-30-PRODUCT METRICS

day, 30 May 2022 7:05 PM

A framework for Product Metrics

⇒ Measures, Metrics, and Indicators

⇒ Measures :

⇒ Measure can be used either as a noun or a verb.

⇒ Within SE context, a measure provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process.

⇒ Measurement is the act of determining a measure.

⇒ When a single data point has been collected (e.g., number of errors uncovered within single software component), a measure has been established.

⇒ Measurement occurs as the result of the collection of one or more data points.

⇒ Metric : measures of different characteristics

The IEEE Standard Glossary of Software Engineering Terminology defines metric as a "quantitative measure of the degree to which a system, component, or process possesses a given attribute".

⇒ Indicators :

⇒ A Software engineer collects measures and develops metrics so that indicators will be obtained.

An indicator is a metric or combination of metrics that provides insight into the software process, a software project, or the product itself.

⇒ An indicator provides insight that enables the project manager or software engineers to adjust the process, the project, or the product to make things better.

⇒ Metrics for the Requirements Model

⇒ Metrics for the Requirements Model

⇒ Technical work in SE begins with the creation of the requirements model.
⇒ It is at this stage that requirements are derived and a foundation for design is established.
⇒ Therefore, Product metrics that provide insight into the quality of analysis model are desirable.
⇒ These metrics examine the requirements model with the intent of predicting the 'size' of the resultant system.
⇒ Size is an indicator of design complexity and is almost an indicator of increased coding, integration, and testing effort.

⇒ Function - Based Metric

⇒ The Function Point (FP) metric can be used effectively as a means for measuring the functionality delivered by a system
⇒ FP metric can be used to :
1) Estimate cost or effort required to design, code and test the software
2) Predict the number of errors that will be encountered during testing.
3) Forecast the no. of components and/or the number of projected source lines in the implemented system.

⇒ Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and qualitative assessments of software complexity.

⇒ Information domain values are defined in the following manner :

1) Number of External Inputs (EI) :
Each external input originates from a user or is transmitted from another application and provides distinct application-oriented data or control information.

2) Number of External Outputs (EOs) :

2) Predict the number of errors that will be encountered during testing.
3) Forecast the no. of components and/or the number of projected source lines in the implemented system.

⇒ Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and qualitative assessments of software complexity.

⇒ Information domain values are defined in the following manner:

1) Number of External Inputs (EI):
Each external input originates from a user or is transmitted from another application and provides distinct application-oriented data or control information.

2) Number of External Outputs (EOs):
Each external output is derived data within the application that provides information to the user. In this context, external outputs refer to reports, screens, error messages and the like.

3) Number of external inquiries (EQs):
An external inquiry is defined as an online input that results in the generation of some immediate software response in the form of an online output.

4) Number of internal logic files (ILFs):
Each internal logical file is a logical grouping of data that resides within the application's boundary and is maintained via external inputs.

5) Number of External Interface Files (EIFs):
Each external interface file is a logical grouping of data that resides external to the application but provides data that may be of use to the application.

Number of External Interface Files (EIFs):

An external interface file is a logical grouping of data that resides external to the application but provides data that may be of use to the application.

FIGURE 30.1 Computing function points

Information Domain Value	Count		Weighting factor: Simple	Average	Complex		
External Inputs (EIs)		3	3	4	6	=	
External Outputs (EOs)		3	4	5	7	=	
External Inquiries (EQs)		3	3	4	6	=	
Internal Logical Files (ILFs)		3	7	10	15	=	
External Interface Files (EIFs)		3	5	7	10	=	
Count total							

3 4 3 7 5
4 5 4 10 7
6 7 6 15 10

⇒ To Compute Function Points (FP),

$$FP = \text{Count total} \times [0.65 + 0.01 \times \Sigma(F_i)]$$

★ Count total = Sum of all FP entries
★ F_i ($i = 1$ to 14) are Value Adjustment factors (VAF) based on responses to following questions ⇒

1. Does the system require reliable backup and recovery?
2. Are specialized data communications required to transfer information to or from the application?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require online data entry?
7. Does the online data entry require the input transaction to be built over multiple screens or operations?
8. Are the ILFs updated online?
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 installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

Each of these questions is answered using an ordinal scale that ranges from 0 (not important or applicable) to 5 (absolutely essential). The constant values in Equation (30.1) and the weighting factors that are applied to information domain counts are determined empirically.

0 ⇒ no influence
1 = Incidental
2 ⇒ Moderate
3 = Average
4 = significant
5 ⇒ Essential (critical)

⇒ Metrics for Specification Quality

⇒ Characteristics used to access quality of the requirements model ⇒
specificity (lack of ambiguity), completeness, correctness, understandability, verifiability, internal and external consistency,

Metrics for Specification Quality

→ Characteristics used to access quality of the requirements model →
specificity (lack of ambiguity), completeness, correctness, understandability, verifiability, internal and external consistency, achievability, concision, traceability, modifiability, precision and reusability.

⇒ We assume there are n_R requirements, such that

$$n_R = n_f + n_{nf}$$

n_f = no. of functional requirements.
n_{nf} = no. of non-functional " .

⇒ To determine specificity (lack of ambiguity) of requirements

$$Q_1 = \frac{n_{ui}}{n_R}$$

n_{ui} = no. of requirements for which all reviewers had identical interpretations.

⇒ Closer the value to 1, lower is the ambiguity.

⇒ The completeness of functional requirements can be determined by computing the ratio.

$$Q_2 = \frac{n_u}{[n_i \times n_s]}$$

* n_u = no. of unique function requirements
* n_i = no. of inputs defined/implied by the specification
* n_s = no. of states specified.

→ Q_2 measures percentage of necessary functions that have been specified for system

→ To address Non-functional requirements in completeness ⇒

$$Q_3 = \frac{n_c}{[n_c + n_{nv}]}$$

* n_c = no. of requirements that have been validated as correct
* n_{nv} = " " " " " not " validated yet.

I-32,33,34. -METRICS FOR SOFTWARE
UALITY,COCOMO-II MODEL,TIME-LINE CHARTS

Monday, 30 May 2022 10:45 PM

⇒ Measuring Quality [CMIV] ⇒ CMI me vansh OR [FCVM]

Following measures provide useful indicators for the project team.

1)- Correctness ⇒ Degree to which the software performs it required function. Defects (lack of correctness) are those problems reported by the users of the program after the program has been released for general use. For quality assessment purposes, defects are counted over a standard period of time, typically a year. The most common measure for correctness is defects per KLOC (per thousand lines of code)

2)- Maintainability: Ease with which a program can be corrected if an error is encountered, adapted if its environment changes or enhanced if customer desires a change in requirements
A simple time oriented metric is mean-time-to-change [MTTC], the time it takes to analyze the change request, design an appropriate modification, implement the change, test it, and distribute the change to all users

3)- Integrity: This attribute measures system's ability to withstand attacks to its security.
To measure integrity, two additional attributes must be defined: threat and security.

Threat: Probability that an attack of a specific type will occur within a given time.
Security: Security is the probability that the attack of a specific type will be repelled.

⇒ Integrity of a system ⇒ $\Sigma [1 - (\text{threat} \times (1 - \text{security}))]$

4)- Usability: Usability is an attempt to quantify ease of use

⇒ Defect Removal Efficiency

Defect Removal Efficiency

32.3.2 Defect Removal Efficiency

A quality metric that provides benefit at both the project and process level is *defect removal efficiency* (DRE). In essence, DRE is a measure of the filtering ability of quality assurance and control actions as they are applied throughout all process framework activities.

When considered for a project as a whole, DRE is defined in the following manner:

$$DRE = \frac{E}{E + D}$$

* E = No. of errors found before delivery
* D = No. of defects found after delivery

where E is the number of errors found before delivery of the software to the end user and D is the number of defects found after delivery.

ADVICE

If DRE is low as you move through analysis and design, spend some time improving the way you conduct formal technical reviews.

The ideal value for DRE is 1. That is, no defects are found in the software. Realistically, D will be greater than 0, but the value of DRE can still approach 1. As E increases (for a given value of D), the overall value of DRE begins to approach 1. In fact, as E increases, it is likely that the final value of D will decrease (errors are filtered out before they become defects). If used as a metric that provides an indicator of the filtering ability of quality control and assurance activities, DRE encourages a software project team to institute techniques for finding as many errors as possible before delivery.

IMP
Past years

DRE can also be used within the project to assess a team's ability to find errors before they are passed to the next framework activity or software engineering task. For example, requirements analysis produces a requirements model that can be reviewed to find and correct errors. Those errors that are not found during the review of the requirements model are passed on to design (where they may or may not be found). When used in this context, we redefine DRE as

$$DRE_i = \frac{E_i}{E_i + E_{i+1}}$$

where E_i is the number of errors found during software engineering action i and E_{i+1} is the number of errors found during software engineering action $i + 1$ that are traceable to errors that were not discovered in software engineering action i.

A quality objective for a software team (or an individual software engineer) is to achieve DRE_i that approaches 1. That is, errors should be filtered out before they are passed on to the next activity or action.

Ch-33 - COCOMO-II model

33.7.2 The COCOMO II Model

In his classic book on software engineering economics, Barry Boehm [Boe81] introduced a hierarchy of software estimation models bearing the name COCOMO, for *COnstructive COst MOdel*. The original COCOMO model became one of the most widely used and discussed software cost estimation models in the industry. It has evolved into a more comprehensive estimation model, called COCOMO II [Boe00]. Like its predecessor, COCOMO II is actually a hierarchy of estimation models that address different "stages" of the software process.

Like all estimation models for software, the COCOMO II models require sizing information. Three different sizing options are available as part of the model hierarchy: object points,[10] function points, and lines of source code.

→ object

→ Ch-34 - Time-Line Charts

34.5.1 Time-Line Charts

When creating a software project schedule, you begin with a set of tasks (the work breakdown structure). If automated tools are used, the work breakdown is input as a task network or task outline. Effort, duration, and start date are then input for each task. In addition, tasks may be assigned to specific individuals.

KEY POINT
A time-line chart enables you to determine what tasks will be conducted at a given point in time.

As a consequence of this input, a *time-line chart*, also called a *Gantt chart*, is generated. A time-line chart can be developed for the entire project. Alternatively, separate charts can be developed for each project function or for each individual working on the project.

Figure 34.3 illustrates the format of a time-line chart. It depicts a part of a software project schedule that emphasizes the **concept scoping** task for a word-processing (WP) software product. All project tasks (for concept scoping) are listed in the left-hand column. The horizontal bars indicate the duration of each task. When multiple bars occur at the same time on the calendar, task concurrency is implied. The diamonds indicate milestones.

Once the information necessary for the generation of a time-line chart has been input, the majority of software project scheduling tools produce *project*

FIGURE 34.3 An example time-line chart

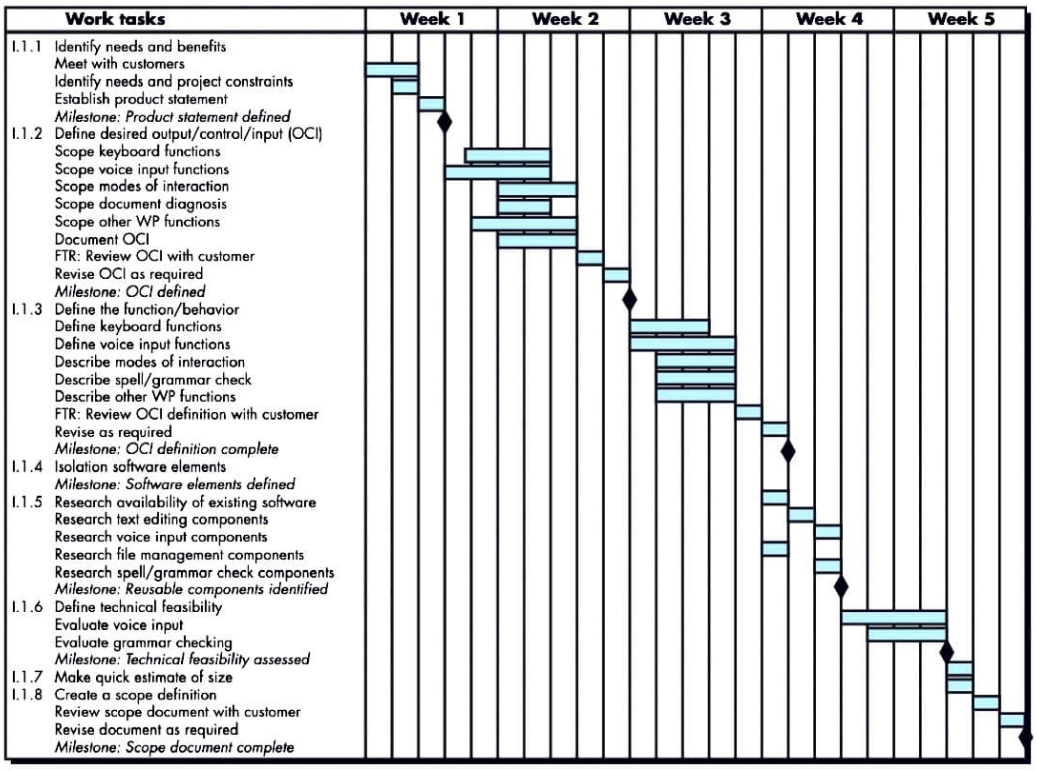

FIGURE 34.4 An example project table

Work tasks	Planned start	Actual start	Planned complete	Actual complete	Assigned person	Effort allocated	Notes
I.1.1 Identify needs and benefits Meet with customers	[redacted]				BLS	2 p-d	Scoping will require more

FIGURE 34.4 An example project table

Work tasks	Planned start	Actual start	Planned complete	Actual complete	Assigned person	Effort allocated	Notes
I.1.1 Identify needs and benefits							Scoping will
Meet with customers	wk1, d1	wk1, d1	wk1, d2	wk1, d2	BLS	2 p-d	require more
Identify needs and project constraints	wk1, d2	wk1, d2	wk1, d2	wk1, d2	JPP	1 p-d	effort/time
Establish product statement	wk1, d3	wk1, d3	wk1, d3	wk1, d3	BLS/JPP	1 p-d	
Milestone: Product statement defined	wk1, d3	wk1, d3	wk1, d3	wk1, d3			
I.1.2 Define desired output/control/input (OCI)							
Scope keyboard functions	wk1, d4	wk1, d4	wk2, d2		BLS	1.5 p-d	
Scope voice input functions	wk1, d3	wk1, d3	wk2, d2		JPP	2 p-d	
Scope modes of interaction	wk2, d1		wk2, d3		MLL	1 p-d	
Scope document diagnostics	wk2, d1		wk2, d2		BLS	1.5 p-d	
Scope other WP functions	wk1, d4	wk1, d4	wk2, d3		JPP	2 p-d	
Document OCI	wk2, d1		wk2, d3		MLL	3 p-d	
FTR: Review OCI with customer	wk2, d3		wk2, d3		all	3 p-d	
Revise OCI as required	wk2, d4		wk2, d4		all	3 p-d	
Milestone: OCI defined	wk2, d5		wk2, d5				
I.1.3 Define the function/behavior							

tables—a tabular listing of all project tasks, their planned and actual start and end dates, and a variety of related information (Figure 34.4). Used in conjunction with the time-line chart, project tables enable you to track progress.

Why Use Gantt Charts?

When you set up a Gantt chart, you need to think through all of the tasks involved in your project. As part of this process, you'll work out who will be responsible for each task, how long each task will take, and what problems your team may encounter.

This detailed thinking helps you ensure that the schedule is workable, that the right people are assigned to each task, and that you have workarounds for potential problems before you start.

They also help you work out practical aspects of a project, such as the minimum time it will take to deliver, and which tasks need to be completed before others can start. Plus, you can use them to identify the critical path – the sequence of tasks that must individually be completed on time if the whole project is to deliver on time.

Finally, you can use them to keep your team and your sponsors informed of progress. Simply update the chart to show schedule changes and their implications, or use it to communicate that key tasks have been completed.