Available online at www.sciencedirect.com
ScienceDirect
The Journal of Systems and Software 81 (2008) 1248–1252
The Journal of Systems and Software
www.elsevier.com/locate/jss

Controversy Corner

Why and how can human-related measures support software development processes? ☆

Orit Hazzan [a,*], Irit Hadar [b]

[a] Department of Education in Technology and Science, Technion – Israel Institute of Technology, Haifa, Israel
[b] Department of Management Information Systems, University of Haifa, Haifa, Israel

Received 24 November 2007; received in revised form 15 January 2008; accepted 19 January 2008
Available online 9 February 2008

Abstract

In this paper we discuss why and how measures related to human aspects should be incorporated into software development processes. This perspective is based on the vast evidence that human aspects are the source of the majority of problems associated with software development projects. Having said that, we do not blame the humans involved in software development processes; rather, we suggest that human-related measures might be one means by which human aspects of software development processes can be supported.

Keywords: Human aspects; Software engineering; Software measures; Process measurement

1. Introduction

When building a measurement[1] model intended to guide software development projects, three main aspects are usually addressed, all of which aim to provide high-quality software. The first is the technical aspect, whose measures deal with activities such as design, implementation, and testing; the second aspect is the managerial aspect whose measures address, for example, time management and schedule; the third aspect is the human aspect which relates, for instance, to communication among teammates, customer collaboration and learning processes.

Although many of the complexities of software development processes are closely related to human aspects, both social and cognitive (Tomayko and Hazzan, 2004), most measures encountered in the literature emphasize the technological and managerial aspects of software projects. While we have the utmost respect for technological- and managerial-related measures and their contribution to software projects processes, we believe that a full view of both system and process can be achieved only by taking all aspects into account, including the influential human aspects of software engineering. This view, we suggest, can help us cope with the intangibility of software and the ensuing challenges.

To deliver our message, we first present a general survey of software measures as documented in the existing literature. We show that most of the measures are technical and illustrate the under-emphasized issue of human-related measures. Second, we discuss the importance of measures

☆ *Controversy corner*. It is the intention of the Journal of Systems and Software to publish, from time to time, articles cut from a different cloth. This is one such article.

The goal of CONTROVERSY CORNER is both to present information and to stimulate thought and discussion. Topics chosen for this coverage are not just traditional formal discussions of research work; they also contain ideas at the fringes of the field's "conventional wisdom".

These articles will succeed only to the extent that they stimulate not just thought, but action. If you have a strong reaction to the article that follows, either positive or negative, send it along to your editor, at card@software.org.

We will publish the best of the responses as CONTROVERSY REVISITED.

* Corresponding author.
E-mail addresses: oritha@techunix.technion.ac.il (O. Hazzan), hadari@mis.haifa.ac.il (I. Hadar).

[1] A note on terminology: Consistently with ISO/IEC Standard 15939, we use in this paper the terms 'measurement' and 'measure' and do not use the term 'metric'.

0164-1212/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2008.01.037

that are related to human aspects of software engineering and reflect on what can be achieved by examining these measures. We conclude with some suggestions for future research aimed at determining and evaluating human-related measures for software development projects.

2. Existing software development measures

Extensive literature exists with respect to the significance of measurement in software development processes (e.g., DeMarco, 1982; Mayrhauser, 1990; Hughes and Cotterell, 2002). Different kinds of measures are suggested for software processes and products. Specifically, measures related to the development process aim at reducing the risk of losing control while gaining confidence with the process progress; measures related to the software product aim at ensuring a high-quality product with respect to customer requirements and preventing defects. In what follows, we outline some of the existing measurement models and frameworks.

One approach used for the measurement of the technical aspects of a software system examines its quality-related attributes. For example, the International Standards Organization (ISO) (Tassey, 2002) details the following main software quality attributes: Functionality, Reliability, Usability, Efficiency, Maintainability and Portability. Each of these attributes is composed of 2–5 more specific sub-characteristics. The calculation of these characteristics is not concretely defines, hence, they constitute more abstract guidelines for evaluating the software as a product, regardless of the technology, paradigm or methodology within which it is developed. On the other hand, concrete measures exist that address very specific software attributes, such as defects found during the software life cycle. Much criticism, however, has been expressed as to the ability of such measures to predict defects (Fenton and Neil, 1999).

Other standards exist that do give more specific guidelines for accurate quantitative software assessment. They usually focus on a very specific aspect of the software system and are usually paradigm dependent. Quality measures for object-oriented software development are quite common. For instance, predictive measures for quality, such as average class number of lines of code (ACLOC), depth of inheritance tree (DIT) and many more are computed to design quality measurements in terms of software system maintainability (Misra, 2005); measurement model for assessing complexity in an object-oriented design (McGregor and Kamath, 1995); measures that deal with inheritance depth or the number of methods overloaded and redefined under the same name (Lorenz and Kidd, 1994); measures concerning the longest path from a specific class to the root of the inheritance tree (Henderson-Sellers et al., 1993; Henderson-Sellers, 1996; Chidamber and Kemerer, 1994); and measures concerning cohesion and coupling between classes (Bieman and Byung-Kyoo, 1998; Lindroos, 2004). Lately, some object-oriented measures were even studied in the context of agile processes (Alshayeb and Li, 2005). While these measures are very explicit and lead to accurate assessments, they refer only to object-oriented development and are designated to specifically assess the aspect of maintainability.

Other measures exist that address not only the product, but also the development processes. For example, in the IEEE's guide to the software engineering body of knowledge (SWEBOK, 2004), product and process measures are defined and an explicit explanation on how to measure them is supplied. According to the SWEBOK, processes must be adapted to local needs, such as organizational context, project size, regulatory requirements, industry practices, and corporate cultures.

One such framework used today is the capability maturity model (CMM), which integrates different methodologies, standards and assessment models. Its more comprehensive framework – capability maturity model® Integration (CMMI) – is targeted mainly at supporting the development processes by showing a measurable benefit for the organization's business objectives and vision. It integrates multiple military, ISO, IEEE and other commercial standards and procedures that cover all aspects of system building (Kasse, 2004). In practice, the CCMI provides guidelines on how to organize and prioritize engineering, people, and business activities; further, it aims to support coordination of multi-discipline activities required, or potentially required for a successful project building (Kasse, 2004).

Other standards exist that include measures for assessing capability levels and software development process attributes. For example, the ISO/IEC 15504 assessment model defines five capability levels and the attributes of their processes. The model enables to rate the processes' attributes using the following grades: "fully achieved" (86–100% achievement), "largely achieved" (51–85%), "partially achieved" (16–50%) and "not achieved" (0–15%). Based on the grades given to the different attributes, it is possible to assess the capability level of the development process (Galin, 2004).

Another common framework is the ISO/IEC 9126, an international standard for software quality that has been accepted by most of the international community, and which some countries, such as Japan, have adopted as a national standard. The standard defines a common language relating to software product quality and is widely recognized as such (Cote et al., 2005). Moreover, it was found, over a decade ago, that at least 70% of the European IT community were familiar with ISO/IEC 9126 (Bazzana et al., 1993). However, some important problems are associated with this standard (Pfleeger, 2001):

- No guidelines are given on how to provide an overall assessment of quality.
- No indications are provided on how to actually measure the quality characteristics.
- Rather than focusing also on the user's perspective of the software, the model's characteristics reflect only the developer's viewpoint.

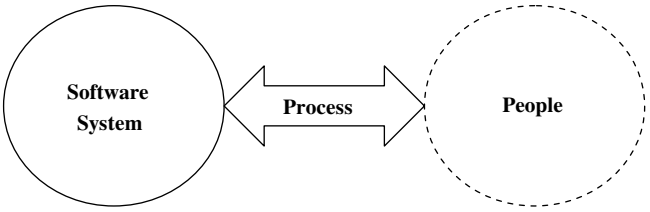

Fig. 1. Three development components – the software system developed, the people who develop it, and the relationship between the two.

The above brief survey of software measures indicates that most of the measures are technical and highlights the under-emphasized issue of human-related measures. One approach that does emphasize human factors in relation to software development is COCOMO II (Boehm et al., 2000) in which human aspects are considered alongside technical and managerial aspects (called product, project and platform) in the estimation of the development time of a software product. These measures (called personnel) relate to inherent properties of the people involved in the project, such as professional capabilities and experience. We suggest that human-related measures should also be monitored during the development process, in addition to prediction-oriented measures, as COCOMO II suggests. In other words, we suggest that in addition to measures collected prior to the project launch (referred to as static measures), dynamic human-related measures should also be considered and controlled during the development process. Examples for such measures are presented in the next section.

Examination of the above-mentioned measurement models reveals that they can be mapped to components of (a) software systems, (b) the people who develop the software systems, and (c) the development processes, which in fact, connects the software systems and the people who develop them. Fig. 1 illustrates the relationship between the three components. For example, the object-oriented code and design-related measures can be mapped to component (a), the COCOMO II measures of software engineer capability and application experience can be mapped to component (b), and the process CMMI measures can be mapped to component (c). As mentioned earlier, measures related to software systems and development processes are largely addressed by the existing software engineering measurement literature. Therefore, the people component in Fig. 1 is sketched with a broken line. We suggest that an addition be made to the exiting body of knowledge whereby measures emphasizing component (b) – people – are formalized. Such measures can relate to cognitive aspects, such as self-learning, and social issues such as collaboration between team members.

3. The importance of the missing human-related measures

So far, we have seen that measures tend to focus on technical issues related to software engineering. We claim that this is not sufficient and is, in fact, the result of the emphasis placed by traditional software development methods on technical aspects. Indeed, if the technical perspective is emphasized, then it makes sense to monitor code-related measures. However, if the people involved in software development environments are put at the center of the development process, as the agile approach, for example, does, then additional and different kinds of measures should be used. In other words, since traditional methods refer mostly to the software life cycle and less to the people involved, the human aspect of software development is addressed in these processes less frequently, as are the measures they inspire. Since awareness to human aspects in software engineering is increasing (see, for example, the agile manifesto (http://agilemanifesto.org/), which specifically stresses the importance of people and interactions), we suggest that this awareness should also be reflected in the measurement model.

At this stage we can speculate why many of the problems that characterize software projects relate to the people involved (Tomayko and Hazzan, 2004). After all, if we do not measure these aspects, how can we approach, fix and improve them? As Kent Beck and Cynthia Andres say: "Good teams don't just do their work, they think about how they are working and why they are working" (Beck and Andres, 2004).

To bridge this gap, we suggest organizing human-related measures as described in Table 1, in which we address cognitive and social aspects, both on the individual and on the team level. We emphasize that this classification is not the only possible one, and other human-centric classifications may encompass the richness of human aspects of software engineering. We propose, however, that this presentation is a good starting point for the exploration of this topic since it emphasizes the main facets of software engineering on the team level.

In what follows, we explore the kind of information that might be supplied by measures in each cell. The challenge is to associate concepts, such as learning and collaboration, with measures, and to explain how such measures can improve software development processes. It should be noted that, from the measurement perspective, the social-team cell is the one most frequently addressed to date.

3.1. The cognitive-individual cell

COCOMO II addresses this category by measuring personal attributes, such as analyst capability and software engineer capability. Measures in this cell might address topics such as personal learning, whereby, for instance,

Table 1
Proposed framework for measuring human-related concerns in software development projects

Level/Aspect	Cognitive	Social
Individual		
Team		

individuals in the team might frequently measure what they have learned during a specific period of time. This can be done at the end of iterations or at the end of any time period as seen fit by the individual. To give such measures numerical values, the practitioner might measure how many times the new knowledge is used, under what circumstances, and how the new knowledge influences the development process. Another example is measuring the improvement in time estimations during the development period and drawing of conclusions, which might address, for example, the reasons for the improvement in time estimations (if such improvement indeed occurred).

3.2. The social-individual cell

This cell addresses measures related to the collaboration and communication between the individual and the team. For example, one might measure the kind of collaboration with other team members, as well as the influence such collaboration has on the practitioner's performance. This implies that developers should be aware of the kind of interactions they have with other team members and how these interactions contribute to, or interfere with their software development tasks. Measures such as these are important since face-to-face communication is so vital in software development process; If, however, such face-to-face interactions impede the individual's progress, they should be either avoided or monitored. On the other hand, if practitioners realize that their interactions with other team members contribute to the performance of their tasks, they should strive to understand what kind of interactions these are and what can be done to improve their quality.

3.3. The cognitive-team cell

This cell relates mainly to knowledge-management in software teams. In practice, team members can establish a knowledge-management system and measure its evolvement, its use, and its contribution to the software development processes. For example, a team might measure how its collective knowledge is enhanced and how this knowledge fosters software development processes. Another example is when the team examines both the ways in which it uses a practice that it decided on and the consequences of such application. One resource for measures that belong to this cell is the knowledge gained in the framework of the knowledge management discipline (see for example, knowledge-management measures, http://www.knowledge-management-online.com/KM-Measures.html and measures in a customer support context: the impact of knowledge management, http://www.systems-thinking.org/mcsc/mcsc.htm#solm).

3.4. The social-team cell

In this case, social issues are measured on the team level. The main emphasize here is on the level of communication within the team and its contribution to the total progress of the team's work. We illustrate this with one of the examples presented in (Dubinsky and Hazzan, 2006), which examined the roles in software teams. Specifically, a role scheme was defined, according to which each team member has another role (such as tracker, in charge of documentation, etc), in addition to his or her role as a developer. As it turns out, the role scheme provides systematic quantitative measures during the development process that reflect the current status at each development phase with respect to team communication. Three measures were defined:

(1) Role time measure (RTM), which measures the ratio between time spent on development tasks and time spent performing role activities.
(2) Role communication measure (RCM), which measures the level of communication within the team at each development stage. This measure evolves over time since each role holder must communicate with other team members in order to perform his or her individual role efficiently.
(3) Role management measure (RMM), which measures the level of project management. Since the role scheme aims to cover all management aspects, a maximal level is obtained when all role holders provide maximum role performance.

4. Conclusion and suggestions for the future

In general, we propose that the software engineering community should aim to build a measurement model for tracking of software processes, which will relate to technical, managerial and human aspects of such processes. From this perspective, questions, such as the following, are derived: What components should a measurement model of this kind contain? How should these components be measured and integrated into the model? How does the measurement model actually guide software development processes from the technical, managerial and human aspects? What processes are appropriate for the assimilation and integration of such a measurement model into the software industry? What are the implications of such a measurement model on the project process and product?

It is not our intention to propose answers to these questions at this stage. We claim, however, that software intangibility can largely explain why many of the main problems associated with software engineering projects are human-related. We further claim that this fact emphasizes the importance attributed to human-related measures, since the different activities related to software development processes are not always transparent. Thus, measures are one means for making the development processes of intangible products, such as software systems, more transparent. This stands in contrast to the development of tangible products whose development is transparent and can be sensed by our

regular senses (seeing, hearing, etc.), and therefore the need for measures related to human aspects is lower.

In other words, since different attributes of product development processes are expressed in software development projects differently than they are in other professions in which people and months are interchangeable (Brooks, 1975, 1995), and since the process control in software processes can not rely on our regular senses, other means should be employed to increase control over the development process. In practice, this implies that we should make an effort to increase software sensibility and make the process more transparent. We suggest that one means to achieve this goal is by using human-related measures that can help us make the intangible software properties more sensible.

References

Alshayeb, M., Li, W., 2005. An empirical study of system design instability metric and design evolution in an agile software process. Journal of Systems and Software 74, 269–274.

Bazzana, G., Andersen, O., Jokela, T., 1993. ISO 9126 and ISO 9000: friends of foes? In: Presented at Software Engineering Standards Symposium.

Beck, K., Andres, C., 2004. Extreme Programming Explained: Embrace Change, second ed. Addison-Wesley.

Bieman, J.M., Byung-Kyoo, K., 1998. Measuring design-level cohesion. IEEE Transactions on Software Engineering 24 (2), 111–124.

Boehm, B.W., Abts, C., Brown, A.W., 2000. Resources Software Cost Estimation with COCOMO II. Prentice-Hall, New Jersey.

Brooks F.P., 1975, 1995. The Mythical Man-Month: Essays on Software Engineering, second ed. 20th Anniversary Ed., Addison-Wesley Professional.

Chidamber, S.R., Kemerer, C.F., 1994. A metrics suite for object oriented design. IEEE Transactions on Software Engineering 20 (6), 476–493.

Cote, M.A., Suryn, W., Laporte, C.Y., Martin, R.A., 2005. The evolution path for industrial software quality evaluation methods applying ISO/IEC 9126:2001 quality model: example of MIRTE's SWAE method. Software Quality Journal 13, 17–30.

DeMarco, T., 1982. Controlling Software Projects: Management Measurement and Evaluation. Yourdon Press.

Dubinsky, Y., Hazzan, O., 2006. Using a role scheme to derive software project metrics. Journal of Systems Architecture 52, 693–699.

Fenton, N.E., Neil, M., 1999. Quantitative analysis of faults and failures in a complex software system. IEEE Transactions on Software Engineering 25 (5), 675–689.

Galin, D., 2004. Software Quality Assurance: From Theory to Implementation. Pearson Addison Wesley.

Henderson-Sellers, B., 1996. Object-Oriented Metrics Measures of Complexity. Prentice Hall PTR, Upper Saddle River, NJ.

Henderson-Sellers, B., Moster, S., Seehusen, S., Weinelt, B., 1993. A proposed multi-dimensional framework for object-oriented metrics. In: First Australian Software Metrics Conference, Sydney.

Hughes, B., Cotterell, M., 2002. Software Project Management, third ed. McGraw-Hill.

ISO/IEC 15939, 2007. Systems and Software Engineering – Measurement Process.

Kasse, T., 2004. Practical Insight into CMMI®: The Look and Feel of a Successful Implementation. Artech House, Boston, London.

Lindroos, J., 2004. Code and design metrics for object-oriented systems. In: Seminar for Quality Models for Software Engineering, Helsinki.

Lorenz, M., Kidd, J., 1994. Object-Oriented Software Metrics. Prentice-Hall PTR, Englewood Cliffs, NJ.

Mayrhauser, V.A., 1990. Software Engineering Methods and Management. Academic Press.

McGregor, J.D., Kamath, S., 1995. A Psychological Complexity Measure at the Domain Analysis Phase for an Object-oriented System. Department of Computer Science Technical Report, Clemson University.

Misra, S.C., 2005. Modeling design/coding factors that drive maintainability of software systems. Software Quality Journal, 297–320.

Pfleeger, S.L., 2001. Software Engineering: Theory and Practice, second ed. Prentice Hall, Upper Saddle River, NJ.

SWEBOK, 2004. Guide to the Software Engineering Body of Knowledge. A project of the IEEE Computer Society Professional Practices Committee, IEEE Computer Society.

Tassey, G., 2002. The Economic Impacts of Inadequate Infrastructure for Software Testing. National Institute of Standards and Technology, <http://www.swebok.org/>.

Tomayko, J., Hazzan, O., 2004. Human Aspects of Software Engineering. Charles River Media.

Orit Hazzan is an associate professor at the Department of Education in Technology and Science of the Technion – Israel Institute of Technology. Her main research topic – human aspects of software engineering – deals with cognitive and social issues of software engineering in general and of agile software development in particular. She co-authored *Human Aspects of Software Engineering* (with Jim Tomayko), published in 2004 by Charles River Media and of *Agile Software Engineering* (co-authored with Yael Dubinsky) to be published by Springer in 2008. Hazzan presents her work at both software engineering conferences and computer science and software engineering education conferences.

Irit Hadar is a lecturer at the Department of Management Information Systems at the University of Haifa, Israel. Her main research areas include cognitive processes that take place during software development, focusing on development according to the object-oriented paradigm; the effect of different methodologies and tools, such as visual models and ontologies, on requirement engineering and software design; and human factors – organizational, management, social and cognitive – and their influence on software quality. Hadar is also interested in the cognitive perspective of the teaching and learning of different software development-related issues.