Evaluation of Visual structure for Industrial size Software Product Line Architecture

Abeer Khalid, Salma Imtiaz
Department of Software engineering
International Islamic university
Islamabad Pakistan
abeer.msse234@iiu.edu.pk, salma.imtiaz@iiu.edu.pk

***Abstract*-Information visualization helps in facilitating comprehension of conceptual information. Information visualization plays a powerful role in software product line architecture. Much work has been done for variability representation, but little consideration has been given to scalability, visualization of traceability links, evolution of variance, etc. There is a greater need for a suitable visual structure that can illustrate "industrial sized" software product line architecture in managing these highlighted factors. There is also a need to perform an evaluation of the visual structures used for visualization of product line architecture, finding out their suitability for visualization of a large and complex software product lines. Our results show that hyperbolic trees are best suited for visualization of product line architecture due to their multiple attributes such as: Exponential growth, scalability, interaction without hindering the structure and clearer (un-complex) perception.**

Keywords-software product line architecture; information visualization; visual structure

I. INTRODUCTION

A software product line is a cluster of software intensive systems, "sharing a common", administered set of features, resulting from the "core assets", in an agreed upon way [2]. Software product line is hierarchal in nature, with intense complexity immersed in it, with increase in size, to industrial level; the intense complexity has increased exponentially.

The focus, on information visualization has increased, during the last decade; due to advanced information processing technology. Its essence is to augment "cognition" with the help of interacted visual illustrations [4], keeping in mind that "visual display provides the highest bandwidth channel from the computer to the human mind" [9].

The representation of software product line architecture in visualized form promotes understanding variation and reduces complexity of the data [1].

The incentive behind representation of software product line data in visual form is to provide an increased perception of data with highest interaction capability.

The benefit of mapping of software product line data to visual structure is best consummated if the visual structure preserves the specified data. There is a need for performing an evaluation of visual structure to identify their strengths and weaknesses. Information visualization has not been able to make much progress in software product line area.

The focus of this paper is to evaluate the existing visual structures for software product line architecture. The scope of the evaluation includes general visual structure along with ones specifically made for software product line visualization. The evaluation criterion are Scope [5], Abstraction [32], Hierarchy [18], Traceability [14][15], Scalability [16], Evolution [17], General visual content [4][8] and Perception [4]. These criterion are carefully chosen from literature, keeping in mind the attributes of software product line architecture and the attributes necessary for good visual structure [4][5][8][14][15][16][17][18][32].

This paper is organized in seven sections: Section I gives a brief introduction to the concerned problem. Section II is concerned with the related work. Section III defines the significance of conducting this evaluation. Section IV defines the visual structures chosen for evaluation. Section V presents and defines the metrics used for evaluation purpose. Section VI presents the evaluation of the visual structure. Section VII presents the conclusion and direction for future work.

II. RELATED WORK

Current visualization techniques in the literature are presented with a different focus: software visualization techniques in general, code visualization techniques, tools and techniques for software architecture visualizations, and visualization of static characteristic of software.

Price et al. [5] proposed a principled systematic approach for categorization of characteristics to be visualized, stating six main categories (scope, context, form, method, interaction, effectiveness) leading to further subcategories, which are further extended to other categories. These categorizations, helped in construction, as well as selection, of visualization techniques, but they lacked in specifying the domains for visualization. In short, visualization techniques, specific to some specialized area, were not considered, but they were reflected upon as an individual entity.

Bassil and Keller [10] evaluated qualitatively and quantitatively, software visualization tools using Price's proposed framework. Maletic et al. [11] proposed a task oriented-approach wherein they incorporated Price's [5] proposed framework, taking into consideration development and maintenance tasks of large scale software's. Gallanger et al. [12] took Price's [5] and Maletic's [11] approaches and proposed their own, focusing on stakeholder perspectives for evaluation of SAVT (Software Architecture visualization tool).

Roman and Cox [6] and Storey et al. [7] emphasized the need for evaluation, and proposed their own taxonomies.

Caserta and Zendra [8] presented a literature review on the static features of visualization techniques and then assessed them on the basis of three points of evolution criteria "changes over versions", "relationship between components" and "evolving of metrics with releases" [8].

The focus and scope of all of the above mentioned papers is different from ours in the sense that, first, our focus is towards evaluating only visual structures. Second, evaluation is done exclusively for software product lines. The scope of this work only includes structures, which are common and most favorable for software product lines.

III. SIGNIFICANCE OF EVALUATION OF VISUAL STRUCTURE FOR SPL ARCHITECTURE

Evaluation of visual structure for software product line architecture has major implications. Its importance is gained from the fact that if visual structure is not aligned with the needs of the concerned data, all the patterns may not be visible; a lot of data may be obscured, and context may not be correctly perceived. In short, departure towards a debacle takes place, if the right representation mechanism for data presentation is not selected [3].

Architecture of the development software is a cause of major apprehension for the stakeholders, and a proper depiction of architecture is raised in priority, therefore, software product line architecture has increased the stakes even more. "By its very nature, architecture is a statement about what we expect to remain constant and what we admit may vary" [13]. This, in short, is the motto of software product line.

The visual structure has a major hold in the visualization of software product line architecture, in the sense, that if visual structure is not up to mark then as many visual interaction techniques are placed; it would never confer a complete image. As stated by Tufte [3], we considered, that "There are right ways and wrong ways to show data; there are displays that reveal the truth and displays that do not" [3]. For this complication to be overcome; there is a major need for evaluation of software product line architecture. Our contribution is on evaluation of visual structures, for the sole purpose of finding a formfitting structure for software product line architecture, which would depict the essence of SPL architecture, with minimum of visual manipulation or interaction techniques needed for stakeholder task completion.

IV. VISUAL STRUCTURES

We have chosen visual structures based on the fact that some of them are already being implemented in SPL and others are more suited for representing Software Product Line Data.

A. Matrix Tables (MT)
B. Cone Tree (CT)
C. Tree Maps (TM)
D. Conventional Trees (CNT)
E. UML Notations (UMLN)
F. Textual Form (TXF)
G. Use Case Map notations (UCM)
H. Hyperbolic Trees (HBT)

Matrix Tables (MT) are a form of representation mechanism used for illustration of SPL architecture data [20][21]. Advantage of this system of demonstration is that it plainly depicts the number of variables linked with an assortment of data [4]. MT are used in the context of hierarchal as well as network data. Additionally, "visualization of data tables is used for detection of data" [4].

Cone Trees (CT) are another form of representation of SPL data. They are favorable in nature because of the fact that they are in essence set in 3D surface space; also, they are a good form of representation for hierarchal data. Their visual structure has a significant effect on the perceptive of the viewer [4].

Tree Maps (TM) are favorable for software product line data representation simply for the fact that they are a space filling technique with a true visual format for representation of hierarchal data [31]. Its essence is to optimize utilization of full window space with a rectangular region mapping the hierarchal structure resulting in a "space filling manner" [31].

Conventional Trees (CNT) either being it be a vertical tree or horizontal tree [32][31] are another established form of representation of SPL data [32]. It is a popular form of depicting hierarchal data using link and containment, with the links being used to connect nodes (containment notation). Convention tree structures are based on the fact that there are no cycles in it and only one axis is used for division of levels in the tree, making it easy to map and extrapolate data [4].

UML notations are probably the most utilized form of representation of architectural data [27][28][26][24][29][25]. Specifically, UML class diagrams are among the most cited in UML representation. Some used natural language with UML based notations for representation purpose. This is a relatively common form of representation for showing generalization, association, composition, inheritance and it is also used for providing platform independence.

Textual Form (TXF) is also used for representation of software product line data [23][22]. This form is useful as software product line data is extremely large in size, and folded axis is used to fit the data in the height of the window and Seesoft [30] form is used in stating file as columns and line of code as colored strip. These are some forms which focus on textual representation of SPL data.

Use case map path notations (UCM) are also used for representation of SPL. It is a concept which is used for capturing "requirements" at a reasonable level of detail. Its use in SPL is focused towards capturing of requirement for construction of architecture [19].

Hyperbolic Trees (HBT) are a unique way of representing structures. Its nature is that parallel lines deviate away from each other, "making the circumference of the circle to grow exponentially with its radius" [18]; thus, resulting in space being accessible with mounting distance.

V. EVALUATION CRITERIA

The evaluation criteria are singled out from the literature on the merit that a visual structure would be most suited for use in illustrating architecture of a software product line if it has these features in its essence rather than having them incorporated externally. Based on the fact that a visual structure is made on the truth that its impact would amplify the "cognition" of the stakeholder, in short increase their perceiving power.

A. *Scope*

Here, scope defines the intake (visual scope) of all the structure in one window and does not require the viewer to scroll or shift windows to see the whole structure. This means that the whole context is depicted in one window and a viewer does not have to rotate between different views [18].

B. *Abstraction*

This metric is chosen for the fact that at an architecture level of SPL, the structure is at an abstract level it is not concerned with the minuscule details such as LOC. Its mean concern is that it can see what "decisions" present are leading to which "features", which further lead to "component", whose code is used for implementation, is not its concern at that level. This is a major metric, in a sense that fine grain detail (are needed in implementation phase are not its concern) are not need of the time at this level.

C. *Hierarchy*

This is one of the major metrics chosen based on the fact that hierarchy is embedded in the nature of software product line. If some structure does not pander towards the hierarchy then its use as part of software product line architecture is not required, also keeping in mind that those structures that do not show hierarchy of the SPL architecture data are of no importance, those structures can be good illustrations for some task, but are not good representation of the whole SPL architecture structure. Hierarchy is subdivided into complex hierarchy and clear hierarchy. Clear hierarchy is a structure that shows hierarchy of data clearly, whereas complex hierarchy makes, perception of data complex.

D. *Traceability*

This is another major metric taken into consideration. Basically the concept behind it is the fact that the comprehension of "what" is affected by some "particular change" is known if "traces" are present [14, 15]. Not just the fact that they are present, but they should be visible as well. This fact is favorable in a sense that it gives viewer the power to not just mentally perceive but visually; see, e.g., forward and backward traceability [32]. This metrics is further subdivided into complex viewer traceability and easy viewer traceability. These sub factors state the fact that if a structure is supporting traceability but that it's not shown clearly results in "complex factor" and a structure which shows trace links clearly falls in "easy factor".

E. *Scalability*

This metrics is drawn because of the simple fact that we are evaluating "industrial size" SPL architecture, which clearly stats the fact that the structure is not going to be small or medium by any standard. This is a major issue and is highlighted by [16], but the literature suggests, that the present techniques might resolve other issues in SPL, but they are not scalable for "industrial size" SPL architecture [16].

F. *Evolution*

This metric is a simple and well cited one in SPL architecture in a sense that evolution happens in "space" and "time" and the structure should be such that it can support evolution and the changes which occur because of it should be traceable back to its source. If not then even if you add changes to its architecture it would bear no consequence because SPL architecture evolution has "dependencies, mismatching of variance, high cognitive complexity etc" incarcerated in it [17].

G. *Perception*

This metrics is extracted from the fact that visual structure is always considered to be "good" if what the data wants to convey is clearly presented, helping in perceiving patterns, association, and relationship and so on. In short, a structure is more effective if the data mapped to it is faster to understand, can express more peculiarity and tend to be less error prone than other mapping techniques [4]. Also, it is a well-known fact [9], that a visual image clearly depends upon the properties of human perception; so a structure is said to be considered good, if it conveys, only the mapped data and not something which is not required.

H. *General visual content*

This general metric contains three sub headings known as overlapping components, overlapping links and data obscuring. These, being chosen on the basis that the obscuring of data is a critical flaw which can lead to disasters results, overlapping links, tend to be misleading in evolution handling matter, overlapping of components can lead to wrong assumption and complexity and shorten the power of perception considerably. These points combined together result in disaster if not taken into view when considering a visual structure for software product line.

VI. EVALUATION OF VISUAL STRUCTURES

The evaluation of visual structures on the basis of criteria mentioned above is depicted in the below mentioned table. The (✓) mark states that the structure supports it, the (✗) mark states that the structure does not support it, and the (○) mark states that the concerned feature is not in contention. Evaluation criteria are a combination of SPL and visual structure attributes. As indicated before, this evaluation does not cover general visual structure (physical data); it only covers, those visual structures which are accounted for, by abstract data, plus they are already in use, in SPL architecture.

TABLE I. TABULAR EVALUATION OF VISUAL STRUCTURES

Evaluation criteria		Visual structure									
		MT	CNT		TM	CT	UMLN		TXF	UCM	HBT
			CNT-V	CNT-H			UMLN-CD	UMLN-NL			
Scope		×	×	×	✓	×	×	×	×	×	✓
Abstraction		✓	✓	✓	✓	✓	×	×	×	×	✓
Hierarchal	**Clear hierarchy**	×	✓	✓	×	×	○	○	○	○	✓
	Complex hierarchy	✓	×	×	✓	✓	○	○	○	○	×
Traceability	**Complex viewing traceability**	✓	×	×	✓	✓	○	○	○	○	×
	Easy viewing traceability	×	✓	✓	×	×	○	○	○	○	✓
Scalability		×	×	×	✓	×	×	×	×	×	✓
Evolution		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Perception		×	✓	✓	×	×	×	×	×	×	✓
General visual content	**Overlapping components**	○	×	×	×	✓	○	○	○	○	×
	Overlapping links	○	×	×	×	✓	○	○	○	○	×
	Data obscuring	○	×	×	×	✓	○	○	○	○	×

The evaluation suggests that the current techniques (UMLN, TXF UCM) which have been used, for representation of software product line architecture are not favorable. They, as a whole, are falling short in all areas, except in evolution handling; the point to be highlighted here is that even evolution handling is not feasible, because they do not support traceability. The only technique in use for SPL, which to some extent, is feasible, is MT. But, that too totally lacks in Scope, Scalability and Perception features. MT partially satisfies hierarchal and traceability features.

Conventional Trees (CNT) are then quite favorable as compared to other visual structures, but, they too also lack in area of scope and scalability.

Cone Tree (CT) falls short in the area of scope, scalability, and perception (affected because of falling short in general visual context).

Tree Maps (TM) are quite good visual structures but the area in which they fall short is hierarchal and traceability feature because, though they support it, they fall in the complex area of the above mentioned features. TM also falls short in the area of perception because of the fact that there structure is so crowded that overlapping of component, overlapping of links, and data obscuring due to component overlapping makes it hard to perceive.

Hyperbolic Trees (HBT) are the ideal structure, which falls, in all the right categories. It is a good structure, for scalability, perceiving, evolution, scope, abstraction, traceability, hierarchy. In short, they are "good" for representation of Software product line architectures.

VII. CONCLUSION AND FUTURE WORK

This paper identified the best suited visual structure and its need, through evaluation of visual structures for software product line architecture. Initially, visual structures were extracted, and then evaluated on the basis of a defined criterion. The result showed that those techniques used by SPL architectures lack a lot of features, only one technique was identified HBT, which was found to be good for use, by software product line architecture.

There is a need for interactive techniques, to be incorporated with HBT for the construction of full information visualization technique for SPL architecture, which is seen as part of our future work.

ACKNOWLEDGMENT

We would like to thank all our teachers and colleagues who helped. A.K, thanks MR. Mushtaq and MR. Abdullah for their endearing support.

REFERENCES

[1] K. Berg, J. Bishop, and D. Muthig, "Tracing Software Product Line Variability — From Problem to Solution Space," In Proceedings of the 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries (SAICSIT '05). South African Institute for Computer Scientists and Information Technologists, South Africa, pp. 182-191, 2005. [Retrieved: June, 2013].

[2] D. John. McGregor, “Software product lines”. Journal of Object Technology, vol. 3, no. 3, pp. 65–74, April/March, 2004. [Retrieved: June, 2013].

[3] R. Edward. Tufte. “Visual explanation:Images and Quantities, Evidence and Narrative”. Cheshire, CT: Graphics Press, 1997.

[4] S. Card, J. Mackinlay, and B. Shneiderman. “Readings in Information Visualization - Using Vision to Think”, Morgan Kaufmann, 1999.

[5] A. Blaine. Price, R. Baecker, and S. Ian. Small, “A Principled Taxonomy of Software Visualization,” J. Visual Languages and Computing, vol. 4, no. 3, pp. 211-266, September, 1993. [Retrieved: June, 2013]. doi: 10.1006/jvlc.1993.1015.

[6] G. Catalin. Roman and C. Kenneth. Cox. “A Taxonomy of Program Visualization Systems,” Computer, vol. 26, no. 12, pp. 11-24, December, 1993. [Retrieved: July, 2013]. doi: 10.1109/2.247643.

[7] M. Storey, F. Fracchia, and H. Muller. “Cognitive Design Elements to Support the Construction of a Mental Model During Software Exploration,” J. Systems and Software, vol. 44, pp. 171-185, January, 1999. [Retrieved: June, 2013]. doi: 10.1016/S0164-1212(98)10055-9.

[8] P. Caserta and O. Zendra. “Visualization of the static aspects of software: A survey,” IEEE Transactions on Visualization and Computer Graphics, vol. 99, no. RapidPosts, August, 2010. [Retrieved: July, 2013]. doi: 10.1109/TVCG.2010.110.

[9] C. Ware. “Information visualization:Perception for design”. Morgan Kaufman Publishers, 2nd ed, 2004.

[10] S. Bassil and R. Keller. “A Qualitative and Quantitative Evaluation of Software Visualization Tools,” Proc. 23rd IEEE Int’l Conf. Software Eng. Workshop Software Visualization, pp. 33-37, 2001. [Retrieved: July, 2013].

[11] J. Maletic, A. Marcus, and M. Collard. “A Task Oriented View of Software Visualization,” Proc. IEEE Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2002), Paris France, pp. 32-40, June. 2002. [Retrieved: July, 2013]. doi: 10.1109/VISSOF.2002.1019792

[12] K. Gallagher, A. Hatch, and M. Munro. “Software Architecture Visualization: An Evaluation Framework and Its Application,” IEEE Trans. Visualization and Computer Graphics, vol. 34, no. 2, pp. 260–270, March/April, 2008. [Retrieved: June, 2013]. doi: 10.1109/TSE.2007.70757.

[13] L. Bass, P. Clements, and R. Kazman. “Software Architecture in practice”, Second Edition, Publisher: Addison Wesley. 2003.

[14] A. Luis. Sequeira. Sousa. “Traceability Support in Software Product Lines”. Thesis report, Lisboa, 2008.

[15] A. Samuel. Ajila and B. Ali. Kaba. “Using Traceability Mechanisms to Support Software Product Line Evolution” Information Reuse and Integration, 2004. IRI 2004. Proceedings of the 2004 IEEE International Conference on , vol., no., pp.157-162, 8-10 November, 2004. [Retrieved: July, 2013]. doi: 10.1109/IRI.2004.1431453.

[16] L. Chen, M. Ali. Babar, and N. Ali. “Variability Management in Software Product Lines: A Systematic Review”.In Proceedings of the 13th International Software Product Line Conference, SPLC ’09, pp. 81–90, Pittsburgh, PA, USA, August, 2009. Carnegie Mellon University. [Retrieved: June, 2013].

[17] J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, J. Henk. Obbink, and K. Pohl. “Variability Issues in Software Product Lines”. Springer. LNCS 2290, pp. 13–21, October, 2002, [Retrieved: June, 2013]. doi: 10.1007/3-540-47833-7_3.

[18] J. Lamping and R. Rao. “Hyperbolic Browser:A focus+context Techniques for visualizing large hierarchies”. Journal of visual languages and computing, vol. 7, no. 1, pp. 33-55, March, 1996. [Retrieved: July, 2013]. doi: 10.1006/jvlc.1996.0003

[19] T. John. Brown, R. Gawley, R. Bashroush, I. Spence, P. Kilpatrick, and C. Gillan. "Weaving behavior into feature models for embedded system families”. Software Product Line Conf. pp. 52-61, August, 2006. Baltimore, Md [Retrieved: July, 2013]. doi: 10.1109/SPLINE.2006.1691577

[20] H. Ye and H. Liu. “Approach to modelling feature variability and dependencies in software product lines”. IEEE. vol. 152, pp. 101-109, June, 2005. [Retrieved: June, 2013]. doi: 10.1049/ip-sen:20045007.

[21] S. Ferber, J. Haag, and J. Savolainen. "Feature Interaction and Dependencies: Modeling Features for Reengineering a Legacy Product Line". Software Product Lines (SPLC2): Springer. pp. 235-256, August, 2002. [Retrieved: July, 2013]. doi: 10.1007/3-540-45652-X_15

[22] K. Chul. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. “FORM: A feature-oriented reuse method with domain specific reference architectures”, Annals of Software Engineering, vol. 5, pp. 143-168, 1998. [Retrieved: July, 2013]. doi: 10.1023/A:1018980625587

[23] A. van Deursen, M. de Jonge, and T. Kuipers. "Feature-Based Product Line Instantiation Using Source-Level Packages". Software Product Lines (SPLC2): Springer. pp. 217-234, August 2002. [Retrieved: June, 2013]. doi: 10.1.1.16.6376.

[24] D. Muthig and C. Atkinson. "Model-Driven Product Line Architecture". Software Product Lines (SPLC2): Springer. pp. 110-129, August, 2002. [Retrieved: June, 2010]. doi: 10.1007/3-540-45652-X_8

[25] D. Fey, R. Fajta, and A. Boros. "Feature Modeling: A Meta- Model to Enhance Usability and Usefulness". Software Product Lines (SPLC2): Springer. pp. 198-216, August, 2002. [Retrieved: June, 2013]. doi: 10.1007/3-540-45652-X_13

[26] S. Salicki and N. Farcet. "Expression and Usage of the Variability in the Software Product Lines". Software Product-Family Eng (PFE-4): Springer. pp. 304-318, October, 2002. [Retrieved: July, 2013]. doi: 10.1007/3-540-47833-7_27.

[27] G. Halmans and K. Pohl. “Communicating the variability of a software-product family to customers”. Software and Systems Modeling, vol. 2, pp. 15-36, March, 2003. [Retrieved: June, 2013]. doi: 10.1007/s10270-003-0019-9.

[28] F. Bachmann, M. Goedicke, J. Leite, R. Nord, K. Pohl, B. Ramesh, and A. Vilbig. "A Meta-model for Representing Variability in Product Family Development". Software Product-Family Eng (PFE-5):

Springer. pp. 66-80, November, 2004. [Retrieved: July, 2013]. doi: 10.1007/978-3-540-24667-1_6.

[29] D. Lyn. Webber and H. Gomaa. “Modeling variability in software product lines with the variation point model”. Sci. Comput. Program., vol. 53. pp. 305-331, December, 2004. [Retrieved: June, 2013]. doi: 10.1016/j.scico.2003.04.004.

[30] S. G. Eick, J. L. Steffen, and E. E. Sumner, "SeeSoft –a Tool for Visualizing Line Oriented Software Statistics". IEEE Transactions on Software Engineering, vol. 18, pp. 957-968, November, 1992. [Retrieved: June, 2013]. doi: 10.1109/32.177365.

[31] B. Johnson and B. Shnedierman. "Tree-maps: a space-filling approach to the visualization of hierarchical information structures," Visualization, 1991. Visualization '91, Proceedings, IEEE Conference on, vol., no., pp. 284-291, 22-25 October, 1991. [Retrieved: July, 2013]. doi: 10.1109/VISUAL.1991.175815.

[32] D. Nestor, L. O'Malley, A. Quigley, E. Sikora, and S. Thiel, "Visualisation of Variability in Software Product Line Engineering," in 1st International Workshop on Variability Modelling of Software Intensive Systems (VaMoS-2007), Limerick, Ireland, 2007. [Retrieved: July, 2013]. doi: 10.1.1.136.9399.