# Effort Estimation

Version 1.2 – 17/12/2020

# Effort estimation

- The goal is to provide a (tentative) estimate for the effort required to build a system
- General techniques are:
  - Analogy based
  - Expert judgment
  - Metrics based

# Metrics based estimation

Effort = Sw_Size × Team_Productivity

- Different size estimation techniques:
  - Function points
  - Use case points
  - …

# Function Points

- Function Point Analysis, developed by Allan J. Albrecht in the late 1970s
- Several variations
  - ISO/IEC 19761 (COSMIC method),
  - ISO/IEC 20926 (IFPUG method)
  - ISO/IEC 20968 (Mk II method),
  - ISO/IEC 24570 (NESMA method), and
  - ISO/IEC 29881 (FiSMA method)

# COSMIC FP – Principles

- Software interacts with
  - its users across a boundary (interface),
  - and with storage
- User requirements can be mapped into unique functional processes.
- Each functional process consists of sub-processes:
  - data movement or
  - data manipulation

# COSMIC FP – Principles

- A data movement moves a single data group
  - Entry: data from user to system.
  - Exit: data from system to user.
  - Write: data from system to persistent storage.
  - Read: data from persistent storage to system.
- Data group: set of attributes that describe a single object of interest
- Each process is started by its triggering Entry data movement.

# USE CASE POINTS

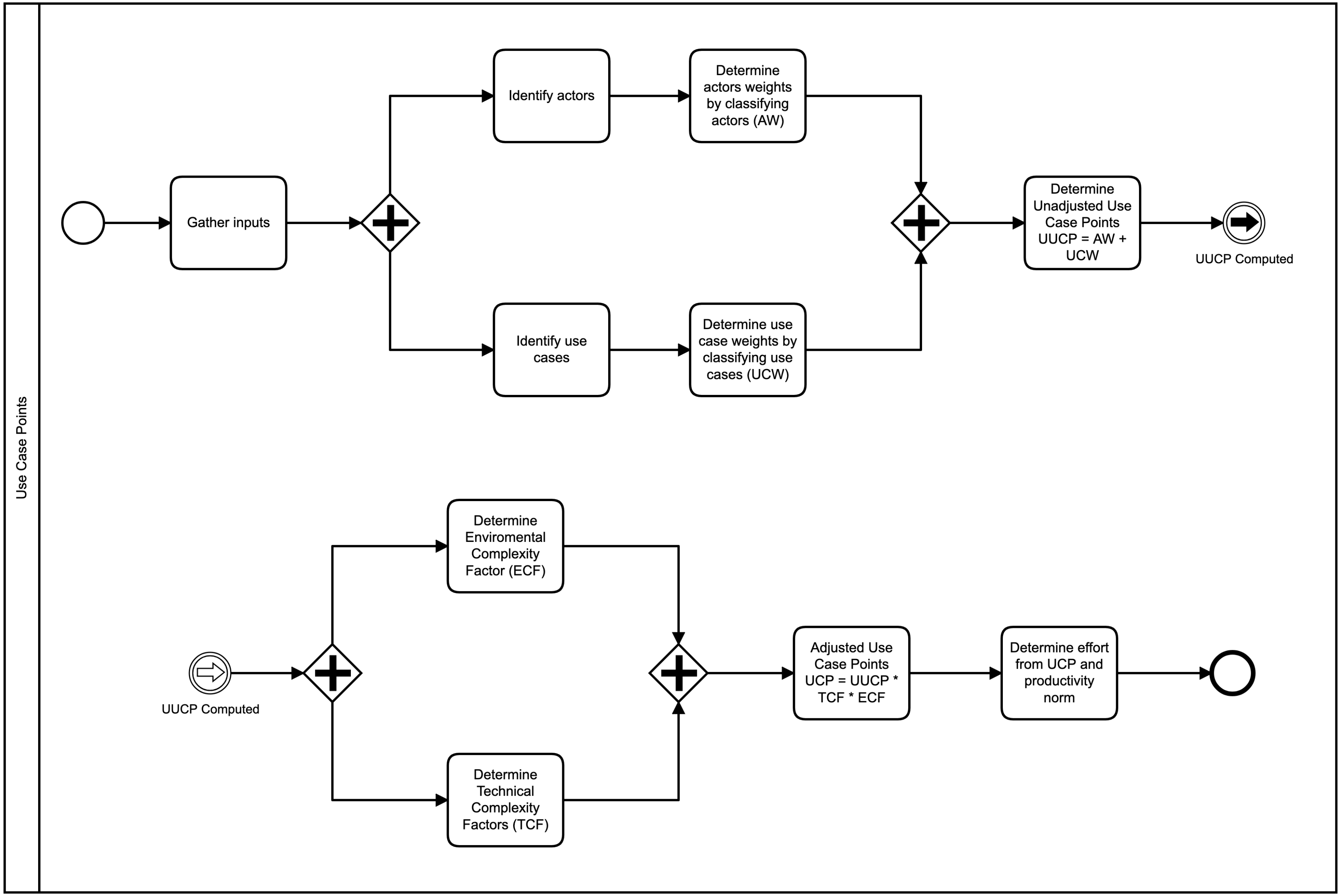
# Use Case Points

- Application size is determined by:
  - Number of actors
  - Number of use cases
  - Contextual factors

# Components

- Technical Complexity Factors (TCF)
- Enviromental Complexity Factors (ECF)
- Productivity norms to determine effort from size

# Process



Use Case Points
Gather inputs
Identify actors
Determine actors weights by classifying actors (AW)
Identify use cases
Determine use case weights by classifying use cases (UCW)
Determine Unadjusted Use Case Points UUCP = AW + UCW
UUCP Computed
UUCP Computed
Determine Enviromental Complexity Factor (ECF)
Determine Technical Complexity Factors (TCF)
Adjusted Use Case Points UCP = UUCP * TCF * ECF
Determine effort from UCP and productivity norm

# Determining Actors Weight

- Identify actors for the system
- Categorize the actors as simple, average and complex
  - A Simple actor represents another system with a defined API.
  - An Average actor is another system interacting through a protocol like TCP/IP or it is a person interacting through a text-based interface (like an ASCII terminal).
  - A complex actor is a user interacting through a GUI interface

# Determining Actors Weight

- Assign weight to each classified actor according to this table:

| Actor Type | Weight Factor |
|---|---|
| Simple | 1 |
| Average | 2 |
| Complex | 3 |

# Determining Use Cases Weight

- Identify use cases for the system
- Determine complexity and hence use case weight based on number of transactions in the use case

# Determining Use Cases Weight

- A transaction is defined as an event occurring between the actor and system, the event being performed completely or not at all

| Use Case Type | No. Transactions | Weight Factor |
|---|---|---|
| Simple | < 4 | 5 |
| Average | 4 – 7 | 10 |
| Complex | > 7 | 15 |

# Unadjusted Use Case Points

- Unadjusted Use Case Points is the sum of actor weights and use case weights:

$$UUCP = AW + UCW$$

- Where:
  - AW is total Actor Weight
  - UCW is total Use Case Weights

# Technical Complexity Factor

| Factor | Description | Weight | Rating (0-5) | TF (W*R) |
|---|---|---|---|---|
| T1 | Distributed System | 2 | | |
| T2 | Response time | 2 | | |
| T3 | End User Efficiency | 1 | | |
| T4 | Complex Internal Processing | 1 | | |
| T5 | Reusable Code | 1 | | |
| T6 | Easy to install | 0.5 | | |
| T7 | Easy to use | 0.5 | | |
| T8 | Cross-platform support | 2 | | |
| T9 | Easy to change | 1 | | |
| T10 | Concurrent | 1 | | |
| T11 | Includes Security Features | 1 | | |
| T12 | Provides Access for 3rd parties | 1 | | |
| T13 | User Training Required | 1 | | |

# T1: Distributed System Required

- The architecture of the solution may be centralized or single-tenant , or it may be distributed (like an n-tier solution) or multi-tenant.
- Higher numbers represent a more complex architecture.

# T2: Response Time Is Important

- The quickness of response for users is an important (and non-trivial) factor.
  - For example, if the server load is expected to be very low, this may be a trivial factor.
- Higher numbers represent increasing importance of response time
  - Search engine would have a high number
  - A daily news aggregator a low number

# T3: End User Efficiency

- Is the application being developed to optimize on user efficiency, or just capability?
- Higher numbers represent projects that rely more heavily on the application to improve user efficiency

# T4: Complex Internal Processing

- Is there a lot of difficult algorithmic work to do and test?
- Complex algorithms (resource leveling, time-domain systems analysis, OLAP cubes) have higher numbers. Simple database queries would have low numbers.

# T5: Reusable Code Is a Focus

- Is heavy code reuse an objective?
  - Code reuse reduces the amount of effort required to deploy a project.
  - It also reduces the amount of time required to debug a project.
    - E.g., a shared library function can be re-used multiple times, and fixing the code in one place can resolve multiple bugs.
- The higher the level of re-use, the lower the number.

# T6: Ease of Installation

- Is ease of installation for end users a key factor?
- The higher the ease required (implying a lower level of competence required from the users), the higher the number.

# T7: Usability

- Is ease of use a primary criteria for acceptance?
- The greater the importance of usability, the higher the number.

# T8: Cross-Platform Support

- Is multi-platform support required?
- The more platforms that have to be supported the higher the value
  - Could be browser versions, mobile devices, or OS (e.g. Windows/OSX/Unix)

# T9: Easy To Change

- Does the customer require the ability to change or customize the application in the future?
- The more change / customization is required in the future, the higher the rating.

# T10: Concurrent

- Will you have to address database locking and other concurrency issues?
  - Concurrency requirements typically bring issues concerning conflicts in data access
- The more attention you have likely to spend to resolving conflicts in the data or application, the higher the value

# T11: Includes Security Features

- Can standard security solutions be leveraged, or must custom code be developed?
- The more custom security work you have to do (field level, page level, or role-based security, for example), the higher the value.

# T12: Access for 3rd parties

- Will the application require the use of third party controls or libraries?
  - Like re-usable code, third party code can reduce the effort required to deploy a solution.
- The more third party code (and the more reliable the third party code), the lower the number.

# T13: User Training Required

- How much user training is required? Is the application complex, or supporting complex activities?
- The longer it takes users achieve a level of mastery of the product, the higher the value.

# Technical Complexity Factor

- TCFactor $= \Sigma$ Tf
  - where Tf is Wt × Rating for each factor

- Tech Complexity Factor

$$\text{TCF} = 0.6 + 0.01 \times \text{TFactor}$$

# Calculating Environmental Complexity Factor

| Factor | Description | Weigth | Rating (0-5) | EF (W*R) |
|---|---|---|---|---|
| F1 | Familiarity With The Project | 1.5 | | |
| F2 | Application Experience | 0.5 | | |
| F3 | Object Oriented Experience | 1 | | |
| F4 | Lead Analyst Capability | 0.5 | | |
| F5 | Motivation | 1 | | |
| F6 | Stable requirements | 2 | | |
| F7 | Part Time Workers | -1 | | |
| F8 | Difficulty of programming language | -1 | | |

# Familiarity With The Project

- How much experience does your team have working in this domain?
  - The domain of the project will be a reflection of what the software is intended to accomplish, not the implementation technology
    - E.g., for an insurance compensation system written in java, you care about the team's experience in the insurance compensation space – not how much java they've written.
- Higher levels of experience get a higher number.

# Application Experience

- How much experience does your team have with the application.
    - This will only be relevant when making changes to an existing application.
- Higher numbers represent more experience.
    - For a new application, everyone’s experience will be 0.

# OO Programming Experience

- How much experience does your team have at OO?
  - It can be easy to forget that many people have no object oriented programming experience if you are used to having it.
  - A user-centric or use-case-driven project will have an inherently OO structure in the implementation.
- Higher numbers represent more OO experience.

# Lead Analyst Capability

- How knowledgeable and capable is the person responsible for the requirements?
  - Bad requirements are the number one killer of projects – the Standish Group reports that 40% to 60% of defects come from bad requirements.
- Higher numbers represent increased skill and knowledge.

# Motivation

- How motivated is your team?
  - Consultants working at the project in the context of a body rental contract will likely be little motivated
- Higher numbers represent more motivation.

# Stable Requirements

- Changes in requirements can cause increases in work.
  - The way to avoid this is by planning for change and instituting a timing system for managing those changes.
  - Most people don’t do this, and some rework will be unavoidable.
- Higher numbers represent less change (or a more effective system for managing change).

# Part Time Staff

- How much of the team staff is working part-time?
  - Often outside consultants, and developers are splitting their time across projects.
  - Context switching and other intangible factors make these team members less efficient
- Higher numbers reflect team members that are mostly part time
- Note: weight for is factor is negative

# Difficult Programming Language

- How difficult is the language for the members of the development team
  - Harder languages represent higher numbers.
  - Difficulty is in the eye of the be-coder
    - Java might be difficult for a Fortran programmer.
    - It is difficulty for the team members, not abstract difficulty.
- Note: weight for is factor is negative.

# Environmental Complexity Factor

- EFactor = Σ Ef
  - where Ef is Wt × Rating for each factor

- Environmental Complexity Factor

$$ECF = 1.4 + (-0.03 \times EFactor)$$

# Adjusted Use Case Points

- Adjusted Use Case Points (UCP) is:

$$\mathrm{UCP} = \mathrm{UUCP} \times \mathrm{TCF} \times \mathrm{ECF}$$

  - where UUCP is unadjusted Use Case Points

# Effort Calculation in person hours

- Let the number of factors below 3 in F1-F6 in the Environment Factor Table be n1

- Let number of factors above 3 in F7-F8 be n2.

- If n1+n2 <= 2 10-20 person hrs per UCP
  1.25-2.5 person days per UCP
  3-4 14-28 person hrs per UCP
  1.75-3.5 person days per UCP
  > 4 18-36 person hrs per UCP
  2.25-4.5 person days per UCP

# UCP – Key Takeaways

- The Use case points method can produce estimates close to actual effort in several projects.
- This indicates that the use case points method may support expert knowledge when a use case model for the project is available.
- Some tailoring to the company may be useful to obtain maximum benefits from the method.
  - Customize the productivity norm for the organization

# EXAMPLE

# Use Case – Waiting List

- A well-known restaurant in a shopping center uses a system for the management of the waiting list.
- When a customer arrives, the waiter, once he knows the number of people in the group, check the estimated waiting time for a table on the system

# Use Case Narrative

- Use Case: Waiting List
- Level: User-goal
- Intention in context: estimate the waiting time
- Primary Actor: Waiter
- Stakeholder interest: customer wants to have a precise waiting time

# Use Case Narrative

- Main Success Scenario
  1. Waiter asks for a time estimate
  2. System requires the number of people
  3. Waiter enters the number of people
  4. System provides the estimate
- Extensions:
  - 4a no available tables, use case fails

# Use Case  Waiting List – AW

- Determine Actor Weight
    - User Waiter actor
    - GUI actor -> complex
    - AW = 3

# Use Case  Waiting List – UCW

- Transactions:
  1. Waiter asks for a time estimate
  2. System requires the number of people
  3. Waiter enters the number of people
  4. System provides the estimate

- Total of 3 transactions (waiter-system)
  - Simple use case, UCW = 5

# Use Case  Waiting List – UUCP

- UUCP = AW + UCW

$$3 + 5 = 8$$

# Calculating Technical Complexity Factor

| Factor | Description | Weight | Rating (0-5) | TF (W*R) |
|---|---|---|---|---|
| T1 | Distributed System | 2 | 1 | 2.0 |
| T2 | Response time | 2 | 2 | 4.0 |
| T3 | End User Efficiency | 1 | 3 | 3.0 |
| T4 | Complex Internal Processing | 1 | 0 | 0.0 |
| T5 | Reusable Code | 1 | 1 | 1.0 |
| T6 | Easy to install | 0.5 | 1 | 0.5 |
| T7 | Easy to use | 0.5 | 1 | 0.5 |
| T8 | Cross-platform support | 2 | 0 | 0.0 |
| T9 | Easy to change | 1 | 1 | 1.0 |
| T10 | Concurrent | 1 | 0 | 0.0 |
| T11 | Includes Security Features | 1 | 2 | 2.0 |
| T12 | Provides Access for 3rd parties | 1 | 0 | 0.0 |
| T13 | User Training Required | 1 | 0 | 0.0 |
|  |  |  | TOTAL | 14.0 |

# Environmental Complexity Factor

| Factor | Description | Weight | Rating (0–5) | EF (W*R) |
|---|---|---|---|---|
| F1 | Familiarity With The Project | 1.5 | 3 | 4.5 |
| F2 | Application Experience | 0.5 | 4 | 2.0 |
| F3 | Object Oriented Experience | 1 | 4 | 4.0 |
| F4 | Lead Analyst Capability | 0.5 | 5 | 2.5 |
| F5 | Motivation | 1 | 5 | 5.0 |
| F6 | Stable requirements | 2 | 5 | 10.0 |
| F7 | Part Time Workers | -1 | 0 | 0.0 |
| F8 | Difficulty of programming language | -1 | 0 | 0.0 |
| | | | TOTAL | 28.0 |

# Use Case  Waiting List – TCF, ECF

TCF = 0.6 + (0.01 * 14) = 0.74

ECF = 1.4 + (-0.03 * 28) = 0.56

# Use Case  Waiting List – Effort

- UUCP = 8
- TCF = 0.74
- ECF = 0.56
- UCP = 8 * 0.74 * 0.56 = 3.32
- Productivity norm:
  - n1 = 0
  - n2 = 0
- 10 Phrs / UCP = 1.25 Pdays / UCP
- Effort = 3.32 * 10 = 33.15  Phrs

# Average person costs

- Junior Developer = 250 € per day
- Senior Developer = 500 € per day
- Junior Analyst = 300 € per day
- Senior Analyst = 600  € per day

# System cost

- Assuming an average developer cost of 300 €/day

- Effort: 33.15 Phrs = 4.14 Pdays
- Cost = 1 243.20 €