Understanding Evolution of Information Systems by Applying the General Definition of Information

Rikard Land
Mälardalen University
Department of Computer Science and Engineering
PO Box 883, SE-721 23 Västerås, Sweden
+46 21 10 70 35
rikard.land@mdh.se
http://www.idt.mdh.se/~rld

Abstract. *Information Systems are continuously evolved for a very long time. Problems with evolving such systems stem from insufficient or outdated documentation, people no longer being available, different (often old) hardware and software technologies being interconnected, and short-term solutions becoming permanent. Crucial for successful evolution of Information Systems is to understand the data and information of the system.*

This paper argues that some of the fundamental concepts and consequences of the General Definition of Information (GDI), presented by the field of Philosophy of Information, can be a complement to approaches such as "data mining" and "data reverse engineering". By applying GDI it becomes possible for the maintainers of Information Systems to ask important questions about the system that can guide the work in a pragmatic way. GDI can become a useful tool that improves the evolution process.

Keywords. General Definition of Information, Information Systems, Legacy Systems, Philosophy of Information.

1 Introduction

Information Systems are systems whose ultimate purpose is to store and manage information (as opposed to e.g. control systems, which are designed to control physical processes, and manages information only as a means to achieve this). Information systems are long-lived and have to incorporate changing requirements, and thus evolve, often over decades. They become *legacy systems*, with problems such as:

- Documentation is lacking or out of date.
- Very few people have overview over the whole system.
- Different technologies from different eras are mixed.

Still, there is no practical option to start over from scratch. Although there are problems with the system, it represents an enormous effort invested in requirements engineering, designing, implementation, testing, debugging, tuning etc. Evolution and maintenance is therefore often carried out through improving the most urgent parts. To understand requirements and design the system is reverse engineered [1]; to get rid of the oldest and most problematic technologies these parts are ported or migrated [3].

Philosophy of Information is concerned with studying information at the most fundamental level [6]. We will start from the General Definition of Information (GDI) and pursue some of its consequences.

Our contribution is a demonstration of how the GDI can be applied to IS problems to guide some of the activities involved in their evolution.

2 General Definition of Information

Asking what information "is" is probably futile – it is what we define it to be. Dictionary definitions of information typically describe information in terms of *communication*, *data*, *message*, *facts*, *knowledge*, *interpretation*, and *understanding*[1]. According to Floridi, "many analyses have converged on a General Definition of Information (GDI) as a semantic content in terms of *data + meaning*" [6]. Information is, according to GDI, *meaningful*, *well-formed data* – if either the meaningfulness, the well-formedness, or the data is

[1] See e.g. *Merriam-Webster Online*, URL: http://www.m-w.com/cgi-bin/dictionary, *Web WordNet*, URL: http://www.cogsci.princeton.edu/cgi-bin/webwn, *Principia Cybernetica Web*, URL: http://pespmcl.vub.ac.be, 2004-02-18.

lacking, we cannot talk about a piece of information.

GDI leaves a number of aspects of data and information open and does not take a certain standpoint, i.e. GDI is *neutral* with respect to these issues. The openness of these "neutralities" provide a good basis for the analysis and discussion of this paper:

- **Typological Neutrality.** Although information cannot be dataless, GDI does not specify *which types of data constitute information*. Of particular interest is the question whether lack of information means negative information. An example would be whether no answer to the question "how many database entries" should be interpreted as "no entries found", or for example "still processing the query", "stuck in a loop", etc. GDI does not choose an interpretation. Choosing the correct interpretation requires additional information about the data (e.g. meta-data).
- **Taxonomic Neutrality.** Every piece of data is a relational entity. This means that nothing can be regarded as data or information in isolation, but in relation to something else. As an example, a black dot is a black dot only in relation to its background (and the background is a background only in relation to something else). But GDI does not by itself identify either the dot or the background as data, it is neutral with respect to *how data and its relata are identified*.
- **Ontological Neutrality.** GDI rejects the possibility of dataless information: no information without data representation. This has been interpreted as "no information without physical or material implementation" [12], which sounds intuitive when working in the field of computers. Others have interpreted this in another way: that the universe itself at the deepest level is made of information (not matter or energy), exemplified by the phrase "it from bit" [19]. GDI is neutral, however, to the *choice of representation*.
- **Genetic Neutrality.** According to GDI, a piece of information can have semantics not only in the mind of an informee, but also *independent of any informee*.
- **Alethic Neutrality.** According to GDI, information consists of meaningful and well-formed data, independent of whether it is true or false (or contains no truth value at all). That is, GDI does not discuss the *truthfulness of data* (alethic value). This poses a problem for the GDI, and some solutions have been proposed [6]. For our purposes it is enough though to just highlight the issue of truth.

3 Problems Encountered in IS Evolution

For the purpose of this paper, some of the most striking features of an Information System in terms of *information* will be listed. We have chosen to consider the information produced in three development phases, present in all development projects [13,15]. The *requirements* on the system are information. The *design* of the system can be seen as something separate from the implementation – a model of the implementation. For example, the system's architecture, different types of conceptual diagrams are information. As Information Systems are typically divided into (persistent) data and program(s) working on this data, the discussion will consider the design of the data storage separately from the design of the programs. The *implementation* can also be considered information, which again is divided into the implementation of data storage and the programs manipulating data (the programs can be said to embody (part of) the semantics of the stored data).

In line with the taxonomic neutrality of GDI (see above) the relation between each of the above will also be considered as information. This paper will investigate four relations (see Figure 1), because they seem the most natural starting point. First, the relation between *requirements and data design*. That is, why has the data been designed as it is? Second, the relation between *requirements and program design*. That is, why has the programs been designed as they are? Third, the relation between *data design and data implementation*, and fourth, the relation between *program design and program implementation*.

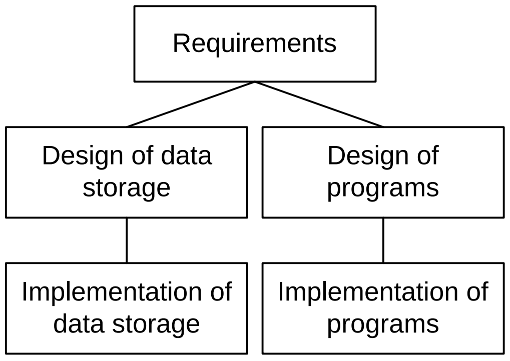

Figure 1: The information considered.

In addition to these relations, one could consider all possible relations between requirements, design, and implementation (i.e. all possible arcs between the boxes in the figure). The relation between requirements and implementation will be touched on, but this is otherwise left as future work.

Before continuing, some notes on the terminology used in the rest of this paper. “Information artifacts” denotes the information produced and the relations between them (that is, the five boxes and the four lines in the drawing). “Developers” denote the people that developed and maintained the system previously. It could be understood as “producers of information artifacts” and includes not only software engineers but also people involved in requirements elicitation. “Maintainers” are the people carrying out the current maintenance and evolution activities, i.e. “consumers of information artifacts”.

4 Addressing the Problems

Each of the neutralities will now be applied to the IS problems, one at a time.

4.1 Typological Neutrality

Applied to Information Systems, the *typological neutrality* helps us choosing an interpretation when no information is found. We have chosen the following interpretation: when no information is found, this does not mean that there is no information, but only that we cannot find it.

The opposite seems unlikely. If no information is found, it seems impossible that the information system was built without some notion of requirements and a design (the implementation is of course there, otherwise there is no information system to talk about). There was arguably originally some information, at least in the heads of the developers. This instead leads to the conclusion that the apparent lack of information means that information has been lost. Either it only existed in the heads of the developers, or if it was documented the development organization lacks a document archive. This could lead to either (or both) of the following actions:

- The information should be searched for (if there is the least possibility to find the original developers or old documents).
- An information artifact can be reverse engineered to reconstruct the lacking information. That is, a binary executable can be decompiled, an implementation can be analyzed (even automatically) to produce a higher-level description (i.e. design), and the design artifacts can be analyzed to understand the original requirements on the system.

In addition to this, the organization should learn its lesson and improve its documentation practices, to avoid the same situation in the future.

4.2 Taxonomic Neutrality

The *taxonomic neutrality* means that GDI does not by itself identify a piece of data in relation to something else. In the context of Information Systems, the identification of information as a contrast to something else is very much fixed. For example, the characters constituting the text of documents or the symbols constituting diagrams, as contrasted to the background of paper, has been defined elsewhere and is only *used*. The same is true for the implementation: a language is used with a fixed syntax, which builds on sequences of characters – or if we like, as sequences of bits. But we can widen the question and ask what the information at hand can be contrasted with, in the sense “what is not there but could have been?” And the next question must be “why?”

Of course, one cannot document everything. As a basis, someone writing documentation assumes that the readers will know the language used (e.g. English or UML). But somewhere there is a borderline between what can be assumed, and where it is possible that the reader will misunderstand the intention of the writer. Some terms may be specific to a particular technical domain, or are used in a specific sense in a particular system. Some requirements may have been considered so fundamental that they are never documented as requirements.

Another reason some information is lacking, in the eyes of the maintainer, may be that the documentation practices at the time (or in the company culture) documentation was prepared was different from today’s. For example, good documentation practices for architectural descriptions [5,9] (and the very notion of software architecture) is recent, and older systems’ architectures may have only been documented very rudimentary [11].

The discussion so far concerns documentation, that is: requirements and design of data storage and programs. For implementation of data storage and programs, matters are different. The information produced could hardly be different (without being erroneous). Perhaps this is because the information in this case is so-called “instructional information” [7], i.e. directions to make something happen, for example a recipe or a sequence of instructions to be executed by a CPU.

4.3 Ontological Neutrality

The *ontological neutrality* states that the information can never be decoupled from its representation. This means that the information was once specified using some data representation. A consequence of this seemingly trivial observation is that the representation chosen possibly affects the actual content of the information. With this in mind, there are numerous questions that should be asked during evolution activities:

Requirements. How were requirements originally represented [10]? Only very informally in the heads of the developers? In a more formal document using natural language? Using some structured notation with natural language (such as a numbered list or a tree structure)? Using some formal language? The answer is itself a piece of information that should be utilized in subsequent evolution activities. Maybe the requirements documentation can be improved by translating it into some more formal form? How did the representation chosen affect the actual requirements – were the requirements focused on functionality or on extra-functional requirements (such as performance, data consistency or robustness)? Focused on data or on behavior?

Programs design. Which languages (textual and graphical) have been used? Flowcharts? Are there architectural descriptions [5]? Is the vocabulary of architectural patterns [4] or design patterns [8] used? The level at which design is made (high abstraction level, such as architecture, or lower level, such as the one modeled with flowcharts) is reflected and affected by the choice of language. This information can be used as a starting point to infer information about the design itself. In case of natural language, do some terms have a specific meaning? Can the choice of language(s) give some clues about the design choices made? The choice of language is partially colored by its popularity at the time the design was made (which may be decades ago) and would not necessarily be the best choice today.

Data design. The same reasoning as for design of programs can be applied to data design, although the languages used would be different: there are e.g. UML [18], so-called "crowfoot" notation, and others.

Programs implementation. How is the implementation represented, i.e. what programming language or languages are used? The choice of language may reflect some conscious decisions based on the information itself, i.e. the program. Especially in newer systems, there are a variety of programming languages to choose from: assembler languages (may indicate a focus on performance), interface definition languages (indicate a component-based approach) [17], logic languages (indicate a rule-based approach) [16], web-based languages (indicates a client-server approach) [14], or procedural languages (may just indicate that a mainstream language was chosen).

Data implementation. The choice of language for implementing data is probably fixed once a database has been determined. Still, the same vendor may offer a variety of languages and technologies for data implementation, and the choice between these reveal some conscious decisions. Standard SQL [2] may have been chosen to achieve portability, or proprietary extensions may have been used, indicate that some vendor specific features were considered more important than portability. For example, non-standard SQL constructs (such as procedures stored in the database server) may have been chosen for performance, security, or data consistency reasons.

Please note that the implementation discussions concern not the implementation on its own but the relation between requirements and implementation – how requirements may have been reflected in the choice of implementation language. Reasoning based on the ontological neutrality is thus one clue (among many) to reverse engineering aiming at understanding the original requirements, in this case mainly the extra-functional requirements. It may even be the case that the decision to use a specific language lays not so much in its technical characteristics as in its political consequences. One example would be choosing a language based on popularity, assuming it will be easy to attract skilled personnel in the future.

Requirements/Data design; Requirements/ Programs design. The relation between requirements and design may have been made explicit in some way. For example, design artifacts such as documents and diagrams may contain references to requirements (in the form of the requirements representation, e.g. the format used to number requirements). Although it is not sure the original developers put effort into making requirements traceable in the design, any clue found is valuable – and it is easier to find these references if they are searched for.

Data design/Data implementation; Programs design/Programs implementation. Source code can easily be searched to find strings used in the design, such as names of higher-level abstractions. Of course, the design description may also explicitly include names of items (such as database tables, column names, interfaces, or classes). Such strings originating from the design could be found in program code (variables, function names etc.) or possibly in comments.

4.4 Genetic Neutrality

According to the *genetic neutrality* of GDI, a piece of information can have semantics independent of any informee. Applied to our discussion, this means that although a maintainer may not understand the information (e.g. the requirements, the design etc.) this does not mean that the information does not have a specific meaning. This may sound as a repetition of the typological neutrality, but there is a difference. The typological neutrality forced us to ask questions about what lack of data mean; the genetic neutrality force us to ask to what extent we understand the data as intended. If the current maintainers do not understand e.g. original design diagrams, these may still have been written in a specific language that were understood at the time of writing.

The genetic neutrality does not, however, state that seemingly unclear texts or diagrams *must* have a meaning that can be discovered if we know the language used. Even if the language used is well known (e.g. English or UML), the information under scrutiny may not conform fully to the language. And it is not uncommon that diagrams are created using an ad-hoc notation with boxes and arrows, without providing a key. A seemingly vague requirement may indeed be vague, even if we know the full semantics of the language used.

Perhaps the genetic neutrality is most useful if interpreted as a procedure: as a maintainer, one should first embrace the attitude that there is information to be retrieved even if it is not understood at once. The language used can provide a key, and to understand the information, one may have to learn the language. This language may be a particular use of natural language (which can be at least partially learned by scrutinizing other documents) or a particular graphical notation (standardized or more ad hoc). The original author, if available, is of course a key person to explain the language used.

4.5 Alethic Neutrality

The *alethic neutrality* highlights the issue of truth: is a certain piece of information true? It seems unlikely that any of the information artifacts would be untruthful on purpose. But there are situations when documents are not trustworthy, which need to be taken into account when using the information contained therein as a basis for evolution activities. Mapped to our information artifacts, it seems unnatural to call requirements, design, implementation, or data untrustworthy by themselves – only when related to each other can they become untrustworthy. This may mean:

Requirements/Programs design; Requirements/Data design. The design might have been insufficient to fulfill the requirements, and therefore the design may be said to be untrustworthy with respect to the requirements. This is particularly common for extra-functional requirements, which are often not analyzed before the system is built (and after it is built it is too late to change the design to fulfill these requirements).

Programs design/Programs implementation; Data design/Data implementation. The implementation may have evolved while the documentation has not. Or the opposite, the implementation never implemented the design fully (due to e.g. time restrictions, which also would explain why the design document was not revised).

As a general principle for software maintainers, the alethic neutrality therefore gives by hand that the information at hand should be met with a sound amount of suspicion. The information should be checked against other information.

5 Discussion and Conclusion

The neutralities of the General Definition of Information have been applied to ten listed information artifacts present in Information Systems, with the aim of discovering important issues to consider while evolving these systems. Most of what have been found is not new; it might rather be seen as old discussions with a new terminology. But there are some things to learn:

- The content and the form of the information are not completely decoupled. So a clue to understand the information is to consider the representation chosen to embed the information: natural language, ad-hoc graphical notations, formal languages, etc. The information reflects the chosen language's strengths and weaknesses.
- Studying the representation used, i.e. what programming languages and design languages were used, may reveal what considerations were important at the time when the information was produced. In particular, this may give clues to the original extra-functional requirements (such as performance or robustness), even if they were not explicit.
- There was once information, even if it is not understood now. Documents and diagrams were written in a language that was understandable for the developer even if they are not clear for the maintainer. Some terms may be left undefined because they were considered trivial by the developer. As a consequence, this attitude leads maintainers to actively search for lost information.

We believe that applying Philosophy of Information to research fields such as that of information system may give birth to new insights. This paper is a first attempt to do this, and there is much left to be done. Future work may include the following:

- We chose three artifacts, from three different development phases (requirements, design, implementation). There are other phases as well to include, e.g. testing. The design phase could be divided into high-level (architectural) and low-level design.

- The actual data stored within the system is also clearly information (hence the term "Information System"). How can the GDI help in understanding and managing this data?
- The other relations between the information listed could be investigated. We are particularly curious about investigating the traceability from implementation and stored data back to requirements.
- Another possible division would be based not on development phases but on architecture, which would typically in an Information System be user interface, business logic, and database.

6 References

[1] Aiken P. H., *Data Reverse Engineering : Slaying the Legacy Dragon*, ISBN 0-07-000748-9, McGraw Hill, 1996.

[2] Bowman J. S., Emerson S. L., and Darnovsky M., *The Practical SQL Handbook: Using SQL Variants* (4th edition), ISBN 0201703092, Pearson Educational, 2001.

[3] Brodie M. L. and Stonebraker M., *Migrating Legacy Systems: Gateways, Interfaces & the Incremental Approach*, Morgan Kaufmann Series in Data Management Systems, ISBN 1558603301, Morgan Kaufmann, 1995.

[4] Bushmann F., Meunier R., Rohnert H., Sommerlad P., and Stal M., *Pattern-Oriented Software Architecture - A System of Patterns*, ISBN 0-471-95869-7, John Wiley & Sons, 1996.

[5] Clements P., Bachmann F., Bass L., Garlan D., Ivers J., Little R., Nord R., and Stafford J., *Documenting Software Architectures: Views and Beyond*, ISBN 0-201-70372-6, Addison-Wesley, 2002.

[6] Floridi L., "Information", in Floridi L. (editor): *The Blackwell Guide to the Philosophy of Computing and Information*, ISBN 0-631-22919-1, Blackwell Publishing Ltd, 2004.

[7] Floridi L., "Information", in Mitcham C. (editor): *Encyclopedia of Science, Technology and Ethics*, Macmillan (forthcoming, see URL: http://www.wolfson.ox.ac.uk/~floridi/pdf/este.pdf), 2004.

[8] Gamma E., Helm R., Johnson R., and Vlissidies J., *Design Patterns - Elements of Reusable Object-Oriented Software*, ISBN 0-201-63361-2, Addison-Wesley, 1995.

[9] IEEE, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, ISBN 0-7381-2518-0, Institute of Electrical and Electronics Engineers, 2000.

[10] Kotonya G. and Sommerville I., *Requirements Engineering: Processes and Techniques*, ISBN 0471972088, John Wiley & Sons, 1998.

[11] Land R., "Applying the IEEE 1471-2000 Recommended Practice to a Software Integration Project", In *Proceedings of International Conference on Software Engineering Research and Practice (SERP'03)*, CSREA Press, 2003.

[12] Landauer R., "Information is Physical", In *Physics Today*, volume 44, pp. 23-29, 1991.

[13] Pfleeger S. L., *Software Engineering, Theory and Practice* (International Edition edition), ISBN 0-13-081272-2, Prentice-Hall, Inc., 1998.

[14] Powell T. A., *Web Design Complete Reference* (2nd edition), ISBN 0072224428, McGraw-Hill Osborne Media, 2002.

[15] Sommerville I., *Software Engineering*, ISBN 0-201-39815-X, Addison-Wesley, 2001.

[16] Sterling L. and Shapiro E., *The Art of Prolog* (2nd edition), ISBN 0262193388, MIT Press, 1994.

[17] Szyperski C., *Component Software - Beyond Object-Oriented Programming* (2nd edition), ISBN 0-201-74572-0, Addison-Wesley, 2002.

[18] UML, *UML Home Page*, URL: http://www.uml.org/, 2003.

[19] Wheeler J.A., "Information, physics, quantum: the search for links.", in Zureck W. (editor): *Complexity, Entropy, and the Physics of Information.*, Addison Wesley, 1990.