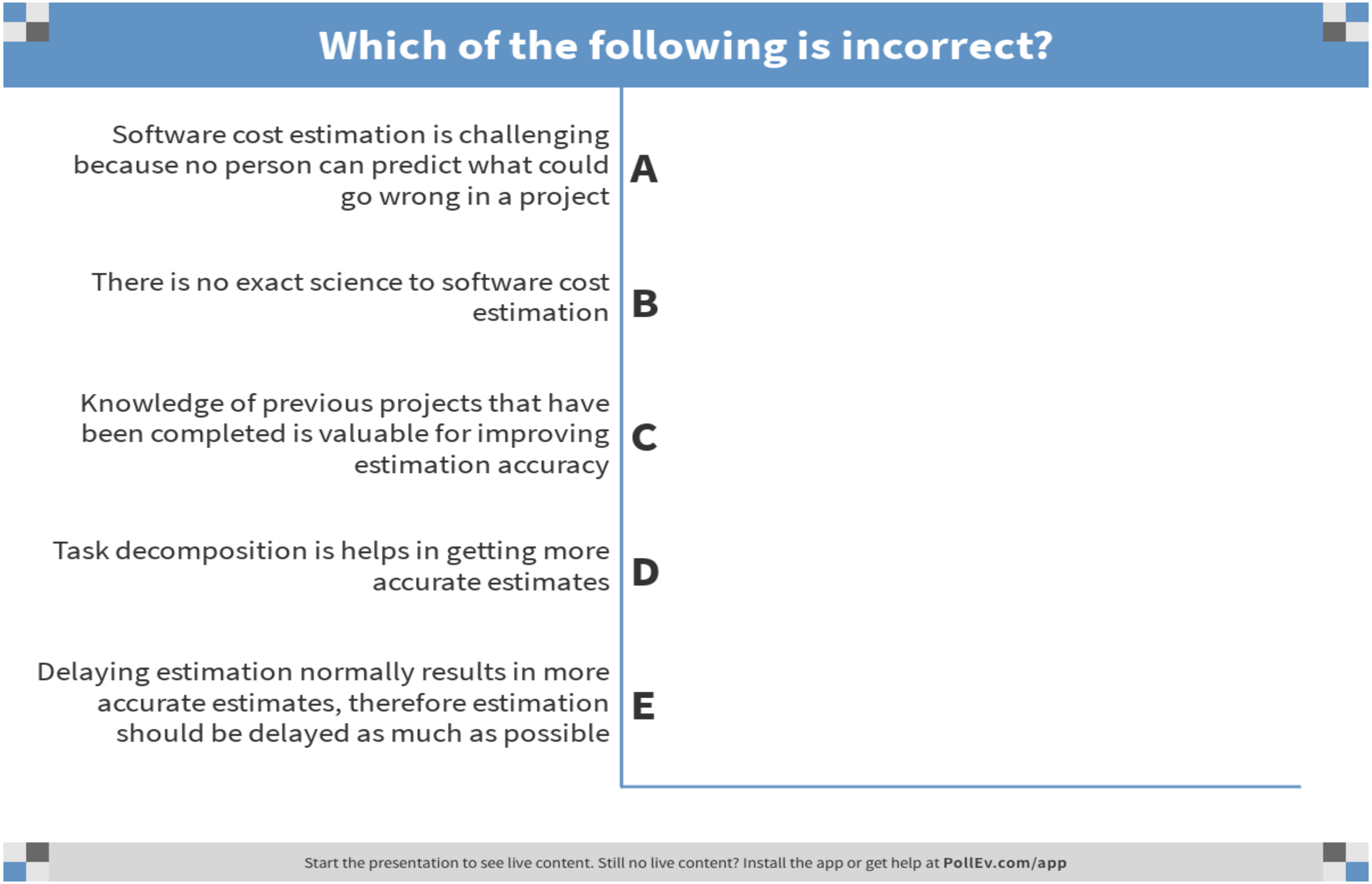
$$UCP = (UUCW + UAW) \times TCF \times ECF$$

Use Case Points

- **Advantages of Use Case Points**
  - UCPs are based on use cases and can be measured very early in the project life cycle
  - UCP based estimates are found to be close to actuals when estimation is performed by experienced people
  - UCPs are easy to use and do not call for additional analysis
  - Use cases are being used vastly as a method of choice to describe requirements

Use Case Points

- **Disadvantages of Use Case Points**
  - UCP can be used only when requirements are written in the form of use cases
  - Dependant on goal-orientated, well written use cases
  - Technical and environmental factors have a high impact on UCP
  - Not as well established as FPs



Which of the following is incorrect?

Software cost estimation is challenging because no person can predict what could go wrong in a project **A**

There is no exact science to software cost estimation **B**

Knowledge of previous projects that have been completed is valuable for improving estimation accuracy **C**

Task decomposition is helps in getting more accurate estimates **D**

Delaying estimation normally results in more accurate estimates, therefore estimation should be delayed as much as possible **E**

Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**

Learning Outcomes

1. Understand the importance of cost estimation and the challenges involved
2. Understand the techniques used for cost estimation
3. Understand software size estimation techniques
4. Understand the principles of the COCOMO II model for algorithmic cost estimation
5. Understand cost estimation techniques used in Agile software development lifecycles

COCOMO Model

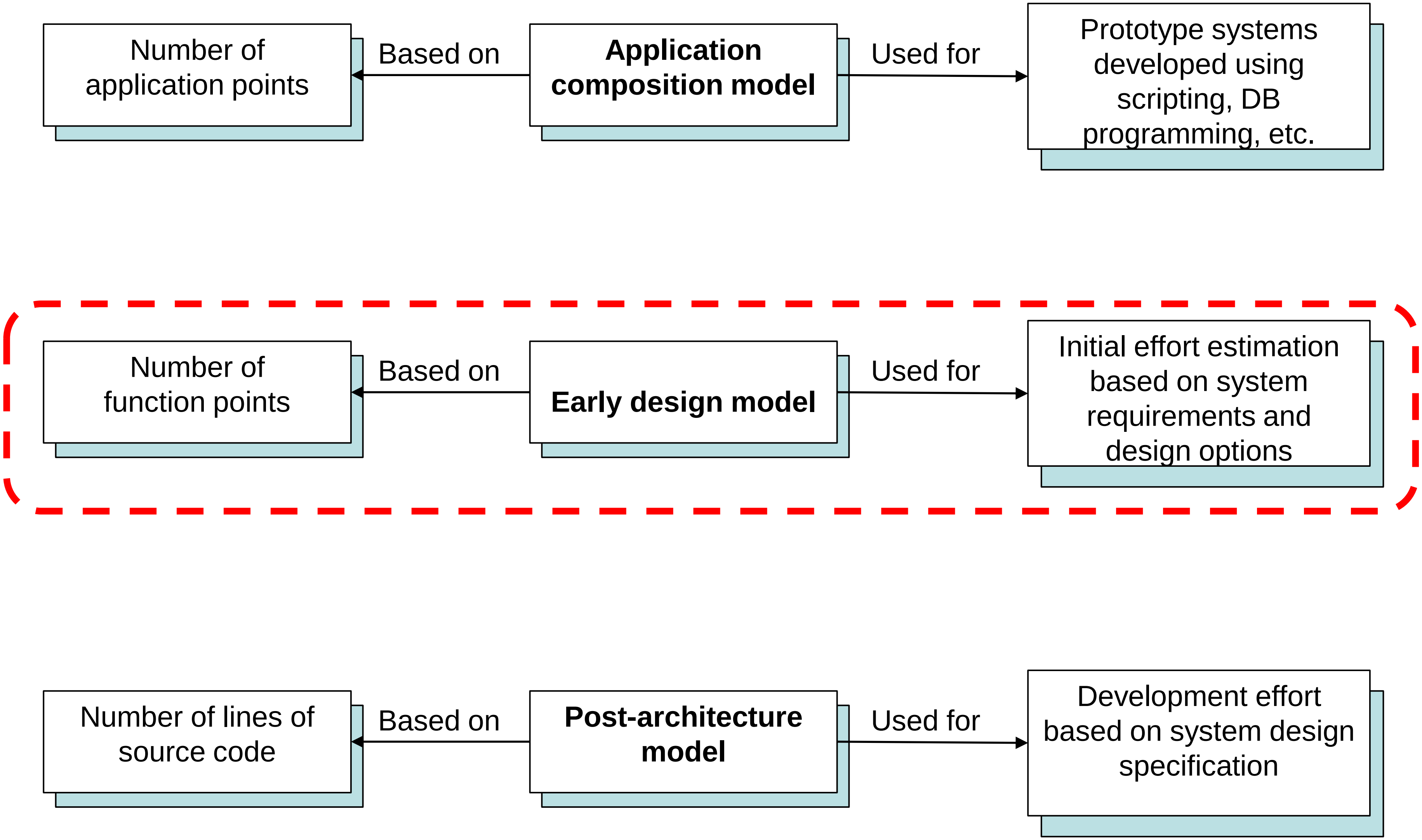
- Derived from collecting data from a large number of software projects and deriving formulae that best fits the observations – *an empirical model*

- It has been widely used and evaluated in a range of organizations

- Well documented, available in the public domain and is well supported by tools

- It has been in use for a long time:
  - first proposed in 1982 (Boehm_1981)
  - most recent version, COCOMO II was published in 2000 (Boehm_2000)

COCOMO II


Number of application points
Based on
Application composition model
Used for
Prototype systems developed using scripting, DB programming, etc.
Number of function points
Based on
Early design model
Used for
Initial effort estimation based on system requirements and design options
Number of lines of source code
Based on
Post-architecture model
Used for
Development effort based on system design specification

COCOMO II – Early Design Model

$$Effort = A \times Size^{B} \times M$$

A - a constant factor that depends on the organizational practices

$Size$ - size of the software estimated in a metric of choice

B - a value between 1 and 1.5 derived experimentally

M − a multiplier made by combining process, product and developmet attributes such as sability of requirement, experience of team

$Effort$ - total effort measured in person-months

In the COCOMO II early design model:

Parameter	**Value**	**Notes**
A	2.94	Estimated empirically
Size	KSLOC (thousands of lines of code)	Computed from FPs
B	1.01 – 1.26	Explained later
M	Computed based on project and process characteristic	Explained later

COCOMO II – Early Design Model

COCOMO II: Estimating *Size* in KSLOC $(Effort = A \times Size^{B} \times M)$

– Size is estimated based on logical lines of code

– Can be estimated based on FPs using the table below

Language	Average	Median	Low	High
Ada	154	-	104	205
Assembler	209	203	91	320
C	148	107	22	704
C++	59	53	20	178
C#	58	59	51	66
Fortran	90	118	35	-
Java	55	53	9	214
Perl	57	57	45	60
Visual Basic	50	52	14	276

Number of logical lines of code per FP

e.g. 200 FPs: in C = `29.6 KSLOC`; in Java = `11 KSLOC`

COCOMO II – Early Design Model

COCOMO II: Estimating parameter B $(Effort = A \times Size^{B} \times M)$

$$B = 1.01 + .01 \sum_{i=1}^{5} W_i$$

W_i - a scaling factor value ranging from 0-5 as per table below.

Precedentedness	Familiarity of the application domain: 0 (thoroughly familiar) to 5 (completely unprecedented).
Development flexibility	The level of flexibility in development process, methods, and tools, ranging from 0 (general goals) to 5 (rigorous).
Architecture completed and risks eliminated	20%: 5; 40%: 4; 60%: 3; 80%: 2; 90%: 1; 100%: 0
Team cohesion	The level of interaction within the team. Ranging from 0 (seamless interactions) to 5 (very difficult interactions).
Process maturity	A ranking use the Software Engineering Institute’s Capability Maturity Model, ranging from 0 (chaotic, following no processes) to 5 (actively measuring and optimising processes)

COCOMO II – Early Design Model

COCOMO II: Estimating M $(Effort = A \times Size^{B} \times M)$

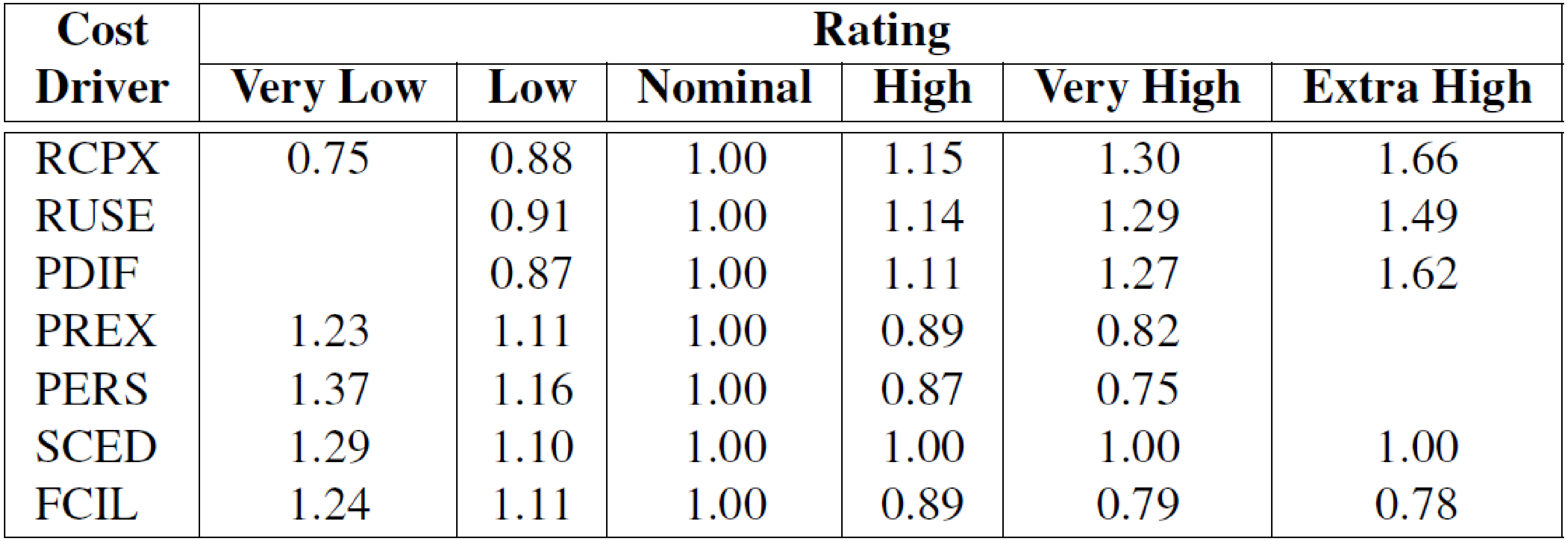
- the cost drivers consist of seven different factors
- each factors is rated on a six point scale; *very low* to *extra high*

$$M = RCPX \times RUSE \times PDIF \times PREX \times PERS \times SCED \times FCIL$$

COCOMO II – Early Design Model

RCPX	The expected complexity of the internal processes, and the level of reliability required for the system.
RUSE	The level of reuse that this code developed in this system is expected to offer to future systems.
PDIF	The level of platform difficulty. This refers to the constraints placed on the system by the platform on which it runs, such as the amount of processor time and storage available.
PREX	The experience of the personnel on the project. Ranging from less than 2 months (very low) to more than 6 years (very high).
PERS	The capability of the personnel on the project. Ranging from the 15th percentile (very low) to the 90 percentile (very high).
SCED	The constraints placed upon the project schedule, rated as a percentage of the "stretch-out" of the schedule. Schedules that are highly compressed (not stretched-out), require more effort than the optimal to complete the project on time. A rating of very low is a schedule that is 75% the length of the nominal project. A rating of very high is a project schedule that is 160%+ of the nominal. Empirical evidence suggests that compressing the schedule, and therefore expending less effort, results in lower quality software.
FCIL	The team support facilities.

COCOMO II – Early Design Model



Cost Driver	Rating					
	Very Low	Low	Nominal	High	Very High	Extra High
RCPX	0.75	0.88	1.00	1.15	1.30	1.66
RUSE		0.91	1.00	1.14	1.29	1.49
PDIF		0.87	1.00	1.11	1.27	1.62
PREX	1.23	1.11	1.00	0.89	0.82	
PERS	1.37	1.16	1.00	0.87	0.75	
SCED	1.29	1.10	1.00	1.00	1.00	1.00
FCIL	1.24	1.11	1.00	0.89	0.79	0.78

Cost driver ratings for the COCOMO II early-design phase model.

COCOMO II – Early Design Model

COCOMO II: Estimating Effort

- Based on the computed parameter values the effort can be estimated:

$$Effort = A \times Size^{B} \times M$$

COCOMO II: Estimating Time (T) and Number of Personnel (N)

- Formula for estimating the nominal delivery time:

$$T = 2.5 \times Effort^{(0.33+0.2(B-1.01))}$$

$$N = \frac{Effort}{T}$$

Learning Outcomes

1. Understand the importance of cost estimation and the challenges involved
2. Understand the techniques used for cost estimation
3. Understand software size estimation techniques
4. Understand the principles of the COCOMO II model for algorithmic cost estimation
5. Understand cost estimation techniques used in Agile software development lifecycles

Effort Estimation in Agile

Two key concepts that are used to effort estimation:

Story points: a story point is a relative measure of the size of a user story (recall that the requirements of the system are documented using user stories)

Velocity: velocity is a measure of productivity of team, which is represented by the number of story points delivered in a specified time period

Agile Estimation Process

1. Develop user stories for the system.
2. Estimate the number of story points for each story, basing the estimate on the number of story points from previous stories, using a chosen technique (discussed later).
3. Use the team’s velocity from previous experience to estimate the delivery time of the project - in the case of fixed-scope release planning develop a release burn-down chart.
4. During development, measure the actual velocity of the team.
5. Using this velocity, re-estimate the time it will take to deliver the product.

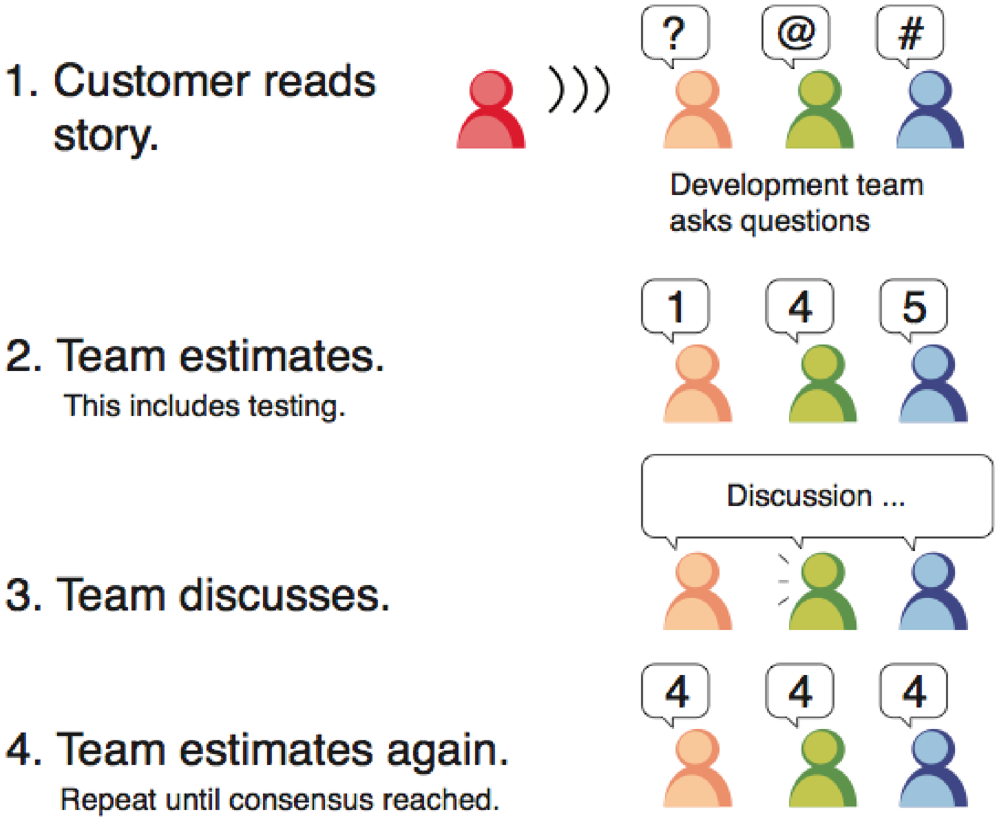
Agile Estimation Guidelines

- **Estimate by analogy**
  - There are no units for story points, always base our measures on other stories. If story A is about the same size as story B, they should have the same number of story points.
- **Decompose a story**
  - By decomposing a story into the tasks that are required to complete the story, we can find measures that we know about the tasks, and combine them to provide a total measure.
- **Use the right units**
  - The relative units should not be too fine grained. A pattern-based scale is used. For example, measures can only be 1, 2, 4, 8, or 12 or numbers in the Fibonacci sequence.
- **Use group-based estimations**
  - For a story that is to be implemented by a team, the whole team should provide estimates. Techniques such as the Delphi method or its adaptations can be used to reach consensus.

Agile Estimation Techniques

- Agile Estimation Techniques:
  - Planning Poker
  - Bucket System
  - Relative Mass Valuation
  - T-Shirt Sizes
  - Affinity Estimation
  - Dot Voting

Planning Poker



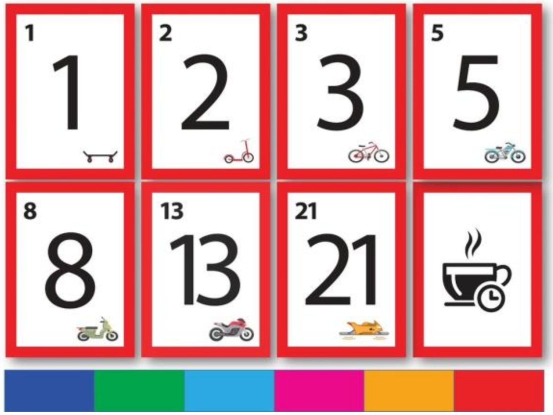
1. Customer reads story.

Development team asks questions

2. Team estimates.
This includes testing.

3. Team discusses.

4. Team estimates again.
Repeat until consensus reached.

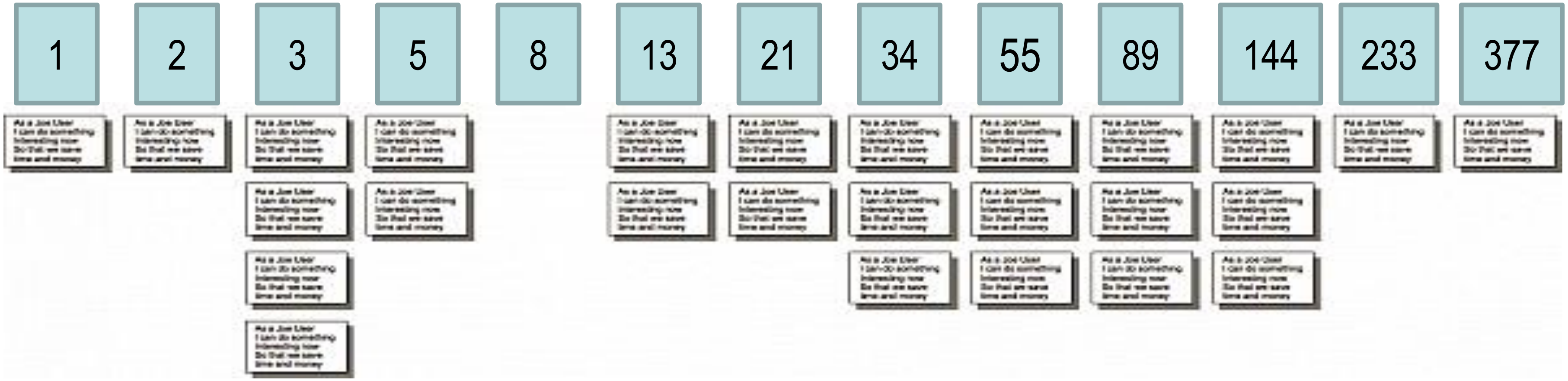


https://www.sitepoint.com/3-powerful-estimation-techniques-for-agile-teams/

BUCKET SYSTEM

http://www.agileadvice.com/2013/07/30/referenceinformation/agile-estimation-with-the-bucket-system/

BUCKET SYSTEM



- The team sitting at a table picks a user story card randomly and places is in bucket 8
- The next few cards are randomly picked one at a time, discussed agreed on, and placed in a bucket relative to the previous ones
- Then each person is allocated a set of cards and they are placed in a appropriate bucket, based on individual judgement (Divide and conquer)
- Finally the team reviews the placements and reach agreement

Relative Mass Valuation

1. Set up a large table so the stories can be moved around easily relative to each other.
2. Pick any story to start, team estimates whether they think that it is relatively: Large, Medium, Small.
3. Large story one end on the table. Medium story in the middle and Small story the other end
4. Continue through steps 2 & 3
5. The next step is to assign points values based on the position of the stories on the table. Start with the easiest story that is worth assigning points to, and call it a 1.
6. Then move up the list of cards, assigning a value of 1 to every story until you get to one that seems at least twice as difficult as the first one. That story gets a 2.

http://www.flowless.eu/relative-mass-valuation/

Computing Velocity

$$V = \frac{SP}{T_i}$$

V - velocity

SP - number of story points completed

T_i - time period over which they were completed

Common methods for measuring velocity:

- Using historical data
- Using data from previous iterations

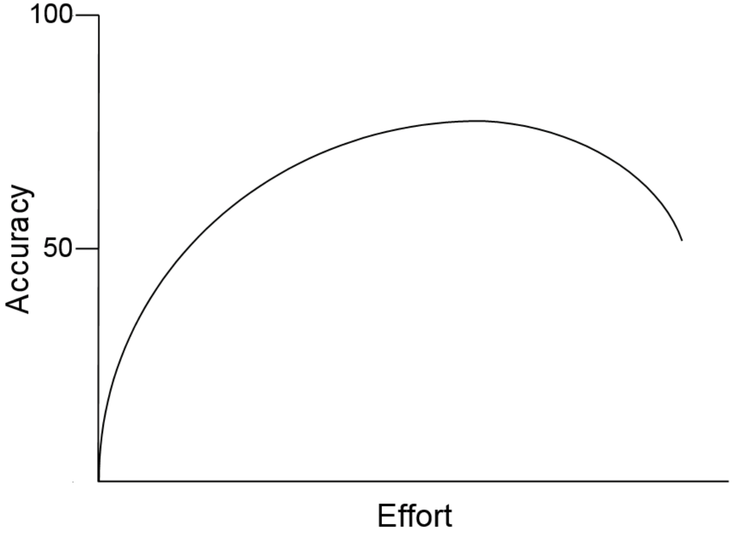
Estimated Delivery Time

$$T = \frac{\sum_{i=1}^{n} SP_i}{V}$$

T - estimate delivery time
V - velocity
SP_i - number of story points in the i-th user story
n - total number of user stories

Final comments on estimation

- Allow enough time to do a proper project estimate - rushed estimates are inaccurate, high-risk estimates
- There is diminishing return on time spent estimating



- Agile teams choose to be closer to the left
- Know that you cannot eliminate uncertainty from estimates but small efforts are rewarded with big gains

Learning Outcomes

1. Understand the importance of cost estimation and the challenges involved
2. Understand the techniques used for cost estimation
3. Understand software size estimation techniques
4. Understand the principles of the COCOMO II model for algorithmic cost estimation
5. Understand cost estimation techniques used in Agile software development lifecycles

Molkken, Kjetil and Jrgensen, Magne

Molkken, Kjetil and Jrgensen, Magne

References

1. B. Boehm, C. Abts, W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece. Software Cost Estimation with Cocomo II. Prentice Hall, 2000.
2. R. S. Pressman. Software Engineering: A Practitioner’s Approach. McGraw Hill, seventh edition, 2009.
3. I Somerville. Software Engineering, Addison-Wesley Publishing, ninth edition, 2010.
4. K. Molkken and M. Jrgensen, A Review of Surveys on Software Effort Estimation, Proceedings of the 2003 International Symposium on Empirical Software Engineering

cont…

References

Molkken, Kjetil and Jrgensen, Magne

5. Danh Nguyen-Cong and De Tran-Cao, A review of effort estimation studies in agile, iterative and incremental software development, The 2013 RIVF International Conference on Computing Communication Technologies - Research, Innovation, and Vision for Future (RIVF)

6. M. Usman, E, Mendes and F. Weidt and R. Britto, Effort Estimation in Agile Software Development: A Systematic Literature Review, Proceedings of the 10th International Conference on Predictive Models in Software Engineering, 2014

7. PMI's PULSE of the PROFESSION -https://www.pmi.org/learning/thought-leadership/pulse/pulse-of-the-profession-2017

Molkken, Kjetil and Jrgensen, Magne

Molkken, Kjetil and Jrgensen, Magne
Molkken, Kjetil and Jrgensen, Magne

References

1. B. Boehm, C. Abts, W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece. Software Cost Estimation with Cocomo II. Prentice Hall, 2000.
2. R. S. Pressman. Software Engineering: A Practitioner's Approach. McGraw Hill, seventh edition, 2009.
3. I Somerville. Software Engineering, Addison-Wesley Publishing, ninth edition, 2010.
4. A Review of Surveys on Software Effort Estimation
5. A review of effort estimation studies in agile, iterative and incremental software development, The 2013 RIVF International Conference on Computing Communication Technologies - Research, Innovation, and Vision for Future (RIVF)