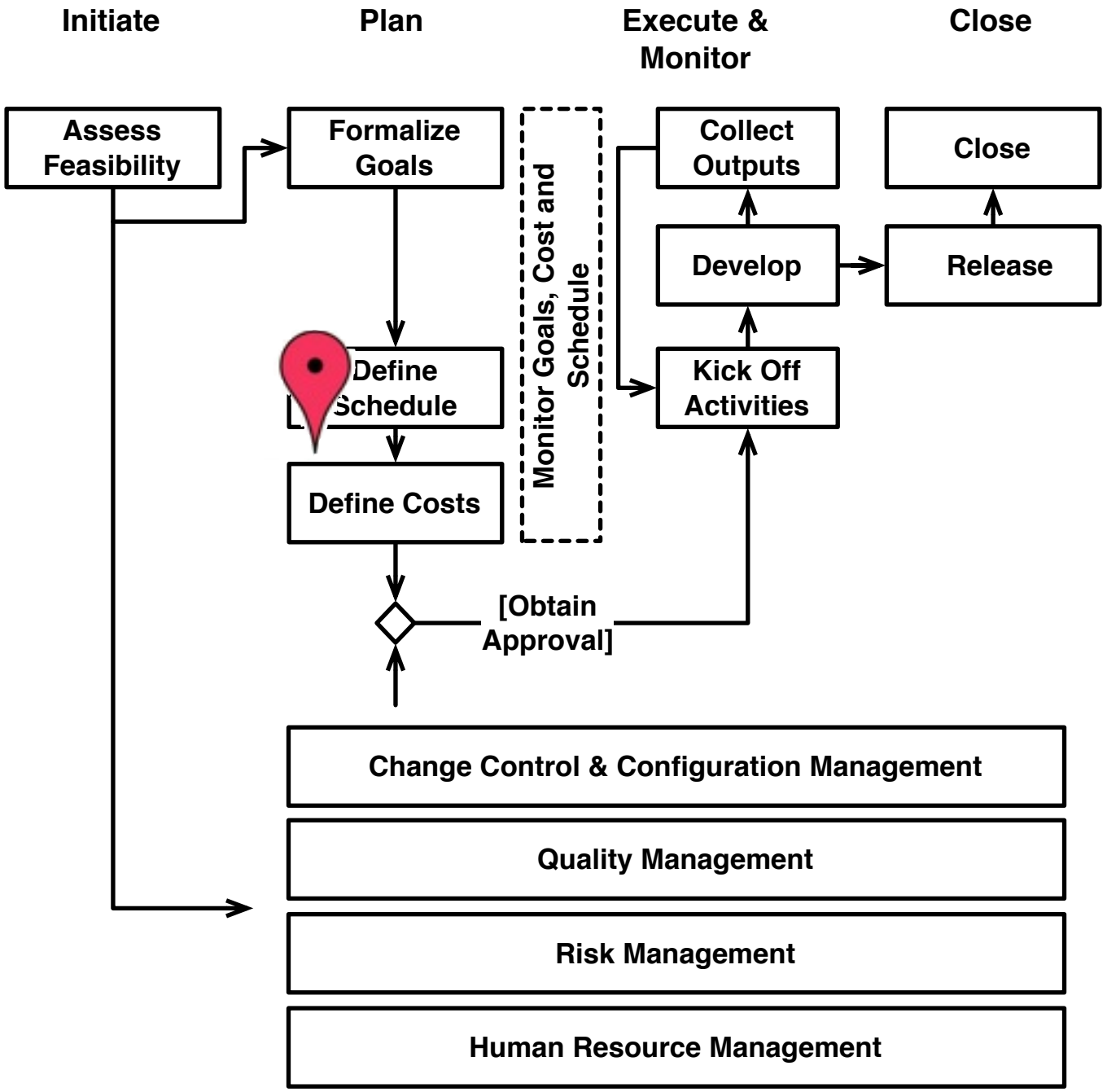
Algorithmic Techniques for Estimation: Function Points


Initiate
Plan
Execute & Monitor
Close
Assess Feasibility
Formalize Goals
Monitor Goals, Cost and Schedule
Collect Outputs
Close
Develop
Release
Define Schedule
Kick Off Activities
Define Costs
[Obtain Approval]
Change Control & Configuration Management
Quality Management
Risk Management
Human Resource Management

Goals of the Unit

- Understand the basic principles behind algorithmic techniques and, more in details, about Function Point Estimation

Recap: Principles of Algorithmic Techniques

Introduction

- Goal: find a way to systematically determine the effort (duration) required for an (arbitrary) task/project
- Ideally:
  - Identify a set of **measurable** characteristics of a project that determine the project's **effort/duration**
  - Define a function that, given the characteristics mentioned above, computes the **effort/duration**

$$f(x_1, \ldots, x_n) = e$$

Problem: how do you find f, x_1, ..., x_n?

Solution

- Look at existing projects/datasets; each project is represented by a vector:

  $< a_1, \ldots, a_n, effort >$

- Find correlations between (some of the) variables in the datasets:

  $f(a_{1, \ldots,} a_k) \propto effort$

- Find appropriate measurement means for the variables at the beginning of a project (so that we can apply the function to a new project)

Productivity Metrics

- There are two main classes of metrics to measure the effort required to develop an application:
  - **Function-based metrics:** it measure the number and complexity of the functions to develop
  - **Size-based metrics:** it measures the size of an application (in terms of lines of code, for instance)

Function Based Metrics

- Advantages:
  - Functional complexity can be estimated when requirements are ready
  - It does not depend upon a specific programming language
- Disadvantages and critiques:
  - More difficult to link to productivity
  - More difficult to measure than size metrics

Size Metrics

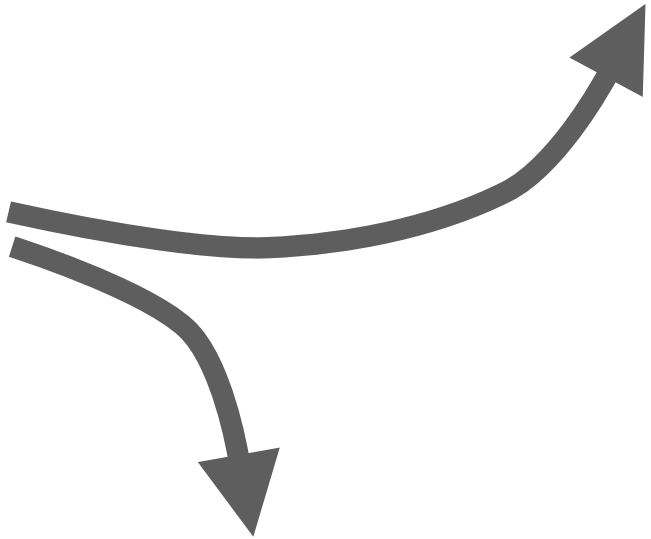
- Advantages:
  - They can be precisely measured (once you have the source code and adopt coding standards)
  - Measuring tools available
  - More easily linkable to productivity
- Disadvantages and critiques:
  - They are know precisely when the system is built (… a bit too late if you want to use then for estimation)
  - Sensitive to programming language, coding standards
  - They do not measure the complexity of code

What is a line of code?

```
System.out.println("Hello world");
```

```
if (a == 1) {
return true;
}
else {
return false;
}
```

```
return a == 1? true : false
```



```
if (a == 1) { return true; } else { return false; }
```

Productivity Examples

- **From Sommerville, Software Engineering:**
  - **Real-time embedded systems:**
    * 40-160 LOC/P-month
  - **Systems programs:**
    * 150-400 LOC/P-month
  - **Commercial applications:**
    * 200-900 LOC/P-month
- **An example (wikipedia):**
  - RH7.1 is about 30 MSLOC, it would have required 8000 person years if developed according to conventional proprietary practices

Function Points

Function Points

- Based on a combination of program characteristics derivable from the requirements (function-based metrics)
- First proposed by Albrecht in the seventies
- First application in business systems
- Evolved to embrace a wide range of systems and applications.
- Large user base and an IFPUG, which trains in the application of the technique

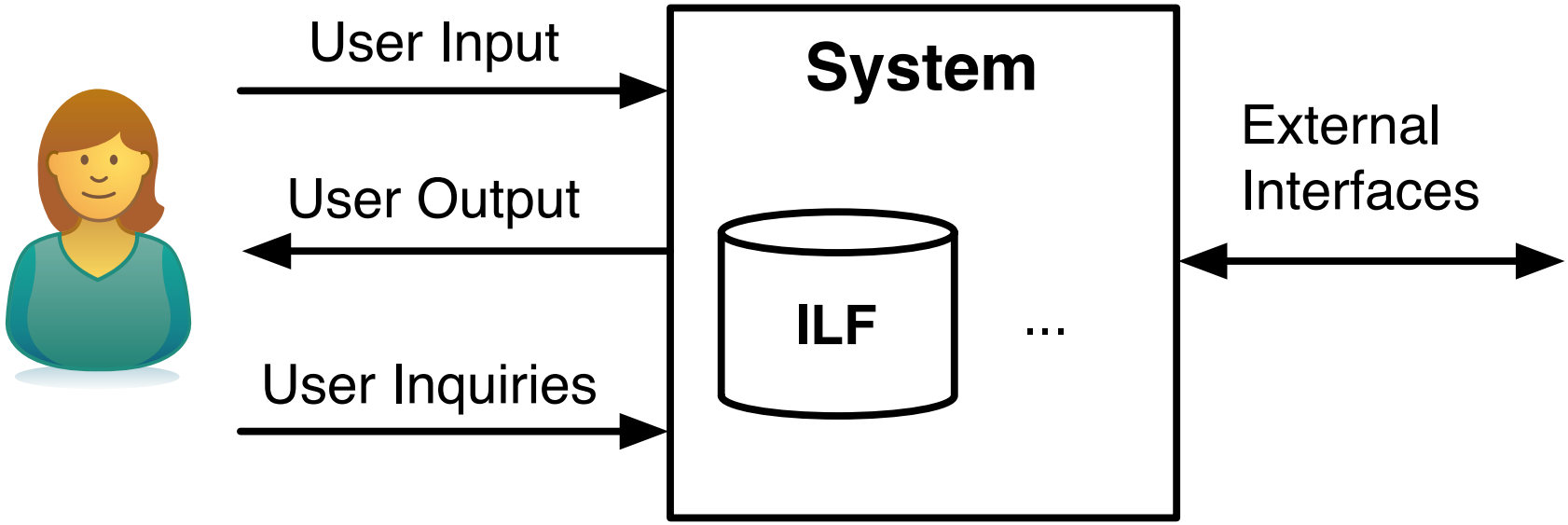
The Process at a Glance

1. Define the **boundary** of your analysis
2. Measure a set of characteristics related to the system's **functional requirements** and use a formula to get the **Unadjusted Function Points** (UFP)
3. Measure a set of characteristics related to the **non-functional requirements**: compute the **value adjustment factor** (VAF)
4. Compute the **(Adjusted) Function Points** (FP): FP = f(VAF, UFP)

UFP: Inputs

- **User inputs:** the total number of user inputs and their complexity
- **User outputs:** the total number of user outputs and their complexity
- **User inquiries:** the total number of user inputs that generate a software response, such as word count, search result, or software status
- **Internal Logical Files:** the total number of files created and used dynamically by the system
- **External interfaces:** the total number of external files that connect with the software to an external system. For instance, if the software communicates with a device, 1 external interface

Count number of each, classified as easy, average, or complex


User Input
User Output
User Inquiries
System
ILF
...
External
Interfaces

VAF: Inputs

The system requires reliable saves and backups?

The system requires communication of data?

Elaboration is distributed?

Performance is a critical factor?

The system will work on a well known environment?

The system requires live data entry?

Data entry is articulated in complex transactions?

ILF are updated on-line?

Data are complex?

Internal elaboration is complex?

Code must be reusable?

Project includes also installation and data conversion?

System must be used in different organizations?

System must be easy to maintain and simple to use?

Answers from 0 (irrelevant) to 5 (extremely relevant)

Function Points Formulas

$$UFP = \sum_{i=1}^{5} \begin{bmatrix} k_i^E & k_i^A & k_i^C \end{bmatrix} \cdot \begin{bmatrix} n_i^E \\ n_i^A \\ n_i^C \end{bmatrix}$$

$$FP = UFP \cdot (0.65 + 0.01 \cdot \sum_{i=1}^{14} C_i)$$

Formulas Comment

- **UFP, unadjusted function points** is the weighted sum of external inputs, external outputs, ...
- The value of the constants (**k^E, k^A, k^C**) is provided by the method
- **FP, (adjusted function points)** takes into account the general system characteristics (the 14 questions)
- FP in in the range **[0.65 UFP, 1.35 UFP]**

UFP: Program Characteristics

- The counting of UFPs is performed by looking at the requirements
- Analysts look for the following elements:
  - DET: data element types
  - FTR: file type referenced
  - RET: record element type
- Tables convert the numbers above into the five characteristics
- Training is required to apply the technique accurately

ILF/EIF		DET	
RET	1..19	20..50	> 50
0..1	Low	Low	Average
2..5	Low	Average	High
>5	Average	High	High

UI		DET	
FTR	1..4	5..15	> 15
0..1	Low	Low	Average
2	Low	Average	High
> 2	Average	High	High

UO/EQ		DET	
FTR	1..5	6..19	> 19
0..1	Low	Low	Average
2..3	Low	Average	High
>3	Average	High	High

FP: Program Characteristics

- Joel Henry (Introduction to SPM) proposes a “practical” approach to the counting:
  - UI: number of dialog boxes classified as simple, average, and complex
  - UO: number of reports, classified as simple, average, and complex
  - UI: total number of user inputs that generate a software response
  - ILF: number of files and complex internal data structures, classified as simple, average, and complex
  - EIF: number of external files and connected device classified as simple, average, and complex
- Rule of the thumb, it provides a rough estimation

What do you do once you have FP?

- An FP estimation measures with a number the “size” of a system which has to be built
- You can now:
  - Relate FP directly to effort, through productivity measures, if your organization keeps this kind of data:
    * FP per man month
    * FP per calendar month
  - Convert FP to SLOC and use another estimation method (e.g. COCOMO)
  - Use it to cross-check estimations provided by other types of estimations

Function Points Productivity Metrics

- Each company should start its own productivity measuring program
- Some “general” data and studies available (see, e.g., http://www.softwaremetrics.com/Articles/estimatingdata.htm)
- Productivity changes with system size (http://www.softwaremetrics.com/Articles/estimatingdata.htm):
  - 1.3 hours/FP for a system of 50 FP
  - 12.1 hours/FP for a system of about 7000 FP
  - 133.6 hours/FP for a system of about 15000 FPs

Relationship FP/SLOC

- FP can also be related to SLOC through FP/SLOC metrics
- This allows to:
  - Apply other algorithmic techniques (e.g. COCOMO)
  - Measure progress using SLOCs

Source: http://www.qsm.com/resources/function-point-languages-table

Language	Avg	Median	Low	High
ABAP (SAP)	**18**	**18**	**16**	**20**
Access *	36	38	15	47
Ada	154	-	104	205
Advantage	38	38	38	38
Assembler *	209	203	91	320
C *	**148**	**107**	**22**	**704**
C++ *	**59**	**53**	**20**	**178**
C# *	**58**	**59**	**51**	**66**
Clipper *	40	39	26	53
COBOL *	80	78	8	400
ColdFusion	68	56	52	105
Cool:Gen/IEF *	37	35	10	180
Culprit	51	-	-	-
Datastage	67	79	16	85
DBase IV	52	-	-	-
Easytrieve+	33	34	25	41
Excel	47	46	31	63
FORTRAN	90	118	35	-
J2EE *	57	50	50	67
Java *	**55**	**53**	**9**	**214**

FP: Final Considerations

- **Disadvantages:**
  - it requires specific training to be applied correctly
  - it can introduce rigidity in the development process (especially if the technique is used for pricing: consider a change to the requirements)
- **Advantages:**
  - it provides an “objective” measure of the complexity of a system and it can be used to agree on pricing
  - it can be computed after requirements! (the development effort of a system more difficult to estimate than writing requirements)
  - the analysis, even if conducted naively, allows one to get a better grasping of the system to build