

Evaluating Software Degradation through Entropy

Alessandro Bianchi, Danilo Caivano, Filippo Lanubile, Giuseppe Visaggio
Dipartimento di Informatica – Università di Bari
Via Orabona, 4, 70126 Bari – Italy
{bianchi, caivano, lanubile, visaggio}@di.uniba.it

Abstract

Software systems are affected by degradation as an effect of continuous change. Since late interventions are too much onerous, software degradation should be detected early in the software lifetime. Software degradation is currently detected by using many different complexity metrics, but their use to monitor maintenance activities is costly. These metrics are difficult to interpret, because each emphasizes a particular aspect of degradation and the aspects shown by different metrics are not orthogonal. The purpose of our research is to measure the entropy of a software system to assess its degradation. In this paper, we partially validate the entropy class of metrics by a case study, replicated on successive releases of a set of software systems. The validity is shown through direct measures of software quality such as the number of detected defects, the maintenance effort and the number of slipped defects.

1. Introduction

Software systems, during their lifetime, are affected by a phenomenon of degradation: reliability gets worse, system structure becomes corrupted, consistency of documentation is lost, maintainability decreases. Such degradation has to be detected as soon as it arises in a software system, otherwise the system starts aging and rejuvenation becomes too onerous [Vis97a].

The main cause of degradation can be addressed in the maintenance activities upon a software system [Leh80, Vis99]. In order to overcome, or at least to minimize, the degradation induced by maintenance, we need to monitor the maintenance activities and then identify the specific features that are getting worse [Sne97].

It is widely accepted that the complexity of a software system affects its maintainability. For example, Kafura and Reddy used seven complexity metrics as predictors of the maintenance task effort [Kaf87]. The same conclusion was expressed by Rombach [Rom87], who used five complexity metrics. Despite of the great amount of

complexity metrics available in literature (for example Zuse in [Zus91] presents more than 100 complexity metrics), monitoring maintenance activities through this kind of metrics is in general costly and not always reliable [She88, Fen99]. Furthermore, these metrics are difficult to interpret, because each metric emphasizes a particular aspect of degradation and the aspects shown by different metrics are not orthogonal [Fen99].

The goal of our research is to define and validate a class of metrics: the *entropy of software systems* (hereafter simply referred as *entropy*). All the instances of the entropy class are aimed at assessing software degradation. For each instance, we analyze the conceptual and practical meaning. Through this analysis a first conceptual validation has been executed. An empirical validation has been conducted using data collected in a replicated case study performed by students.

The paper is organized as follows: section 2 defines the entropy of software systems; section 3 presents the direct measures of software quality used for validation; section 4 describes the case study and section 5 discusses the results; section 6 draws conclusions and future works. Some tabular data are reported in the Appendixes.

2. Entropy of a Software System

This paper uses only one class of metrics, entropy, which is embedded in a general framework. In the following, we summarize the framework and, within it, define the class of entropy metrics.

2.1. Basic Concepts

According to [Jac92], a software system is considered as a collection of *software models* (or simply, *models*), including the requirements model, the analysis model, the design model and the implementation model. Each model is composed of a collection of components, which are connected, according to an architectural style, through relationships. In [Lan95] the concepts of internal and external traceability relationships were first introduced.

The *internal traceability* concerns relations connecting components in the same model. The *external traceability* concerns relations connecting components belonging to models at different abstraction levels.

The concepts of internal and external traceability relationships are used to define the *Model Dependency Descriptor (MDD)*, which represents the software system as a graph $MDD = (System, Rel)$, with $System = C_a \cup C_d \cup C_i$ and $Rel = E_i \cup E_e$; where:

- C_a , C_d and C_i are the sets of components from the requirements, analysis, design and implementation models, respectively;
- $E_i \subseteq C_a \times C_a \cup C_d \times C_d \cup C_i \times C_i$ is the internal traceability relationship;
- $E_e \subseteq C_a \times C_d \cup C_d \times C_i$ is the external traceability relationship.

In other words, MDD models the software workproduct set as a graph of software lifecycle objects connected by internal and external traceability relationships. The MDD does not include the requirements specification model because requirements are in general expressed in textual form and are difficult to formalize.

2.2. Entropy definition

The *entropy* of a software system is a class of metrics to assess the degree of disorder in a software system structure. Entropy covers all the components of a software system at different abstraction levels, as well as the traceability relationships among them.

According to Arnold and Bohner [Arn93], we assume that during impact analysis, first a set of *primary* entities is identified including candidates for change, and then the set of *secondary* candidates for change is found through a ripple-effect analysis. A *ripple effect* is defined as “the effect caused by making a small change to a system which affects many other parts of a system” [Ste74].

In the following, we provide a set of definitions for the entropy metrics.

Definition 1. Let, C_k and C_j be two components, either belonging to the same or to different abstraction models. C_j is said to be *directly impacted* by C_k iff C_j and C_k are connected by a link.

Definition 2. Let, C_k and C_j be two components, either belonging to the same or to different abstraction models. C_j is said to be *indirectly impacted* by C_k iff C_j is not *directly impacted* by C_k , but a path $P=[C_1, C_2, \dots, C_n]$ there exists such that:

- C_1 is directly impacted by C_k ;
- C_i is directly impacted by C_{i-1} , with $i = 1, 2, \dots, n$;

- C_j is directly impacted by C_n .

Definition 3. Let PC_H be the set of *primary components*, which includes the components to change in the model at higher abstraction level (M_H).

Definition 4. Let PC_l be the set of *primary components* in the model M_l at abstraction level l , induced by PC_{l-1} with $H < l \leq L+1$, where L is the lowest abstraction level. It includes the components in M_l directly impacted by components to modify in PC_{l-1} .

Definition 5. The number of direct impacts between the i -th component in PC_l and any other component in the model M_l , $H \leq l \leq L$ is said the number of *direct internal links (DIL_i)*.

Definition 6. The number of direct impacts between the i -th component in PC_l and any other component in the model M_{l+1} , $H \leq l \leq L$ is said the number of *direct external links (DEL_i)*.

Definition 7. The i -th component in PC_l is connected to the other components in the model M_l through both a number of direct internal links (DIL_i) and a number of indirect internal links (IIL_i) (according to definition 2). The sum of $EIL_i = DIL_i + IIL_i$ is said the number of *extended internal links*.

Definition 8. The i -th component of the generic model M_l is associated with a number of direct external impacts which is DEL_i (according to definition 6); moreover, the same component has a number of indirect impacts to the components of PC_{l-2}, \dots, PC_H which are respectively called $IEL_{i, l-2}, \dots, IEL_{i, H}$. We call number of *extended external links*

$$EEL_i = DEL_i + \sum_{k=H}^{l-2} IEL_{i,k}$$

Finally, let's define the Entropy. Each M_l is associated with a set of entropy H_l generally defined as:

$$H_l = - \sum_i p_i \ln p_i,$$

where:

- i is the number identifying the links considered in H_l ,
- n_i is the number of links starting from i -th component;
- $p_j = \frac{n_j}{\sum n_j}$, where the sum is extended to all the components in M_l ; it represents the probability to involve the links in the impact analysis of the change to the i -th component.

The previous definition specifies four instances for each abstraction level l different from L , for this abstraction level do not exist the third and fourth instances:

- the *direct internal entropy*, when $n_j = \text{DIL}_j$ of the PC_i ;
- the *extended internal entropy*, when $n_j = \text{EIL}_j$ of the PC_i ;
- the *direct external entropy*, when $n_j = \text{DEL}_j$ of the PC_i ;
- the *extended holistic entropy*, when $n_j = \text{EEL}_j$ of the PC_i .

Based on [Kit95], the metrics belonging to the entropy class are different from each other with respect to the entities and attributes they observe. On the contrary, the relationship between entities and attributes, even though formally different, are similar in their meaning. Finally, all the metrics are equals with respect to value, unit, scale type and measurement instrument.

2.3. Entropy meaning

In order to analyze the entropy meaning, a preliminary consideration on p_j is useful.

If the number of links starting from the j -th component in a generic M_l has an increasing trend, then the $\sum p_j$ has an increasing trend too; therefore the p_j -s have a decreasing trends; finally, H_l increases.

For example, let's suppose to consider a software system with two representation models, M_l and M_{l-1} ; let C_1 and C_2 be the components in the first one, and let's leave unspecified the components in the second one, in that they are not relevant for our purposes. Moreover, let's suppose that the links starting from C_1 and C_2 are established, both internally and toward M_{l-1} , as shown in figure 1.a.

Then it results $p_j = 1/7$ and

$$H_l = -\sum_{i=1}^7 \frac{1}{7} \ln \frac{1}{7} = \ln 7 = 1.95$$

On the other hand, if we suppose that between M_l and M_{l-1} there are more links, as shown in figure 1.b, then the result will be

$$H_l = -\sum_{i=1}^9 \frac{1}{9} \ln \frac{1}{9} = \ln 9 = 2.2$$

Physically, if the number of links starting from one or more components in a representation model increases, then, also the mean number of the paths through which the effect of a change propagates.

The *direct internal entropy* expresses the links that software engineers can see through the workpackages they are currently using. In fact, the links considered by this entropy are the links the analysts and the designers can read on their blueprints, while programmers follow the control flows of their programs. Therefore, this entropy synthetically expresses the coupling among components in each model.

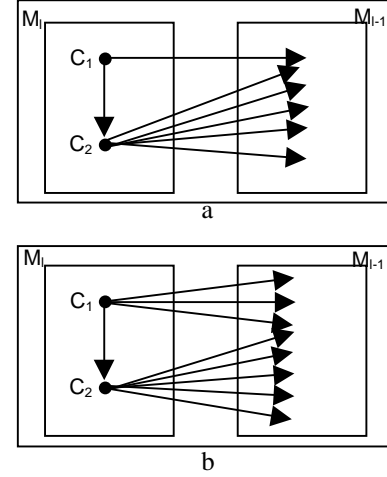


Figure 1. Graphical representation of links among components.

The *direct external entropy* considers the traceability links to the next lower abstraction model. The traceability links are conceptually partitioned into two types: *structural* and *cognitive*.

Structural links connect each higher-level component to one or more lower abstraction level components in which the first one is transformed, according to the working approach currently adopted. For example: an essential class A_i in analysis is transformed in a design class D_j using the data transformation methods, and in a second design class D_k using the data detecting and control methods. In this case, there are two structural links starting from A_i : one going to D_j , and one going to D_k .

On the other side, *cognitive links* express transformations derived from design or implementation decisions, instead of the techniques provided by the currently used approach. For example: the class A_i referenced in the previous example can be detailed in design in two classes (D_j^1 and D_j^2), or in only one class (D_j): this is a design decision. Therefore, if a software engineer would assume the first decision, two links start from A_i going, respectively, to D_j^1

and D_j^2 , which are justified by the design decision rather than the adopted design approach. Therefore, these two links are cognitive.

Then, the extended entropy expresses the adequacy of a solution (expressed with a design model) to a given problem (expressed with an analysis model). In fact, if a very articulated solution is adopted to solve a problem, there are many links starting from a model to the next one. Therefore, a simple change to a higher abstraction level component might have a great impact in the lower abstraction level: this is expressed by the amount of the value of H .

An interesting role is assumed by the cognitive links when a project is developed following the information hiding principle. In fact, supposing that a system has to be modified because the behavior of a function has changed, then all the links derived from hiding information other than behavioral ones, do not generate any impact. This is the same as saying that these links can be dropped in the impact analysis of such a change, i.e., the actual number of links involved in the change decreases and therefore the actual extended entropy decreases. This is coherent with the information hiding goal to make maintenance easier.

The *extended internal entropy* has the same meaning as the direct internal entropy, but other than coupling as seen by software engineers on their documents, it also includes the indirect relations among components.

Finally, the *extended holistic entropy* is the disorder of the whole system perceived from a model.

3. Quality Factors

According to Schneidewind [Sch92], an indirect measure of software quality can be validated by showing that its values are statistically associated with some metrics that directly measure software quality. These direct metrics can be mapped to quality factors, which are attributes of software that contribute to its quality, as perceived by users. For example, the effort spent for a change is usually mapped to the maintainability quality factor.

In order to assess software system degradation we have considered three direct measures of software degradation:

- the *number of discovered defects*, i.e. the number of defects, recorded for each model, discovered during the inspections of analysis and design, and during testing;
- the *number of slipped defects*, i.e. the number of defects, recorded for each model, missed during inspections and tests but discovered during the use of the system;

- the *maintenance effort*, i.e. the person hours spent to execute the maintenance process for each model.

We have not chosen these metrics in advance, but we have made a selection within a previously existing data set. However, the selected metrics express well software degradation, as discussed in the following:

- Degraded software is difficult to maintain. Therefore, the number of discovered defects has an increasing trend when degradation grows too.
- Degraded software is hardly readable, comprehensible and testable, therefore some defects existing in it cannot be found during inspections or tests. So, the number of slipped defects increases when degradation grows too.
- Degraded software is more complex than usual, and then performing maintenance activities requires more effort than usual. So, an increasing maintenance effort is a symptom of software system degradation.

4. Case Study

Students of a software engineering course at the University of Bari were asked to develop and then maintain a software system. The application domain of the system, scheduling the courses in a computer science department, was taken from a software engineering textbook [Jal97].

4.1. Process for data collection

In the first session, all students were assigned to the development of the first version of the same software system, whose requirements were produced by the teachers. Students, working in 3-person groups, performed the analysis, design and implementation activities. Analysis and design ended with an inspection; implementation was followed by testing (unit, integration and system test). The following deliverables were provided:

- analysis and design documentation;
- inspection reports including discovered defects;
- timesheets for measuring the effort spent for analysis, design, implementation, inspection and test; inspection effort was included in the analysis and design sheets, while test effort was included in the implementation sheet.

Instructors then executed acceptance test over all delivered workproducts, with the help of final-year undergraduate students. The acceptance test was used to find slipped defects.

The timesheets were weekly delivered. Data were checked with respect to their internal consistency. In case of problems, students were asked to explain or correct.

All the groups had to finish the first system version within a period of 30 days.

The best four systems were then corrected by the final-year undergraduates to remove the slipped defects. The results represent the initial systems of our replicated case study: $\{S_{1,0}, S_{2,0}, S_{3,0}, S_{4,0}\}$.

Systems were C++ object-oriented programs, whose size was about 3000 LOC, In the next session, each group was randomly assigned to one of the four previously developed systems and received a maintenance request. In order to preserve documentation consistency, each maintenance request had to be satisfied by each group according to the iterative enhancement model of maintenance [Bas75, Bas90].

All the maintenance requests had an impact on the analysis workpackage as H abstraction level. In this way, all the representation models were involved by each maintenance request.

The session workproducts, included process data, were always the same as for the first session. The process was analogous to the one followed in the first session, and again there were 30 days available to satisfy the maintenance request.

At the end of the i -th session, the best four systems were chosen to be included in the set $\{S_{1,i}, S_{2,i}, S_{3,i}, S_{4,i}\}$, which goes on in the next session, to be further maintained.

At the end of the course, four complete sets of changes were collected. Among them, we do not consider the set $S_{k,0}$, because the system was built by scratch and then it is outside of our interest. Table 1 summarizes collected data.

After delivering the case studies, final-year undergraduate students built the MDD models at five increasing granularity levels and then inserted the MDD models into an impact software analysis tool, called ANALYST [Fas99, Cim99, Bia00]:

- in Granularity Level 1 (shortened in $GRAN_1$) each model considers as components the essential classes.
- in Granularity level 2 ($GRAN_2$) each model considers as components the same classes and the methods.
- in Granularity level 3 ($GRAN_3$) each model considers as components the essential classes, the methods and the attributes.
- in Granularity level 4 ($GRAN_4$), besides of the components included in the previous model, in the design and the code all the main supporting components appear (reading data, management of persistent data, ...).
- in Granularity level 5 ($GRAN_5$), besides of the components included in the previous model they are added, in design and code the details of methods.

In the analysis model, $Gran_3$, $Gran_4$, and $Gran_5$ are the same, because it does not make sense to give a deeper description. As a consequence, the two internal entropy metrics calculated for such a model at these three granularity levels will be the same.

5. Results

In this section, first we analyze the results of the replicated case study, then we validate entropy as class of metrics.

	analysis			Design			implementation		
	#detected defects	effort (person hours)	#slipped defects	#detected defects	effort (person hours)	#slipped defects	#detected defects	effort (person hours)	#slipped defects
$S_{1,1}$	15	6	5	41	50	5	33	80	19
$S_{1,2}$	18	8	7	47	54	7	47	111	19
$S_{1,3}$	23	11	8	49	62	8	52	123	22
$S_{2,1}$	9	5	2	15	17	2	12	39	5
$S_{2,2}$	11	5	3	19	21	3	17	40	7
$S_{2,3}$	15	7	5	29	25	5	20	42	8
$S_{3,1}$	12	6	3	21	19	3	30	66	14
$S_{3,2}$	13	7	4	27	23	5	32	79	15
$S_{3,3}$	17	9	6	31	33	6	35	87	19
$S_{4,1}$	14	9	4	21	18	4	22	47	11
$S_{4,2}$	20	12	7	33	33	6	28	53	13
$S_{4,3}$	22	13	11	51	65	9	45	105	17

Table 1. Data collected during the maintenance processes of four systems.

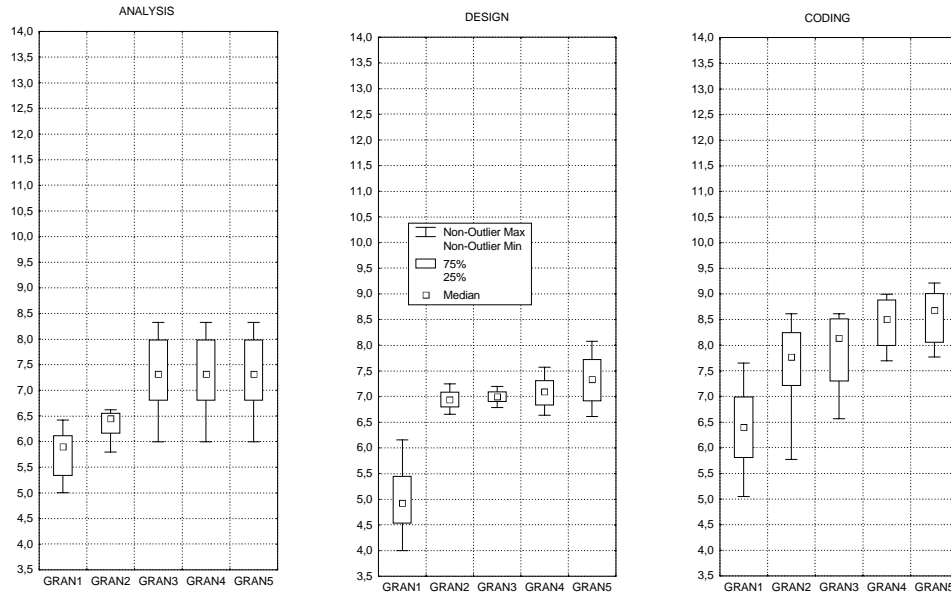


Figure 2. Direct Internal Entropy.

5.1. Entropy values

Tables A.1 to A.10 in appendix A report the values of entropy for each system, for each granularity level, and for each abstraction level model. In this section, we only emphasize the more interesting observations.

In figure 2 the box plots of direct internal entropy are shown. As can it be seen, for each model and entropy metric, the finer the granularity the greater is the value of entropy. This is because of the feature of coarser grain, which hides inside itself some disorder. A part of this disorder is explicated when the grain becomes finer. Physically, this means that the finer grain of representation models makes the impact analysis more coherent with the real disorder of the software system.

In the same figure, it is shown that entropy values for design are lower than for analysis. This confirms that design makes easier to read and comprehend source code.

Figure 3 shows the extended holistic entropy. Here the entropy values in the design model get very high when granularity increases. This means that a coarser grain hides design defects.

5.2. Validation

Schneidewind in [Sch92] proposed a metrics validation methodology based on six validity criteria. Each validity criterion specifies a specific quantitative relationship that should exist between factors and metrics.

We have applied only two of these validity criteria: *consistency* and *tracking*. We could not perform validation with respect to other three criteria (*predictability*, *repeatability* and *association*) because of the few available observations in our case study. We also could not validate with respect to the *discriminative power* criterion because there was no baseline to discriminate between degraded and non-degraded systems, based on our direct measures. Anyway, available data would not allow us to empirically define any reasonable threshold.

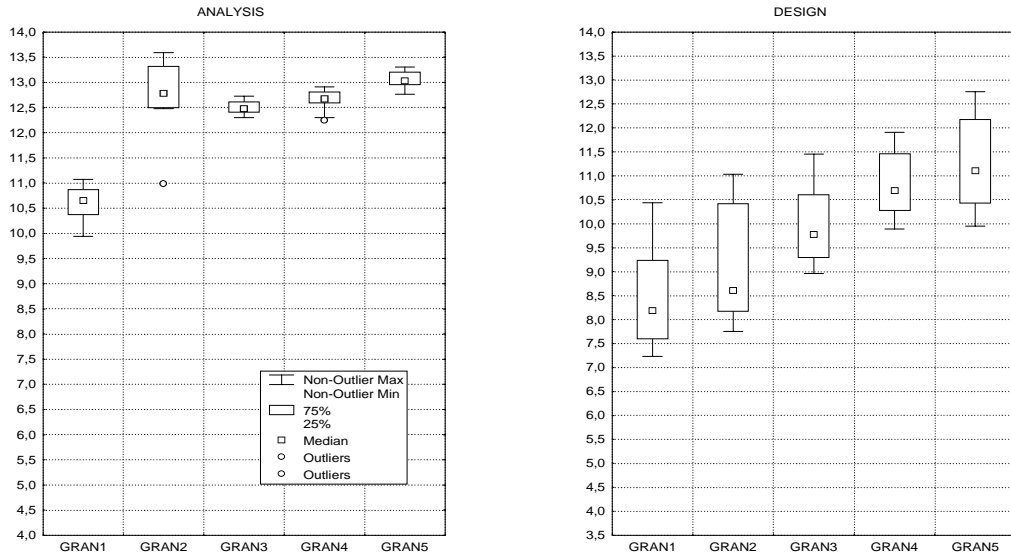


Figure 3. Extended Holistic Entropy.

Consistency

The consistency criterion evaluates whether there is adequate consistency between the ranks of the factor, or direct measure, and the ranks of the indirect metric, in our case the entropy class of metrics.

A rank order correlation coefficient, such as Spearman R , is applied to verify that the coefficient R exceeds a given threshold (β_c) with a specified significance level (α). In our case, we established the following values: $\beta_c = 0.7$ and $\alpha = 0.05$. Using our representative sample, if R is higher than 0.7 with p less than 0.05, we might conclude that the entropy metric, at a given granularity level, is consistent with the direct measure of degradation. This means that if after a change entropy grows then we might say that software degradation (directly measured) would increase too.

Tables B.1 to B.10 in appendix B report the values of correlation (R). The results for consistency are summarized in Table 2. There are 12 observations (4

systems per 3 releases). Each cell contains the minimum granularity level that satisfies the condition $R > \beta_c$ with an acceptable confidence level ($p < 0.05$). For example, the extended internal entropy is consistent with the effort in the design model starting from the granularity level 2.

Empty cells correspond to the direct external and extended holistic entropy when measured for the implementation model. This is not due to lacking consistency but because there are no external links and so the values for these metrics cannot be computed.

A finding is that all the computed instances of entropy are consistent with the direct measures of degradation whether an adequate granularity level is considered.

Furthermore, we can say that the minimal level of granularity, which is adequate for all the quality factors and entropy metrics is the granularity level 3.

Quality Factor	#detected defects			effort			#slipped defects		
	analysis	design	Implem.	Analysis	design	implem.	analysis	design	implem.
Direct Internal	Gran 2	Gran 1	Gran 1	Gran 2	Gran 1	Gran 1	Gran 2	Gran 1	Gran 1
Extended Internal	Gran 1	Gran 1	Gran 1	Gran 2	Gran 1	Gran 1	Gran 2	Gran 2	Gran 1
Direct External	Gran 2	Gran 2		Gran 3	Gran 2		Gran 2	Gran 3	
Extended Holistic	Gran 1	Gran 2		Gran 2	Gran 2		Gran 1	Gran 2	

Table 2. Minimal granularity required to validate consistency.

Tracking

The tracking criterion evaluates whether the indirect metric is able to adequately track changes in the factor, or direct metric, so that to be confident that it is actually an indirect metric.

Figures 4 to 6 show the trends of the direct metrics with respect to each entropy metric. Each figure includes the graphs concerning a model.

The trends of the four system versions are all equal: when the direct metric decreases, the entropy metric decreases too. Therefore, the tracking validity criterion holds for all the defined entropy metrics.

By comparing the design of $S_{1,i}$ to $S_{2,i}$, $S_{3,i}$, $S_{4,i}$, it is clear that the quality of design strongly influences the

value of the entropy. In fact, figure 6 shows the greatest changes in values. Once the design has settled on a bad entropy value, this determines the quality of the code, as it is highlighted in figure 6.

The same quality gradient is not evident in the analysis model. This again confirms that design is critical to develop and maintain software systems.

Furthermore, the number of detected defects, even if shows the same trend of entropy, does not have any analogous gradient. On the contrary, the number of slipped defects follows more coherently the entropy gradient. An interpretation might be that when the quality of a software system degrades it is more likely that many defects persist in it nevertheless inspection and testing.

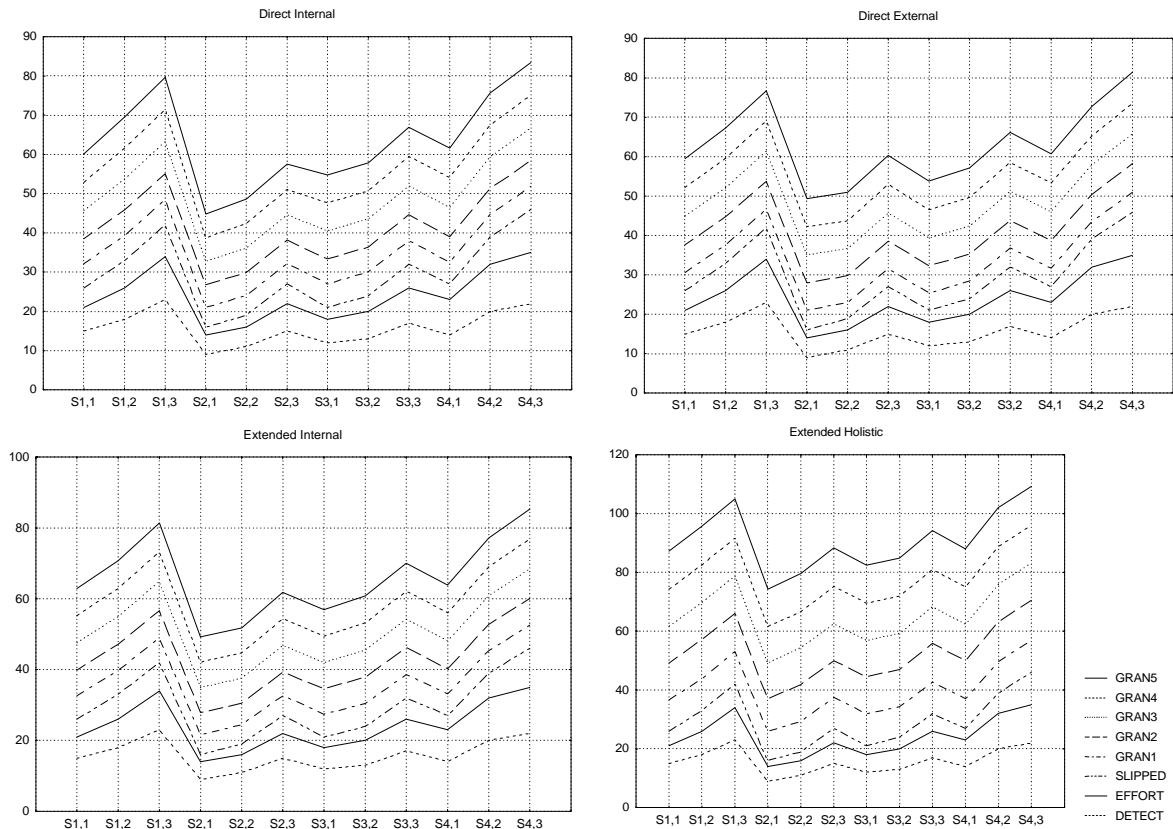


Figure 4. Comparison of trends with respect to analysis.

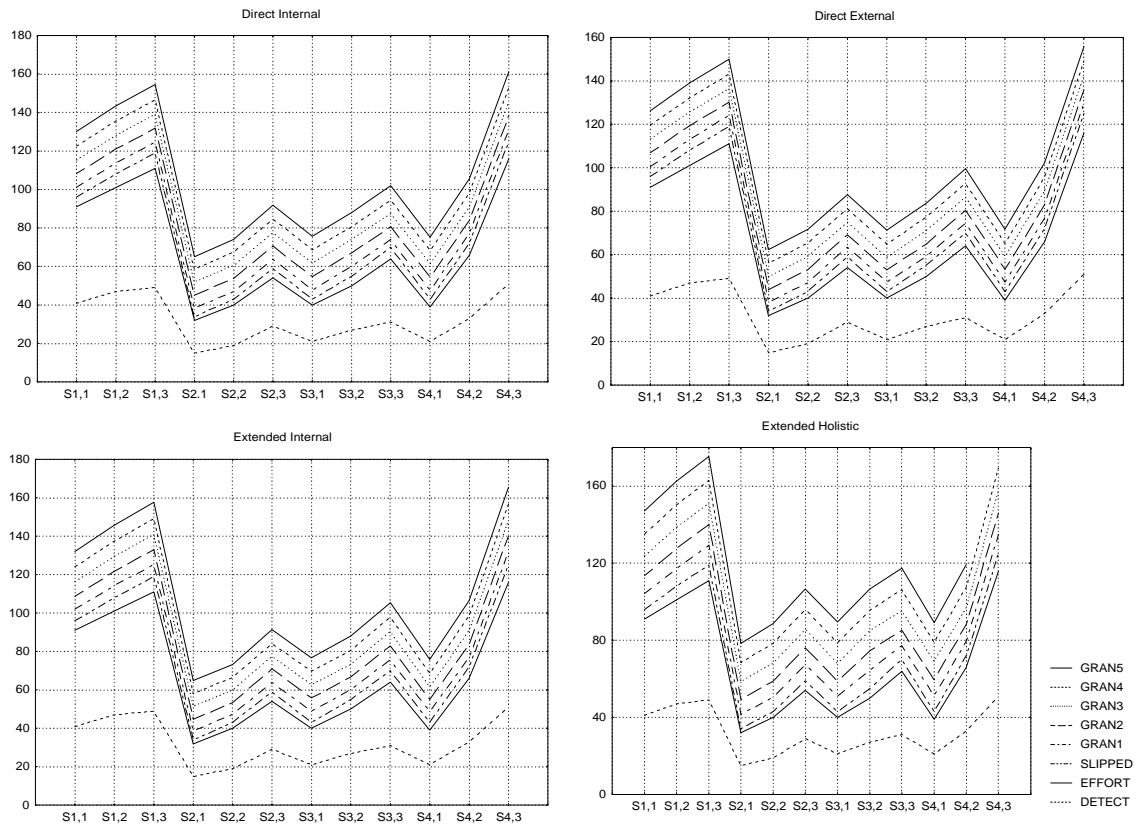


Figure 5. Comparison of trends with respect to design.

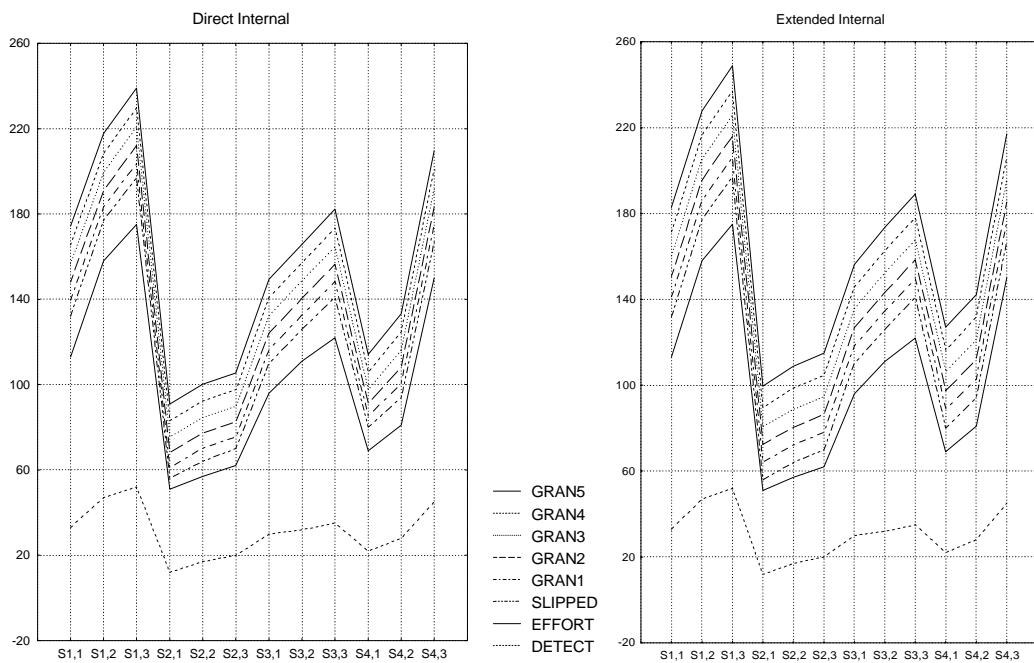


Figure 6. Comparison of trends with respect to implementation.

6. Conclusions

In this paper the definitions of entropy, previously introduced in [Vis97b], have been modified and extended to monitor software system degradation.

A tool, based on a software representation model aimed to support maintenance, can automatically compute the entropy metrics before and after each maintenance intervention. This means that the entropy metrics allow a software engineer to frequently monitor a software system, getting slower its aging.

The entropy metrics also locate the representation model in which degradation is injected. As a consequence, it is possible to improve system quality starting from the more adequate abstraction level.

We evaluated the consistency of the entropy metrics and the ability to track changes with respect to three quality factors, i.e., direct measures of software degradation: the number of detected defects, the maintenance effort and the number of slipped defects. Validation has been performed using data collected from a replicated case study. We showed that the granularity level of software system representation strongly influences the entropy.

A limitation of our work is that collected data are not enough to determine the optimal granularity level. Another consequence for the lack of available observations is that we could not perform a richer validity testing, considering for example, predictability, repeatability, association and discriminative power criteria. These limits can only be overcome by collecting more sample systems and changes, preferably in an industrial environment.

References

- [Arn93] R.S. Arnold, S. A. Bohner, "Impact Analysis – Towards a Framework for Comparison", *Proc. of Conf. on Software Maintenance*, IEEE CS Press, Los Alamitos, CA, 1993, pp. 292- 301
- [Bas75] V. R. Basili, A. J. Turner, "Iterative Enhancement: A Practical Technique for Software Development", *IEEE Trans. Software Engineering*, Dec. 1975, pp. 390- 396.
- [Bas90] V. R. Basili, "Viewing Maintenance as Reuse-Oriented Software Development", *IEEE Software*, Jan. 1990, pp. 19-25
- [Bia00] A. Bianchi, A. Fasolino and G. Visaggio "An Exploratory Case Study about the Maintenance Effectiveness of Traceability Models", *Proc. of the 8th*

- International Workshop on Program Comprehension*, Limerick, Ireland, 2000, pp. 149-158.
- [Cim99] A. Cimitile, A.R. Fasolino, G. Visaggio, "A Software Model for Impact Analysis: a Validation Experiment", *Proc. of the 6th Working Conference on Reverse Engineering*, IEEE Computer Society, Atlanta, Georgia, 1999, pp. 212-222
- [Fas99] A.R. Fasolino, G. Visaggio, "Improving Software Comprehension through an Automated Dependency Tracer ", *Proc. of the 7th International Workshop on Program Comprehension*, Pittsburgh, Pennsylvania, 1999, pp.58-65.
- [Fen99] N.E. Fenton, M. Neill, "A Critique of Software Defect Prediction Models", *IEEE Transactions on Software Engineering*, Vol. 25 no.5, 1999, pp.675-689.
- [Jal97] P. Jalote, *An Integrated Approach to Software Engineering*, 2nd ed., Springer-Verlag, New York Inc., 1997.
- [Kaf87] D. Kafura and G.R. Reddy, "The Use of Software Complexity Metrics in Software Maintenance", *IEEE Transactions on Software Engineering*, Vol. 13 no. 3, March 1987, pp.335-343.
- [Kit95] B. Kitchenham, S.L. Pfleeger, N. Fenton, "Towards a Framework for Software Measurement Validation", *IEEE Transactions on Software Engineering*, vol.21, no.12, 1995, pp.929-944.
- [Lan95] F. Lanubile, and G. Visaggio, "Decision-driven maintenance", *Journal of Software Maintenance: Research and Practice*, vol.7, no.2, March-April 1995, pp.91-115.
- [Leh80] M. Lehman, "Program, Life-Cycle, and the Law of Program Evolution", *Proceedings of the IEEE*, 1980, pp 1060-1076.
- [Rom87] H.D. Rombach, "A Controlled Experiment on the Impact of Software Structure on Maintainability ", *IEEE Transactions on Software Engineering*, Vol. 13 no. 3, March 1987, pp.344-354.
- [Sch92] N.F. Schneidewind, "Methodology for Validating Software Metrics", *IEEE Trans. On Software Engineering*, vol.18, No.5, May1992, pp.410-422.
- [She88] M.J. Shepperd, "A Critique of the Cyclomatic Complexity as a Software Metric", *Software Engineering Journal*, vol.3 no.2, 1988, pp.30-36.
- [Sne97] H. Sneed, "Measuring the performance of a Software Maintenance Department", *Proc. of 1st Euromicro Working Conference on Software Maintenance*, Berlin – Germany, 1997.
- [Ste74] W. Stevens, G. Meyers, L. Constantine, "Structured Design", *IBM Systems J.*, 13 (2), 1974.
- [Vis97a]G. Visaggio, "Comprehending the Knowledge Stored in Aged Legacy Systems to Improve their Qualities with a Renewal Process", ISERN-97-26 International Software Engineering Research Network, 1997.
- [Vis97b]G. Visaggio, "Structural Information as a Quality Metric in Software Systems Organization", *Proceedings*

of *IEEE International Conference on Software Maintenance*, Bari, Italy, 1997, pp. 92-99.

[Vis99] G. Visaggio, "Assessing the Maintenance Process through Replicated, Controlled Experiment", *The Journal of Systems and Software*, Elsevier Science, Vol. 44, N°3, 1999, pp. 187-197.

[Zus91] H. Zuse, *Software Complexity measure and methods*, Walter de Gruyter, 1991.

Appendix A. Values of Entropy

In this Appendix, entropy data are reported for all the systems and granularity levels:

- Tables A.1 through A.4 show the four instances of entropy for analysis model;
- Tables A.5 through A.8 show the four instances of entropy for design model;
- Tables A.9 and A.10 show direct internal and extended internal entropy for implementation model.

System	Gran1	Gran2	Gran3	Gran4	Gran5
S _{1,1}	6,152	6,365	7,198	7,198	7,198
S _{1,2}	6,359	6,531	7,865	7,865	7,865
S _{1,3}	6,421	6,624	8,185	8,185	8,185
S _{2,1}	5,003	5,798	5,995	5,995	5,995
S _{2,2}	5,065	5,859	6,235	6,235	6,235
S _{2,3}	5,189	5,974	6,458	6,458	6,458
S _{3,1}	5,976	6,377	7,147	7,147	7,147
S _{3,2}	5,997	6,389	7,160	7,160	7,160
S _{3,3}	6,082	6,527	7,439	7,439	7,439
S _{4,1}	5,499	6,500	7,525	7,525	7,525
S _{4,2}	5,744	6,566	8,102	8,102	8,102
S _{4,3}	5,826	6,588	8,326	8,326	8,326

Table A.1. Report of the direct internal entropy in the analysis model.

System	Gran1	Gran2	Gran3	Gran4	Gran5
S _{1,1}	6,666	7,222	7,672	7,672	7,672
S _{1,2}	6,771	7,410	7,867	7,867	7,867
S _{1,3}	6,914	7,727	8,275	8,275	8,275
S _{2,1}	5,595	6,293	7,126	7,126	7,126
S _{2,2}	5,466	6,106	7,065	7,065	7,065
S _{2,3}	5,714	6,586	7,513	7,513	7,513
S _{3,1}	6,406	7,228	7,421	7,421	7,421
S _{3,2}	6,450	7,350	7,677	7,677	7,677
S _{3,3}	6,599	7,658	7,943	7,943	7,943
S _{4,1}	6,197	7,029	7,894	7,894	7,894
S _{4,2}	6,385	7,407	8,132	8,132	8,132
S _{4,3}	6,538	7,533	8,399	8,399	8,399

Table A.2. Report of the extended internal entropy in the analysis model.

System	Gran1	Gran2	Gran3	Gran4	Gran5
S _{1,1}	4,595	7,028	7,272	7,293	7,312
S _{1,2}	4,676	7,131	7,367	7,480	7,619
S _{1,3}	4,639	7,110	7,599	7,694	7,712
S _{2,1}	5,010	6,936	7,134	7,169	7,127
S _{2,2}	4,088	6,748	6,884	7,015	7,212
S _{2,3}	4,614	6,926	7,204	7,224	7,305
S _{3,1}	4,381	6,872	7,099	7,195	7,279
S _{3,2}	4,439	6,910	7,098	7,291	7,378
S _{3,3}	4,845	7,022	7,301	7,425	7,560
S _{4,1}	4,698	7,034	7,291	7,245	7,471
S _{4,2}	4,490	6,948	7,398	7,408	7,484
S _{4,3}	5,000	7,252	7,589	7,727	7,851

Table A.3. Report of the direct external entropy in the analysis model.

System	Gran1	Gran2	Gran3	Gran4	Gran5
S _{1,1}	10,602	12,566	12,515	12,609	13,000
S _{1,2}	10,751	13,479	12,459	12,820	13,181
S _{1,3}	11,070	12,954	12,725	12,911	13,305
S _{2,1}	9,939	10,983	12,356	12,247	12,763
S _{2,2}	10,354	12,510	12,478	12,299	12,944
S _{2,3}	10,496	12,476	12,560	12,700	13,065
S _{3,1}	10,988	12,488	12,466	12,576	12,937
S _{3,2}	10,396	12,601	12,299	12,608	12,967
S _{3,3}	10,692	13,165	12,478	12,737	13,117
S _{4,1}	10,016	12,985	12,359	12,641	13,006
S _{4,2}	10,714	13,588	12,667	12,883	13,228
S _{4,3}	10,999	13,471	12,717	12,799	13,302

Table A.4. Report of the extended holistic entropy in the analysis model.

System	Gran1	Gran2	Gran3	Gran4	Gran5
S _{1,1}	5,208	7,053	7,056	7,252	7,592
S _{1,2}	5,984	7,124	7,126	7,367	7,853
S _{1,3}	5,692	7,216	7,201	7,505	7,989
S _{2,1}	4,366	6,659	6,795	6,639	6,700
S _{2,2}	4,002	6,775	6,899	6,815	6,615
S _{2,3}	4,799	6,883	7,065	6,995	7,176
S _{3,1}	4,825	6,895	6,915	7,135	6,901
S _{3,2}	5,024	7,035	6,972	6,684	7,234
S _{3,3}	4,005	6,672	6,789	7,053	7,534
S _{4,1}	4,699	6,826	6,908	6,861	6,946
S _{4,2}	5,071	6,973	7,014	7,247	7,430
S _{4,3}	6,157	7,251	7,194	7,573	8,075

Table A.5. Report of the direct internal entropy in the design model.

System	Gran1	Gran2	Gran3	Gran4	Gran5
S _{1,1}	6,015	6,645	7,726	7,751	7,893
S _{1,2}	6,331	7,296	7,785	7,931	8,270
S _{1,3}	6,198	7,801	7,965	8,101	8,570
S _{2,1}	4,754	6,074	6,615	6,693	6,735
S _{2,2}	4,259	6,245	6,412	6,556	6,774
S _{2,3}	4,858	7,177	6,433	6,580	7,298
S _{3,1}	5,931	6,893	6,993	6,998	7,006
S _{3,2}	5,149	6,521	6,525	7,355	7,417
S _{3,3}	5,633	7,216	7,414	7,520	7,532
S _{4,1}	5,715	6,413	6,558	6,952	7,258
S _{4,2}	5,128	6,631	7,254	7,744	7,802
S _{4,3}	6,968	8,018	8,294	8,344	8,802

Table A.6. Report of the extended internal entropy in the design model.

System	Gran1	Gran2	Gran3	Gran4	Gran5
S _{1,1}	4,567	6,374	6,292	6,327	6,655
S _{1,2}	5,063	6,294	6,358	6,454	6,811
S _{1,3}	5,136	6,022	6,352	6,463	6,955
S _{2,1}	4,125	5,742	5,989	6,179	6,293
S _{2,2}	4,279	5,918	6,012	6,215	6,310
S _{2,3}	4,514	5,618	5,736	6,255	6,584
S _{3,1}	4,380	5,598	5,875	6,012	6,382
S _{3,2}	4,311	5,500	6,126	6,320	6,401
S _{3,3}	4,619	5,833	6,023	6,302	6,785
S _{4,1}	4,454	5,813	5,876	6,115	6,451
S _{4,2}	4,803	6,214	6,382	6,345	6,551
S _{4,3}	4,321	6,450	6,538	6,576	7,016

Table A.7. Report of the direct external entropy in the design model.

System	Gran1	Gran2	Gran3	Gran4	Gran5
S _{1,1}	8,257	9,336	10,237	11,259	12,052
S _{1,2}	9,124	10,662	10,843	11,758	12,299
S _{1,3}	10,438	10,588	10,990	11,910	12,503
S _{2,1}	7,569	7,757	9,202	9,893	9,951
S _{2,2}	7,555	8,318	9,386	10,150	10,272
S _{2,3}	8,529	8,550	9,572	10,405	10,592
S _{3,1}	7,853	8,032	9,136	10,726	10,736
S _{3,2}	9,346	10,257	10,366	10,655	10,987
S _{3,3}	7,232	8,013	9,983	11,033	11,230
S _{4,1}	8,111	8,357	9,563	10,013	10,128
S _{4,2}	7,635	8,655	8,965	10,456	11,532
S _{4,3}	10,059	11,036	11,454	11,659	12,753

Table A.8. Report of the extended holistic entropy in the design model.

System	Gran1	Gran2	Gran3	Gran4	Gran5
S _{1,1}	7,605	8,425	8,521	8,867	8,960
S _{1,2}	6,354	7,686	8,522	8,895	9,157
S _{1,3}	6,565	8,615	8,611	8,998	9,212
S _{2,1}	5,048	7,192	7,214	7,699	7,773
S _{2,2}	6,125	7,242	7,254	7,735	7,826
S _{2,3}	5,486	6,998	7,341	7,771	7,885
S _{3,1}	6,326	7,852	8,088	8,571	8,600
S _{3,2}	6,658	7,854	8,211	8,322	8,752
S _{3,3}	7,325	8,068	8,197	8,827	8,856
S _{4,1}	5,286	5,775	6,569	8,223	8,232
S _{4,2}	6,453	7,698	7,995	8,443	8,574
S _{4,3}	7,654	8,474	8,500	8,961	9,053

Table A.9. Report of the direct internal entropy in the implementation model.

System	Gran1	Gran2	Gran3	Gran4	Gran5
S _{1,1}	9,173	9,542	9,596	11,162	11,386
S _{1,2}	8,995	9,542	9,610	11,208	11,482
S _{1,3}	9,185	9,750	9,787	11,403	11,681
S _{2,1}	8,167	8,266	8,308	8,846	10,048
S _{2,2}	8,172	8,189	8,434	9,866	10,130
S _{2,3}	8,170	8,223	8,372	9,938	10,216
S _{3,1}	8,348	8,457	8,751	10,115	10,521
S _{3,2}	8,597	8,613	9,210	10,258	10,844
S _{3,3}	8,652	8,833	9,170	10,553	10,856
S _{4,1}	8,724	8,811	8,830	10,051	10,560
S _{4,2}	8,856	8,996	9,128	10,589	10,703
S _{4,3}	9,025	9,232	9,977	10,722	11,143

Table A.10. Report of the extended internal entropy in the implementation model.

Appendix B. Values of Correlation

In this Appendix, correlation of entropy with the quality factors are reported. More precisely:

- Tables B.1 – B.4 report correlation of the four instances of entropy for analysis model;
- Tables B.5 – B.8 report correlation of the four instances of entropy for design model;
- Tables B.9 and B.10 report correlation of direct and extended internal entropy for implementation model.

Entropy vs Quality Factor	Gran1	Gran2	Gran3	Gran4	Gran5
#detected defects	0,606	0,893	0,907	0,907	0,907
effort	0,366	0,923	0,923	0,923	0,923
#slipped defects	0,585	0,873	0,898	0,898	0,898

Table B.1. Correlation of direct internal entropy with quality factors in analysis model.

Entropy vs Quality Factor	Gran ₁	Gran ₂	Gran ₃	Gran ₄	Gran ₅
#detected defects	0,704	0,827	0,893	0,893	0,893
Effort	0,472	0,771	0,972	0,972	0,972
#slipped defects	0,694	0,813	0,880	0,880	0,880

Table B.2. Correlation of extended internal entropy with quality factors in analysis model.

Entropy vs Quality Factor	Gran ₁	Gran ₂	Gran ₃	Gran ₄	Gran ₅
#detected defects	0,270	0,757	0,928	0,928	0,914
Effort	0,352	0,694	0,866	0,835	0,905
#slipped defects	0,289	0,768	0,898	0,944	0,926

Table B.3. Correlation of direct external entropy with quality factors in analysis model.

Entropy vs Quality Factor	Gran ₁	Gran ₂	Gran ₃	Gran ₄	Gran ₅
#detected defects	0,767	0,739	0,718	0,967	0,981
Effort	0,592	0,817	0,518	0,873	0,901
#slipped defects	0,768	0,743	0,669	0,937	0,961

Table B.4. Correlation of extended holistic entropy with quality factors in analysis model.

Entropy vs Quality Factor	Gran ₁	Gran ₂	Gran ₃	Gran ₄	Gran ₅
#detected defects	0,848	0,848	0,809	0,914	0,977
effort	0,778	0,809	0,771	0,883	0,942
#slipped defects	0,739	0,749	0,714	0,813	0,944

Table B.5. Correlation of direct internal entropy with quality factors in design model.

Entropy vs Quality Factor	Gran ₁	Gran ₂	Gran ₃	Gran ₄	Gran ₅
#detected defects	0,781	0,862	0,841	0,914	0,991
Effort	0,701	0,848	0,795	0,865	0,963
#slipped defects	0,682	0,852	0,763	0,859	0,954

Table B.6. Correlation of extended internal entropy with quality factors in design model.

Entropy vs Quality Factor	Gran ₁	Gran ₂	Gran ₃	Gran ₄	Gran ₅
#detected defects	0,690	0,725	0,774	0,914	0,953
effort	0,634	0,753	0,788	0,935	0,928
#slipped defects	0,671	0,629	0,777	0,912	0,940

Table B.7. Correlation of direct external entropy with quality factors in design model.

Entropy vs Quality Factor	Gran ₁	Gran ₂	Gran ₃	Gran ₄	Gran ₅
#detected defects	0,648	0,820	0,725	0,872	0,963
Effort	0,585	0,795	0,743	0,879	0,963
#slipped defects	0,594	0,777	0,721	0,802	0,908

Table B.8. Correlation of extended holistic entropy with quality factors in design model.

Entropy vs Quality Factor	Gran ₁	Gran ₂	Gran ₃	Gran ₄	Gran ₅
#detected defects	0,748	0,783	0,923	0,965	0,993
Effort	0,748	0,783	0,923	0,965	0,993
#slipped defects	0,768	0,803	0,916	0,930	0,965

Table B.9. Correlation of direct internal entropy with quality factors in implementation model.

Entropy vs Quality Factor	Gran ₁	Gran ₂	Gran ₃	Gran ₄	Gran ₅
#detected defects	0,825	0,865	0,930	0,930	0,958
effort	0,825	0,865	0,930	0,930	0,958
#slipped defects	0,824	0,868	0,873	0,923	0,958

Table B.10. Correlation of extended internal entropy with quality factors in implementation model.