



THE UNIVERSITY OF  
MELBOURNE

SWEN90016  
Software Processes & Project Management

# Cost Estimation

*Marion Zalk*

*Department of Computing and Information Systems*

*The University of Melbourne*

*mzalk@unimelb.edu.au*

2021– Semester 1

Lecture 7

Copyright University of Melbourne 2018

# Assignment 2

Your Assignment 2 report should include screen shots of your Gantt Chart or Kanban board in section 6.5.

You have an option of also including a link to your online tool, such as Trello, if you think this will aid your marker, as sometimes the screen shots are fuzzy and difficult to read.

If you grant (read) permissions to the generic SWEN90016 tutor email address inside your tool,

`swen90016tutors@groups.unimelb.edu.au`

then all tutors who click on the link inside your report will have access to read.

# 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

## Redecorate Room

### Prepare materials

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

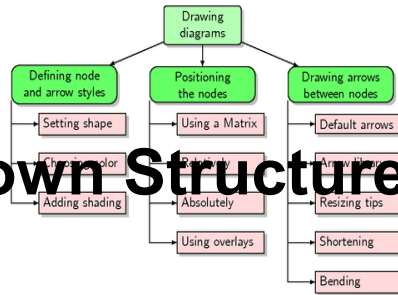
### Prepare room

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

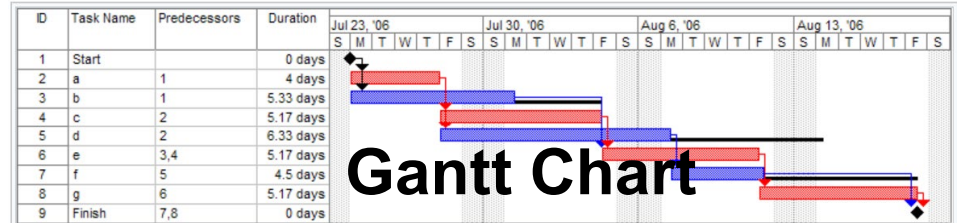
### Paint the room

### Clean up the room

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

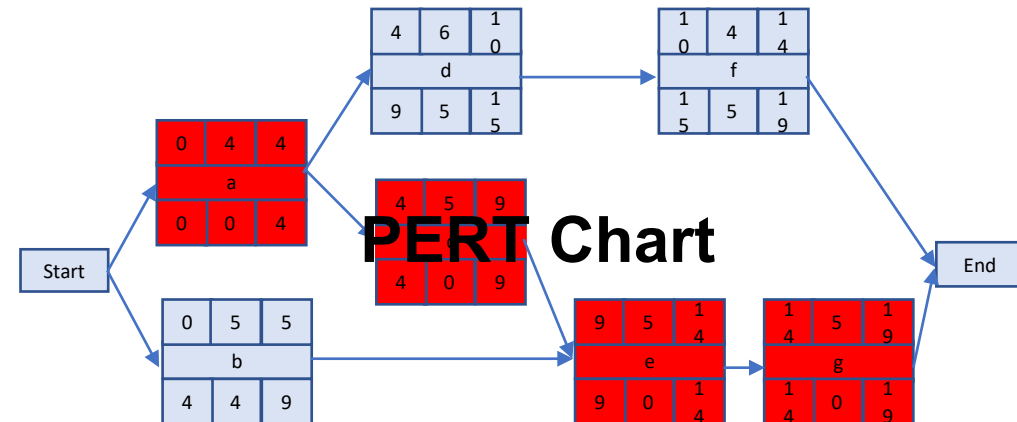


## Work Breakdown Structure



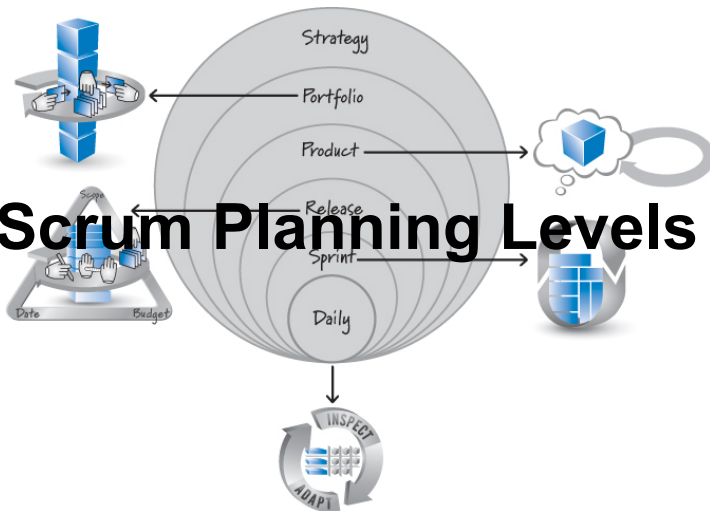
## Gantt Chart

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.



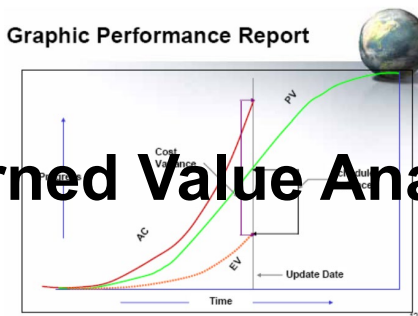
## PERT Chart

## Scrum Planning Levels



## Graphic Performance Report

## Earned Value Analysis



# Module 12– Cost Estimation



Today's lecture

## Formal

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

## Agile

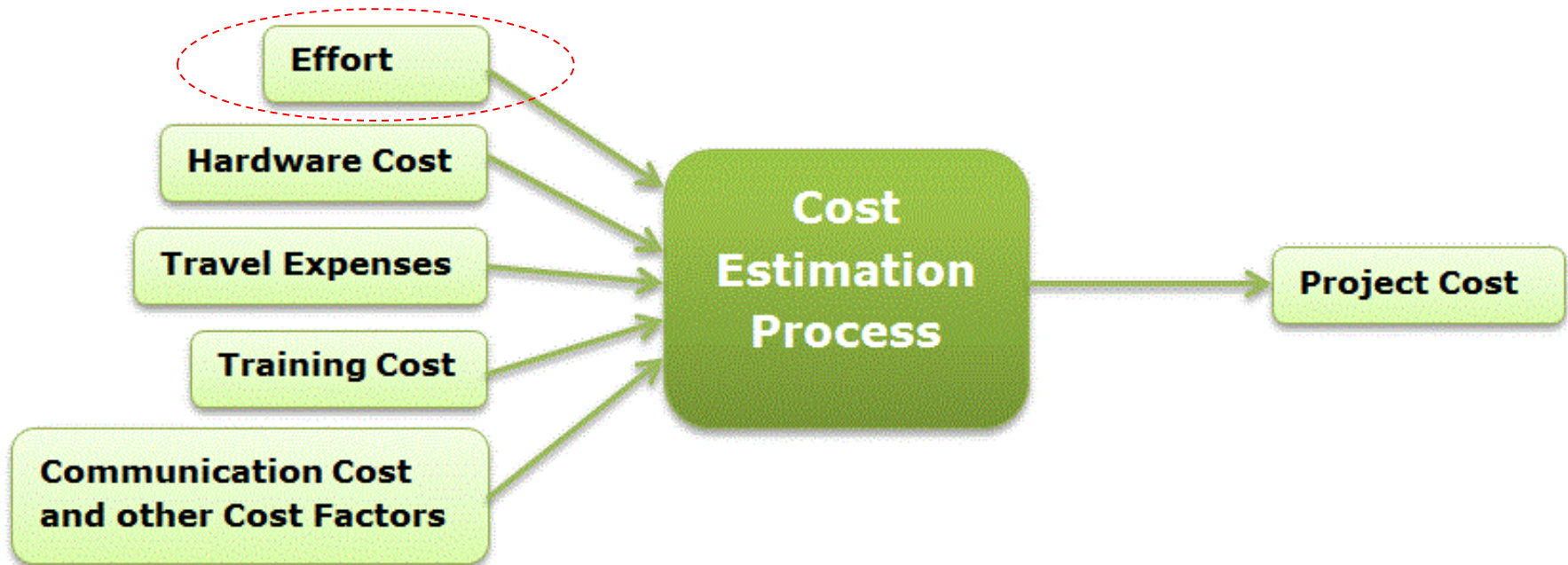
Understand cost estimation techniques used in Agile software development lifecycles



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



# 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



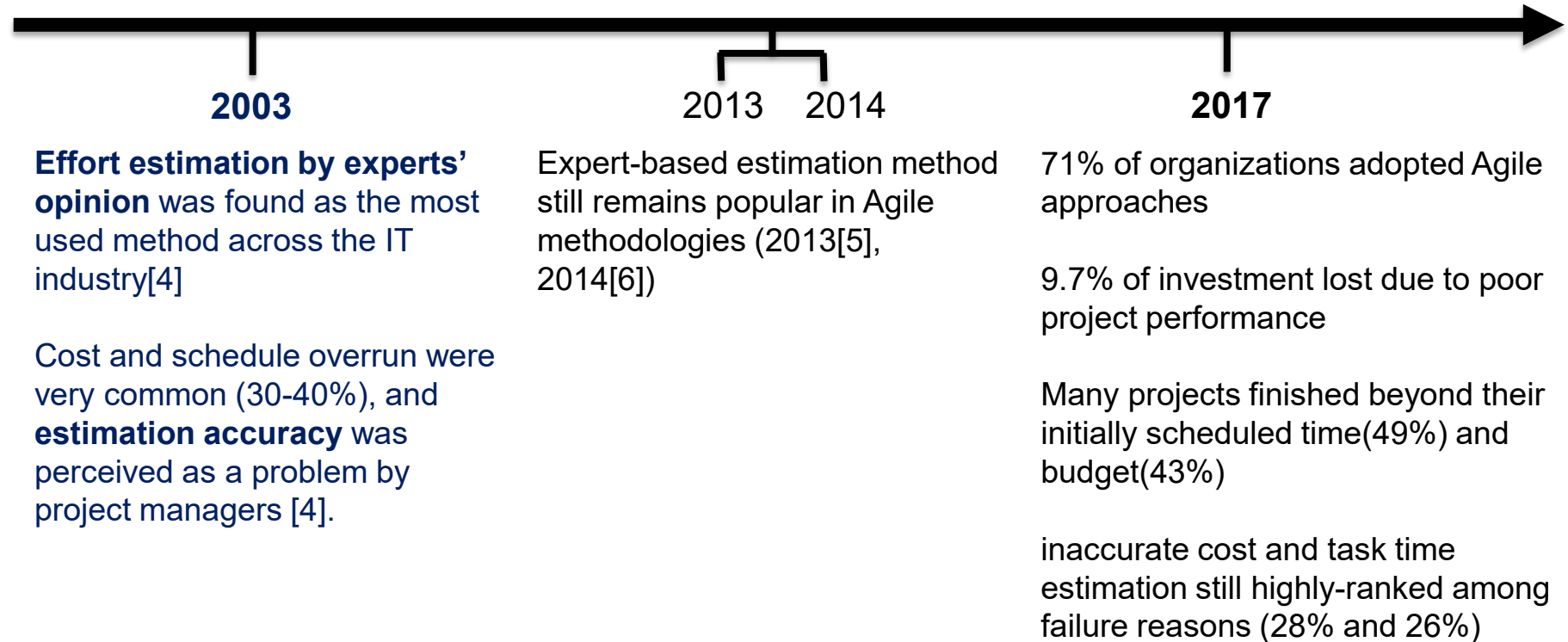
- There is no exact science for cost estimation – it will never be considered as all accurate
- No person can reasonably predict what can go wrong in the project
- Most estimation methods assume things will proceed as expected and simply adds some slack to account for what can go wrong



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



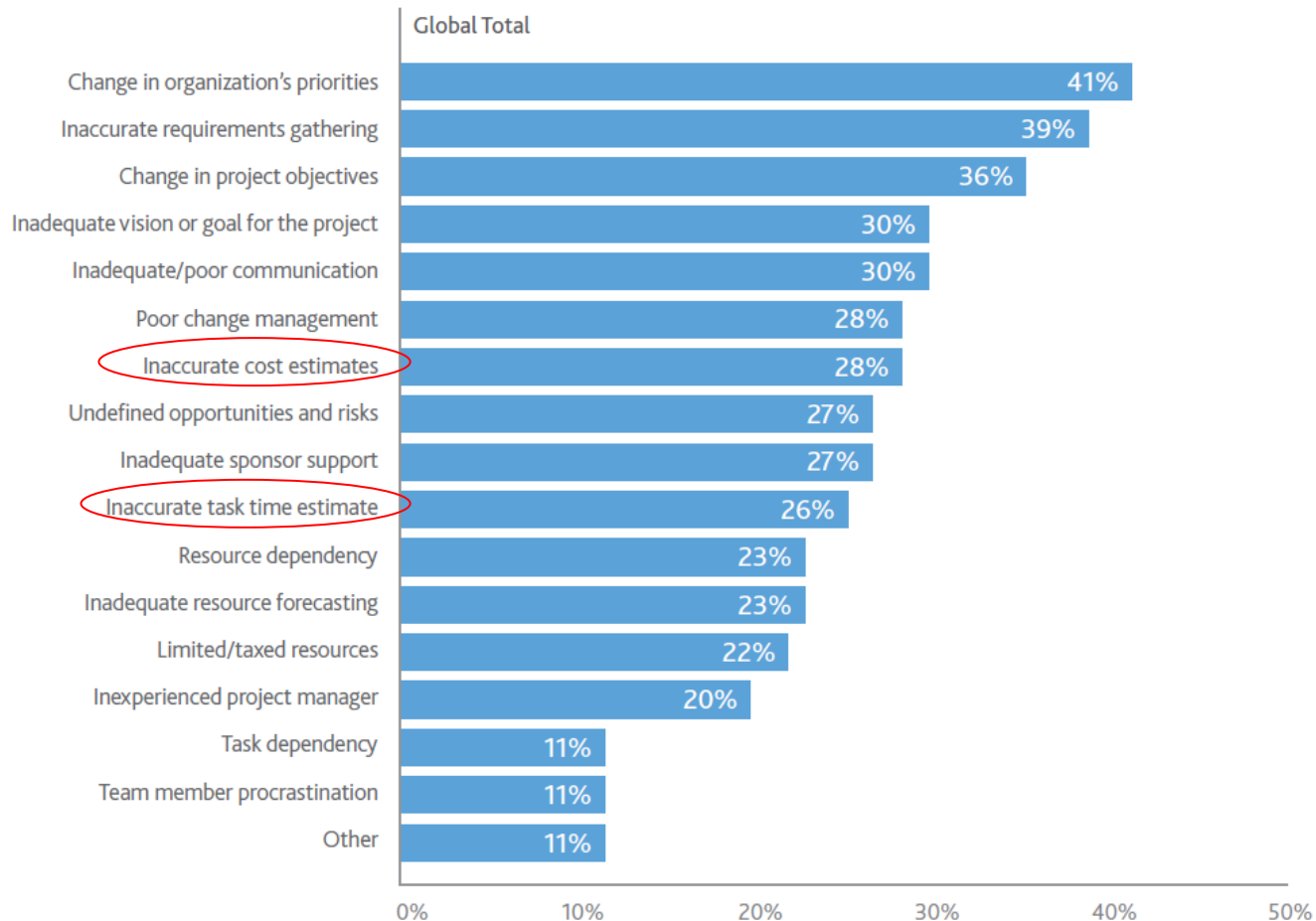
## Why do we need to improve effort estimation?



Source: Literature Review PhD Student Jirat Pasuksmit

MELBOURNE

**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>

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

## 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.



## 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

## 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 development attributes such as stability of requirement, experience of the team



## 5. Algorithmic cost modelling - cont...

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

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***



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

- 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



- 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
<pre># include &lt;stdio.h&gt;  int main() {     printf("\nHello world\n"); }</pre>	<pre>identification division. program-id. hello . procedure division. display "hello world" goback . end program hello .</pre>
Lines of code: 4 (excluding whitespace)	Lines of code: 6 (excluding whitespace)



- 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

- 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





MELBOURNE

- 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)*



- **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

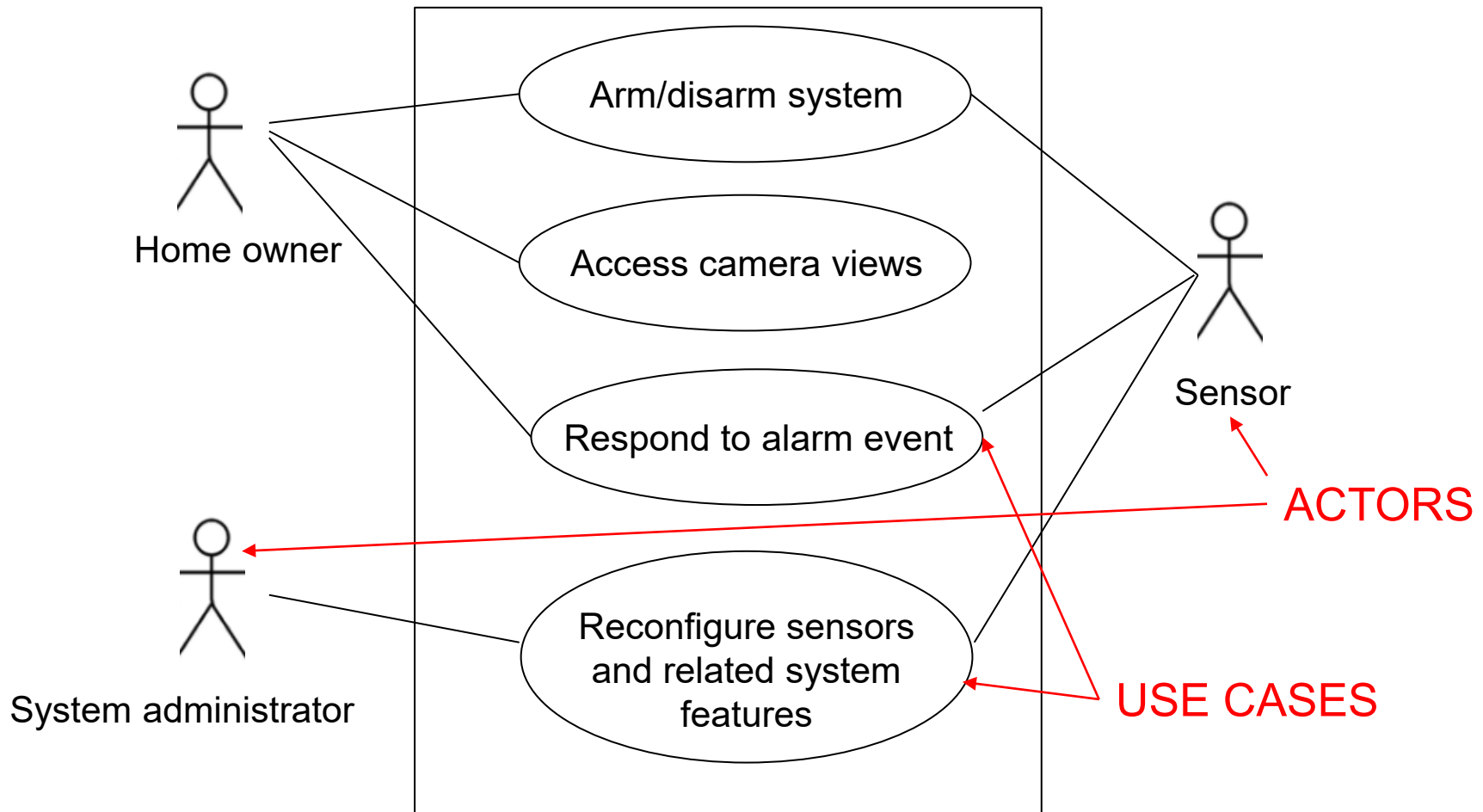
- **Disadvantages of Function Points**
  - A well defined requirements specification is necessary
  - Gaining proficiency is not easy, the learning curve is quite long
  - Could be quite time-consuming thus could be costly

**We will be going through an example**

- 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 cases describe the functionality of the system.
- Use cases model the dialog between the actors and the system.
- Primary purpose is to document functional requirements but also used for testing purposes,

# Use Case Diagram



**Use Case Diagram for a SafeHome Security System**

## 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.

....



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



## 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

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

## 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

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

## 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

$$TCF = 0.6 + TF/100$$

### 3. Compute Technical Complexity Factor (TCF)

Factor	Description	Weight	Score
T1	Distributed system	2.0	0
T2	Response time	1.0	5
T3	End-user efficiency	1.0	4
T4	Internal processing complexity	1.0	3.0
T5	Code reusability	1.0	0
T6	Easy to install	0.5	1
T7	Easy to use	0.5	2
T8	Portability to other platforms	2.0	1
T9	System maintenance	1.0	3
T10	Concurrent/parallel processing	1.0	4
T11	Security features	1.0	5
T12	Access for third parties	1.0	1
T13	End user training	1.0	3

Score each factor between **0 – 5**

- **0 - irrelevant**
- **5 - essential**

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

*S<sub>i</sub>* - Score for the *i*-th factor  
*W<sub>i</sub>*- Weight of the *i*-th factor

$$TCF = 0.6 + TF/100$$

## 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

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



## 5. Compute the final size estimate

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

- **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



- **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



## BREAK

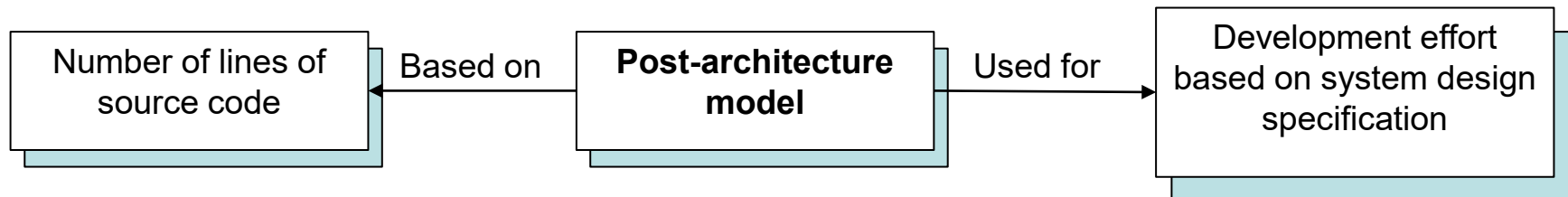
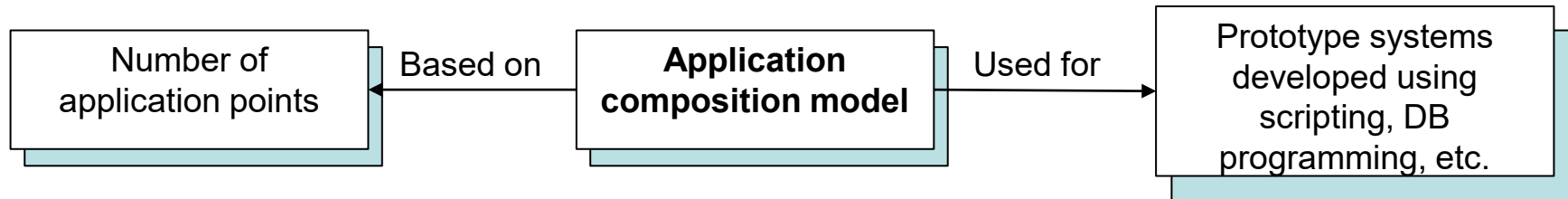
Please return promptly as the  
Lecture will re-start in **5 mins**



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

## WHY COCOMO?

- 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)



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

- 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 development attributes such as stability of requirement, experience of team
- Effort* - total effort measured in person-months

In the COCOMO II early design model:

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

## 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 =  $29.6$   $(200 \times 148 / 1000)$  KSLOC; in Java =  $11$   $(200 \times 55 / 1000)$  KSLOC  
(per 1000 lines of code)

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.

<b>Precedentedness</b>	Familiarity of the application domain: 0 (thoroughly familiar) to 5 (completely unprecedented).
<b>Development flexibility</b>	The level of flexibility in development process, methods, and tools, ranging from 0 (general goals) to 5 (rigorous).
<b>Architecture completed and risks eliminated</b>	20%: 5; 40%: 4; 60%: 3; 80%: 2; 90%: 1; 100%: 0
<b>Team cohesion</b>	The level of interaction within the team. Ranging from 0 (seamless interactions) to 5 (very difficult interactions).
<b>Process maturity</b>	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) ( <b>Quality Management</b> )

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*

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





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.

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: 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:

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

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



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

## 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

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.



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



# Planning Poker

1. Customer reads story.

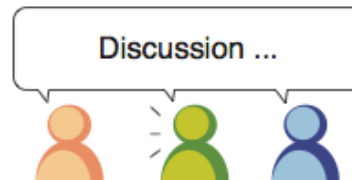


Development team  
asks questions

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/>

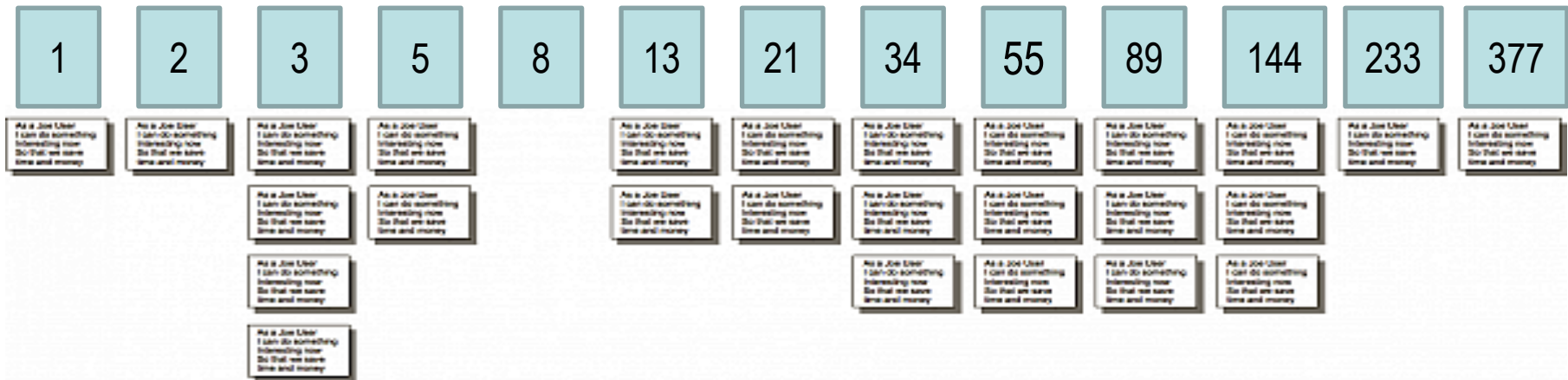


MELBOURNE



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

MELBOURNE



- The team sitting at a table picks a user story card randomly and places it 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

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/>

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

$V$  - 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

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

$T$  - estimate delivery time

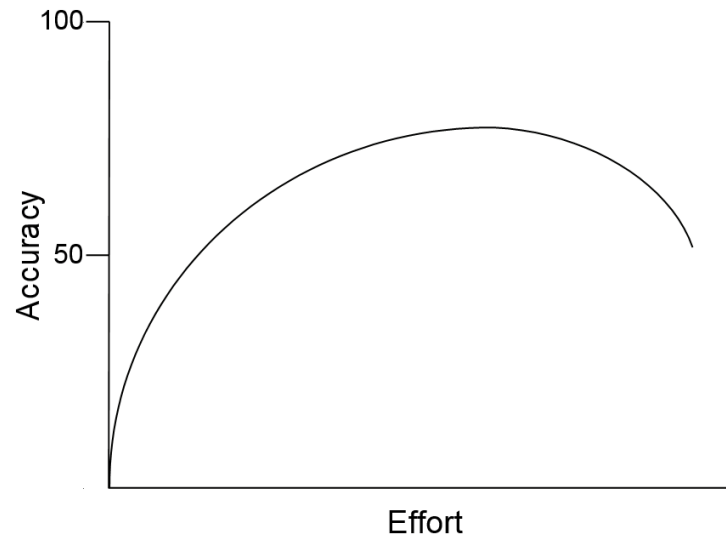
$V$  - velocity

$SP_i$  - 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

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



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...



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>



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)