LIME: A THREE-DIMENSIONAL MEASUREMENT MODEL FOR LIFE CYCLE PROJECT MANAGEMENT

LUIGI BUGLIONE
European Software Institute
SPI Measurement Product Line
Parque Tecnológico de Zamudio #204
E-48170 Vizcaya, Spain
E-mail: luigi.buglione@esi.es
Tel: (34) 94.420.9519
Fax: (34) 94.420.9420

ALAIN ABRAN
Software Engineering Management Research Laboratory
Université du Québec à Montréal
C.P. 8888, Succ. Centre-Ville
Montréal, Québec, Canada
E-mail: abran.alain@uqam.ca
Tel: +1 (514) 987-3000 (8900)
Fax: +1 (514) 987-8477

ABSTRACT

Organizational performance models are usually based on accounting systems, and therefore take into account mostly the economic-financial viewpoint, or the *tangible asset* part, of it using performance management terminology. In the IT field, the Earned Value model has been promoted to be present project performance during the project life cycle. However, these types of models oversimplify performance representation with a single performance index, while in reality multiple viewpoints must be managed simultaneously for proper performance management.

This work shows how an *open* three-dimensional measurement model of software project performance functions. Called **LIME** (**LI**fecycle **ME**asurement), it extends the structure of a previous model to a dynamic context I applies to software production during all SLC phases, which are classified following a generic 6-step and scheme waterfall standard.

A quantitative and qualitative analysis of the project is effected considering the three distinctive but connected areas of interest, each of them represent has a dimension of performance:

- **economic dimension**, from the *managers'* viewpoint, with a particular attention to cost and schedule drivers;
- **social dimension**, from the *users'* viewpoint, with particular attention to the quality-in-use drivers;
- **technical dimension**, from the *developers'* viewpoint, with particular attention to technical quality, which has a different impact during each SLC phase.

KEYWORDS: Performance Measurement, Software Product Quality, Metrics, Function Point Analysis, ISO/IEC 9126, GQM approach, SLC, QEST model.

1 INTRODUCTION

Over the past few years, the Software Engineering field has developed an increased awareness of the need to measure of both process and product, in other words to improve the management of the software development life cycle. For example, there already exist a significant number of one-dimensional models of performance which integrate individual measurements into a single performance index. The usual performance model of an organization is derived from information within its accounting system, thereby taking into account the economic-financial viewpoint (the *tangible asset* measurement issue in Performance Management frameworks like the Balanced Scorecard, Intangible Asset Monitor and Skandia Navigator). In the technical field, one of the most valuable and proven approaches is *Earned Value* (EV), a method for monitoring project expenditures versus expected progress at each project phase[1].

[1] [40] is the reference document; see also the website http://www.acq.osd.mil/pm/. Furthermore, see [29] for an application of an Earned Value Tracking System in the

However, these types of models are too oversimplified to adequately reflect the reality of the multidimensional nature of performance and of the analytical requirements of management when various "viewpoints" must be taken into account simultaneously.

In addition to multidimensional analysis (here *dimension* is referred to as "viewpoint" or "perspective"[2]), many other issues have been identified by researchers, such as a mix of quantitative and qualitative aspects, as well as integration of product and process analyses [1, 3, 31].

In multidimensional analysis, distinct but related areas of interest can be taken into account simultaneously, each representing a distinct dimension of performance; for example:

- **economic** dimension - the perspective of the managers[3];
- **social** dimension - the perspective of the users;
- **technical** dimension - the perspective of the developers.

An open model called **QEST** (**Q**uality factor + **E**conomic, **S**ocial & **T**echnical dimensions) has been developed to handle, simultaneously and concurrently, the three-dimensional perspectives of performance [7 to 12]. This model had been developed initially to represent multiple views of performance of completed projects. It originally represented a static view of projects, once completed.

This paper presents an extension to this QEST model, which allows the analysis of measurement results throughout the various phases of software development, and considers both process activities and intermediate software product deliverables. Such ongoing analysis during a project life cycle is useful for the continuous monitoring of project progress and for making adjustments to forecasts and schedules of subsequent phases of projects. This extension to the QEST model thereby allows the model to be used dynamically throughout a project's life with the flexibility to represent, for example, distinct views of quality depending on the phase of the life cycle.

This paper is subdivided into five parts:

- a review of topologies of the software life cycle (SLC) models, from a process viewpoint;
- management of quality during a software life cycle
- a description of basic concepts from the QEST model, which constitute the starting reference point considered for extending it to a dynamic SLC context;
- a description of the **LIME** (**LI**fecycle **ME**asurement) model, the newly derived SLC performance measurement model, illustrating its peculiarities and the applicability of the PMAI Cycle over time.
- observations and conclusions

2 SOFTWARE LIFE CYCLE MODELS

A Software Life Cycle (SLC) is defined as "t*he period of time that starts when a software product is conceived and ends when the product is no longer available for use. The software life cycle typically includes a requirements phase, design phase, implementation phase, test phase, installation and check-out phase, operation and maintenance phase, and sometimes, retirement phase*" [17] or as "*a typical sequence of phased activities that represent the various stages of engineering through which software systems pass*" [36]. It can be considered as an instance derived from the Product Life Cycle (PLC), an object of a study in marketing sciences[4] in business schools.

The descriptive and/or prescriptive characterizations of software evolution constitute what is labeled the *Software Life*

software field and [13] for an overview of costs and benefits of an Earned Value Management Process.

2 Opdhal [32] distinguishes between the two concepts, asserting that the first "*emphasizes the physical, logical, mental etc. point from which something is perceived*", while the second "*emphasizes what is perceived from that point*".

3 The viewpoints listed reflect the perspectives of three possible groups of stakeholders, the same ones who were listed in the ISO/IEC JTC1/WG6 work.

4 PLC "*provides insights into a product's competitive dynamics. […]. Product Life Cycle portrays distinct stages in the sales history of a product. Corresponding to these stages are distinct opportunities and problems with respect to marketing strategy and profit potential. By identifying the stage that a product is in, or may be headed toward, companies can formulate better marketing plans.*" [28, pp. 354-380].

Cycle Model[5], which can be used in several ways [36, p.4]:

- to organize, plan, staff, budget, schedule and manage software project work over organizational time, space and computing environments;
- as prescriptive outlines of documents to be produced for clients;
- as a basis for determining what software engineering tools and methodologies will be most appropriate to support different life cycle activities;
- as a framework for analyzing or estimating patterns of resource allocation and consumption during the SLC;
- as comparative descriptive or prescriptive accounts on how software systems come to be the way they are;
- as a basis for conducting empirical studies to determine what the impact will have on software productivity, cost and overall product quality.

SLC models can be classified according to different criteria, such as, for example, number of phases or their topology. Concerning the former, it is possible to find in the literature SLC models with four to eight phases, depending on the level of detail and the definitions given to the term *phase* by different authors.

Concerning the latter, it is possible to identify at least five SLC topologies which can themselves be classified as either traditional or alternative SLC models:

TRADITIONAL TOPOLOGIES OF SLC MODELS

- **Waterfall** [6, 36]: This model is characterized by a set of phases executed in sequential order, where each phase must be completed before the project progresses to the next phase. At the end of each phase, the deliverables are frozen and serve as the baseline for subsequent phases. The software product does not usually become visible to its users until the end of the project, that is, after completion of the last phase. This topology is most adequate when the software product is simple, the requirements stable, the technology used for development is well mastered, and project resources tested. It is necessary for the requirements to be frozen so that only limited change is allowed, which minimizes the risk of cost increases;
- **V-shape model** [16]: Similar to the waterfall model, the V-shape model emphasizes the importance of considering the testing activities up front rather than later in the life cycle. Each test phase is considered as mapping a corresponding development phase. The V-shape model is used in the same context as the waterfall model. This topology, in which the test phases enter in the life cycle earlier has a number of advantages, such as:
  - ✓ Earlier receipt user feedback
  - ✓ Earlier review and evaluation of requirements
  - ✓ Help in developing additional requirements
  - ✓ Upfront monitoring of quality
- **Incremental** [39]: The incremental model allows a project to develop software in incremental stages, adding additional functionality at each stage, each stage including design, code and unit test, integration test and delivery phases. This model allows customers to access functional software much earlier than the two previous models, but of course with less functionality in the early stages.

ALTERNATIVE TOPOLOGIES OF SLC MODELS

- **Prototyping** [2]: Prototyping is a technique used to develop an implementation of the software quickly during the software requirements phase; this allows the customer to use the prototype and to provide feedback to the software developers on the strengths and weaknesses of the prototype. This feedback is the used to improve the prototype. This topology is most adequate when there is limited understanding of user requirements and when requirement volatility is high.

[5] The standardization effort has produced some *de facto* and *de jure* norms on SLC. For the first category it is possible to cite the ESA PSS-05-0 [15], one of the sources for the BOOTSTRAP Improvement model. For the second category, it is possible to mention the ISO effort with the IS 12207 [23] and the omologous series from IEEE [18, 19, 20].

- **Spiral** [4]: This topology represents a risk-driven approach to software process analysis and structuring, incorporating elements from both the waterfall and prototype models. Each spiral addresses major risks that have been identified; the last spiral ends as an ordinary waterfall software life cycle. A spiral model makes it possible to start small, to identify related risks, to adjust the plan to deal with the risk, and then to commit to a specific approach for the next iteration. A spiral model is used for complex projects or when the project issues are poorly identified and understood. The mitigation of risk can become a requirement for the next iteration.

- Boehm et al. have refined the Spiral model to arrive at the **Win-Win Spiral Model** [5], characterized by the presence of different categories of stakeholders and *win* conditions (for the system or subsystems) to be negotiated at each iteration. The possible alternatives must be evaluated with respect to objectives and constraints.

For the purposes of this paper, the simpler but also better known, waterfall model is used; the spiral model similarly uses an SLC with six generic phases:

- Requirements
- Specification
- Design
- Coding
- Testing
- Maintenance

This generic 6-phase SLC model is used in this paper for illustrative purposes only and organizations can easily adapt the LIME model according to their own models needs. This flexibility of the proposed performance measurement model is referred to here as an *open multi-dimensional model of SLC performance*.

3 MANAGEMENT OF QUALITY DURING THE SLC

The propose of Project Management is to ensure that project objectives are met, such as delivering the software functionality on time, on budget and within the specified levels of quality, while optimizing the available resources.

However, in software projects, management must rely on experience due to a lack of measurement techniques and models sophisticated enough to meet simultaneous multidimensional constraints.

An example of quality management during an SLC based on quantitative measurements and quality models is shown in the following figure: typical defect distributions during different phases of the SLC illustrate how later project phases are burdened with a certain amount of *inherited (or injected) defects* . For process improvement, a relationship must be identified during software development between SLC phases in which defects are found and the quality of the process/product.

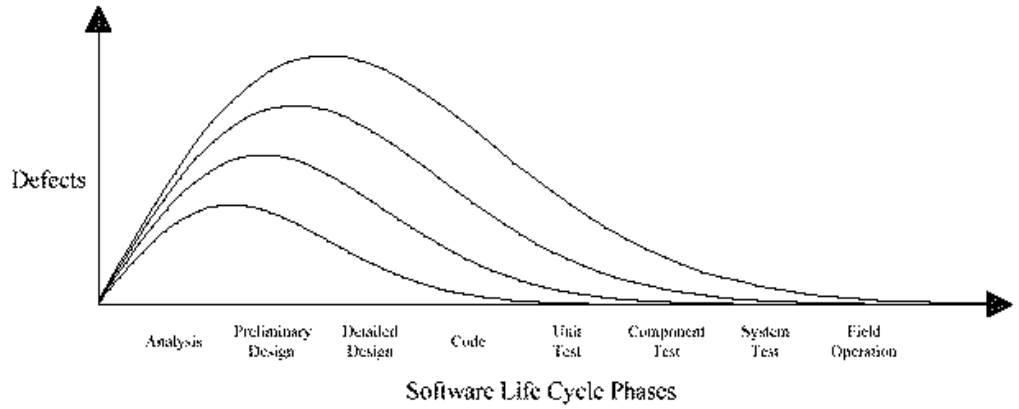

Figure 1 - *Defect Distribution in the SLC with Rayleigh curves*

Table 1 from [27, pp. 159-166] shows a Defect Removal Effectiveness (DRE) statistic. The Rayleigh model used in Figure 1 is based on two fundamental assumptions [27, pp. 182-186]:

- the defect rate observed during the development process is positively correlated with the defect rate in the field[6];
- if more defects are discovered and removed earlier, fewer remain in later steps.

LIFE CYCLE PHASE	INHERITED DEFECTS /KSLOC (A)	INJECTED DEFECTS /KSLOC (B)	SUBTOTAL (C=A+B)	DEFECT REMOVAL EFFECTIVENESS (DRE) (D)	DEFECTS REMOVED PER KSLOC (E=C*D)	RESIDUAL DEFECTS PER KSLOC (F=C-E)
Analysis	N/a	1.2	1.2	N/a	N/a	1.2
Preliminary Design (I0)	1.2	8.6	9.8	74%	7.3	2.5
Detailed Design (I1)	2.5	9.4	11.9	61%	7.3	4.6
Code (I2)	4.6	15.4	20.0	55%	11.0	9.0
Unit Test (UT)	9.0	N/a	9.0	36%	3.2	5.8
Component Test (CT)	5.8	N/a	5.8	67%	3.9	1.9
System Test (ST)	1.9	N/a	1.9	58%	1.1	0.8
Field Operation	0.8	N/a	N/a	N/a	N/a	N/a

Table 1 - *Example of Phase-based Defect Removal Model* [27, pp. 166]

Since the costs of finding and fixing the defects in later phases are much higher that in earlier phases, both assumptions suggest that injected defects must be removed as early as possible, preferably before the testing phases, in order to improve quality and reduce these costs.

However, it must be observed that the quantitative ratios used in this example are based on a measure which does not become available until fairly late in a project SLC, that is, a number of lines of code for which only an approximation with an unknown accuracy level is unavailable until fairly late in the SLC. Similarly, this typical quality model takes into account only a single perspective of quality, that is, the number of defects in the software. However, software quality has many additional dimensions, such as usability, quality of documentation, associated operational costs, etc.

Restricting quality measurement, analysis and management to only a single view of quality throughout an SLC, as illustrated with a typical example of a reliability analysis, seems to be reductive: it does not allow the various outputs from the successive SLC phases that contribute to the final product release to be gathered and analyzed for managing quality on an ongoing basis for continuous monitoring of the multiple views of quality.

An integrated multidimensional view of software Quality is presented in [7 to 12], where as quality can be viewed as the concurrent integration of the three different viewpoints:

- **E**conomical: viewpoint of **management**, who are "*interested in overall quality rather than in a specific quality characteristic [...] and the need to balance quality improvement with management criteria*" [34]
- **S**ocial: viewpoint of the **user**, for whom software quality is achieved by all the properties required to satisfy correctly and

[6] Kan demonstrates the significant correlation (using Spearman's ranking order coefficient) observed for I2, CT, ST and all phases combined (I0, I1, I2, CT and ST) with the field defect rate, while for I0 and I1 the correlations are not significant, since the two are the earliest development phases and the Rayleigh curve peaks just after I1.

efficiently, the present and future real needs of whoever buys it and uses it;

- Technical: viewpoint of the **developer**, for whom software quality is achieved by "*conformity to functional and performance requirements explicitly stated, to development standards explicitly documented and to implied characteristics supposed for every software product developed in a professional way*" [25].

Therefore, an extension is needed with respect not only to quality as the object of measurement, analysis and management, but **performance**, defined as "*the degree to which a system or a component accomplishes its designated functions within given constraints*" [17], moving up from *reliability analysis* to *performance analysis*.

So, in a competitive market period such as the current one, where the capability of a company to react on time to customers' requests and to minimize the cost of goods and services offered is a fundamental and absolute necessity, where measuring performance levels becomes a key component in improving the planning and monitoring the delivery of goods and services, as well as for the design of improvement programs.

A model is proposed in the next section to measure software project performance, which is referred to as the **QEST** (**Q**uality factor + **E**conomic, **S**ocial and **T**echnical dimensions) model.

4 THE QEST MODEL

In the QEST model, the measurement of performance (**p**) is given by the *integration* of an instrument-based measurement process (expressed in the model by the component **RP** - *Rough Productivity*) with a perception-based measurement process based on the subjective perception of quality (expressed in the model by the component **QF** - *Quality Factor*). The QEST model[7] provides a multidimensional *structured shell*, which can then be filled according to management objectives for any specific project: it is therefore referred to as an *open model*. This topology of performance models makes it possible to handle the multiple and distinct viewpoints already discussed, all of which can exist concurrently in any software project. This section presents the design of this *open model* for the measurement of software project performance.

The basic purpose of the structured shell of the open model is, as stated above, to express performance as the combination of the specific measures (or sets of measures) selected for each of the three dimensions, these values being derived from both an instrument-based measurement of productivity and a perception-based measurement of quality.

A three-dimensional geometrical representation of a *regular tetrahedron* was selected as the basis for the model, and is illustrated in Figure 2. Furthermore:

- the three dimensions (E, S, T) in the space correspond to the corners of the pyramid's base, and the convergence of the edges to the P vertex, which describes the top performance level;
- when the three sides are of equal length, the solid shape that represents this three-dimensional concept is therefore a pyramid with its triangular base and sides of equal length (*tetrahedron*).

This pyramid-type representation imposes the following constraint: the sides must be equal, and this is achieved through giving equal weights to each of the three different dimensions chosen – and with sides of length exactly equal to 1 (*regular tetrahedron*); in this way, the dimensions are represented through a normalized value between 0 and 1 for each of them on a ratio scale, for ease of understanding.

[7] Several publications cover the different aspects of the QEST Model:
- Theoretical aspects [10, 11, 12]
- Geometrical and statistical foundations [9]
- Implementation of the model [8]
- Quality Factor [7]

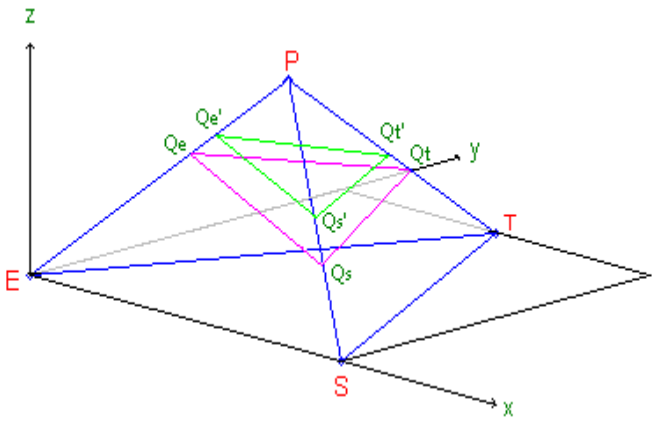

Figure 2 - *QEST model and its hyperplane sections*

With this 3D representation it is possible to determine and represent performance considering distinct geometrical concepts (distance, area and volume); in this 3D representation, the ratio between the volume of the lower part of the truncated tetrahedron and the total volume of the tetrahedron represents the normalized performance level of a project being assessed[8].

Key features of the QEST model:

- **Integrated quantitative and qualitative evaluation from three concurrent organizational viewpoints:** management (economic viewpoint), users (social viewpoint) and technical personnel (technical viewpoint). Performance is not a single one-dimensional concept. It is not enough to meet a single specific target in an unconstrained environment. It is a multidimensional concept that must integrate multiple viewpoints, most of which are present simultaneously in the software development process. The other fundamental point is the integration of qualitative and quantitative evaluations, intended as a 2-sided face of the same problem. Only if these dimensions are managed concurrently in an integrated model for performance measurement can they be adequately assessed;
- **Use of de facto and de jure standards,** such as the ISO/IEC 9126 standard on Software Quality Characteristics and Sub-characteristics and Function Point related measures [21] for the functional size of software. The use of standards is strongly recommended because they give the international software engineering community the opportunity to share and use the same definitions of objects and concepts in their work, reducing the potential for misinterpretation when communicating. For this reason, two important de jure (ISO/IEC 9126) and de facto[9] standards are recommended when implementing the model;
- **A 3D geometrical construction to gather a single SLC phase value for each project** – The geometrical approach permits representation of the measurement of performance in a simple and visual way for immediate impact and optimal understanding. The original selection of the regular tetrahedron was also suggested by the idea that the vertex of the 3D shape represents, from a conceptual viewpoint, the convergence of different viewpoint evaluations into a final, single one. Another important factor to take into account is the use of normalized values in order to give Management greater value readability for taking decisions;

5 THE LIME MODEL

We are proposing to use QEST model concepts, and extend them to a dynamic context, such that the model will be applicable to each step of any topology of SLC selected. For illustrative purposes here only, this new model, called **LIME** (**LI**fecycle **ME**asurement), considers a generic 6-phase waterfall SLC structure.

The intrinsic SLC dynamicity and sequentiality necessarily implies the adoption of a notation to describe the process and its flows. From the various possible notations found in the

[8] Refer to [9] for the geometrical and statistical foundations and demonstrations.

[9] While ISO 14143 [24] and the IEEE standardization work on the Functional System Method (FSM) are going on, the COSMIC initiative (http://www.cosmicom.com) is developing and promoting the next generation of functional size measurement methods.

technical literature[10], the **ETVX** (Entry-Task-Validation-eXit) notation [35] in this paper[11].

Figure 3 expresses the relationship between activities and results, referred to was selected for used OO the n^{th} phase of a project. along a certain time-flow *t*.

In this notation system, the output of the $(n-1)^{th}$ phase represents the input for the n^{th} one; processing produces the n^{th} output, which will be the input for the $(n+1)^{th}$ phase, and so on.

It must be noted that the measurement results (I_1, ..., I_6, O_1, ..., O_6) can be added since they have been normalized within the QEST model to facilitate an understanding of them and a representation of them in a 3D space:

[10] See http://source.asset.com/stars/loral/process/guide/notation.htm for a review of the main ones.

[11] See [30] for an ETVX application.

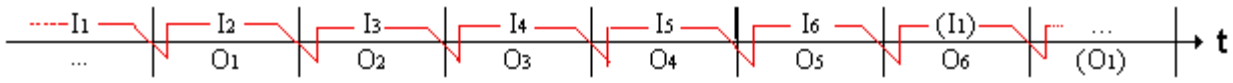

Figure 3 - *Input and output overlapping per phase*

The key features added in the LIME model are:

1. **Flexibility of distinct relative contributions from the three dimensions (E, S, T) in each phase**: Each project phase is characterized by a more or less marked presence of every group of interest; therefore selected ratios values must be weighted according to this criterion. For instance, during the Coding phase, user presence is not so relevant; similarly for managers in the Testing or Requirement phases; by contrast, technical staff presence is predominant during the Coding and Testing phases. So, in each phase the relative contribution of the three groups must be determined quantitatively in a proper way;

2. **Flexibility of distinct relative contributions between quantitative and qualitative evaluations in each phase**: Similarly, the characteristics of the relative distribution of quantitative and qualitative evaluations constitute one of the basic the QEST model. The extension of the QEST model to the whole SLC requires variable tuples of relative distribution values for each step;

3. **Different sources for QF calculation**: A key difference in the structure of the LIME model is the treatment of the Quality Factor (QF). As defined in [7], it arises from a sample analysis the measurement object of which is given by the released software product. The referenced procedure can be put in to action just in the SLC technical phases (Testing and Maintenance). In the other phases, it will be sufficient to consider a list of ratios that measure qualitative aspects of the project (i.e. in the Requirement phase, the number of *verbs* or *shalls*) always in the balance chosen by the organization between quantity and quality evaluations;

4. **Flexibility in selecting suitable measures and ratios for each SLC phase**: The treatment of each single SLC phase, unlike the original QEST model, makes it possible to select for each company dimension suitable measures and ratios for each step. A partial listing of measurable inputs and outputs is provided in [26, pp. 148-153]; the *open* structure of the QEST and LIME models gives the flexibility to use the preferred and most suitable measures as recommended for each different project by the organizational measurement group in co-operation with the various teams for each dimension.

The framework for the LIME model is the following:

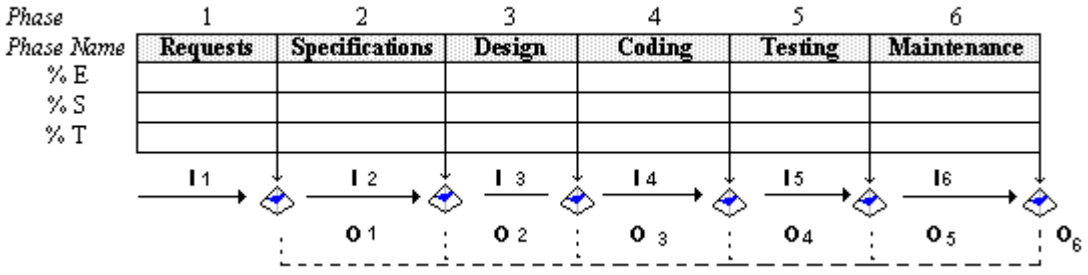

Figure 4 - *The LIME model*

The iterative definition, collection and analysis of multidimensional measures at each life cycle phase offers, therefore, the feedback required to make adjustments to the project processes in a timely fashion, both for the next

phase and for designing future improvements to the process of the preceding phase.

This is illustrated in Figure 5 with the 12 process improvement steps (Plan – Measure – Analyze – Improve) for each of the 6 phases of the waterfall model considered, based, in every phase, on a complete QEST assessment:

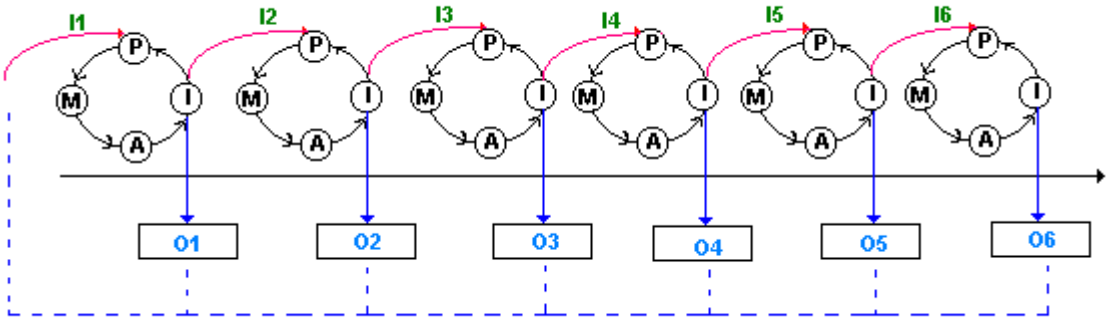

Figure 5 - *LIME model and PMAI Cycle*

6 CONCLUSIONS & PROSPECTS

Improving software project management is a must for project managers. The starting point of the analysis is necessarily the Software Life Cycle approach chosen, so that its strengths and weaknesses can be identified and a better solution designed.

After the main traditional and alternative SLC models had been reviewed, a quality management approach for the whole SLC was presented, extending the Earned Value model and the generic technical vision of quality (*defectiveness*) to multiple simultaneous distinct perspectives, moving from one-dimensional *analysis (earned value or reliability analysis)* to multidimensional *performance analysis*.

A dynamic multidimensional model, called **LIME** (**LI**fecycle **ME**asurement) is proposed, based on basic concepts of a previous static model (**QEST** - **Q**uality factor + **E**conomical, **S**ocial & **T**echnical dimensions).

This model exist addresses the need to obtain a richer multidimensional, combined-view software performance measurement tool, and includes the various aspects - technical, economic and social - that concurrently in every organization. These simultaneous views are usually analyzed in isolation in one-dimensional models. The framework of the LIME model is presented and the key concept is illustrated through a generic 6-phase waterfall SLC model, where each output of the $(n-1)^{th}$ phase represents the input for the n^{th} one, and, so on thereby improving the overall quality of deliverables for subsequent processes.

This approach will make measurement results more comprehensive and useful by providing a multidimensional representation of performance, with the possibility of analyzing each single dimension and any SLC phase of a project.

ACKNOWLEDGMENTS

We wish to thank the Bell Canada and the Natural Sciences and Engineering Research Council of Canada for research funding. The opinions expressed in this paper are solely those of the authors.

REFERENCES

[1] ABRAN A., *Quality – The Intersection of Product and Process*, The 6th IEEE International Software Engineering Standards Symposium, ISESS'95, Montréal, Québec, Canada, August 21-25, 1995, IEEE Computer Society Press.

[2] BALZER R., GOLDMAN N. & WILE D., *Operational Specifications as the Basis for Rapid Prototyping*, ACM Software

Engineering Notes, Vol. 7, no 5, pp. 3-16, 1982.

[3] BIRK A., DERKS P., HAMANN D., HIRVENSALO J., OIVO M., RODENBACH E., VAN SOLINGEN R. & TARAMAA J., *Applications of Measurement in Product-Focused Process Improvement: A Comparative Industrial Case Study*, ISERN, Techical Report ISERN-98-25, August 1998.

[4] BOEHM B., *A Spiral Model of Software Development and Enhancement*, ACM Software Engineering Notes, Vol. 11, no 4, 1986, pp. 22-42.

[5] BOEHM B., EGYED A., KWAN J., PORT D., SHAH A & MADACHY R., *Using the Win-Win Spiral Model: A Case Study*, IEEE Computer, July 1998, pp. 33-44.

[6] BOEHM B., *Software Engineering*, IEEE Transactions on Computers, C-25, Vol. 12, December 1976, pp. 1226-1241.

[7] BUGLIONE L & ABRAN A., *A Quality Factor for Software*, Proceedings of QUALITA99, 3rd International Conference on Quality and Reliability, Paris, France, 25-26 March 1999, pp- 335-344.

[8] BUGLIONE L. & ABRAN A, *Implementation of a Three-Dimensional Software Performance Measurement Model*, Technical Report, Université du Québec à Montréal, to be published, 1999.

[9] BUGLIONE L. & ABRAN A., *Geometrical and Statistical Foundations of a Three-dimensional Model of Performance*, accepted for publication in the "International Journal of Advances in Engineering Software", Elsevier Publisher, 1999.

[10] BUGLIONE L. & ABRAN A., *Multidimensional Software Performance Measurement Models: A Tetrahedron-based Design*, in "Software Measurement: Current Trends in Research and Practice", R. Dumke/A. Abran (eds.), Deutscher Universitats Verlag GmbH, pp. 93-107, 1999

[11] BUGLIONE L. & ABRAN A., *Multidimensional Software Performance Measurement Models: A Tetrahedron-based Design*, Rapport de Recherche No, 99/01, Départment d'informatique, UQAM, Université du Québec à Montréal, 14 Mai 1999.

[12] BUGLIONE L., *Software Process Assessment & Improvement: un nuovo starting-point per il Management aziendale*, Tesi di Dottorato, XI Ciclo, Dottorato di Ricerca in "Sistemi Informativi Aziendali", Università di Roma "La Sapienza", Roma, Italia, 21 Dicembre 1998.

[13] CHRISTENSEN D., *The Costs and Benefits of the Earned Value Management Process*, Acquisition Review Quarterly, Fall 1998, pp. 373-386.

[14] DEMING, W.E., *Out of the Crisis*, MIT Press, 1986.

[15] EUROPEAN SPACE AGENCY, *ESA Software Engineering Standards ESA PSS-05-0*, Issue 2, Paris, February 1991.

[16] GOLDBERG R., *Software Engineering: An Emerging Discipline*, IBM Systems Journal, Vol. 25, no 3-4, 1987.

[17] IEEE, *Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12, 1990.

[18] IEEE/EIA, *Guide for ISO/IEC 12207 – Standard for Information Technology – Software Cycle Processes – Life Cycle Data,* Std 12207.1, 1997.

[19] IEEE/EIA, *Guide for ISO/IEC 12207 – Standard for Information Technology – Software Cycle Processes – Implementation Considerations,* Std 12207.2, 1997.

[20] IEEE/EIA, *Standard for Industry Implementation of International Standard ISO/IEC 12207:1995 – Standard for Information Technology – Software Cycle Processes,* Std 12207.0, 1996.

[21] IFPUG, *Function Points as Assets: Reporting to Management*, 1992.

[22] ISO, *International Standard 8402: Quality - Vocabulary,* 1986.

[23] ISO/IEC JTC1/SC7/WG7, *International Standard 12207 – Information*

Technology: Software Life Cycle Processes, 22/02/95.

[24] ISO/IEC, *International Standard 14143-1 – Information Technology – Software Measurement – Functional Size Measurement – Part 1: definition of concepts*, December 1996.

[25] ISO/IEC, *International Standard 9126: Information Technology – Software product evaluation – Quality characteristics and guidelines for their use,* 1991.

[26] JOINT LOGISTIC COMMANDER*, Practical Software Measurement – A Foundation for Objective Project Management*, Version 3.1, Joint Group on Systems Engineering*,* April 17, 1998.

[27] KAN S.H., *Metrics and Models in Software Quality Engineering*, Addison-Wesley, 1995.

[28] KOTLER P., *Marketing Management: Analysis, Planning, Implementation and Control,* Prentice Hall, 8 /e, 1994.

[29] LETT S.H., *Earned Value Tracking System for Self-Directed Software Teams*, SEPG 1998 Conference, March 9-12 1998, Chicago, USA.

[30] MC ANDREWS D.R., *Establishing a Software Measuring Process,* SEI Technical Report, CMU/SEI-93-TR-16, July 1993.

[31] MCGARRY F., *Product-Driven Process Improvement*, Software Process Newsletter, IEEE Technical Council on Software Engineering, no 2, Spring 1995, pp. 1-3.

[32] OPDHAL A.L.*, A Comparison of Four Families of Multi-Perspective Problem Analysis Method*, IRIS 20 Conference, Hanko, Norway, August 9-12, 1997.

[33] PARK R.E., *Software Size Measurement: A Framework for Counting Source Statements*, SEI Technical Report, CMU/SEI-92-TR-20, September 1992.

[34] PRESSMAN R., *Software Engineering: a beginner's guide*, McGraw-Hill, 1988.

[35] RADICE, R. A. & PHILLIPS, R.W., *Software Engineering: An Industrial Approach*, Prentice-Hall, 1988.

[36] ROYCE W.W., *Managing the Development of Large Software Systems*, Proceedings of the 9th International Conference on Software Engineering (ICSE), IEEE Computer Society, 1987.

[37] SCACCHI W., *Models of Software Evolution: Life Cycle and Process*, SEI Curriculum Module, SEI-CM-10-1.0, October 1987.

[38] SHEWHART W.A., *Statistical Method from the Viewpoint of Quality Control, 2/e,* Dover Publication, 1986.

[39] TULLY C., *Software Development Models*, Proceedings of IEEE Software Process Workshop, IEEE Computer Society, pp. 37-44, 1984.

[40] US AIR FORCE, *Cost/Schedule Status Report (C/SSR) Joint Guide, May 1, 1996.*