Software Testing and Quality Assurance – Unit 2

"WHEN YOU CAN MEASURE WHAT YOU ARE SPEAKING ABOUT AND EXPRESS IT IN NUMBERS, YOU KNOW SOMETHING ABOUT IT; BUT WHEN YOU CANNOT MEASURE IT, WHEN YOU CANNOT EXPRESS IT IN NUMBER, YOUR KNOWLEDGE IS OF A MEAGER UNSATISFACTORY KIND: IT MAY BE THE BEGINNING OF KNOWLEDGE, BUT YOU HAVE SCARCELY, IN YOUR THOUGHT, ADVANCED TO THE STAGE OF A SCIENCE"

-LORD KELVIN

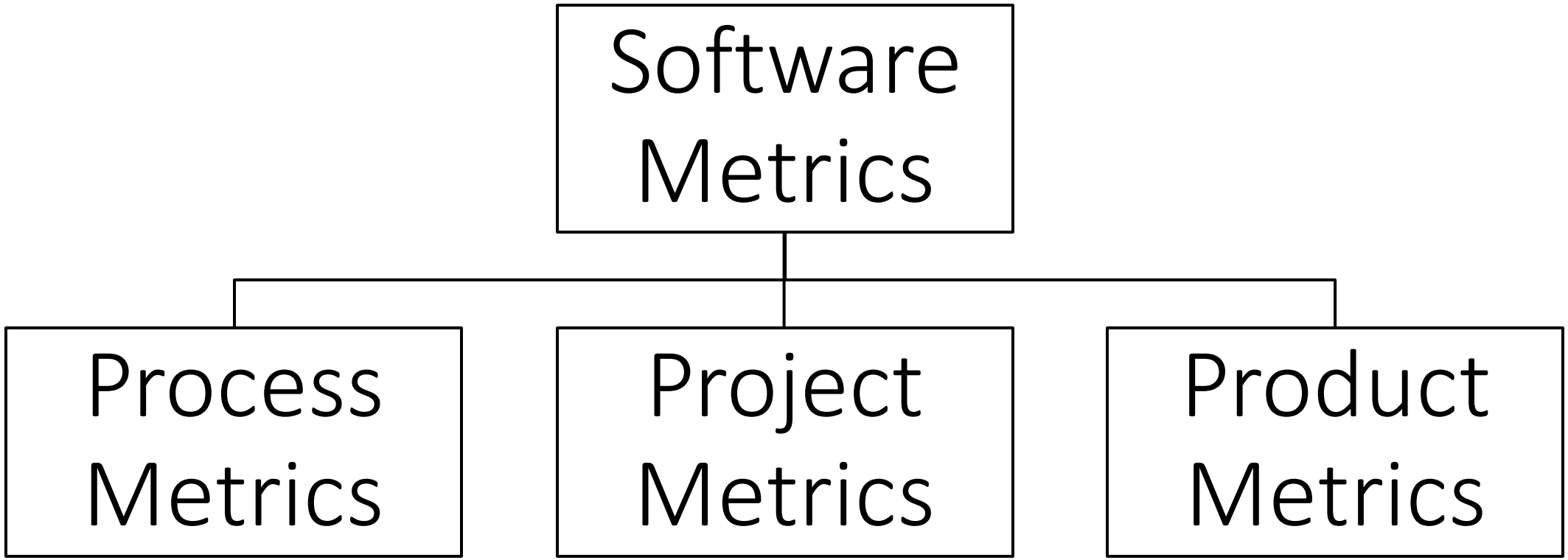
Introduction

Measurements in the physical world can be categorized in two ways: direct measures (e.g. length, weight, etc.) and indirect measures (e.g. 'quality'). Software metrics can be categorized similarly.

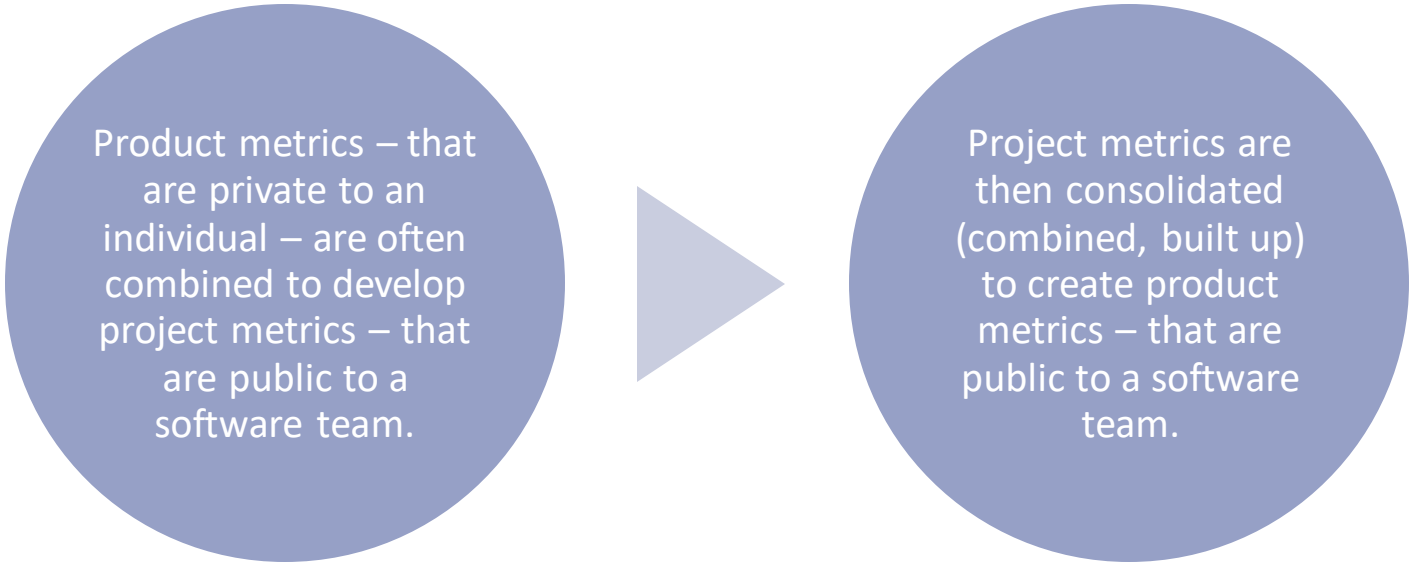
Direct measures of the software process include the cost and effort applied in terms of the lines of code (LOC) produced, execution speed, memory size, and defects reported over some set period of time. Indirect measures include functionality, quality, quantity, complexity, efficiency, reliability, maintainability and many other "-abilities".

The cost and effort required to build software, the no. of LOC produced, ad other direct measures are easy to collect as long as certain measurement conventions are defined beforehand. However, indirect measures such as efficiency, quality, reliability, maintainability, etc. are more difficult to assess and can only be measured indirectly.

Software Metrics



Software Metrics



How does an organization combine these metrics that come from different individuals or software team?

An example.

Individuals on 2 different project teams record and categorize all errors that they find during the software testing process. Individual measures are then combined to develop team measures. Team A found 342 measures and Team B found 184 errors. All other things being equal, which team is more effective in uncovering errors throughout the process?

Because we do not now the size or complexity of the project, we cannot answer this question. However, if the measures are normalized, it is possible to create software metrics that enable comparison to broader organizational averages.

Size-Oriented Metrics

Size-oriented metrics are derived by normalizing quality and/or productivity measures by considering the *size* of the software that has been produced.

Size-oriented Metrics

Project	LOC	Effort	K$	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6

The table lists all each software development project that has been completed over the past few years and corresponding measures for the project.

Size-Oriented Metrics

PROS

Is an "artifact" of all software development projects, that can be easily counted.

Many existing software estimation models use LOC/KLOC as a key input for estimation.

A large body of literature and data predicated on LOC already exists.

CONS

LOC measures are programming language dependent.

When productivity is considered, these types of metrics penalize well-designed but shorter programs.

They cannot easily accommodate non-procedural languages.

Their use in estimation requires a level of detail that may be difficult to achieve.

Function-Oriented Metrics

Function-oriented metrics use a measure of the functionality derived by the application as a normalization value. Computation of the function point is based on characteristics of the software's information domain and complexity.

Function-Oriented Metrics

PROS

These metrics are programming language independent. Ideal for applications using conventional and nonprocedural languages.

Based on data that are more likely to be known in the early evolution of a project. Making FP more attractive as an estimation approach.

CONS

The method requires some "sleight of hand" in that computation is based on subjective rather than objective data.

Counts of the information domain can be difficult to collect after the fact.

FP has no physical meaning – it's just a number.

Reconciling LOC and FP Metrics

The relationship between LOC and FP is dependent on the programming language and the quality of design of the software.

The following table provides rough estimates of the average number of LOC required to build one FP in various programming languages.

Programming Language	LOC per FP			
	Average	Median	Low	High
C	162	109	33	704
C++	66	53	29	178
Java	42	31	24	57
JavaScript	63	53	77	-
JSP	59	-	-	-
Oracle	30	35	4	217
SQL	26	19	10	55

Object-Oriented Metrics

Conventional software project metrics can be used to estimate object-oriented software projects. However, these metrics do not provide enough granularity for the schedule and effort adjustments as you iterate through an evolutionary or incremental process.

Object-Oriented Metrics

Number of Scenario Scripts: a Scenario Script is a detailed sequence of steps that describe the interaction between the user and the application. No. of such scripts is directly correlated to the size of the application and to the number of test cases that must be developed.

Number of Key Classes: are the "highly independent components" that are defined early in the object-oriented analysis. They are central to the problem domain, number of such classes is an indicator to the amount of effort required and to the potential amount of reuse to be applied.

Number of Support Classes: are required to implement the system but are not immediately related to the problem domain. Support classes can be defined for each Key class. The may also be defined iteratively throughout an evolutionary process. This is also an indicator to the amount of effort required and to the potential amount of reuse to be applied.

Object-Oriented Metrics

Average number of Support Classes per Key Class: if the average number of support classes per key class were known for a given problem domain, estimating (based on total number of classes) would be greatly simplified.

Number of Subsystems: is an aggregation of classes that support a function that is visible to the end user of a system. Once subsystems are identified, it is easier to lay out a reasonable schedule in which work on subsystems is partitioned among project staff.