Regular Paper

Software Analytics for Manual Activities using Developer Work Elements

Peraphon Sopahtsathit[1,a)]

Received: April 27, 2019, Accepted: January 16, 2020

Abstract: Software Engineering is a diverse and highly flexible discipline that can be practiced using a development model of the developer's choosing. Unfortunately, existing state-of-the-practice software engineering development models do not take human effort into consideration since there are no applicable metrics to gauge the associated manual activity. This study presents a novel discretization technique as a software analytic to estimate the manual effort expended on software development process. The proposed technique classifies three manual activity domains, namely, abstract, concrete, and unclassified. The units of classification are called Developer Work Elements (DevWE). The sequence of DevWE denotes a development analytic in three visual aids, namely, symbolic flow map, operation chart, and workload breakdown chart. These give rise to the determination of efforts expended which are measured by COSMIC Function Point. The result can be combined with those traditional software measurable activities to yield accurate total project effort estimation. Major contributions of this prospectus encompass (1) discretization DevWE analytic for manual effort estimation, (2) visual chart aids for operation tracing, monitoring, improving, and control, and (3) discovering that almost half of the estimation effort stems from manual activity.

Keywords: developer work elements, symbolic flow map, operation chart, workload breakdown chart, COSMIC function point, manual activity

1. Introduction

Software development is one part of a lengthy, costly, and human intensive process, regardless of the underlying model being implemented, i.e., traditional waterfall, object-oriented, or agile approach. These models require extensive human involvement which makes them difficult to automate. Despite numerous CASE tools for development assistance, manual activity is still a prevalent part of the development effort. One cannot entirely remove the human factor out of the project management equation and fully automate the process using machine learning technology. One compelling issue persists—effort estimation. In principle, effort estimation method is an essential software process to predict how much resources are required to complete the project as accurately as possible. These estimation methods are, in many cases, model-based estimation that is confined to model characteristics, nature of application coverage, assumptions, quantitative setting, etc. They might fit one project but fail in another. Consequently, the predicted effort usually suffers from error, model over/under-fitting, and other variations of wrong estimations.

Meanwhile, to reach an agreed-upon effort estimation outcome, some forms of measurement must be established as baselines for estimation analytics. A common handle that characterizes most estimation methods is the use of output or end-result measures for effort estimation. Lines-of-code (LOC) [7], [8], function point (FP), use case point (UCP) [26], object point (OP) [4], and COSMIC function point (CFP) [1], [29], [35] are a few popular process effort estimation analytics.

The motivation of this work stemmed from newly Computer Science/Software Engineering graduates who are embarking on their professional career in IT business and software industry. They usually work with state-of-the-practice models and techniques that mainly involve rewriting patched code to maintain aging software [32] or existing production applications. The maintenance process incorporated considerable manual activities such as meeting with writers of the old code, users, project manager, planning the new patches, adjustments, reviews, etc. Existing state-of-the-practice effort estimation models, methods, tools, and on-going related researches have been attempted over the years to reckon with accurate estimation. In fact, the sheer volume of a project involves many subjective and human-oriented operations that are hard to quantify procedurally, and therefore the project cost could not be accurately estimated.

Consider two practical maintenance change examples. How long will it take to decide on choosing 'Yes' or 'OK' as the label for the *accept* button that would fit different users' culture, background, and familiarity? This could take as few as 5 minutes to discuss and reach the conclusion. On the contrary, a more extensive change customization of web layout to fit individual locales might involve users from relevant cultures and ethnic backgrounds to collaborate. This could take days or weeks to plan, lookup/search, think/analyze, meeting/discussion, before arriving at the final specifications. How do these two analyses account for the effort spent? The answer is based on performance measure-

[1] Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University, Phayathai Road Bangkok 10330, Thailand
[a)] sperahon@gmail.com

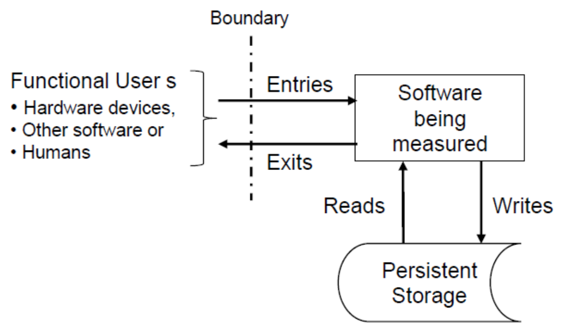


Fig. 1 The four types of data movements in CFP [35].

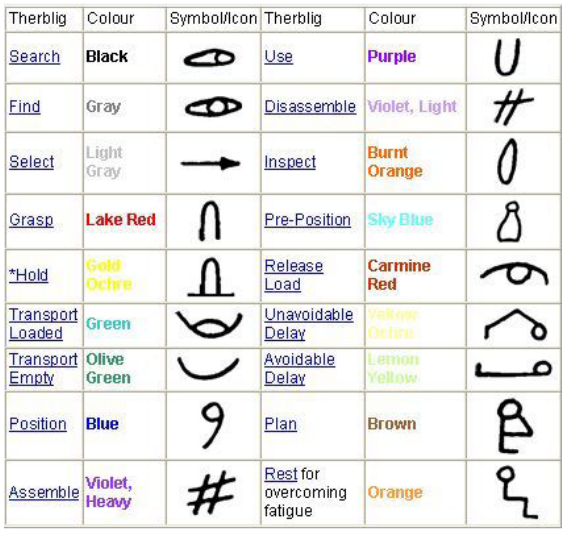
ment which in this study will adopt data movements of COSMIC Function Point (CFP) metrics [35] as illustrated in **Fig. 1**.

Both tasks require one entry (the problem), one read (problem requirements), one write (the analysis result), one exit (conclusion), and lots of discussions in between that cannot be taken into account. The total data movements in terms of CFP for both tasks are the same. Hence, these examples represent innumerable manual activities that are not straightforward enough to measure the effort required. The first well-established standard for discretizing manual operations is Method-Time Measurement (MTM) [13] in Industrial Engineering. It is the time required to perform a specific task depending on the method chosen for the activity. The basic building blocks of all 18 manual motions or Therbligs [15] are defined as shown in **Fig. 2**.

Therbligs came about after the industrial revolution in the 19th century to automate factory systems. Today, manual operations still persist in some factories. The need to measure performance of these activities remain. For example, moving object X (small enough to be picked up by fingers) from point A to B can be discretized as follows (all italicized terms are Therbligs definition): move the empty hand (*transport empty*) toward point A where object X is located—*position* above object at point A—pick up (*grasp*) the object—*hold* the object—move the hand holding the object toward point B (*transport loaded*)—*position* at point B—put down object X (*release load*) for a total of seven operations. In MTM, icons are used to represent the above motions in a visually succinct manual operation chart without the need for any descriptions that are subject to confusion, ambiguity, or variations of interpretation. This research attempts to delve into such human-oriented activity (the term will be used interchangeably with manual activity, or manual operation) to create a measurable set of iconic operations that represent human-oriented activities called developer work elements or DevWE.

Major contributions of this study are: (1) discretization DevWE analytic for manual effort estimation, (2) the visual three charts serve as a tool for operation trace, monitor, improve, and control, and (3) discovering that almost half of the estimation effort stems from manual activity. The prospectus also lends itself to future automation if these DevWEs and the processes are recognized and standardized.

The remaining portion of this article is organized as follows. Section 2 describes some related work to this research. Section 3 elucidates the basis, rationale, and configuration development of the manual operations. Section 4 exhibits the experimental results of the proposed approach in comparison with the conventional estimation approach. A few final thoughts and future work are summarized in the Conclusion Section.



Therblig	Colour	Symbol/Icon	Therblig	Colour	Symbol/Icon
Search	Black		Use	Purple	U
Find	Gray		Disassemble	Violet, Light	#
Select	Light Gray	→	Inspect	Burnt Orange	0
Grasp	Lake Red	∩	Pre-Position	Sky Blue	8
*Hold	Gold Ochre		Release Load	Carmine Red	
Transport Loaded	Green		Unavoidable Delay	Yellow Ochre	
Transport Empty	Olive Green	◡	Avoidable Delay	Lemon Yellow	
Position	Blue	9	Plan	Brown	
Assemble	Violet, Heavy	#	Rest for overcoming fatigue	Orange	

*Hold---Therblig first appeared in Mogensen's book and later in Dr. Barnes'.

Fig. 2 Definition of Therbligs [15].

2. Related Work

There have been numerous research attempts to establish project effort estimation, thereby project cost can be derived accordingly. Several well-established models have been practiced such as SLIM model [33], COnstructive COst MOdel or COCOMO 81 [7], and COCOMO II [8], Walston-Felix model [36], Bailey-Basili model [3], Albrecht-Gaffney model [2], Kemerer empirical model [27], and Matson, Barrett and Mellichamp model [28]. These models are supported by extensive research that corroborate practical costing formula based on well-defined software metrics, namely, lines of code (LOC), function point (FP), use case point (UCP) [26], application point (AP) [8], and their variants such as source lines of code (SLOC), delivered source instructions (DSI), and unadjusted use case point (UUCP). These forerunners set a common ground on many state-of-the-practice software estimation techniques, ranging from effort measurement techniques [16], effort/cost estimation techniques [9], predictive models for effort/cost estimation [18], [31], and phase-wise cost estimation [37], etc.

Perhaps one of the most systematic guidelines for estimation methods was introduced by Briand et al. [10], who proposed a straightforward and systematic classification schema for cost estimation methods, namely, model based and non-model based methods. The former was further broken down into generic (proprietary and not proprietary) and specific (data driven and composite) methods. Further study on some definitions of the above methods from references provided by Jorgensen et al. [24] found that suitable estimating criteria would help decide what method to choose. A few suggested methods summarized by Kalach [25] were bottom-up, 3-point, parametric, and analogous estimates. Billows [6] explained 3-point estimating which took risk factor into account. Chung [14] provided some comparative bases for analogous and parametric estimating methods such as similar

Table 1 Comparison of bottom-up [17] and DevWE.

Step	Bottom-up estimating	DevWE
1	use WBS to breakdown all required project work	use MTM to breakdown the manual part of a project development stage
2	produce task estimates and dependencies from team members, stakeholders, and experts	produce manual activity or PWE using BSH process to create DevWE breakdown (Table 4)
3		use Little-JIL and symbolic flow map to express the dependencies
4	gather resource, experience, skill, type, unit, equipment, supplies requirements	compute standard effort for use in operation and workload breakdown assignment to team members
5	use an accurate schedule to determine the resource needed and task completion	add to functional activity to obtain the total effort required

scope from past projects/activities and scaling. Despite their dependence on historical data, analogous estimating is based on 'analogy', whereas parametric estimating relies on a unit cost of historical data that can be scaled to give the estimation. Unfortunately, these methods from past studies did not provide any estimating schemes for manual activity.

Guerrera [17] elucidated bottom-up estimating in 5 steps, namely, (1) identify all project tasks, (2) estimate them using work breakdown structure (WBS), (3) identify task dependencies, (4) identify the resources required to compute all tasks, and (5) determine when resources were needed to complete these tasks. These steps exhibited some resemblance to the MTM framework that was adopted as the basis for the proposed DevWE development.

The bottom-up method takes WBS artifacts to look for any similar, applicable, or matching methods that subsume those artifacts. An estimate of each artifact is derived using some of the above methods such as 3-point, analogous, or parametric. For example, to compute the estimate of a story (which is a WBS artifact), the participating team members might use planning poker scoring to vote, namely, 0, 1/2, 1, 2, 3, 5, 8, 13, 20, 40, 100, ?, Break (details are omitted for brevity). The final estimate of each story represents a working increment, when combined with other working increments, make up the product backlog and eventually the project estimate. The philosophy of DevWE is similar. It starts from defining each manual activity using a succinct and measurable element. Then applies some predetermined rating to determine the standard estimate. This standard estimate is substituted into the DevWE activity breakdown. By assembling all the DevWEs (by way of bottom-up integration), the final estimate will represent a manual *measurement of work*. Similar parts of both processes lie in the use of estimating scheme and bottom-up integration to obtain the final estimate. Nonetheless, the major difference is that current bottom-up methods do not account for manual activity, whereas DevWE is pure manual orientation. Since DevWEs are predefined and work with three visual charts to assist the effort estimation process, the visibility and straightforward estimating computations make them less complex, transparent, and repeatable by estimation method criteria [10]. **Table 1** summarizes the comparison between Guerreras's bottom-up method [17] and the proposed DevWE.

A noteworthy observation from software development cost studies [24] is the explicit treatment of manual activity in the proposed DevWE. As exemplified earlier, this treatment could render the manual activity to stand out as an observable process that is manageable to achieving better estimating accuracy. This is what makes the proposed DevWE analytics different from all software project cost estimating methods.

The method of operation breakdown to discrete motion elements [34] are procedurally set up and carried out in the design process with the help of Little-JIL [12]. Details will be described in the sections that follow.

3. Fine-grained Developer Work Elements

This section will elucidate the problem statement, the proposed model, the detailed breakdown of DevWe framework, the activity trace, the analysis of performance assessment, and the application to manual operation estimation.

3.1 Problem Statements

Suppose an estimated 1-day login story for an ordinary web application took 2 programmers to do, given the wage of \$100/man-day. This login story would be counted as 2 man-days and costed \$200.

1. How does one obtain the 1 day estimation as the operating time?
2. In addition to the determination of conventional development effort such as LOC or FP count, are there any manual activities performed for which have never been accounted?

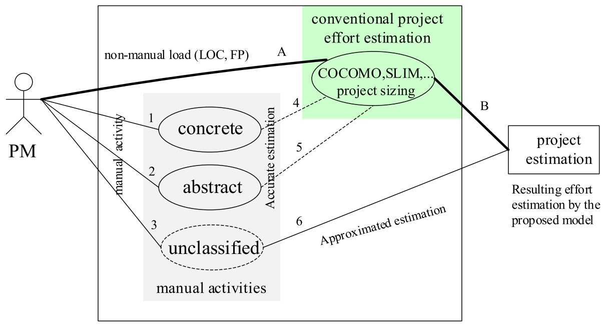
What is important is not just being able to compute the sum of effort estimation, but how the systematic procedure of the manual process is broken down and enumerated. This study will dichotomize the manual and functional activities (described in Section 3.3) by breaking down the process into developer work elements (DevWE) so as to determine the manual effort involved. Details will be further described in the next sections.

3.2 The Proposed Model

The proposed model, as shown by a Use Case diagram in **Fig. 3**, accounts for quantitatively measurable (or concrete) and quantitatively non-measurable (or abstract) measurements of work or *domains* to cover the manual activities performed by human within a project. The terms are defined as follows:

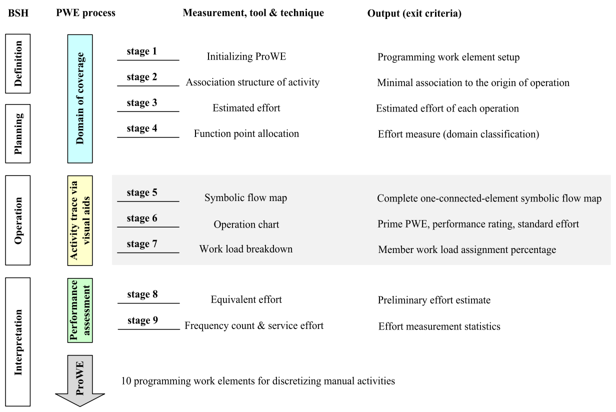
1. The *abstract* domain refers to activities that are subjective, somewhat intangible or unclear, and indirectly measurable.
2. The *concrete* domain refers to activities that are objective, tangible, and directly measurable.
3. An *unclassified* domain is furnished to accommodate activities that are hardware-oriented or difficult to logically define.

In this figure, the project manager (PM) computes conventional measures (non-manual load) in terms of LOC, FP, etc., for project



conventional project effort estimation
non-manual load (LOC, FP)
A
COCOMO,SLIM,... project sizing
B
PM
manual activity
1
concrete
2
abstract
3
unclassified
Accurate estimation
4
5
6
Approximated estimation
manual activities
project estimation
Resulting effort estimation by the proposed model

Fig. 3 Use Case of the proposed model.



BSH
PWE process
Measurement, tool & technique
Output (exit criteria)
Definition
Planning
Operation
Interpretation
Domain of coverage
Activity trace via visual aids
Performance assessment
ProWE
stage 1
Initializing ProWE
Programming work element setup
stage 2
Association structure of activity
Minimal association to the origin of operation
stage 3
Estimated effort
Estimated effort of each operation
stage 4
Function point allocation
Effort measure (domain classification)
stage 5
Symbolic flow map
Complete one-connected-element symbolic flow map
stage 6
Operation chart
Prime PWE, performance rating, standard effort
stage 7
Work load breakdown
Member work load assignment percentage
stage 8
Equivalent effort
Preliminary effort estimate
stage 9
Frequency count & service effort
Effort measurement statistics
10 programming work elements for discretizing manual activities

Fig. 4 Breakdown of the proposed model.

sizing estimation, denoted by A. In the meantime, manual activities, designated by three lines connecting to concrete, abstract, and unclassified activity group, denoted by 1, 2, and 3, respectively, will be determined for additional inputs to project estimation. The intermediate outputs of the manual effort computed by the proposed approach are concrete (4), abstract (5), and unclassified (6) efforts, respectively, where the numbers and letters denote labelling for easy reference. The resulting directly measurable effort estimation by the proposed model is equal to B = A + 4 + 5 units. The indirectly measurable effort estimation is equal to 6. Thus, the total effort estimation of this project becomes B + 6 units. It can be seen that the benefits from the proposed model are (*i*) separation of manual activity estimations (4, 5, 6) from conventional model-based estimation (A) and (*ii*) visual breakdown of manual activities which give rise to better monitoring and control of these activities. Such benefits will materialize in the sections that follow.

The above model can be broken up into three steps by adopting the experimentation established by Basili-Selby-Hutchens (BSH) [5], denoted in parenthesis, (1) define the domain of coverage (definition and planning), (2) trace the activity via visual aids (operation), and (3) perform the assessment (interpretation). This setup is depicted in **Fig. 4**. The first step *defines* the elements of DevWE and *plans* the corresponding process, accompanied by relevant metrics to be used in effort estimation. The second step expresses the division of manual *operation* in terms of DevWE using three visual aids. The last step sets up the equivalent effort and measurement with respect to conventional methods to *interpret* the results obtained from those visual aids. The three steps constitute the Programmer Work Elements (ProWE) which culminate in the ten DevWEs. Note that all italicized PWE processes signify the correspondence with BSH processes, i.e., definition, planning, operation, and interpretation. Details on each step will be described below.

3.3 Detailed Breakdown of DevWE Framework

The manual operation breakdown starts from classification of various development process activities based on the proposed model. The first two classifications are to some extent sizable-which can be measuredby conventional software metrics, namely, LOC and FP. This work employs CFP as the starting measurement metric during the early stages of the development process. The result will then be compared with the well-established standard FP metric to demonstrate their benchmarking conversion. Consider the standard FP allocation which is illustrated in **Table 2** [16], where quantifiable weights are allotted to different complexity levels of estimation factors.

Table 2 Function point allocation (FP) [16].

Description	Simple	Average	Complex
External Input	3	4	6
External Output	4	5	7
External Inquiry	3	4	6
External Interface File	5	7	10
Internal File	7	10	15

To illustrate how FP allocation is computed. Suppose a 'simple' program consists of two inputs, two outputs, three inquiries, one interface file, and one internal file. The size of this program is 35 FP ($2 * 3 + 2 * 4 + 3 * 3 + 1 * 5 + 1 * 7$).

Based on the above related works in Section 2 and the proposed model reference architecture depicted in Fig. 3, ten elements of DevWE are categorically derived as follows. The abstract domain encompasses four major manual activities, namely, operation (that subsequently may be tallied, checked, or obtained the result), read/inquire (that may be tested, or answered), meeting/discussion (that subsequently may be reported or summarized), and planning/decision (that indirectly may be verified, acted upon, or followed). The concrete domain accounts for adjustment (that can be specifically noted), code (that can be counted using proper metrics), and inspection/review (that can be prepared in advance). An unclassified domain consists of storage (what media, ownership), movement (from/to designated source/destination), transfer (to where, speed, volume, QoS, reliability), and personal delay. All DevWE elements along with their descriptions are summarized in **Table 3**.

These elements of activities are grouped into two groups for practical adoption, namely, manual (MP + RDAI) and functional (OSTC) activities. The former combines two abstract domains that are difficult to quantify, i.e., M, P, and one abstract (R), one unclassified (D), two concrete domains (A, I) that are somewhat objectively measurable. The latter combines one abstract (O), one concrete (C), and two unclassified (S, T) domains that are functionally technology oriented. This will be visibly clearer during the experiment.

In this study, the "Forgotten password" story [30] is used as an explanation-by-example to describe the prospectus detail shown in **Table 4**. There are six columns denoting (1) Traditional phase encompassing four steps, namely, requirements gathering, design, code, and unit testing, (2) DevWE equating to traditional phases as think/analyze, plan, discuss, lookup/search to requirements gathering, design to design, code to code, and debug, test, allowance to unit testing. This logical work flow is methodically set up by means of Little-JIL technique as demonstrated in **Fig. 5** to transform procedural data and control flows into COSMIC statistics for subsequent analysis and evaluation. (3) Descriptions of the steps being broken down, where square brackets and parentheses denote measurable manual efforts by CFP and FP, respectively, (4) Activity measures of each phase in FP, (5) Approximate duration by DevWE, and (6) corresponding CFP measures. This story is estimatedto take one day to complete.

Table 3 The 10 developer work elements (DevWEs).

Symbol	Name	code	Description
○	operation	O	action or computation pertaining to logical work product
⛁	read, inquire	R	obtain input information (through system interface)
⊗	meeting, discussion	M	discussion and resolution on issues pertinent to the work product
◇	planning, decision	P	planning and decision by individual or collaborative members
▽	storage	S	archive information content
⇨	movement, transfer	T	send or receive message, file, or any form of information media
D	delay	D	any delays caused by personal, duty, assignment, whether intentional, unintentional, avoidable, unavoidable, or mandatory
⏢	adjustment	A	update or delete artifact or work product
▭	code	C	Construction/configuration of work product
□	inspection, review	I	verification and validation of work product or service

The design process can be broken down (based on an 8-hour or 480-min working day) into DevWE according to the ProWE process shown in Fig. 4 as follows. In stage 1, proper DevWE is selected to represent the activities to be compared with traditional approach. The structure of the activity breakdown for stage 2 consists of two inputs from user information read (IR) in *unit testing*, one output for result message (RM) in *unit testing*, one inquiry for prompt (P) in *requirements gathering*, one internal file for password read (PR) in *code*, and one external interface file for user GUI screen (UI) in *requirements gathering*. The total effort amounts to $2*3+1*4+1*3+1*5+1*7 = 25$ FP. By comparing with DevWE measure, think/analyze use 1 read and 1 write for 2 CFP. Design takes 3 writes for 3 CFP, code takes 1 write for 1 CFP. Debug uses 1 read and 1 write, while test uses 1 read and 1 write for 2 CFP each. The total effort amounts to 10 CFPs. This is shown in Table 4.

The work time of stage 3, i.e., analysis, planning, and search activities are estimated at 1/3 [11] of one-day work or approximately three hours. Design activities take one and a half hours. Coding and debugging are counted for 1/6 [11] or one hour. Test-

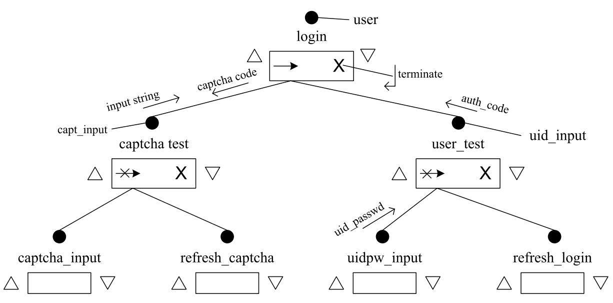


Fig. 5 Login validation exhibited in Little-JIL.

Table 4 DevWE breakdown with estimated working time and performance measurement.

Traditional phase	DevWE	Description	Phase(FP)	DevWE (min)	CFP
requirements gathering	think /analyze	Password trigger mechanism, authentication, new entry and confirmation, save new password [1 R and 1 W] (…)	1(P) / 1(UI)	180	2
	plan	Event flow, test (checkpoint) plan			0
	discuss	Discuss the plan			0
	lookup /search	Search for related library, package, sample code to be reused (external inquiry/external interface file)			0
design	design	1. UML diagrams [1 W] 2. UI screens for new password and associate code [1 W] 3. test case construction [1 W] (…)		90	3
code	code	Write code [1 W] (internal file)	1(PR)	60	1
unit testing	debug	Trial runs (unit test) with test cases [1 R and 1 W] (input)	2(IR) / 1(RM)		2
	test	integration tests [1 R and 1 W] (input/output)		120	2
	allowance	**Personal** allowances		30*	0
	Total		25	480	10

Note: (…) denotes FP measurement and […] denotes COSMIC measurement, R=read, W=write

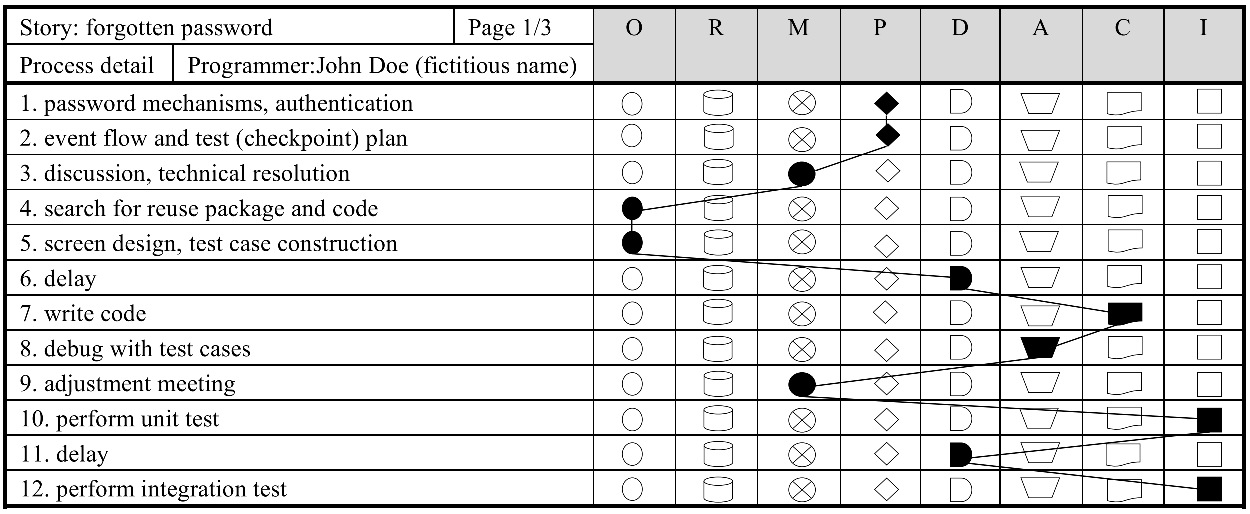

Story: forgotten password
Page 1/3
Process detail
Programmer:John Doe (fictitious name)
O
R
M
P
D
A
C
I
1. password mechanisms, authentication
2. event flow and test (checkpoint) plan
3. discussion, technical resolution
4. search for reuse package and code
5. screen design, test case construction
6. delay
7. write code
8. debug with test cases
9. adjustment meeting
10. perform unit test
11. delay
12. perform integration test

Fig. 6 A symbolic flow map of 'forgotten password' breakdown to DevWE.

ing consumes 1/4 [11] or two hours. Hence, the total equivalent effort becomes 7.5 man-hours. The half hour different between working time and equivalent effort will be allocated for allowances. From the above time allotment, both measurements yield different effort estimations, i.e., FP gives 480/25 = 19.2 min/FP and CFP becomes 480/10 = 48 min/CFP, or equivalently 1 CFP = 2.5 FP for this study.

Note that all descriptions and symbols in Table 3, representing stage 4, are the results obtained from the BSH planning step.

3.4 Activity Trace

This step focuses on the visual aids of activities for operation management, covering stage 5–6. The process must already be represented in DevWE symbols to furnish good visibility. This step adopts MTM process chart that will visually interface between information flow and operation.

There are three charts involved in activity trace, namely, a symbolic flow map, an operation chart, and a workload breakdown chart. A symbolic flow map furnishes a visual trace of the information flow and manual operation to discover any inherent inefficiencies that are created by human. **Figure 6** shows the symbolic flow map that denotes the sequence of DevWEs constituting the forgotten password story. The first four symbols (O, R, M, P) are concrete types and the last three (A, C, I) are abstract types. The only observable but unaccountable symbol is delay (D) which is a sizable element that can happen any time for any random duration. Hence, they are setup in the symbolic flow map based on DevWE descriptions in Table 3.

In this figure, each activity, denoted by a filled DevWE symbol, is successively connected to the next symbol that represents the succeeding activity. For example, Step 1 password mechanisms/authentication is denoted by P (decision) for password checking, so the diamond symbol of P is filled and the rest of the symbols are left blank in line 1. Step 2 is also denoted by P (decision), and is connected to the first step P (diamond) by a short vertical line to demonstrate the flow of event from step 1 to step 2. Step 3 is an M (discussion), so the cross-circle symbol on line 3 is filled with a line connecting to step 2 to denote the continuation of the flow of events. This process continues in succession until reaching the I (inspection) of step 12. Hence, the flow of events connecting all the symbols will convey a visual operating sequence of the designated task. This will enhance in-depth discussion for better task planning and improvement which will be demonstrated in Section 3.6.

Table 5 shows an operation chart that describes effort constraints for each DevWE as part of the built-in measures. This chart serves as a worksheet for estimated effort, performance rating, and standard effort derivation. A dark filled estimated effort cell denotes the prime DevWE, representing manual activity that involves direct effort conducive to project success. A gray filled estimated effort cell denotes idling (due to delay) or overhead (due to discussion) that somehow is inherent to humans. The future machine learning substitute (if it is realized) might not need these gray filled elements and could be eliminated.

Each DevWE performed by an average (qualified and trained) developer is measured as the normal effort bearing the relationship

$$\begin{aligned}&\text{Normal effort (NE)}\\&\quad = \text{estimated effort} * \text{performance rating } (pr) \qquad (1)\end{aligned}$$

where estimated effort is the effort required to perform the DevWE, pr is the performance rating as decided by the project manager (PM). An average developer is rated at 100%. The standard effort of each DevWE is an unstressed achievable effort by the average developer plus allowances expressed in fractions (%) of NE. That is,

$$\text{Standard effort} = \text{NE} + (\text{allowances} * \text{NE}) \qquad (2)$$

Therefore, the operation chart expresses every measurable DevWE of the process under investigation. Computations are carried out as follows. The normal effort of password mechanisms/authentication operation (#1) is equal to 0.50 ∗ 100% = 0.50, which gives the standard effort of 0.50+(5%∗0.50) = 0.525. Other operations proceed in the same fashion. The activity operation summary is the sum of all prime DevWEs, that is, the sum

Table 5 Operation chart.

Story: forgotten password		Programmer: John Doe	
Activity ID:	Build:	Doc#:	Page: 2 of 3
Date:	GroupID:	Duration:	
Authorized by:		Project:	

Operation	*Pr*	Estimated effort		Allowance (%)		Standard effort
1. password mechanisms, authentication	100	0.50	⬛	5	🟨	0.525
2. event flow and test (checkpoint) plan	100	0.50	⬛	0	🟨	0.500
3. discussion, technical resolution	100	0.67	▨	0	🟦	0.667
4. search for reuse package and code	100	2.00	⬛	5	🟨	2.100
5. screen design, test case construction	100	2.00	⬛	5	🟨	2.100
6. delay	100	0.11	▨	0	🟦	0.111
7. write code	100	1.00	⬛	5	🟨	1.050
8. debug with test cases	110	1.00	⬛	8	🟨	1.188
9. adjustment meeting	100	0.67	▨	0	🟦	0.667
10. perform unit test	100	1.00	⬛	2	🟨	1.020
11. delay	100	0.11	▨	0	🟦	0.111
12. perform integration test	100	1.00	⬛	2	🟨	1.020

Summary

Activity	Effort	Overhead	yield Effort	Remarks
Operation	9.503	1.334	10.837	
Idle	0.000	0.222	0.222	
Utilization (%)	87.69	12.31	97.99	

Estimated effort: ⬛ prime DevWE ▨ idle/overhead *Pr*: performance rating

of #1, 2, 4, 5, 7, 8, 10, and 12 = 9.503. Idle is taken from the two delays, namely, #6 and #11, yielding 0.111 + 0.111 = 0.222. The overhead comes from discussion/technical resolution (#3) and adjustment meeting (#9), that is, 0.667 + 0.667 = 1.334. The operating time is 9.503 + 1.334 = 10.837 and the overall task time is 10.837 + 0.222 = 11.059. Therefore, the utilization percentage of effort and yield effort are 9.503/10.837 ∗ 100 = 87.69% and 10.837/11.059 ∗ 100 = 97.99%, respectively.

3.5 Performance Assessment

Performance evaluation can be determined quantitatively in a step-by-step standard effort yield by Eq. (2). The procedure can be extended to cover multiple task assignments or project level as follows.

Let X_{kj}, $k = 1, \ldots, Q$ denotes the effort expended on DevWE(X_k), and j denotes programmer j performing X_k. The estimated effort E_j of programmer j participating in a project can be determined by

$$\mathrm{E}_j = X_{1j} + X_{2j} + \cdots + X_{kj}, \quad k = 1, \ldots, Q, \quad j = 1, \ldots, m \tag{3}$$

where m denotes the number of programmers in the team. The normal effort (NE) of programmer j becomes

$$\mathrm{NE}_j = \mathrm{E}_j * Pr_j \tag{4}$$

where Pr_j denotes the performance rating of programmer j. Hence, the total normal effort (TNE) of the project can be computed by

$$\mathrm{TNE} = \sum_{j=1}^{m} \mathrm{NE}_j \tag{5}$$

The total standard effort (TSE) is therefore equal to

$$\mathrm{TSE} = \mathrm{TNE} + (Aw * \mathrm{TNE}) \tag{6}$$

where Aw denotes the allowances for all personnel. One may contend that the allowance factor should not be made 'one size fits all'. Mathematically, this factor can be established on an individual basis. In this article, it is intended to simplify some complicated issues such as 'Why did testers get higher allowances than programmers?'

This generalization can serve as a trace back to the above task assignment by means of a workload breakdown chart of stage 7 as shown in **Table 6**. This chart not only provides work load breakdown information to all parties involved, but also reveals potential process improvement since inefficient bottlenecks will be uncovered from the analysis of each story. All of these results demonstrate the equivalent effort computation in stage 8. The shaded slots denote the person in charge of the designated DevWE. For example, P1 (blue/gray) is in charge of task #3, 9, 11, 12. Note that some DevWEs, such as meeting/discussion (#3 and #9), call for all members to participate. Thus, the total percentage of work load distribution tallied from all members may exceed 100% scaling. For example, participation of P3 is (0.67+2.00+2.00+0.11+1.00+1.00+0.67+1.00)∗100/10.56 = 80.02%, and the total participating proportion becomes 23.20 + 21.16 + 80.02 = 124.38%.

Consider the effort expended by each programmer. The total standard effort of this forgotten password story can be determined from Eqs. (3)–(6), assuming the performance rating of P1,

Table 6 Workload breakdown chart by team member.

Story: forgotten password				Doc#:		Page: 3/3
Project:				P1: John Doe (leader)		
Build:		GroupID:		P2: Jane Doe		
Date:		Duration:		P3: Jim Smith		
Operation	code	Estimated effort		P1	P2	P3
1. password mechanisms, authentication	P	0.50	■		🟨	
2. event flow and test (checkpoint) plan	P	0.50	■		🟨	
3. discussion, technical resolution	M	0.67	▒	🟦	🟨	🟩
4. search for reuse package and code	O	2.00	■			🟩
5. screen design, test case construction	O	2.00	■			🟩
6. delay	D	0.11	▒			🟩
7. write code	C	1.00	■			🟩
8. debug with test cases	A	1.00	■			🟩
9. adjustment meeting	M	0.67	▒	🟦	🟨	🟩
10. perform unit test	I	1.00	■			🟩
11. delay	D	0.11	▒	🟦		
12. perform integration test	I	1.00	■	🟦		
Total		10.56		23.20%	21.16%	80.02%

P2, and P3 are 100%, 100%, and 120%, respectively, and the allowances are set to 5%.

$$E_{P1} = M_3+M_9+D_{11}+I_{12} = 0.67+0.67+0.11+1.00 = 2.45$$
$$NE_{P1} = 2.45*1.0 = 2.45$$
$$NE_{P2} = (P_1+P_2+M_3+M_9)*1.0$$
$$= (0.5+0.5+0.67+0.67)*1.0 = 2.34$$
$$NE_{P3} = (M_3+O_4+O_5+D_6+C_7+A_8+M_9+I_{10})*1.2$$
$$= (0.67+2.00+2.00+0.11+1.00+1.00+0.67+1.00)*1.2$$
$$= 10.14$$
$$TNE = 2.45+2.34+10.14 = 14.93$$
$$TSE = 14.93+(0.05*14.93) = 15.677\ \text{m-h}$$

Note that M_3 of NE_{P2} stands for operation M#3 or 'discussion, technical resolution' of programmer P2, while A_8 of NE_{P3} stands for operation A#8 or 'debug with test cases' of programmer P3, and so on. The full subscript notation of X_{kj}, i.e., M_{3P2} or A_{8P3}, is omitted for brevity.

The standard effort will be converted to monetary terms by '*piecework*' in stage 9. Let p_j be the pay rate for job classification j, e.g., system analyst, tester, programmer, etc. The cost (Ct) of this task can be determined as follows:

$$Ct = \text{standard effort}_j * p_j \qquad (7)$$

The only concern of this assessment scheme is a fair pay rate for all job classifications of the development process. How this amount should be allotted is beyond the scope of this work. Using the above example, the cost of this story becomes

$$Ct = SE_{P1} * p1 + SE_{P2} * p2 + SE_{P3} * p3$$
$$= (2.45 + 0.05 * 2.45) * 18 + (2.34 + 0.05 * 2.34) * 15$$
$$+ (10.14 + 0.05 * 10.14) * 15$$
$$= 242.865$$

assuming $p1$ of P1 = \$18/m-h, $p2$ of P2 = \$15/m-h, and $p3$ of P3 = \$15/m-h.

3.6 Application

In order to see how this novel prospectus is applied in real project effort estimation, a closer look into the symbolic flow map, operation chart, and workload breakdown chart of the forgotten password story demonstrates a methodical process by means of a UML activity chart in **Fig. 7**.

The figure reflects the process control of activities in the forgotten password story (see Fig. 5 for procedural detail). From the entry point in main activity section, the process details exhibit a visual trace of activities in input and refresh sections that are performed by the developers. This process trace reveals a few activity checkpoints [19] for design improvements. The first improvement has to do with concurrency of user and CAPTCHA tests (steps 2 and 8). If this application were to run on a limited resource mobile device, the concurrency would be rendered unnecessary since it required too much computation resources. In light of Green Technology, the less power consumed, the higher the consumption economy becomes. Thus, the process can be sequentially re-ordered to perform user test first, followed by CAPTCHA test.

The second improvement could consider the sequence of process control flow in the above activity chart that jumped from one stage to the next. This pattern could be visualized from the symbolic flow map where process flow sweeps from one side to the other, changing flow direction intermittently. Consider swapping between #9 (**adjustment meeting**) and #10 (**perform unit test**), the zigzag flow will alter the work pace as follows: #8 (debug with test cases), #9 (perform unit test), and #10 (adjustment meeting). That is to say, the original ordering started from A-M-I-D, that is, sitting down to debug, getting up and left for adjustment meeting, and coming back to sitting down again to perform unit

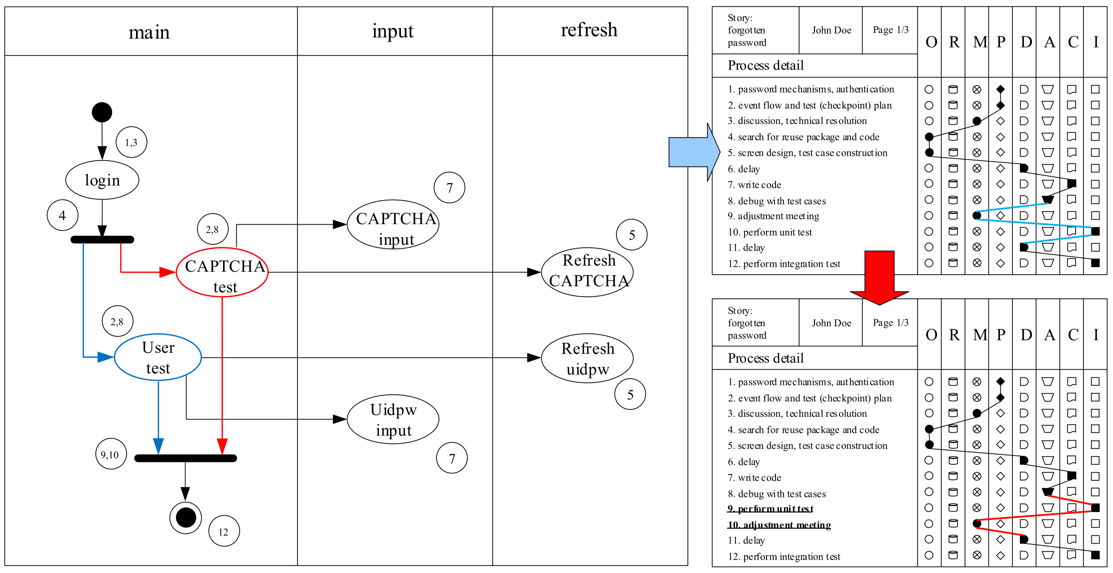


Fig. 7 Forgotten password process layout improvement.

test, while the re-group ordering became A-I-M-D, that is, sitting down to debug, continue sitting down to perform unit test, and getting up and leaving for adjustment meeting. Apparently, the re-group ordering was physically less disruptive than the original ordering. However, if the project mandates required that there should be a formal review ('adjustment meeting' in this example) before any tests can begin, then the swap cannot be carried out.

4. Experiment and Results

Two experiments were setup to compare the proposed and traditional analytics as illustrated in **Fig. 8**. Details will be described in the sections below.

4.1 Subjects

Since this was a pioneer attempt to explore these fine-grained building blocks for manual operations, there were no existing software standards, disciplines, or practices to gauge the novelty of the proposed approach. Consequently, no industrial software organization was willing to participate since they could not afford the expenses and losses of productivity.

It was decided to carry out the experiment on senior students in computer science (CS) major as they were ideal representatives of this study. For one thing, these students have undergone rigorous test and train processes. The host Mathematics and Computer Science Department is highly ranked of the nation. The applicants' entrance exam score had to be in top ranking to be admitted. Their collegiate training underwent many mathematics and computer science mandatory classes such as Calculus I, II, III, Probability and Statistics, Differential Equations, Linear Algebra, Discrete Mathematics, Programming Techniques, Data Structures and Fundamental Algorithms, Computer Systems, Operating Systems, Database Systems, Theory of Computations, Algorithm Designs and Analysis, Programming Languages and Compiler, etc. All of them were in average standing as the academic atmosphere was highly competitive. In addition, the students have gone through their junior summer internship in many local organizations which prepared them to be qualified work-

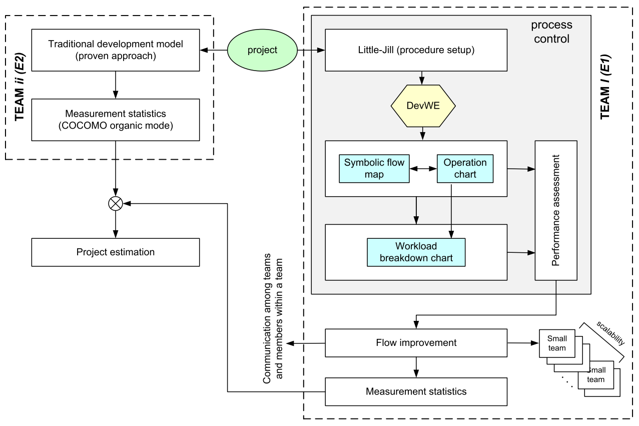


Fig. 8 Comparison of DevWE and conventional analytics.

force for the industry. Hence, they were not just any IT enthusiasts who volunteered to participate in this study.

4.2 Experimental Setup

Setup of the proposed approach was arranged as follows. (1) assigned a small sized software project so that it could be completed in four months by a team of 5 members. Due to the limited number of 10 members to be allocated, the team was organized as follows: divided the 5-member into two groups of 2 members and a shared PM per group, i.e., 2 + PM and PM + 2. The shared PM could oversee activities of both groups to ensure proper load balancing with the help of the instructor. His role was somewhat demanding in that he had to switch his thoughts, planning, assisting, and supervising each group when he was engaging with the group. Fortunately, the nature of experiment was relatively straightforward. As such, there would be no psychological differences between both groups as they worked toward a common goal. The workload spanned 3 hours per day, 13 days per month. Should this be a real life production software project (having ample members to select), it would be better to employ two independent members since the shared member might not be able to fulfill his assigned responsibilities well. (2) measured manual and functional activities.

4.3 Procedures

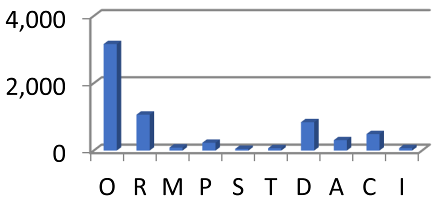
Two semester class experiments were conducted using one team per class. The first semester experimented on team *i* (E1) while the second semester experimented on team *ii* (E2). Variations among member qualifications of the two teams were negligible owing to the aforementioned stringent subject selection process. In order to minimize differences between the two teams, both experiments worked on familiar web-based applications to avoid any unfamiliar guesswork and difficulties on the project problem and the results so obtained. From Fig. 8, team E1 employed Little-Jill to set up the procedure as demonstrated in Table 4. They then deployed DevWE analytic using the three charts as their performance assessment aids, made necessary adjustments on flow improvements with a lot of communication among the small teams. All manual activities represented by DevWEs resulted in output measurement statistics to be compared with the other team's output. Team E2, on the other hand, worked on a similar assignment using conventional approach and sizing metrics, i.e., COCOMO, organic mode, LOC, and FP, to arrive at a total effort estimation. In the meantime, both teams had to produce four deliverables, namely, software requirements specification (SRS), design, an interim report which summarized all the problems, modifications, and corrections to the previous two deliverables, and complete documents. Every member also served in the role of a document writer to describe their own assignment, hence first-hand data was being recorded.

4.4 Results

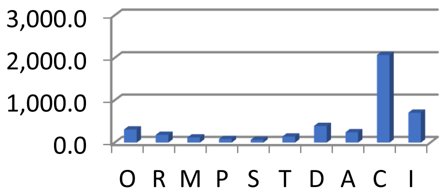
The following descriptions and statistics were collected from team *i*. Students started by breaking down requirements into activities based on the procedures described in Table 4. They built symbolic flow map and operation chart. The PM collected project data to build workload breakdown chart of the members, summary of frequency count, and effort by DevWE measured in man-hours. The class overall results are shown **Fig. 9** (a), and 9 (b), respectively.

The results show that O (operation) and R (read) dominate the most frequently performed DevWE, while C (code) and I (inspect) take the heaviest and second heaviest effort, respectively. Further analysis revealed that the frequency dominant O and R turned out to be relatively short and easy DevWE from an effort standpoint, they would not noticeably slow the team performance down. In the meantime, I surpasses C to expend the heaviest effort per frequency count. The rationale is because conventional coding focuses on *pure* quantitative coding measurement in the form of LOC/man-day, the fine-grained scrutiny of DevWE separates manual I into code organization planning and reviews. Hence, both manual activities were revealed to use up higher effort than the rest of manual activities. The average overall effort is 865.4 man-hours per team.

Figure 10 summarizes the statistics according to the experimental setup, i.e., manual versus functional activities. Manual activities encompass MP + RDAI for a total of 2,612 man-hours, while the functional activities encompass OSTC for 3,776 man-

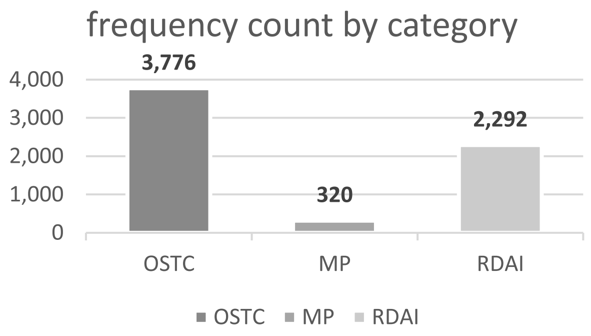


(a)

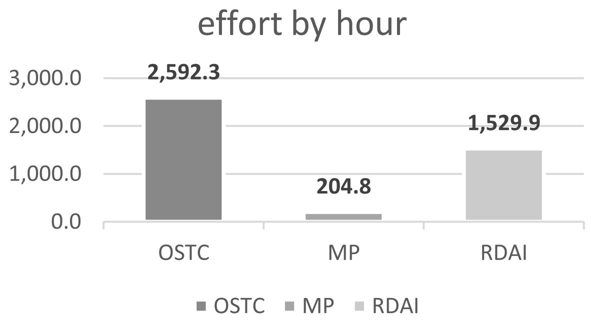


(b)

Fig. 9 (a) Class overall frequency count, (b) Class overall effort.



(a)



(b)

Fig. 10 (a) Frequency count of manual vs. functional activities, (b) Effort by hour of manual vs. functional.

hours. From Fig. 10 (a), the ratio of manual activity frequency to functional activity is $2612/3776 = 69\%$. That is to say, manual activity occurs $2612/(2612 + 3776) = 41\%$ of the time in the entire project activities. A similar story holds for the effort by hour depicted in Fig. 10 (b), that is, the ratio of manual effort to functional effort is $1735/2592 = 67\%$ and the effort by hour for the entire project is $1735/(1735 + 2592) = 40\%$. What these pictures convey is that *almost half of the effort estimation actually is manual activity*.

The experiment of team *ii* was considerably more straightforward than that of team *i* since there was no explicit treatment of manual operations. Planning was performed at the outset of the project to set up FP estimation for early sizing since there was no coding to collect the corresponding LOC. All necessary functions were tallied to obtain project FP estimate which subsequently was converted to equivalent LOC (see **Table 7**). For example, suppose the project contained 66 function points, if it were using VB

Table 7 Ratios of logical source-code statements to function points for selected programming languages (Jones, 1995).

Language	Nominal	Source statements per function point		
	Level	Low	Mean	High
First generation	1.00	220	320	500
Basic assembly	1.00	200	320	450
Macro assembly	1.50	130	213	300
C	2.50	60	128	170
Basic (interpreted)	2.50	70	128	165
Second generation	3.00	55	107	165
Fortran	3.00	75	107	160
Algol	3.00	68	107	165
Cobol	3.00	65	107	170
CMS2	3.00	70	107	135
Jovial	3.00	70	107	165
Pascal	3.50	50	91	125
Third generation	4.00	45	80	125
PUI	4.00	65	80	95
Modula 2	4.00	70	80	90
Ada 83	4.50	60	71	80
Lisp	5.00	25	64	80
Forth	5.00	27	64	85
Quick Basic	5.50	38	58	90
C++	6.00	30	53	125
Ada9X	6.50	28	49	110
Database	8.00	25	40	75
Visual Basic (Windows)	10.00	**20**	32	37
APL (default value)	10.00	10	32	45
Smalltalk	15.00	15	21	40
Generators	20.00	10	16	20
Screen painters	20.00	8	16	30
SQL	27.00	7	12	15
Spreadsheets	50.00	3	6	9

(Windows) low 20 statements per function point, the converted LOC would have become $66 * \underline{\mathbf{20}} = 2{,}952$ LOC. For team *ii*, their estimated sizing became 2.952 KDSI.

Since the project was a familiar web-based application, students found no difficulty in getting themselves up to speed. This was treating as an organic mode work effort according to COCOMO estimation as follows:

$$\text{Effort} = \text{A} * (\text{KDSI})^{b} \tag{8}$$

where A denotes processing mode, i.e., organic = 2.4, semi-detach = 3.0, and embedded = 3.6, b denotes a processing mode constant corresponding to organic, semi-detach, and embedded modes, i.e., 1.05, 1.12, and 1.20, respectively. Effort is measured in man-month (MM), yielding

$$\begin{aligned}\text{Effort} &= 2.4 * (2.952)^{1.05}\\ &= 4.537\,\text{MM}\\ &= 884.8\,\text{man-hours}\end{aligned}$$

The effort of team *ii* did not deviate much from that of team *i*, i.e., 864.4 vs. 884.8 man-hours. The fact that team member

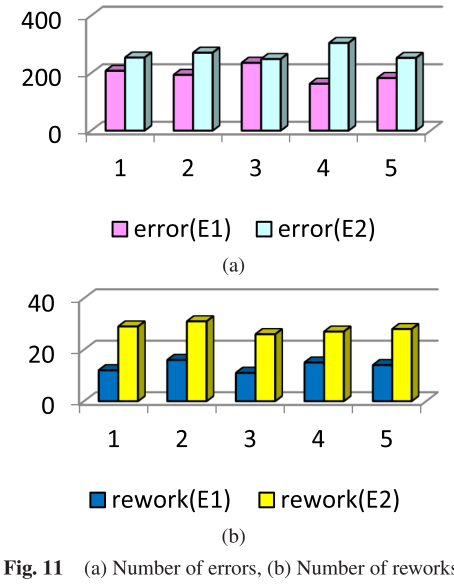


Fig. 11 (a) Number of errors, (b) Number of reworks.

qualifications were quite uniform owing to the university highly-valued admission criteria contributed to their output deliverables of comparable quality in accordance with the project SRS. The only difference was the execution exercised by both teams. Thus, the number of errors and reworks committed by them was comparably consistent. **Figure 11** (a) and 11 (b) show the amount of errors and reworks for both experiments. The error counts stemming from typographic, parameter setup, syntax, etc., were relatively comparable from various standpoints, e.g., project assignment, team ability, size, and complexity. The rework tallies of the second experiment (E2), on the other hand, turned out to be somewhat higher since it was less thorough, detailed, and monitored than the first one (E1). This was due to traditional project setting of E2 that encompassed phase-wise execution where design and logical coding errors that needed to be reworked might not be unveiled until late in the project. The proposed approach scrutinized every step using DevWE breakdowns that could uncover the errors to be reworked early. Fortunately, since the project was small, straightforward, and familiar to the teams, the extra reworks produced by team E2 did not require any higher efforts to be expended than team E1. All in all, effort estimation was relatively close and team E1 demonstrated slightly better performance than their E2 counterpart. For production scale projects, however, the amount of errors and reworks could adversely affect the accuracy of project effort estimation.

5. Conclusion

A routine car tune-up usually costs the same at any certified auto shops that does such work. Why should technology-oriented estimation of manual operations be unpredictably different, especially in an outsourcing situation? What hinders the standard costing to be established?

This study introduces a novel analytic to measure manual activity that is inherent to software development by means of a set of quantifiable metrics called Developer Work Elements (DevWE). Two experiments were conducted to gauge the efficacy of the proposed analytics in comparison with traditional approach and metrics. The results show that DevWE helps reduce the number of errors and reworks of the project considerably. The benefits are attributed to fine-grained scrutiny of manual work that helps spot early human errors before they slip through subsequent phases. In addition, the visual charting technique offers discretization and traceability of human-oriented operations.

By virtue of the proposed analytics, many manual activities pertaining to the development process can be compiled into a checklist that is easily verified according to DevWE definitions. As a consequence, the manual operating costs of estimation effort can be standardized in the same manner as car tuning.

Future prospect could focus on computerizing this DevWE measurement methodology to lessen the manual burden of software developers so that they can spend more time and effort on production work. Hence, the quality of work product or service can be accomplished by any software development teams in the same way as the Therbligs have contributed to the Industrial Engineering counterpart.

References

[1] AACE International Recommended Practice No.74R-13, TCM Framework: 7.3—Cost Estimating and Budgeting (2014).

[2] Albrecht, A. and Gaffney, J.J.: Software function, source lines of code, and development effort prediction: A software science validation, *IEEE Trans. Software Engineering*, Vol.SE-9, No.6, pp.639–648 (1983).

[3] Bailey, J.W. and Basili, V.R.: A meta-model for software development resource expenditures, *Proc. 5th International Conference on Software Engineering*, pp.107–116 (1981).

[4] Banker, R., Kauffman, R. and Kumar, R: An empirical test of object-based output measurement metrics in a computer aidedsoftware engineering (case) environment, *Journal of Management Information Systems*, Vol.8, No.3, pp.127–150 (1991).

[5] Basili, V.R., Selby, R.W. and Hutchens, D.H.: Experimentation in Software Engineering, *IEEE Trans. Software Engineering*, Vol.SE-12, No.7, pp.733–743 (1986).

[6] Billows, D.: How To Do 3 Point Estimating, *Project Management Tools* (2018), available from ⟨https://4pm.com/2018/01/05/3-point-estimating-2⟩ (accessed 2019-10-02).

[7] Boehm, B.: *Software Engineering Economics*, Upper Saddle River, NJ: Prentice Hall PTR (1981).

[8] Boehm, B., Abts, C., Brown, A., Chulani, S., Clark, B., Horowitz, E., Madachy, R., Reifer, D. and Steece, B.: *Software Cost Estimation with COCOMO II*, Upper Saddle River, NJ: Prentice Hall PTR (2000).

[9] Briand, L.C., Emam, K. and Bomarius, F.: COBRA: A hybrid method for software cost estimation, benchmarking, and risk management, *Proc. 20th International Conference on Software Engineering*, pp.390–399 (1998).

[10] Briand, L.C. and Wieczorek, I.: Resource Estimation in Software Engineering, *International Software Engineering Research Network, Technical Report*, also appears in *Encyclopedia of Software Engineering*, New York, John Wiley & Sons, pp.1160–1196 (2002).

[11] Brooks, F.P.: The Mythical Man-Month, *An Essays Software Engineering Anniversary Edition*, Addison-Wesley (1995).

[12] Cass, A.G., Lerner, B.S., McCall, E.K., Osterweil, L.J., Sutton Jr., S.M. and Wise, A.: Little-JILJuliette: A Process Definition Language and Interpreter, *Proc. 22nd International Conference on Software Engineering*, Limerick, Ireland, pp.754–757 (2000).

[13] Christmansson, M., Falck, A.C., Amprazis, J., Forsman, M., Rasmusson, K.L. and Kadefors, R.: Modified method time measurements for ergonomic planning of production systems in the manufacturing industry, *International Journal of Production Research*, Vol.38, No.17, pp.4051–4059 (2000).

[14] Chung, E.: Analogous Estimating vs Parametric Estimating for PMP Exam, *The Complete PMP Certification and Study Guide 2019* (2019), available from ⟨https://edward-designer.com/web/analogous-estimating-vs-parametric-estimating-for-pmp-exam⟩ (accessed 2019-10-02).

[15] Ferguson, D.: Therbligs: The Keys to Simplifying Work, *The Gilbreth Network* (2000), available from ⟨http://gilbrethnetwork.tripod.com/therbligs.html⟩ (accessed 2017-08).

[16] Finnie, G.R., Wittig, G.E. and Desharnais, J.M.: A Comparison of Software Effort Estimation Techniques: Using Function Points with Neural Networks, Case-Based Reasoning and Regression Models, *J. Systems Software*, No.39, pp.281–289 (1997).

[17] Guerrera, G.: 5 Steps to Bottom-Up Estimating, *IT Project Blog* (2010), available from ⟨http://www.nuwavetech.com/it-project-blog/bid/44872/5-Steps-to-Bottom-Up-Estimating⟩ (accessed 2019-10-01).

[18] Heiat, A.: Comparison of artificial neural network and regression models for estimating software development effort, *Information and Software Technology*, Vol.44, pp.911–922 (2002).

[19] Humphrey, W.S.: *Introduction to the Personal Software Process*, Addison Wesley Longman, Inc. (1997).

[20] International Function Point Users Group (IFPUG): *Function Point Counting Practices Manual*, Release 4.0, Blendonview Office Park, 5008-28 Pine Creek Drive, Westerville, OH 43081-4899 (1994).

[21] Johnson, P.M.: Searching under the Streetlight for Useful Software Analytics, *IEEE Software*, pp.57–63 (2013).

[22] Jones, C.: Software Cost Estimating Methods for Large Projects, *CrossTalk: The Journal of Defense Software Engineering*, pp.8–12 (2005).

[23] Jones, C.: Backfiring: Converting lines of code to function points, *Computer*, pp.87–88 (1995).

[24] Jørgensen, M. and Shepperd, M.: A Systematic Review of Software Development Cost Estimation Studies, *IEEE Trans. Software Engineering*, Vol.33, No.1, pp.33–53 (2007).

[25] Kalach, M.: Bottom-Up estimating, *Techniques Wiki* (2019), available from ⟨https://www.projectmanagement.com/wikis/368761/Bottom-Up-estimating⟩ (accessed 2019-10-02).

[26] Karner, G.: Resource estimation for objectory projects, *Objective Systems* SF AB (1993).

[27] Kemerer, C.F.: An empirical validation of software cost estimation models, *IEEE Trans. Software Engineering*, Vol.30, No.5, pp.416–429 (1987).

[28] Matson, J.E., Barrett, B.E. and Mellichamp, J.M.: Software development cost estimation using function points, *IEEE Trans. Software Engineering*, Vol.20, No.4, pp.275–287 (1994).

[29] nesma, *Early Function Point Analysis*, version: July 15, 2015, available from ⟨https://nesma.org/wp-content/uploads/2015/11/Early-Function-Point-Analysis-vs-2015-07-15-EN.pdf⟩ (accessed 2019-10-01).

[30] Newkirk, J. and Martin, R.C.: *Extreme Programming in Practice*, Addison-Wesley (2001).

[31] Park, H. and Baek S.: An empirical validation of a neural network model for software effort estimation, *Expert Systems with Applications*, Vol.35, pp.929–937 (2008).

[32] Parnas, D.L.: Software Aging, *Proc. 16th International Conference on Software Engineering* (*ICSE '94*), pp.279–287 (1994).

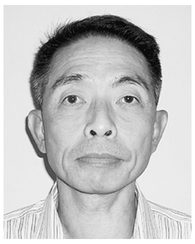
[33] Putnam, L.: A general empirical solution to the macro software sizing and estimating problem, *IEEE Trans. Software Engineering*, Vol.SE-4, No.4, pp.345–361 (1978).

[34] Sophatsathit, P.: Fine-Grained Work Element Standardization for Project Effort Estimation, *Journal of Software Engineering and Applications*, Vol.7, pp.655–669 (2014).

[35] Symons, C.R. and Lesterhuis, A.: *COSMIC: Introduction to the COSMIC method of measuring software*, version 1.1 (2016), available from ⟨https://cosmic-sizing.org/publications/⟩ (accessed 2019-10-01).

[36] Walston, C.E. and Felix, C.P.: A method of Programming Measurement and Estimation, *IBM Systems Journal*, Vol.16, No.1, pp.54–73 (1977).

[37] Yang, Y., He, M., Li, M., Wang, Q. and Boehm, B.: Phase distribution of software development effort, *Proc. 2nd ACM-IEEE International Symposium on Empirical SoftwareEngineering and Measurement*, pp.61–69 (2008).



Peraphon Sophatsathit received his B.S. in Industrial Engineering from Chulalongkorn University, Thailand, his M.S. degrees in Industrial Engineering and Computer Science from University of Texas at Arlington, and his Ph.D. degree in Computer Science from Arizona State University. He is an associate professor of computer science at the Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University. His research interests include software engineering, design, and construction.