COCOMO II Parameters and IDPD: Bilateral Relevances

Ramin Moazeni
Computer Science Department
University of Southern California
Los Angeles, U.S.A.
moazeni@usc.edu

Daniel Link
Computer Science Department
University of Southern California
Los Angeles, U.S.A.
dlink@usc.edu

Barry Boehm
Computer Science Department
University of Southern California
Los Angeles, U.S.A.
boehm@usc.edu

ABSTRACT

The phenomenon called Incremental Development Productivity Decline (IDPD) is presumed to be present in all incremental software projects to some extent.

COCOMO II is a popular parametric cost estimation model that has not yet been adapted to account for the challenges that IDPD poses to cost estimation. Instead, its cost driver and scale factors stay constant throughout the increments of a project. While a simple response could be to make these parameters variable per increment, questions are raised as to whether the existing parameters are enough to predict the behavior of an incrementally developed project even in that case. Individual COCOMO II parameters are evaluated with regard to their development over the course of increments and how they influence IDPD. The reverse is also done. In light of data collected in recent experimental projects, additional new variable parameters that either extend COCOMO II or could stand on their own are proposed.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management – *cost estimation, life cycle, productivity.*

General Terms

Management, Measurement, Economics, and Human Factors.

Keywords

Parametric cost estimation; IDPD; incremental development; cost drivers; scale factors

1. INTRODUCTION

IDPD is the phenomenon of an overall productivity decline over the course of several increments.

For the purposes of IDPD research, productivity is measured in SLOC/Effort. While SLOC are not an ideal measure of software productivity because they are not very useful in improving it [1], much useful work is not concerned with the creation or testing of code and different management perspectives should be taken into account when measuring productivity [2]. IDPD research uses them as a measure because they are subjective and difficult to measure. In addition, code churn, which includes debug code churn and refactoring code churn, is more associated with defect density than productivity [3] [4].

The IDPD between two increments is the percentage by which the productivity in SLOC/Effort is diminished between them. (If the productivity of a given increment is 80% of a given previous one, this is called an IDPD of 20%.)

IDPD research is trying to answer the question of how much new output an organization can expect for a new increment.

1.1 Incremental Development

This paper focuses on those incremental development projects whose increments are 1) more than one, 2) not less than a tenth of the previous one as expressed in SLOC (otherwise it will be counted with the previous one, 3) not just a bug fix of a previous increment, 4) each adding new functionality, and 5) depending on and coherent with previous increments. Section 2 describes the status of IDPD research. Section 3 gives an overview about software cost estimation. Section 4 looks at IDPD in relation to the COCOMO II cost estimation model and potential new parameters.

2. Status of IDPD research

The concept has been introduced with some first observations [3]. It has been found to be supported by Lehman's Laws of Software Evolution [4].

Research remains to be done on several areas of interest such as IDPD variations between different project domains, a predictive model of IDPD and whether the choice of a life cycle model or other management influences can mitigate IDPD.

2.1 Research Methodology

2.1.1 Approach

In order to arrive at a statistical model of IDPD, the attributes of increments as well as the parameters of the projects, their personnel and their environments have to be collected. The collected environmental data (cost drivers, scale factors) are then evaluated with regard to their development over the course of increments and whether additional new variable parameters are needed or to either extend COCOMO II existing parameters.

Table 1: Characteristics of our data sources

	Typical KSLOC	SLOC collection	Parameter collection	Main issue(s)
Software Industry	>100	Various tools, unverifiable	Unverifiable	Secrecy, data quality
Controlled experiments	<5	UCC, source available	Authors	Individuals have great influence
Open source	Any	UCC, source available	Authors	Hard to collect cost parameters

2.1.2 Data collection

Data has been collected from the software industry, controlled experiments, and open source. Table 1 shows the characteristics of our data sources.

The core attributes of each data point are actual effort, actual size (amount of code added, modified and deleted), and rating levels of the COCOMO II cost drivers and scale factors. Effort is collected in person-hour and converted into person-month using the standard COCOMO model that defines 152 hours per person-month.

The size metrics are collected by using code counting tools to compare differentials between two baselines of the source program. The size metrics are based on the logical SLOC definition, adopted from the Software Engineering Institute [7].

2.1.3 Controlled Experiments

To better control over data collection and project parameters, a controlled experiment was conducted with 21 graduate students. The amount of new requirements and personnel composition of the projects and related teams were changed throughout the project at irregular intervals. Data was collected from the students before the formation of teams, after each week on their projects and at the end of the semester.

2.1.4 Open Source Projects

The problem with open source projects lies in obtaining effort data. In order to mitigate that fact, data has only been collected on projects that are major, and only over limited amounts of time per set of observations. This makes it reasonable to assume a relatively constant number of contributors, which in turn makes it possible to assume a constant ratio of SLOC output to effort.

Table 2. COCOMO II Cost Drivers and Scale Factors expected to have a decrease in effort

Name	Description	Comments/Special cases
PREC	Project precedence	Project becomes more familiar
TEAM	Team interactions	If continuity is good, customer-developer teams grow closer
PMAT	Process maturity	CMMI level increases should reduce rework
RUSE	Instances of reuse	Reused components may need further capabilities Later increments will tend to see less reuse with the exception of systems that are built top to bottom
DOCU	Amount of documentation	Foundational increments need the most and best documentation.
PVOL	Development platform volatility	In the beginning, platform has not solidified yet
ACAP	Analyst capability	Analysts will gain capabilities if personnel continuity good
PCAP	Programmer capability	Programmers will gain capabilities if personnel continuity good
TOOL	Quality of development tools	Better tools will be acquired over time
SITE	Collocation and communication	If continuity good, teams grow closer and communications improve

3. Cost Estimation

Organizations planning sizable incrementally developed projects will be interested in estimating the cost of those increments. This applies to projects that have their number of increments or requirements determined and set in advance (like custom-built applications) as well as to open-ended projects (e.g. operating systems).

The types of possible cost estimation models that have been identified [8] and could be applied to non-incremental development can also be applied to incremental development include Algorithmic (also known as parametric), Expert Judgment, Analogy, Parkinson, Price-to-Win, Top-Down and Bottom-Up. Out of these, a parametric model has the advantage that it is predictive, does not depend on the availability of resources outside the project (save for the ability to set the parameters reasonably) and that its accuracy and quality can be objectively measured on the basis of existing project data.

Though not all parameters of all models can be mapped or converted to each other, the large majority of them can be [5]. For this reason, it becomes feasible to evaluate data collected using any model with any other model.
While it would therefore be scientifically viable to base an IDPD model on parameters from each of these models or a synthesis of all of them, the scope of this document dictated that only one be chosen. The choice fell on COCOMO II not because it is of higher quality than the others, but because data collected using its parameters was most readily available and the fact that if an IDPD model could be based on COCOMO II parameters, it should be possible to map it to the other models.

It may be helpful to consider how cost drivers and scale factors behave over the course of an incremental project.Collected industry data suggests cost drivers to be variable across increments. In contrast to this, the incremental development model for COCOMO II cost estimation model assumes that the cost drivers remain constant across the increments of the whole project [6].

Constant cost drivers are unable to explain the changes in productivity over the course of an incrementally developed project. Therefore, this model in its current form is not equipped for the estimation of the cost of incremental development.

4. IDPD and COCOMO II

Adapting COCOMO II to incremental development in order to predict IDPD also presents an opportunity to re-evaluate some parameters for the current time. While some parameters have held up over time, some may not have a major influence anymore nowadays (TIME and STOR come to mind, defined in **Table 3**).

4.1 COCOMO II parameters

Table 2 and **Table 3** list expectations of whether the effort factor related to a given COCOMO II cost driver is expected to increase, decrease or have no expectation of any trend. The reason why the tables look at the expectation for effort multipliers instead of cost driver or scale factor levels is in order to make the tables more readable for readers unfamiliar with the individ-

Table 3. COCOMO II Cost Drivers and Scale Factors expected to have no specific trend in effort

Name	Description	Comments/Special cases
FLEX	Procedural flexibility	
RESL	Architectural risk resolution	Architecture may require increasing amounts of rework; but familiarity with architecture grows
RELY	Required reliability	May increase for safety/security critical systems
DATA	Test data per line of code	May increase for big-data analytics
CPLX	Complexity	May increase with need to become part of system of systems or to support increasing demands
TIME	Percentage of CPU time used	Only relevant to projects with finite CPU resources or that share their execution time with others, such as in a mainframe environment; will increase then, but such projects are exceptions nowadays
STOR	Percentage of storage space used	Only relevant to projects with hard upper limits on storage or that share their storage with others; will increase then, but such projects are exceptions nowadays
PCON	Personnel continuity	High turnover reduces productivity because new team members need to be trained and affects experience related factors. Therefore, this is a pivotal cost driver for IDPD which depends on how the project is managed.
APEX	Applications experience	Applications experience will increase if personnel continuity good
PLEX	Platform experience	Platform experience will increase if personnel continuity good
LTEX	Language and tools experience	Language and tools experience will increase if personnel continuity good
SCED	Percentage of needed time available	Schedule compression becomes more expensive as system size grows

ual drivers or factors: While for some of those a higher level can mean more effort, for some others it means less effort.

Note that an expectation of no specific effort does not mean that there should not be any trend just that it can go either way depending on the situation of the project. That the expectation for all effort factors is to go in no specific direction or to decrease suggests that in a general case, a well-managed organization which is able to keep turnover under control should experience a gradual decrease in the effort that is expressed in the existing COCOMO II cost drivers when applied to individual increments. This is because the organization will grow more familiar with the project and the developers will become more experienced. This means that in the case of a well-managed organization and in non-exceptional situations, IDPD cannot be captured

Table 4. P-value of weighted multiple regressions between cost drivers and productivity

Drivers \ Project	DR-1	DR-2	DR-3	DR-4
Product Factors	0.961	0.945	0.986	0.001
Platform Factors	0	0.789	0.762	0.139
Personnel Factors	0.025	0.029	0.137	0.169
Project Factors	0.136	0.425	0	0.02
Scale Factors	0.249	0.42	0.651	0.014

Table 6. Weighted regression between cost drivers and productivity (only for cost drivers with correlation > 0.7)

	DR1	DR2	DR3	DR4
PREC	0.05	0.37	0.93	0.01
CPLX	0.34	0.55	0.94	0.45
RUSE	0.02	0.25	0.10	0.94
TIME	0.38	0.47	0.61	0.72
STOR	0.00	0.94	0.29	0.02
LTEX	0.00	0.62	0.76	0.70
SITE(Communication)	0.40	0.94	0.00	0.79

or predicted using existing COCOMO II drivers applied to increments.

While Personnel Continuity (PCON) is a parameter that cannot be directly set by an organization, it is very influential on other parameters. This means that an organization that wants to be productive in iterative and incremental development should

Table 5. P-value (< 0.08) of regression between cost drivers and productivity for 4 DR (directed research) projects

Driver \ Project	DR-1	DR-2	DR-3	DR-4
TIME	0.0022	0.08	0.08	/
STOR	0	0.04	/	/
ACAP	/	0.06	/	/
PCAP	/	0.06	/	/
RELY	/	/	/	0.07
SITE (Collocation)	0.08	0.07	/	/
PREC	/	0.01	/	/
TEAM	0.0031	/	/	/
PMAT	0.05	/	/	/

make efforts to retain its workforce, at least as far as the team of a given project is concerned.

In order to see which cost factors may have a higher influence on productivity and to increase the accuracy of predicting the IDPD factors in the next build, the relationship between individual cost drivers and productivity was evaluated by examining the p-value of regression between cost drivers and productivity.

Since there are 22 cost drivers and scale factors in total, only the ones with p-values within the significant range were reported (< 0.08) (Table 5). In these 4 directed research (DR) projects, TIME is the factor that predicts the productivity the best. Then we have STOR, PCAP and SITE (Collocation,), followed by ACAP, RELY, PREC, TEAM and PMAT.

A weighted regression was conducted on all the cost drivers versus productivity (see Table 6).

The results are still not consistent throughout the four DR projects. APEX seems to be the factor that predicts the productivity the best among all the factors. The ratings of APEX represent the level of applications experience of the project team developing the software system.

Multiple regressions between groups of cost drivers and productivity, weighted by increment sizes, were also performed (Table

4). In order to find better predictions for productivity, the added lines of codes were used as the weight of the regression. The results show that personnel factors and project factors seem to be the best factors to predict productivity within the 4 projects.

4.2 How IDPD influences parameters

In a project whose increment schedules have been planned beforehand, the size of IDPD will influence the SCED driver in subsequent increments in cases where one increment takes longer than expected. This can cause a chain reaction of schedule compressions if adjustments to the schedule aren't made.

In situations where team members are judged by their productivity, a loss in productivity can cause an increase in personnel fluctuation (captured by PCON), which will weigh negatively on other cost drivers as pointed out further above.

4.3 New Cost Estimation Parameters

Some of the main differences between monolithic and incremental projects are: 1) The existence of previous increments that need to be maintained, 2) The passing of time between releases, 3) That a large body of code (when compared to the current increment) needs to be integrated with the current increment, 4) that the previous increments may have a number of known defects, 5) that those increments may be more or less complex, and 6) that later increments may depend on previous ones to a lower or higher degree.

The first of these is trivially true from the second increment on with regard to the existence of previous ones. It has been established that previous increments need to be maintained [7] [4].

The remaining attributes of individual increments do not correspond with COCOMO II cost drivers. Since they are all measurable, albeit to different degrees, they hold interest as potential cost drivers.

4.3.1 Time between increments

The time passing between two increments causes the following phenomena [4]:

- Quality/usefulness of the software will slowly decrease all the way to zero because a system at rest will not keep pace with technological advances.
- Knowledge of the existing increments diminishes by personnel fluctuation or details being forgotten in the organization.
- New "must have" features need to be added to existing increments due to customer or market requests.
- Rework may need to be done due to changing paradigms (e.g. desktop to mobile).

Since all of these factors are exasperated the more time passes, more time between increments should cause the IDPD between them to increase. Increment productivity in terms of new SLOC over effort from several open-source projects has been found to be aligned with the time difference to the previous increment in a statistically significant way.

4.3.2 Defect number and complexity in increments

Defects of the previous increments will need to be fixed in those increments or compensated for in newer ones.

While defect complexity may not be easy to measure, it should be possible to at least categorize them into three categories:

1. Easy (e.g. typo or likely off-by-one)
2. Intermediate (requires fixing some complex interactions, but root cause is known)
3. Difficult (e.g. unknown root causes make system display occasional off-nominal behavior)

4.3.3 Comparative size of current increment

Most projects that data has been collected on have shown a decline in productivity between the first and last increment. The body of code will generally rise over the course of an incremental project. It is therefore of interest to relate the amount of existing code to the productivity of a new increment. Intuition would say that the more code is pre-existing, the more work will have to be done on maintenance and integration. In addition to that, a growing body of code will either become increasingly complex, which causes the effort to master it to grow [4] or work to become necessary to reduce the complexity [4]. Increment productivity in terms of new SLOC over effort from several open-source projects has been found to be aligned with the size of the previous increment in a statistically significant way (Table 7).

4.3.4 Coherence of the project

An influence on the complexity of increments is to what extent these increments need to make calls to functions of other increments. The measures here could lie in the number of calls to other increments, the complexity of these calls, the amount of required data flow over time between increments and the complexity of the increments that are being called (cyclomatic complexity and other measures).

4.3.5 Evaluated new Cost Drivers

Four potential parameters that would extend COCOMO II for increments have been evaluated. These were **RCPLX** (Complexity of Increment I+1 relative to complexity of existing software), **IPCON** (Staff turnover from previous increment's staff), **PRELY (** Previous increments' required reliability) and **IRESL** (Architecture and risk resolution (RESL) rating of increment I+1 relative to RESL of 1..I).

Initial data collected by the software industry did not show any good correlation with IDPD.

Ten projects have been evaluated using measures of effort and size. The best ANOVA p-values have been found when relating new lines to productivity.

4.4 Threats to Validity

4.4.1 Internal validity considerations

The productivity of the first increment of any software project can be atypical due to a number of reasons. Reasons for atypical high productivity in terms of SLOC output include auto-generated code from IDEs (Integrated Development Environments) and that initial code may be based on templates or copied

Table 7. Correlation of Code Size Metrics to Increments

	First to current		Current to last		Previous to current		Size per increment	
Project	Correlation	P-value	Correlation	P-value	Correlation	P-value	Correlation	P-value
common-daemon	-0.67	0.0086	0.17	0.2987	0.08	0.4024	-0.16	0.3008
tomcat 6.0	-0.465	0.0245	0.4663	0.0245	-0.2806	0.1302	-0.4195	0.0326
php-5.4	0.1028	0.3726	0.0384	0.4484	0.107	0.3603	-0.0068	0.4859
sendmail 8.14	0.5548	0.1684	0.5614	0.1631	-0.1356	0.4112	0.5181	0.1158
php-5.3	-0.6464	0.0003	0.698	0.0001	-0.1264	0.2724	-0.7045	0.0001
gsf-kernel	-0.5906	0.0015	0.5905	0.0015	0.1345	0.2772	-0.6184	0.0005
cmirror 0.1	-0.3215	0.1225	0.2063	0.2263	0.3132	0.1304	-0.2302	0.1873
cmirror 2.6.9	-0.0395	0.4111	0.0073	0.4776	0.2264	0.0953	-0.0405	0.4084
cluster 3.1	0.3863	0.2223	-0.7339	0.0498	-0.8014	0.028	0.7578	0.0143

and pasted from elsewhere.

Reasons for atypical low first increment SLOC productivity include exploration, preoccupation with requirements engineering and architecting for the later increments, and that it may end up with a large amount of partially-developed code not included and therefore not counted in the incremental release. (This may also be the case for later releases, but the amount of unused code should decrease when the team gains experience over the course of the project.) While these are concerns, the productivity of all examined projects still decreases between the second and last increments, though to a smaller degree. Self-reported time logs submitted by members of development teams in graded or paid settings can be questionable due to lack of precision and the temptation of embellishments. The threat is mitigated for professional and student developers by the likelihood of distortions being common to all parts of the project, which will not significantly affect the study of the development of their productivity.

4.4.2 External validity considerations

Students are typically facing fewer existential risks for failure than employees in the industry. Students may have productivity peaks and valleys due to concurrent course obligations.

5. Conclusions

- Even when conventional COCOMO II drivers are applied to the increments of such projects and made variable over them, they are unable to explain the cases where a well-managed organization (meaning one which manages personnel education and turnover well) experiences IDPD. Therefore, either new parameters need to complement or replace the existing ones in the model or the prediction has to take place outside the model.
- A controlled experiment with students showed inconclusive results.
- Promising new parameters with statistically significant predictive capabilities in several open-source projects include the size of the current increment and the time between increments and the time between the current and the first increment as well as the time between current and the last increment.
- Data should be collected using parameters that include defect and complexity analysis of code.

6. Future Work

Since no model has been found yet that closely and reliably predicts IDPD, future work should focus on that goal by employing the following methods:

- Collected data needs to be statistically evaluated further under more aspects in order to increase chances of finding patterns that predict productivity
- Project data needs to be collected and evaluated regarding the parameters proposed in this paper
- Additional potential parameters should be evaluated

7. References

1 Scacchi, Walt. UNDERSTANDING SOFTWARE PRODUCTIVITY. In *Software Engineering and Knowledge Engineering: Trends for the Next Decade*. World Scientific Press, 1995.

2 Dale, C J and van der Zee H. Software productivity metrics: who needs them? In *Proceedings of Eurometrics 92* (Nanterre 1992), EC2 Publications, 731-738.

3 Moazeni, Ramin, Link, Daniel, and Boehm, Barry. Incremental Development Productivity Decline. In *Proceedings of the 9th International Conference on Predictive Models in Software Engineering* (Baltimore, MD, USA 2013).

4 Moazeni, Ramin, Link, Daniel, and Boehm, Barry. Lehman's laws and the productivity of increments: Implications for productivity. In *APSEC 2013* (Bangkok, Thailand 2013).

5 Boehm, Barry W and Madachy, Ray. *Comparative Analysis of COCOMO II, SEER-SEM and True-S Software Cost Models*. USC CSSE, 2008.

6 Boehm, B., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., and Horowitz, E. et al. *Software Cost Estimation with COCOMO II*. Prentice Hall, Upper Saddle River, 2000.

7 Lehman, Meir M. Programs, life cycles, and laws of software evolution. In *Proceedings of the IEEE 68.9* (1980).

8 Boehm, Barry, Clark, Brad, Madachy, Ray, and Tan, Thomas. Future Software Sizing Metrics and Estimation Challenges. (Mystic 2011), Systems Engineering Research Center.

9 Boehm, Barry. Software Engineering Economics. (Englewood Cliffs 1981), Prentice-Hall.

10 Boehm, Barry and Turner, Richard. *Balancing agility and discipline: A guide for the perplexed*. Addison-Wesley, Boston, 2003.

11 Moser, Raimund et al. *A case study on the impact of refactoring on quality and productivity in an agile team*.

12 Elbaum, Sebastian G. and Munson, John C. *Code Churn: A Measure for Estimating the Impact of Code Change*.

13 Nagappan, Nachiappan. Use of Relative Code Churn Measures to Predict System Defect Density. In *ICSE '05* (St. Louis, 2005), ACM.

14 Chatman, Vernon V. CHANGE-POINTS: A Proposal for Software Productivity Measurement. *J. SYSTEMS SOFTWARE* (1995), 71-91.

15 SOFTWARE TECHNOLOGY SUPPORT CENTER COST ANALYSIS GROUP. *Software Development Cost Estimating Guidebook*. 2010.

16 Abrahamsson, Pekka et al. Effort Prediction in Iterative Software Development Processes – Incremental Versus Global Prediction Models. In *First International Symposium on Empirical Software Engineering and Measurement* (2007), IEEE, 344-353.

17 Tan, Thomas et al. Productivity trends in incremental and iterative software development. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement* (2009), IEEE Computer Society.

18 Weisberg, S. *Applied Linear Regression*. Wiley, 2005.

19 Larman, Craig. *Agile and iterative development: a manager's guide*. Addison-Wesley, 2004.