IBIMA Publishing
Communications of International Proceedings
https://ibimapublishing.com/p-articles/46SE/2025/4649725/
Vol. 2025 (36), Article ID 4649725

Software Architecture Entropy*

Stanislaw NIEPOSTYN

VIZJA University, Warsaw, Poland

Correspondence should be addressed to: Stanislaw NIEPOSTYN, j.niepostyn@vizja.pl

* Presented at the 46th IBIMA International Conference, 26-27 November 2025, Ronda, Spain

Abstract

In software architecture design, proper attention is often not paid to the relationships between elements across diagrams. However, IT designers, more or less consciously, introduce elements representing the same object instance in different diagrams with nearly identical names. These relationships are called consistency rules and are typically not supported in current software modeling tools. A software architecture without consistency rules is simply an unrelated set of diagrams. Failure to apply consistency rules leads to countless inconsistencies in software architecture. This article presents a mathematical proof demonstrating that applying consistency rules primarily increases the information content of a software architecture. Therefore, the concepts for improving the readability and orderliness of software architecture using consistency rules also have a strong mathematical basis. This article demonstrates the reduction of information entropy (information uncertainty) when applying consistency rules in the design of software architecture of IT systems. It has therefore been proven that labeling selected elements in different diagrams with similar names, indicating the use of consistency rules, is a very effective way to increase the information content of a software architecture while simultaneously improving its orderliness and readability. Considering the constantly increasing quality requirements placed on IT systems, it seems that without IT specialists paying more attention to consistency rules in software development, it will not be possible to significantly improve the current stagnation in software development methods.

Keywords: entropy, software architecture, software architecture metrics

Introduction

In building software architectures, the relationships between elements across diagrams are often overlooked. However, when constructing software architecture, IT designers more or less consciously introduce elements that represent the same object instance on different diagrams with nearly identical names. These connections are called consistency rules and are usually not saved in any way in the modeling tool. A software architecture without consistency rules is simply an unrelated set of diagrams. But these implicit consistency rules significantly facilitate the reading of such software architecture. Consequently, such consistency rules lead to an increase of the software architecture information content.

Designing software architecture without using tools that support its construction using consistency rules usually leads to hundreds, if not thousands, of inconsistencies, as Marchezan et al. (2023) showed, describing the detection of 39,683 inconsistencies. However, as described by Torre et al. (2023), consistency rules are not met with much enthusiasm by respondents, even those selected from among the participants of scientific conferences. It turns out

Cite this Article as: Stanislaw NIEPOSTYN, Vol. 2025 (36) “Software Architecture Entropy" Communications of International Proceedings, Vol. 2025 (36), Article ID 4649725, https://doi.org/10.5171/2025.4649725

that 44% of such respondents indicated that consistency rules can be important for verifying the consistency of software architecture. Therefore, despite this high understanding of the role of consistency rules among MDE (Model-Driven Engineering) experts, it is rather difficult to expect widespread use in the IT industry of solutions designed solely for identifying the consistency of software architecture during the development of information systems, as mentioned Marchezan et al. (2024). It seems that similar, recently proposed solutions by Hagel et al. (2025) or Klare et al (2021) are not of great importance for the development of information systems, because they only detect inconsistencies without indicating ways to fix them. Therefore, it is worth mentioning a tool proposed by Marchezan et al. (2023) for implementing repairs of inconsistencies in which they were detected. Even though the use of software architecture is not of great importance in most IT system projects, in the face of increasingly higher quality requirements for software, it seems that without IT specialists' attention to the rules of consistency in software development, it will not lead to a significant improvement in the current stagnation in software development methods.

In this article, a proof of the decrease in entropy of an architectural model was conducted using consistency rules between different elements in different diagrams in the construction of software architecture IT systems. The uncertainty of software architecture construction is thus reduced when introducing consistency rules, and therefore, the orderliness and consistency of the entire software architecture improve.

Therefore, marking selected elements in different models using similar names is an implicit way to increase the information content of the software architecture and, at the same time, to improve its orderliness and readability. This explains the use of consistency rules in the construction of software architecture, in order to cut its design time, as compared to designing without paying attention to consistency rules.

Related work

The term "entropy" is used in thermodynamics, probability theory, information theory, or the theory of dynamic systems. However, as Thims (2012) pointed out, Shannon's equation regarding information theory has no relation to similar equations used in thermodynamics or statistical mechanics. In the further part of the work, the term "entropy" will be understood in the meaning assigned to it by Shannon in accordance with the formula:

$$\text{Entropy} = -\sum_{i=1}^{n_1}(P_i \log_2 P_i) \tag{1}$$

where n_1 is the total number of all elements, and P_i is the probability of occurrence of a set i of possible elements.

Entropy may be perceived as a measure of uncertainty associated with a discrete distribution with appropriate probabilities. In research on UML diagrams, published by OMG (2015), it is assumed that the probability distribution of a given UML element is the quotient of the number of occurrences of a given UML element by the number of occurrences of all UML elements in a UML diagram. For example, the probability distribution for one class and for one attribute is identical and amounts to 0.5 for a UML diagram consisting of only one class (one occurrence) and one attribute (one occurrence).

For a UML diagram composed of only one class, entropy would be equal to 0, because the probability distribution for this one class would be equal to 1.0 (a certain event). Hence, a UML diagram with only one class does not contain any information about the given system.

Using the software complexity metrics with entropy, it can be proved that for the same diagram configuration, the decrease in entropy occurs due to the increase in the number of consistency rules (links) between the elements of these diagrams. Relationships between individual elements of diagrams should be interpreted as displaying the existence of consistency rules between elements. These rules do not introduce additional elements to the diagrams' configuration (relationships between elements of diagrams) but are expressed in practice through similar names of the elements being linked.

The definition of inconsistencies in models was provided by Spanoudakis and Zisman (2001), indicating that the ultimate goal of management consistency is to maintain consistency during the design of the system. However, often researchers, including Straeten (2005), indicate that this is not realistic in a real project, where several architects work in parallel. On the other hand, it is worth mentioning that many studies on inconsistencies, along with the area of application of consistency rules according to Ha and Kang (2008), Sapna and Mohanty (2007), and Shinkawa (2006), have come to interesting proposals for defining and applying individual properties of various UML models in the field of software development. In addition, recent publications in this area, by Torre et al (2014, 2015), seem to aim to develop a full list of UML diagram consistency rules according to the new proposals for their classification after Allaki et al. (2015).

AICC and CDE software complexity metrics

One of the first metrics based on estimating the information content (entropy) of the software data structure was the AICC (Average Information Content Classification) by Harrison (1992) and the CDE (Class Design Entropy) metrics by Bansiya et al. (1999), which calculated the complexity of the software code, the first referring to structural languages (in particular PL/I) and the other object-oriented languages. It is worth noting that the AICC and CDE metrics have been proposed for estimating the source code. The proposed mathematical formulas of these metrics were used in their later applications to assess, among others, UML diagrams.

The Average Information Content Classification metrics (AICC) describes the complexity of the software based on the concept of entropy and was proposed to estimate the complexity of the source code. The AICC metrics includes the following parameters: N_I is the total number of (non-unique) symbols of the language used in the code, and f_i, where $1 \leq i \leq \eta_1$ is the number of occurrences of the i-th language symbol appearing in the source code. The formula for calculating the AICC metrics is given below.

$$\mathrm{AICC} = -\sum_{i=1}^{\eta_1} \frac{f_i}{N_1} \log_2 \frac{f_i}{N_1} \tag{2}$$

The interpretation of this metric is that a program with a higher value of this metric should be less complex (complex) than a program with a lower value of this metric. The AICC values are not additive, nor is the comparison of its values for two different modules is not meaningful, but already these values, for the same module, could indicate justification for using, for example, some complex software libraries. For large systems, the AICC metric can take values between 1.7 and 5.1. In AICC-based diagrams describing UML diagrams, the elements from which a given diagram can be built are used as language symbols, whereas η_1 means the number of groups of elements of the same type.

The Class Design Entropy metrics, like the AICC metrics, has also been proposed for estimating the source code. An identical formula was proposed for calculating the value of this metric as for the AICC metric, where, instead of all language symbols, only the names of identifiers appearing in the tested part of the software (class definitions), such as class name, variable, constant, class, and symbols characteristic for a given language, are included. On the other hand, symbols characteristic of a given language, such as keywords or operators, are omitted. The proposed CDE metrics includes the following parameters: N1 is the total number of occurrences of identifiers (non-unique) used in the definition of the tested class, and $\hat{f}_i$, where $1 \leq i \leq n_1$ is the number of occurrences of the i-th identifier in the definition of this class, n_1 is the number of group identifiers with the same name (unique identifier names) of the class definition being examined. The formula for calculating the CDE metrics, which is identical to (2), is given below, but the symbols used have different meanings.

$$\mathrm{CDE} = -\sum_{i=1}^{n_1} \frac{\hat{f}_i}{N_1} \log_2 \frac{\hat{f}_i}{N_1} \tag{3}$$

In the diagrams based on the above formula, describing UML diagrams, the name of the UML element of the analyzed diagram is usually taken as software code identifiers, whereas n_1 means the number of groups of UML elements having similar names.

The AICC and CDE metrics have been used in this work to demonstrate that the application of consistency rules, understood as linking elements of identical interpretation in any independent diagram, results in a decrease in its entropy, i.e., an increase in the information content of the model and an increase in its orderliness.

Proof of the decrease in entropy when applying consistency rules

Below is proof of the decrease in entropy when linking UML elements of different diagrams using identical names, i.e. it will be proved that $E_{indep} > E_{dep}$, where E_{indep} is the entropy of the configuration of independent diagrams, and E_{dep} is the entropy of the configuration of dependent diagrams using consistency rules. The formula for the AICC metrics described in (2) will be used, as well as the formula for the CDE metrics, given by (3).

Fig. 1 shows two configurations with two UML diagrams. On the left is a configuration with independent diagrams, and on the right a configuration with related diagrams. The configuration shown on the left has 4 occurrences of independent UML elements (two occurrences of UML Activity elements named "a" and "c", one occurrence of UML UseCase element named "d", and one occurrence of UML ControlFlow element without a name). Thus, according to (2), parameter N1 is the number of all occurrences of UML elements (4 UML elements), parameter η_1 is the number of all types of elements in the diagram (3 types of elements: UML Activity, UML UseCase, UML ControlFlow), and the number of occurrences of individual UML elements is as follows: $f_{UML\ Activity} = 2$ (2 elements of UML Activity); $f_{UML\ UseCase} = 1$ (1 element of UML UseCase); $f_{UML\ ControlFlow} = 1$ (1 element of UML ControlFlow).

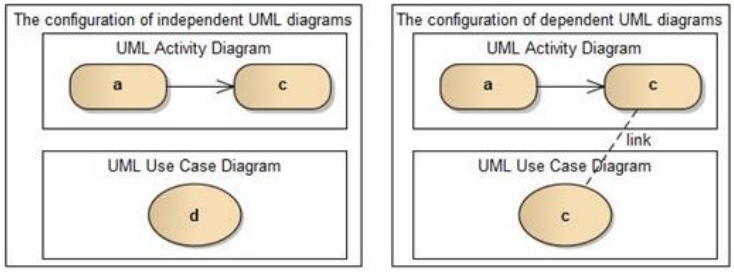

Fig. 1. Configuration with two UML diagrams with a minimum number of elements.

On the other hand, the configuration shown on the right has 2 independent elements (UML Activity element named "a" and an unnamed UML ControlFlow connection) and 1 dependent element (element named "c" appearing as UML Activity element in the top diagram and the same element occurring as UML UseCase in the bottom diagram). The connection drawn with a dashed line and named "link" is not part of the configuration but indicates the consistency rule. Thus, according to (3), parameter N1 means the number of all UML elements (4 non-unique identifiers, i.e., all UML elements), parameter η_1 is the number of all elements on the diagram having different names (3 unique identifiers: "a", "c", and an unnamed identifier). Moreover, the number of occurrences of individual UML elements with different names is as follows: $\hat{f}_a = 1$ (1 UML element with the identifier "a"); $\hat{f}_c = 2$ (2 UML elements with the identifier "c"); $\hat{f}_{unnamed} = 1$ (1 UML element with id unnamed).

It is worth noting that unrelated UML elements occur only once in a diagram system, whereas UML elements related in a given configuration occur many times in the form of various UML elements (e.g. the UML element named "c" occurs once as part of the UML Activity, and for the second time as UML UseCase element).

Assuming that the entropy E_{indep} for an unrelated configuration result from the formula for AICC, and is given by the formula:

$$\mathrm{E_{indep}} = \mathrm{AICC} = -\sum_{i=1}^{\eta_1} \frac{f_i}{N} \log_2 \frac{f_i}{N}, N \in \mathbb{N}^+, f_i \in \mathbb{N}^+ \tag{4}$$

where the symbol N denotes the number of all elements of the configuration of independent (unrelated) diagrams, the symbol η_1 denotes the number of occurrences of elements of type i, and f_i is the number of occurrences of elements of the i-th type i = 0,1,2 ...

On the other hand, entropy E_{dep} for the related same configuration results from the formula for CDE and is given by the following formula:

$$\mathrm{E_{dep}} = \mathrm{CDE} = -\sum_{i=1}^{\eta_2} \frac{\hat{f}_i}{N} \log_2 \frac{\hat{f}_i}{N}, N \in \mathbb{N}^+, \hat{f}_i \in \mathbb{N}^+ \tag{5}$$

The symbol N denotes the number of all configuration elements of related diagrams, the symbol η_2 represents the number of occurrences of elements with the given name i, and $\hat{f}_i$ the number of occurrences of the i-th element with the given name i = 0, 1, 2 ...

For the configuration of independent diagrams, there is equality: $f_1 = f_2 = \cdots = f_{\eta_1} = 1$. By inequalities $0 \leq id \leq \eta_2$ let us indicate the number of elements (in the configuration of related diagrams) such that $\hat{f}_k = 1$. Without loss of generality, we can assume that:

$$\hat{f}_1 = \hat{f}_2 = \cdots = \hat{f}_{id} = 1 \tag{6}$$

In addition, equality is satisfied:

$$\sum_{i=1}^{\eta_1} f_i = \sum_{i=1}^{\eta_2} \hat{f}_i = N \tag{7}$$

and inequality $\eta_1 > \eta_2$. Thus:

$$\mathrm{E_{indep}} - \mathrm{E_{dep}} = -\sum_{i=1}^{\eta_1} \frac{f_i}{N} \log_2 \frac{f_i}{N} + \sum_{i=1}^{\eta_2} \frac{\hat{f}_i}{N} \log_2 \frac{\hat{f}_i}{N} =$$

$$-\sum_{i=1}^{id}\frac{f_i}{N}\log_2\frac{f_i}{N}-\sum_{j=id+1}^{\eta_1}\frac{f_j}{N}\log_2\frac{f_j}{N}+\sum_{i=1}^{id}\frac{\hat{f}_i}{N}\log_2\frac{\hat{f}_i}{N}+\sum_{j=id+1}^{\eta_2}\frac{\hat{f}_j}{N}\log_2\frac{\hat{f}_j}{N}=$$
$$-\sum_{i=1}^{id}\frac{1}{N}\log_2\frac{1}{N}-\sum_{i=id+1}^{\eta_1}\frac{1}{N}\log_2\frac{1}{N}+\sum_{i=1}^{id}\frac{1}{N}\log_2\frac{1}{N}+\sum_{j=id+1}^{\eta_2}\frac{\hat{f}_j}{N}\log_2\frac{\hat{f}_j}{N}=$$
$$-\sum_{i=id+1}^{\eta_1}\frac{1}{N}\log_2\frac{1}{N}+\sum_{j=id+1}^{\eta_2}\frac{\hat{f}_j}{N}\log_2\frac{\hat{f}_j}{N} \tag{8}$$

Equations (6) and (7) show that:

$$\sum_{j=id+1}^{\eta_2}\hat{f}_j=\eta_1-id \tag{9}$$

Thus, the above equality can be written as:

$$\sum_{j=id+1}^{\eta_2}\left(\frac{\hat{f}_j}{N}\log_2\frac{\hat{f}_j}{N}-\frac{\hat{f}_j}{N}\log_2\frac{1}{N}\right)=\sum_{j=id+1}^{\eta_2}\frac{\hat{f}_j}{N}\left(\log_2\frac{\hat{f}_j}{N}-\log_2\frac{1}{N}\right) \tag{10}$$

Because $\forall j\in\{id+1,\cdots,\eta_2\}$ then the inequality $\hat{f_j}>1$ occurs, and from the monotonicity of the function $\log_2 x$ we get:

$$\sum_{j=id+1}^{\eta_2}\frac{\hat{f}_j}{N}\left(\log_2\frac{\hat{f}_j}{N}-\log_2\frac{1}{N}\right)>0 \tag{11}$$

What ends the proof that $E_{indep} > E_{dep}$.

Conclusion

The above evidence shows that, for the configuration of diagrams with consistency rules, created by assigning similar names to diagram elements, the system's entropy decreases compared to the configuration with the same number of elements but without consistency rules. Thus, the introduction of consistency rules (assigning similar names to corresponding elements) reduces information uncertainty and consequently increases the overall orderliness and consistency of the model. Typically, this method of building software architecture is performed by experienced IT designers. Unfortunately, there is a lack of appropriate tools to support these practices when building software for large or complex IT systems. Furthermore, there is often a lack of awareness and conviction in the explicit application of consistency rules in software development. The evidence presented demonstrates the significant benefits of using consistency rules in software architecture development.

Currently, the number of articles devoted to the application of consistency rules in building software architecture is practically negligible. It also appears that the use of software architecture is not very important in most IT system projects, despite IT experts' emphasis on the importance of consistency rules in software architecture design. However, in the face of increasingly higher quality requirements for software, it seems that without IT designers paying attention to the application of consistency principles in software development, the current stagnation in software development methods is unlikely to be significantly improved.

References

- Allaki, D., Dahchour, M. and En-Nouaary, A., (2015), '*A new taxonomy of inconsistencies in UML models with their detection methods for better MDE,*" International Journal of Computer Science and Applications, Vol. 12 (No. 1), pp. 48–65.
- Bansiya, J., Davis, C. and Etzkorn, L., (1999), '*An entropy-based complexity measure for object-oriented designs,*" Theory and Practice of Object Systems, Volume 5, Issue 2, pp. 111–118.
- Ha, I., Kang, B., (2008), '*Cross Checking Rules to Improve Consistency between UML Static Diagram and Dynamic Diagram,*' International Conference on Intelligent Data Engineering and Automated Learning - IDEAL 2008, LNCS, vol. 5326, pp. 436–443.
- Hagel, N., Hili, N., Bartel, A. and Koziolek, A., (2025), '*Towards LLM-Powered Consistency in Model-Based Low-Code Platforms,*' 2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C), pp. 364–369.
- Harrison, W., (1992), '*An Entropy-Based Measure of Software Complexity,*" IEEE Transactions on Software Engineering, Volume 18, Issue 11.
- Klare, H., Kramer, ME., Langhammer, M., Werle, D., Burger, E. and Reussner, R., (2021), '*Enabling consistency in view-based system development — The Vitruvius approach,*' J. Syst. Softw., vol. 171, p. 110815.
- Marchezan, L., Homolka, M., Blokhin, A., Assunção, WKG., Herac, E. and Egyed, A., (2024), '*A Tool for Collaborative Consistency Checking During Modeling,*' ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, 2024, pp. 655–659.
- Marchezan L., Kretschmer, R., Assunção, WKG., Reder, A. and Egyed, A. (2023), '*Generating repairs for inconsistent models,'* Softw. Syst. Model., vol. 22, no. 1, pp. 297–329.
- Object Management Group, (2015). '*Unified Modeling Language*,' Object Management Group. [Online], [Retrieved December 10, 2025], https://www.omg.org/spec/UML/.
- Sapna, PG., Mohanty, H., (2007), '*Ensuring Consistency in Relational Repository of UML Models,'* 10th International Conference on Information Technology, pp. 217-222.
- Shinkawa, Y., (2006), '*Inter-Model Consistency in UML Based on CPN Formalism,'* 13th Asia Pacific Software Engineering Conference, pp. 414-418.
- Spanoudakis, G. and Zisman, A., (2001), '*Inconsistency management in software engineering: survey and open research issues,*' Handbook of Software Engineering and Knowledge Engineering, World Scientific Publishing Co., Singapore, pp. 329-380.
- Straeten, R., (2005), '*Inconsistency Management in Model-Driven Engineering: An Approach Using Description Logics,'* Ph.D. dissertation, Vrije Universiteit Brussel, Belgium.
- Thims, L., (2012), '*Thermodynamics ≠ Information Theory: Science's Greatest Sokal Affair,*' Journal of Human Thermodynamics, 8(1), pp. 1-120.
- Torre, D., Genero, M., Labiche, Y. and Elaasar, M., (2023), '*How consistency is handled in model-driven software engineering and UML: an expert opinion survey,*' *Softw. Qual. J.*, vol. 31, no. 1, pp. 1–54.
- Torre, D., Labiche, Y. and Genero, M., (2014), '*UML consistency rules: a systematic mapping study,*" 18th International Conference on Evaluation and Assessment in Software Engineering.
- Torre, D., Labiche, Y., Genero, M. and Elaasar, M., (2015), '*A systematic identification of consistency rules for UML diagrams,*" Carleton University, Ottawa.