The Potential for Synergy between Information Visualization and Software Engineering Visualization

Orlena C.Z. Gotel[1], Francis T. Marchese[1], Stephen J. Morris[2]
[1]Pace University, New York, USA, [2]City University, London, UK
{ogotel@pace.edu, fmarchese@pace.edu, sjm@soi.city.ac.uk}

Abstract

To be provocative, it could be argued that information visualization is a tool in search of an application. This viewpoint becomes most apparent when one seeks to adopt and adapt practices from the information visualization field and attempt to apply them elsewhere. Software engineering is an appealing area in which a number of researchers have been seeking to leverage some of the benefits that information visualization can bring. Through an examination of the two fields, and their underlying motivations and foci, we highlight an as yet untapped area in which future research efforts should be directed to gain the most impact in software engineering. We also highlight recent concerns from the information visualization field to emphasize the role of establishing criteria through which new contributions can be assessed.

Keywords--- **Information Visualization, Requirements Engineering, Software Engineering Visualization**.

1. Introduction

Both 'information visualization' and 'software engineering visualization' are data intensive activities but their motivations are markedly different. In software engineering visualization, associated as it is with a primary design function, the fixing and communication of structure are paramount, whereas in information visualization the revealing and understanding of structure are the principal concern. In some parts of the overall software development process, in particular the 'software visualization' of implemented code, the understanding of hidden structures is important, as it may also become in the as yet unrealized field of 'requirements engineering visualization'.

Given this basic disparity in current motivations, it is difficult to formulate any type of overarching framework for both fields, although a semiotic view may provide some insights. Each field also exhibits unique problems, in software engineering visualization the semantic and syntactic complexities of defining a universally applicable form of structuring for software systems via the Unified Modeling Language (UML), and in information visualization the need to discover or create appropriate metaphors to guide the structuring of new data sets. Both fields share the need for standards of effectiveness though, whether based purely pragmatically on survival and broad use, or more theoretically on visual language definition or cognition. In this paper, we suggest that the most fruitful area for future development is the emerging field of requirements engineering visualization, and we draw upon information visualization efforts to provide a basis upon which to develop and assess the effectiveness of contributions.

This paper is therefore a first attempt to define the overlap of the two fields of software engineering and information visualization and to look for where the opportunities lie for a more fruitful marriage of ideas. In sections 2 and 3, the paper examines the contrasting motivations behind software engineering visualization and information visualization, and explains those areas in which commonality or overlap occurs. This highlights the potential for activity in requirements engineering. Section 4 suggests a framework within which to examine the state of the art and gives possible reasons for the predominantly textual focus in requirements engineering to date. Section 5 lists problems unique to each field and outlines a common problem area that demands attention as research moves forward. Specifically, establishing criteria through which the effectiveness of visual contributions should be examined.

2. Visualization in Software Engineering

The wide scope of what might be called visualization in the domain of software engineering suggests that broad, vernacular definitions would be appropriate. Standard definitions, *"the action or fact of visualizing; the power or process of forming a mental picture or vision of something not actually present to the sight; a picture thus formed"* [26], place emphasis both on the cognitive activities and their products. In software engineering, visualization as an activity (or non-textually based process) and visualizations as artifacts (or process products) have played fundamental roles from the earliest days of automated computing [11] and since the advent of the first software engineering notation [23].

Over the past two decades, the focus of the visualization efforts associated with software engineering has been in two main areas. Firstly, much attention has been paid to the development of visual notations and techniques for defining and communicating the understanding of a problem, its requirements and possible designs. The dominant approaches today are fully described in most of the popular software engineering texts (e.g., [7, 33]). The demand for shared conventions has ultimately led to the UML [25]. The first goal of visualization in software engineering is clearly to fix and communicate structures so as to facilitate development (visualization as artifact). Secondly, there has been recent interest in the creation and use of innovative visualizations to assist with the downstream activities of algorithm and program analysis, testing and debugging, giving rise to the term 'software visualization' [1, 17, 36]. The second goal of visualization in software engineering is, by contrast, to reveal and understand hidden structures (visualization as activity).

Software engineering is a discipline much of which now revolves around the creation and use of models. These models describe stakeholder problems and needs, at varying levels of abstraction, from loose requirements statements through to the concrete programming code written to satisfy these requirements. The most data-intensive and media-rich aspects of software engineering are clearly those early requirements engineering activities in which stakeholders are determined, problems explored and goals defined, so the period in which informal aspirations converge to an agreed statement of the problem and requirements specification. Despite the emphasis on information-seeking behaviors and knowledge creation amongst the multiple parties involved pre-requirements specification, and amongst the multiple activities which can take place as conflicts arise, negotiation occurs and decisions get taken, the exploitation of any techniques or ideas from the field of information visualization to support such activities is rare.

There are long sustained views that requirements engineering is the software development phase in which the most errors are introduced and that it is the cheapest phase in which to fix errors [3]. Coupled with its role contributing to ongoing software development project failure [35], it is surprising that visualization techniques which could potentially mitigate the introduction of misconceptions, especially when formulating and communicating requirements, are strikingly absent from mainstream requirements engineering practice and literature. This is evidenced in fifteen years of international conference proceedings in the area where the predominant visualizations are either associated with UML diagrams or i* goal models [15] and concerned with fixing conventions. Attention has only really been paid to this visual void over the past two years with the introduction of a new workshop series on requirements engineering visualization [16]. Papers from these workshops, in addition to furthering attention on visualizing requirements in UML models and on visual techniques to support goal-based requirements engineering approaches, have begun to focus on the challenges of visualizing:

- individual requirements and requirements collections to assess the health of the requirements;
- requirements relationships to support traceability;
- requirements variability to support decision making about product line requirements; and
- risks to requirements to support requirements-driven forms of risk assessment.

These initial efforts, however, have yet to capitalize upon the active role visualization can play in making sense of what can be complex and initially fragmented data. This is where further work is needed. Whether visualization can also assist with the very early lifecycle activities where these initial tentative data are first generated for future analysis also merits attention.

3. Overlapping Concerns

In contrast to software engineering visualization efforts, information visualization focuses almost entirely on the *act* of visualization as its primary goal and on using vision to think [4]. Where the objective of information visualization is to arouse consciousness and insight, the focus is on the transformation of data for easier assimilation by an individual's sense of sight, on the creation of a visual artifact. Also, the concern is with those mechanisms within humans and computers that allow for the perception, use and communication of sensory information and so facilitate the desired visual activity. For this field, visualization is primarily a cognitive activity [34, 38]. As such, information visualization draws upon many fields for its foundations, including: computer graphics, computer vision, computer science, human computer interaction, art and design, cognitive science and artificial intelligence. In computer-supported information visualization, complex data is mapped to perceptual representations in such a way as to maximize human understanding and communication, and to engender a deeper understanding of information, physical phenomena or the underlying processes related to them [38].

For Chen [5], information visualization is more concerned initially with methods for finding and extracting backbone structures from a complex set of information and subsequently with techniques for generating spatial layouts and graph drawing techniques. His examples all involve potentially large or very large unstructured data sets. The degree of semantic structure inherent in the UML, even if not defined in a fully formal manner, illustrates an important distinction between much information visualization as generally defined and the types of visualization characteristic of software engineering.

The structuring effort in software engineering has, in significant parts of the field, already gone into the preparation and agreement of conventions for visualization of important parts of its activities. If structure is, in general, something requiring visualization in software engineering design it is not because of the generally unstructured nature of data sets, but because of the broad value of conventionalized visualizations as communicative artifacts in the development process, or because there may be different structures derived from the same underlying model at some particular stage of the process or potentially as the process progresses from one stage to another.

Such structures are also an essential determinant of the data available, as a result of collection, analysis, design, implementation or execution. The nature of the structures and their visualizations are, in the cases of requirements collection and code execution, consequent representations of pre-existing antecedent, whereas with analysis, design and implementation it is likely, although not certain, that they will be precedent representations of subsequent objects. At its most simple, this is a distinction between the descriptive used for analysis and the prescriptive used for design.

During and after the implementation of software designs, increasing complexity has made it necessary to investigate aspects of this complexity, for example the dependencies between various components of a software system [8] and ‘software visualization’ for code artifacts has become a field in its own right requiring its own taxonomies [20, 28].

The overlaps between software engineering visualization and information visualization are depicted in Figure 1. They mostly occur where the exploration and understanding of hidden structures is critical, as is necessary for testing and maintaining implemented code (‘software visualization’) where structure may be obscured by complexity, and in requirements engineering where structure is yet to be formed. Representational and metaphorical techniques, the concerns of information and also knowledge visualization [9] activities, offer areas of promise for conveying and discovering such structures.

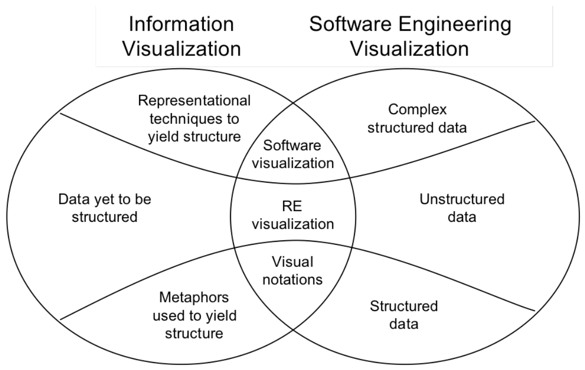
Information Visualization
Software Engineering Visualization
Representational techniques to yield structure
Complex structured data
Software visualization
Data yet to be structured
RE visualization
Unstructured data
Visual notations
Metaphors used to yield structure
Structured data

Figure 1. Overlapping Concerns and Possibilities

4. Conceptual Framework

The importance of these disparate roles of visualization as an activity and visualization as an artifact has been somewhat concealed in software engineering because of what some semioticians would call the ‘primary modeling systems’ being employed. In the semiotic discipline, all sign systems serve as a means of modeling, cognizing and explaining the world and particular cultures. Whether an explanation of all such systems should be based on the framework of natural language remains a contentious issue, but for the Moscow-Tartu School of semiotics [14] the ‘primary modeling system’ (PMS) is natural language, the proper object of linguistics.

Natural language serves as the universal meta-language for the interpretation of ‘secondary modeling systems’ (SMS) which are realized by correlation with the system of natural language and which use it as their material, whilst at the same time adding to it further structures. The classic early application of such structuring revealed a standard framework within Russian folk tales [29]. In general, any structured text serving a descriptive rhetorical function is definable as a SMS. In software engineering, texts structured as use case descriptions or scenarios are an important class of examples [19].

It does not violate the overall principle to posit a class of ‘tertiary modeling systems’ (TMS) that also depend on natural language as a meta-language of interpretation but employ exclusively non-textual components as the foundation for representation. A UML use case diagram is an example in software engineering, as would be any visualization of structured text used for a particular requirements engineering technique, for example a Volere requirements template visualization [13].

In this three-level hierarchy of modeling systems, text is at the top of the hierarchy and visualizations (in the vernacular sense) are at the bottom. This position may explain the relative lack of attention paid to visualizations in the earliest stages of software engineering when this agreement with respect language has yet to be established, one of the chief aims of requirements engineering being to agree upon language use.

It is clear, however, that natural language text does not remain the PMS throughout any comprehensive software engineering process. For proponents of ‘agile software development’ [2], ‘rapid or evolutionary prototyping’ [22] and related approaches, the PMS is the programming language itself, or some class of such languages. The dominant role of the programming language or paradigm as the PMS leaves little room for the creation of any special type of SMS, although secondary artifacts are used. In those standard software engineering processes and practices where the programming language is only the implicit PMS, as most importantly is the case in the use of object-oriented programming languages, the SMS has become the subject of continuous and intensive development culminating in the definition of the UML [25]. In the UML, visualizations of a software system, at all stages of its development, take the form of diagrammatic representations.

Before attention shifted to the object-oriented paradigm, diagrammatic representations as visualizations of software models had also played an essential part in ‘structured analysis and design’ and similar techniques [7, 39]. In this middle ground of software engineering, between upstream text-oriented requirements engineering and downstream implementation-oriented ‘software visualization’, the ‘model’ and the ‘modeling language’ have now themselves become a PMS in ‘model driven architecture’ [18], again without a specific SMS except perhaps for recognizable ‘patterns’ [10].

5. Towards Effective Visualizations

The two fields of software engineering visualization and information visualization focus on unique problem areas. The problems that are unique to software engineering visualization include the search for generalized solutions, hence visual language definition and advances with the UML. Despite their manifest function as visualizations of software systems at some stage of development, the visual characteristics of all software engineering notations have, since the early days of standardization in UML v0.9, been subordinated to the difficult problems of defining agreed semantics, ironically partly in highly complex natural language. As a consequence of very effective use of pragmatically defined visual notations, software engineering has seen no need, as yet, to exploit the refutation of the argument that logical soundness cannot be obtained in diagrammatical reasoning [31], nor advances in the formal specification of visual languages [21] and hence to deploy a visual language as a PMS.

However, the specific graphic forms used for all these visualizations do vitally affect their usefulness as communicative artifacts. According to Tversky [37], *“Effective visualizations conform to two principles:*

- *Congruity: structure and content of visualization should match structure and content of desired representation.*
- *Apprehension: structure and content of visualization should be readily perceived and comprehended.”*

In the former case, a UML representation of the underlying conceptual domain’s classes and their associations should appropriately reflect a system’s congruity. But the principle of apprehension remains problematic because most culturally based visualization theory has not addressed the practical problems regarding the use of visual graphical notations, most importantly how it is possible to guarantee reproducibility and uniformity of interpretation. In its oldest form, this is the problem of designing a notational system that will guarantee the integrity of a work of art such as a piece of music as it passes from score to performances and possibly from performance back to score, or from an architect’s drawing to a building, directly analogous to the problem of instantiating a software model in executable code.

The main work in this field [12], subtitled as ‘An approach to a theory of symbols’, defines a notational system and a set of criteria that must be met for there to be invariant relationships between separate, distinguishable symbols and separate, distinguishable elements that they denote. Application of this scheme and criteria has already provided useful insights into the UML [24].

When Goodman’s criteria fail, often because of the lack of any certain definition at all, representations simply become ‘sketches.’ Visualizations based on metaphors are one such example, which nevertheless remain valuable because the conventions for their use and the context in which they are used sufficiently mitigate ambiguity of meaning. Engineering drawings may have syntactic problems with individual component (character) differentiation, but the semantics are fixed by the delineation within the drawing plane of a separate space and its occupation by a table specifying types, identifiers, names, etc. [32].

Goodman’s tests do provide an initial means of analyzing notations and visualizations in general and of deciding if particular criteria are clearly satisfied. However, his definition of a notational system says little about the nature of syntactic or semantic structure and nothing about the nature of the morphisms associating them, or about levels of abstraction. This is a general problem in information visualization in which data of various degrees of abstraction, dimensions, degrees of freedom, and relatedness are correlated employing graphical means. In such situations, it may be useful to transform this data into another conceptual structure so as to make it more readily perceptible. This transformational process amounts to finding a metaphor that aptly represents and communicates the information to be perceived. Gotel, Marchese and Morris have put forward this approach in the area of requirements engineering [13], suggesting that the right metaphors could immediately make complex abstract data perceptible, thus fulfilling Tversky’s principle of apprehension.

Finally, one problem that is common to both fields of endeavor is agreement on a definition of effectiveness. The majority of visualization effects, particularly in software engineering, tend to lack any conclusive validation as to effectiveness in use. Other than a purely pragmatic measure of survival and use (as per the UML), what are the determinants of an effective visualization? Without clear criteria, it will be difficult to assess the value of what is likely to become a growing number of contributions in an under-explored field. This is an area open to research [6, 27, 30].

6. Conclusions

Software engineering is a field that is characterized, in its earliest stages, by the need to reconcile multiple viewpoints, fuse data from disparate sources and develop agreed models. Requirements engineering is that part of the software engineering discipline that has used visualization the least creatively and, we argue, the least successfully to date, and we suggest it is the area most open to future development and practical potential. The information visualization field is replete with exemplars that attempt to discover structure in complex unstructured data sets and this is where the synergy lies.

7. References

[1] *ACM Symposium on Software Visualization (SOFTVIS).* Conference Proceedings from 2003, 2005, 2006.

[2] Agile Software Development Alliance. *Manifesto for Agile Software Development.* Online at http://agilemanifesto.org/, February 2001.

[3] Boehm, B.W. and Papaccio, P.N. Understanding and Controlling Software Costs. *IEEE Transactions on Software Engineering*, Vol. 14, No. 10, pp.1462-1477, October 1988.

[4] Card, S.K., Mackinlay, J.D. and Shneiderman, B. *Readings in Information Visualization: Using Vision to Think.* Morgan Kaufmann, 1999.

[5] Chen, S. *Information Visualization: Beyond the Horizon*, 2nd edition. London: Springer-Verlag, 2004.

[6] Chen, C. and Czerwinski, M.P. Empirical evaluation of information visualizations: an introduction. *International Journal of Human-Computer Studies,* Vol. 53, No. 5 pp.631-635, November 2000.

[7] Davis, A.M. *Software Requirements: Analysis and Specification.* Prentice-Hall, Inc., 1990.

[8] Drew, N.S. and Hendley, R.J. Visualising Complex Interacting Systems. CHI 95 Conference Companion, ACM, pp.204-205, 1995.

[9] Eppler, M.J. and Burkhard, R.A. *Knowledge Visualization: Towards a New Discipline and its Fields of Application.* Paper #2/2004, July 2004-07-28, Version 2.5, Institute for Corporate Communication, Faculty of Communication Sciences, Universita delia Svizzera italiana.

[10] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns.* Reading, Mass.: Addison Wesley, 1995.

[11] Goldstine, H.H. and von Neumann, J. Planning and coding of problems for an electronic computing instrument, Part II, Volume 1. Report prepared for US Army Ordnance Department, April 1947. In A.H. Traub (Ed.), *John von Neumann, Collected Works Volume V, Design of computers, theory of automata and numerical analysis.* Pergamon Press, Oxford, pp 80-151, 1963.

[12] Goodman, N. *Languages of Art: An approach to a theory of symbols,* 2nd ed. Indianapolis, Ind.: Hackett, 1976.

[13] Gotel, O.C.Z., Marchese, F.T. and Morris, S.J. On Requirements Visualization. In *Proceedings of the Second International Workshop on Requirements Engineering Visualization (REV'07),* New Delhi, India: IEEE, 2007.

[14] Groden, M. and Kreiswirth, M. (Eds). Moscow-Tartu School. In *The Johns Hopkins Guide to Literary Theory and Criticism.* Baltimore: The Johns Hopkins University Press, 1997.

[15] *IEEE International Requirements Engineering Conference (RE).* Annual Conference Proceedings from 1994 through 2007.

[16] *IEEE International Workshop on Requirements Engineering Visualization (REV).* Workshop Proceedings from 2006, 2007.

[17] *IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT).* Conference Proceedings from 2002, 2003, 2005, 2007.

[18] Kleppe, A., Warmer, J. and Bast, W. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Professional, 2003.

[19] Maiden, N. CREWS-SAVRE: Scenarios for acquiring and validating requirements. *Automated Software Engineering Journal*, Vol. 11, No. 3, pp.183-192, 1998.

[20] Maletic, J.I., Marcus, A. and Collard, M.L. A Task Oriented View of Software Visualization. In *Proceedings of the First International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT'02)*, Los Alamitos, CA: IEEE, 2002.

[21] Marriott, K. and Meyer, B. (Eds.) *Visual Language Theory.* New York: Springer, 1998.

[22] McConnell, S. *Rapid Development: Taming Wild Software Schedules.* Microsoft Press, 1996.

[23] Morris, S. and Gotel, O. Flow Diagrams: Rise and Fall of the First Software Engineering Notation. In *Proceedings of the 4th International Conference on the Theory and Application of Diagrams (Diagrams'06)*, Stanford, USA, 2006 (Lecture Notes in Computer Science 4045, Springer-Verlag).

[24] Morris, S.J. and Spanoudakis, G. UML: An Evaluation of the Visual Syntax of the Language. In *Proceedings of the 34th Annual Hawaii International Conference on Systems Sciences (HICSS'01),* Los Alamitos, CA: IEEE Press, 2001.

[25] Object Management Group (OMG). *Unified Modeling Language,* Version 2.1.2. Online at http://www.omg.org/spec/UML/2.1.2/, November 2007.

[26] *Oxford English Dictionary (OED).* Online at http://www.oed.com, February 2008.

[27] Plaisant, C. The challenge of information visualization evaluation. In *Proceedings of the Working Conference on Advanced Visual interfaces* (Gallipoli, Italy, May 25 - 28, 2004). AVI '04. ACM, New York, NY, pp.109-116, 2004.

[28] Price, B.A., Baecker, R.M. and Small, I.S. A Principled Taxonomy of Software Visualization. *Journal of Visual Languages and Computing*, Vol. 4, No. 3, pp.211-266, 1993.

[29] Propp, V. *Morphology of the folktale*, translated by L. Scott, 2nd edition. Austin: University of Texas Press, 1968.

[30] Reilly, D.F. and Inkpen, K.M. White rooms and morphing don't mix: setting and the evaluation of visualization techniques. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (San Jose, California, USA, April 28 - May 03, 2007). CHI '07. ACM, New York, NY, pp.111-120, 2007.

[31] Shin, S.-J. *The Logical Status of Diagrams*. Cambridge: CUP, 1994.

[32] Simmons, C.H. and Maguire, D.E. *Manual of Engineering Drawing to British and International Standards*, 2nd edition. Butterworth-Heinemann, 2004.

[33] Sommerville, I. *Software Engineering*, 8th edition. Addison Wesley, 2006.

[34] Spence, R. *Information Visualization*. ACM Press, 2001.

[35] The Standish Group. *CHAOS Chronicles III*. Online at http://www.standishgroup.com/chaos/toc.php, 2003.

[36] Stasko, J., Domingue, J., Brown, M.H. and Price, B.A. *Software Visualization: Programming as a Multimedia Experience*. Cambridge, Mass.: MIT Press, 1998.

[37] Tversky, B., Morrison, J.B. and Betrancourt, M. Animation: Can it facilitate? *International Journal of Human Computer Systems*, Vol. 57, pp. 247-262, 2002.

[38] Ware, C. *Information Visualization: Perception for Design*, 2nd edition. Morgan Kaufmann, 2004.

[39] Yourdon, E. *Modern Structured Analysis*. Englewood Cliffs, N.J.: Yourdon Press, 1989.