Software Project Management

- The Project
- Common Sense Approach
- Critical Software Practices
- Software Project Plan
- The W5HH Principle
- To Get to the Essence of a Project - W5HH Approach
- Few Keywords about Software Measures
- Introduce the necessity for software metrics
- Differentiate between process, project and product metrics
- Compare and contrast Lines-Of-Code (LOC) and Function Point (FP) metrics
- Why opt for FP based metrics
- Computing Function Points

The Project

- Projects get into jeopardy / trouble / risk when …
  - Software people don’t understand their customer’s needs.
  - The product scope is poorly defined.
  - Changes are managed poorly.
  - The chosen technology changes.
  - Business needs change [or are ill-defined].
  - Deadlines are unrealistic.
  - Users are resistant.
  - Sponsorship is lost [or was never properly obtained].
  - The project team lacks people with appropriate skills.
  - Managers [and practitioners] avoid best practices and lessons learned.

Common-Sense Approach

- *Start on the right foot.* This is accomplished by working hard (very hard) to understand the problem that is to be solved and then setting realistic objectives and expectations.
- *Maintain momentum. The* project manager must provide incentives to keep turnover of personnel to an absolute minimum, the team should emphasize quality in every task it performs, and senior management should do everything possible to stay out of the team's way.
- *Track progress.* For a software project, progress is tracked as work products (e.g., models, source code, sets of test cases) are produced and approved (using formal technical reviews) as part of a quality assurance activity.
- *Make smart decisions.* In essence, the decisions of the project manager and the software team should be to "keep it simple."
- *Conduct a postmortem analysis.* Establish a consistent mechanism for extracting lessons learned for each project. Evaluate plan, schedule, analysis of project, customer feedback, etc in written form.

Critical Software Practices

The Airlie Council has developed a list of “**critical software practices for performance-based management.”**
These practices are “consistently used by, and considered critical by, highly successful software projects and organizations whose ‘bottom line’ performance is consistently much better than industry averages” [AIR99].

In an effort to enable a software organization to determine whether a specific project has implemented critical practices, the Airlie Council has developed a set of QuickLook” questions [AIR99] for a project:

Formal risk management. What are the top ten risks for this project? For each of the risks, what is the chance that the risk will become a problem and what is the impact if it does?

Empirical cost and schedule estimation. What is the current estimated size of the application software (excluding system software) that will be delivered into operation? How was it derived?

Metric-based project management. Do you have in place a metrics program to give an early indication of evolving problems? If so, what is the current requirements volatility?

Earned value tracking. Do you report monthly earned value metrics? If so, are these metrics computed from an activity network of tasks for the entire effort to the next delivery?

Defect tracking against quality targets. Do you track and periodically report the number of defects found by each inspection (formal technical review) and execution test from program inception and the number of defects currently closed and open?

People-aware program management. What is the average staff turnover for the past three months for each of the suppliers/developers involved in the development of software for this system?

If a software project team cannot answer these questions or answers them

SOFTWARE PROJECT PLAN :

The software project plan mainly will be a consequence of function and performance allocations performed as part of the systems engineering study of the project planning phase. Estimation is accomplished using one of a number of techniques that also rely on historical productivity data as well as the methodology that is chosen.

SCOPE : Project objectives, Major functions, Development Scenario / Development platforms.

RESOURCES : Human Resources, Hardware Resources & Software Resources, Availability windows.

GRAY AREAS : Identification of Gray Areas, Efforts required from buyer of software and from supplier of software to estimate, evaluate and resolve Gray Areas.

COSTS : Any differences between the buyer and supplier of software can be reviewed by the steering Committee .
After the estimates are presented, the actual cost can be worked out by applying these estimates on the rates prevailing as fixed and escalation terms. These will be discussed in commercial terms. Here, at this point of software project plan, the estimates of the efforts will be signed off by both supplier of software and the buyer of software.

SCHEDULE : Task network, Gant chart / Bar chart, Task resource table

THE W5HH PRINCIPLE

- In an excellent paper on software process and projects, Barry **Boehm** [BOE96] states:

- “you need an organizing principle that scales down to provide simple [project] plans for simple projects.”

  Boehm suggests an approach that addresses project objectives, milestones and schedules, responsibilities, management and technical approaches, and required resources.

- He calls it the **WWWWWHH** principle, after a series of questions that lead to a definition of key project characteristics and the resultant project plan:

To Get to the Essence of a Project - W^5HH Approach

- **Boehm** suggests an approach(W^5HH) that addresses project objectives, milestones and schedules, responsibilities, management and technical approaches, and required resources.

- Why is the system being developed?
  - Enables all parties to assess the validity of business reasons for the software work
- What will be done?
  - Establish the task set that will be required.
- When will it be accomplished?
  - Project schedule to achieve milestone.
- Who is responsible?
  - Role and responsibility of each member.
- Where are they organizationally located?
  - Customer, end user and other stakeholders also have responsibility.
- How will the job be done technically and managerially?
  - Management and technical strategy must be defined.
- How much of each resource is needed?
  - Develop estimation.

It is applicable regardless of size or complexity of software project

The Essence of a Project - W^{5}HH Approach

Why is the system being developed? The answer to this question enables all parties to assess the validity of business reasons for the software work. Stated in another way, does the business purpose justify the expenditure of people, time, and money?

What will be done, by when?
The answers to these questions help the team to establish a project schedule by identifying key project tasks and the milestones that are required by the customer.

Who is responsible for a function? Earlier in this chapter, we noted that the role and responsibility of each member of the software team must be defined. The answer to this question helps accomplish this.

Where are they organizationally located? Not all roles and responsibilities reside within the software team itself. The customer, users, and other stakeholders also have responsibilities.

How will the job be done technically and managerially? Management and technical strategy must be defined.

How much of each resource is needed? Develop estimation.

Few Keywords about software measures

- Measure: Quantitative indication of the extent, amount, dimension, or size of some attribute of a product or process. A single data point
- Metrics: The degree to which a system, component, or process possesses a given attribute. Relates several measures (e.g. average number of errors found per person hour)
- Indicators: A combination of metrics that provides insight into the software process, project or product
- Direct Metrics: Immediately measurable attributes (e.g. line of code, execution speed, defects reported)
- Indirect Metrics: Aspects that are not immediately quantifiable (e.g. functionality, quantity, reliability)
- Faults:
  - Errors: Faults found by the practitioners during software development
  - Defects: Faults found by the customers after release

Introduce the necessity for software metrics

Against:

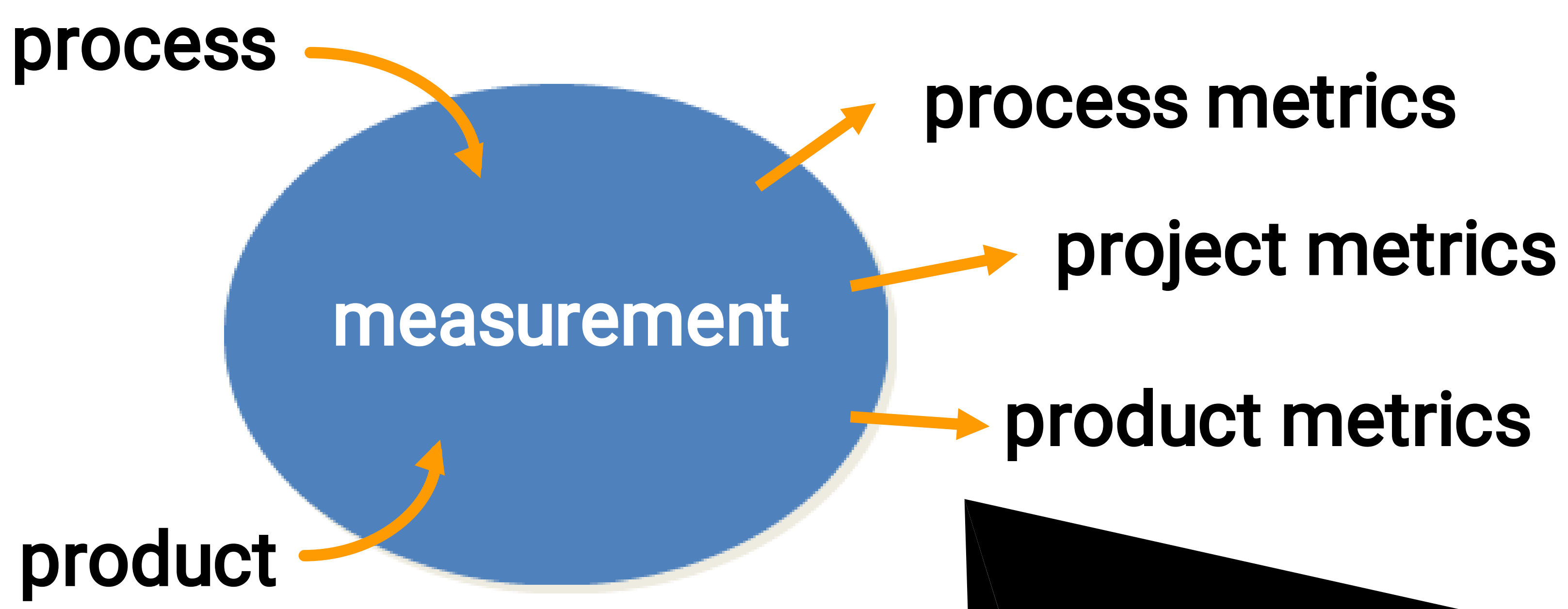
Collecting metrics is too hard … it's too time consuming … it's too political … they can be used against individuals … it won't prove anything

For:

In order to characterize, evaluate, predict and improve the process and product a metric baseline is essential.

"*Anything that you need to quantify can be measured in some way that is superior to not measuring it at all*" Tom Gilb

Differentiate between process, project and product metrics



What do we use as a basis?

- size?
- function?

"Not everything that can be counted counts, and not everything that counts can be counted." - Einstein

Process Metrics

- Focus on quality achieved as a consequence of a repeatable or managed process. Strategic and Long Term.
- Statistical Software Process Improvement (SSPI). Error Categorization and Analysis:
    - All errors and defects are categorized by origin
    - The cost to correct each error and defect is recorded
    - The number of errors and defects in each category is computed
    - Data is analyzed to find categories that result in the highest cost to the organization
    - Plans are developed to modify the process

Project Metrics

- Used by a project manager and software team to adapt project work flow and technical activities. Tactical and Short Term.
- Purpose:
  - Minimize the development schedule by making the necessary adjustments to avoid delays and mitigate problems
  - Assess product quality on an ongoing basis
- Metrics:
  - Effort or time per SE task
  - Errors uncovered per review hour
  - Scheduled vs. actual milestone dates
  - Number of changes and their characteristics
  - Distribution of effort on SE tasks

Product Metrics

- Focus on the quality of deliverables
- Product metrics are combined across several projects to produce process metrics
- Metrics for the product:
  - Measures of the Analysis Model
  - Complexity of the Design Model
    1. Internal algorithmic complexity
    2. Architectural complexity
    3. Data flow complexity
  - Code metrics

Compare and contrast Lines-Of-Code (LOC) and Function Point (FP) metrics

- Both FP and LOC are units of measurement for software size.
- The size of a software that is subject to development is required in order to come up with accurate estimates of effort, cost and duration of a software project.
- Most parametric estimation models such as COCOMO accept size expressed in either FP or LOC as input.
- Size expressed using the FP metric stay constant regardless of which programming language or languages are used.
- Since the industry has more than 700 programming languages and almost every application uses multiple languages, the consistency of FP metric allow economic studies that are not possible using LOC metric.
- Function points can be seen as a universal IT-currency converter, giving a synthetic measure of the size of a software.
- For instance, it takes about 106.7 Cobol statements to construct 1 function point of software. It takes 128 C statements for the same 1 function point.

Compare and contrast Lines-Of-Code (LOC) and Function Point (FP) metrics

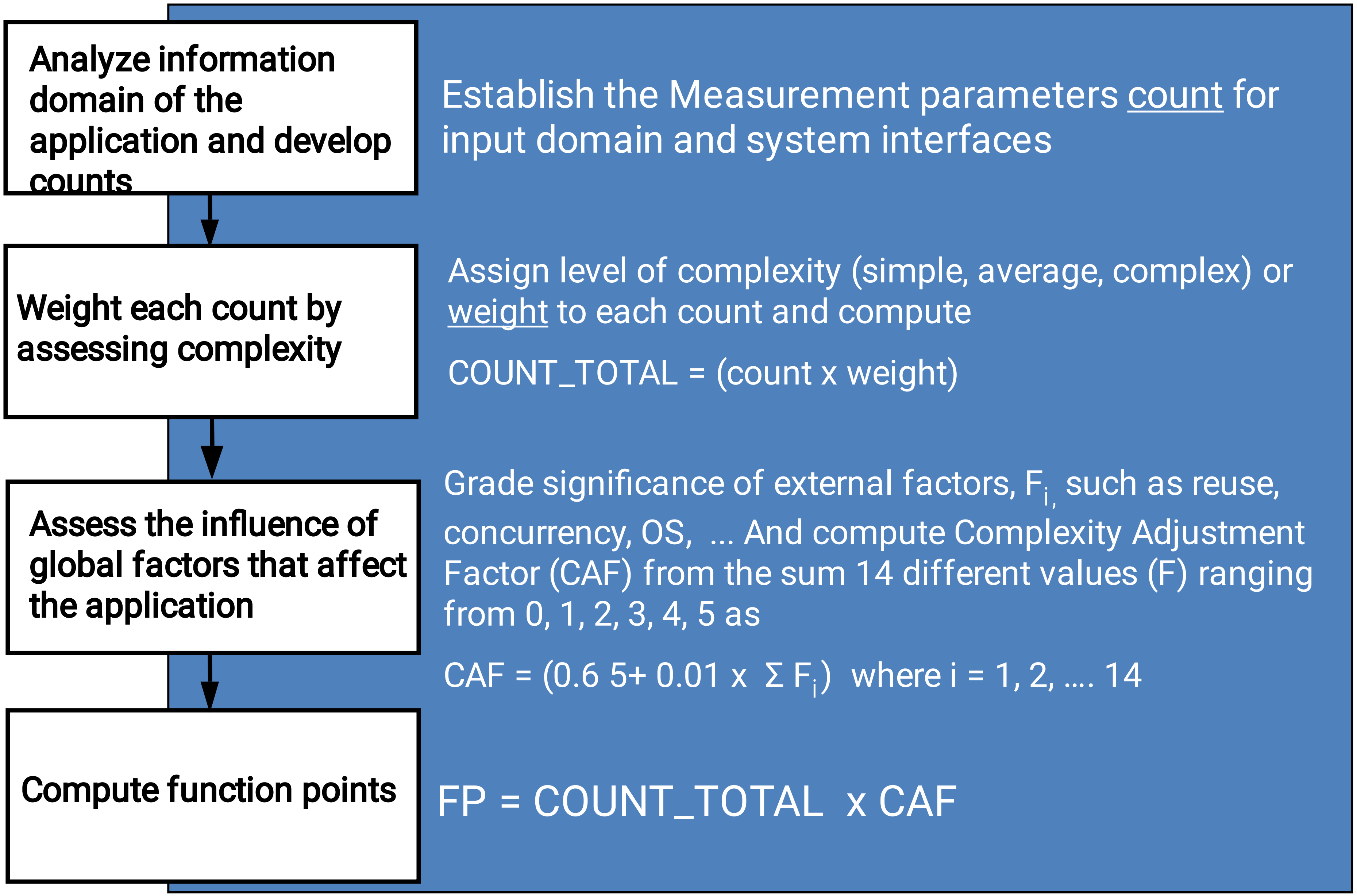
Project	LOC	FP	Effort (P/M)	R(000)	Pp. doc	Errors	Defects	People
alpha	12100	189	24	168	365	134	29	3
beta	27200	388	62	440	1224	321	86	5
gamma	20200	631	43	314	1050	256	64	6

- Size-Oriented:
  - errors per KLOC (thousand lines of code), defects per KLOC, R per LOC, page of documentation per KLOC, errors / person-month, LOC per person-month, R / page of documentation
- Function-Oriented:
  - errors per FP, defects per FP, R per FP, pages of documentation per FP, FP per person-month

Why Opt for FP based metrics

- Independent of programming language. Some programming languages are more compact, e.g. C++ vs. Assembler
- Use readily countable characteristics of the “information domain” of the problem
- Does not “penalize” inventive implementations that require fewer LOC than others
- Makes it easier to accommodate reuse and object-oriented approaches
- Original FP approach good for typical Information Systems applications (interaction complexity)
- Variants (Extended FP and 3D FP) more suitable for real-time and scientific software (algorithm and state transition complexity)

Computing Function Points


Analyze information domain of the application and develop counts
Weight each count by assessing complexity
Assess the influence of global factors that affect the application
Compute function points

Establish the Measurement parameters count for input domain and system interfaces

Assign level of complexity (simple, average, complex) or weight to each count and compute

COUNT_TOTAL = (count x weight)

Grade significance of external factors, $F_{i,}$ such as reuse, concurrency, OS, ... And compute Complexity Adjustment Factor (CAF) from the sum 14 different values (F) ranging from 0, 1, 2, 3, 4, 5 as

CAF = $(0.6\ 5+ 0.01 \text{ x } \Sigma F_i)$ where i = 1, 2, …. 14

FP = COUNT_TOTAL x CAF

Computing Function Points

			weighting factor				
measurement parameter	count		simple	avg.	complex		
number of user inputs		X	3	4	6	=	
number of user outputs		X	4	5	7	=	
number of user inquiries		X	3	4	6	=	
number of files		X	7	10	15	=	
number of ext.interfaces		X	5	7	10	=	
count-total							
complexity multiplier							
function points							

Computing Function Points

Complexity Adjustment Values (F_i) are rated on a scale of 0 (not important) to 5 (very important):

1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. System to be run in an existing, heavily utilized environment?
6. Does the system require on-line data entry?
7. On-line entry requires input over multiple screens or operations?
8. Are the master files updated on-line?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and instillation included in the design?
13. Multiple installations in different organizations?
14. Is the application designed to facilitate change and ease-of-use?

Exercise: Computing Function Points

- Compute the function point value for a project with the following information domain characteristics:
  - Number of user inputs: 3
  - Number of user outputs: 2
  - Number of user enquiries: 2
  - Number of files: 1
  - Number of external interfaces: 4
  - Different Complexity Adjustment Values (F_i) are:

    2, 2, 0, 4, 3, 4, 5, 3, 2, 5, 4, 2, 5, 5
  - Assume that weights are simple.

Exercise: Computing Function Points

STEP 1: Compute the Count_Total from Measurement Parame

measurement parameter	count		weighting factor				
			simple	avg.	complex		
number of user inputs	3	X	[✓ 3]	4	6	=	9
number of user outputs	2	X	[✓ 4]	5	7	=	8
number of user inquiries	2	X	[✓ 3]	4	6	=	6
number of files	1	X	[✓ 7]	10	15	=	7
number of ext.interfaces	4	X	[✓ 5]	[✓ 7]	10	=	22
Count-Total $\longrightarrow$							52

Exercise: Computing Function Points

STEP 2: Compute the Complexity Adjustment Factor

Compute ΣF_i = 2 + 2 + 0 + 4 + 3 + 4 + 5 + 3 + 2 + 5 + 4 + 2 + 5 + 5
= 46

From ΣF_i, compute Complexity Adjustment Factor (CAF)

CAF = $[0.65 + 0.01 \times \Sigma F_i]$

CAF = [0.65 + 0.46] = 1.11

Exercise: Computing Function Points

STEP 3: Compute the Function Point

Compute Funtion Point (FP) = Count_Total x CAF

FP = 52 x 1.11

FP = 57.72 ANSWER

Computing Function Points

- Uses the functionality of the software as a measure
- Analogy: For a given house, we can say how many square meters it has (LOC) or we can say how many bedrooms and bathrooms it has (FP).
- The idea was first put forward by Allan Albrecht of IBM in 1979.
- Derived using an empirical relationship based on countable (direct) measures of software's information domain (i.e., things in the external behavior that will require processing)
- A function point count is a measure of the amount of functionality in a product
- Can be estimated early in project before a lot is known

Thank You