Aspects Oriented Approach towards the Goal Driven Software Lifecycle

G. SuryaTeja
B.Tech Student,
RVR&JC College of Engineering,
Dist: Guntur, AP, India

ABSTRACT

Technology and its development need to be improved ion the aspect of the Information Technology which follows some methodology or process. In the current context we usually talking about the XP, Agile and test driven Approach needs to also change in its aspect oriented. Development of the software needs to be smart unique and proportionally domination if we want to explore in the Digitalized information World. If we see the data analysis of the requirement which having many draw flaws leads us to the next level of the journey of life cycle of SDLC. Aspect Oriented requirement needs to be early discussed and should follow a rapid model of changing with each and every version to base cycle which we represented in this paper making to inspiration next level of the classical technology may not be adoptable . IT industry moves on two major task one is deliverable in time which is based on the requirements is the base Pillar.

Keywords

Early aspects, goals, goals interaction, fuzzy logic, use cases, goal cluster.

1. INTRODUCTION

OOP and the reasons to consider AOP as an extension Object oriented techniques decompose software into modules, in this case classes, because the unit of modularity in OOP languages is the class. Some desired behavior might be common to different classes and therefore ends up spread among several classes. Systemic concerns or concerns that relate to a group of classes, such as security concerns, cannot be encapsulated in a single unit and therefore surface disperses across several classes. The role of programming languages in shaping the abstractions by which software designers and programmers apprehend and organize software cannot be underestimated. This applies for requirements engineering as well. The abstractions that ultimately shape software are heavily influenced by the underlying implementation paradigm, like the prevalence class/object concept. In this evolutionary trend we find more and more conceptual tools, just like objects in OOP provide an abstraction for elements in the real world. Evolving together with the programming languages we find software development methodologies.

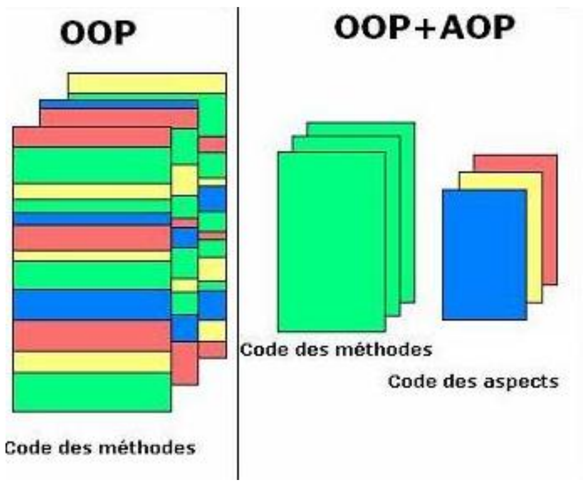

Fig.1.1. Illustration of the Goal Driven Cycle

Nevertheless, the object abstraction along with the composition mechanisms provided in the OOP entail limitations. These limitations have already been discussed in and more in depth by S. Clarke in. S. Clarke clearly demonstrates that the units of modularization in the OOP are structurally different from the units of modularization of requirements specification.

2. RELATED WORK

A number of difficulties for aspect identification, either at requirements or at other stages of software development stem from a definition of aspect that needs to be made more complete and precise. Let us for instance consider the proposal on early aspect identification as in. Their approach towards aspect identification relies on use cases, when a use case extends more than one use case or when a use case is included by one or more viewpoints then it is considered an aspectual use case. There are a number of difficulties associated with aspect identification by doing so, for instance, prioritization of conflicts that stem from different viewpoints is done by hand, and by hand is made also the decision of what is an aspect and what is not an aspect once they identify candidate aspects. It is nevertheless a valuable approach that gives an important insight towards aspect identification.

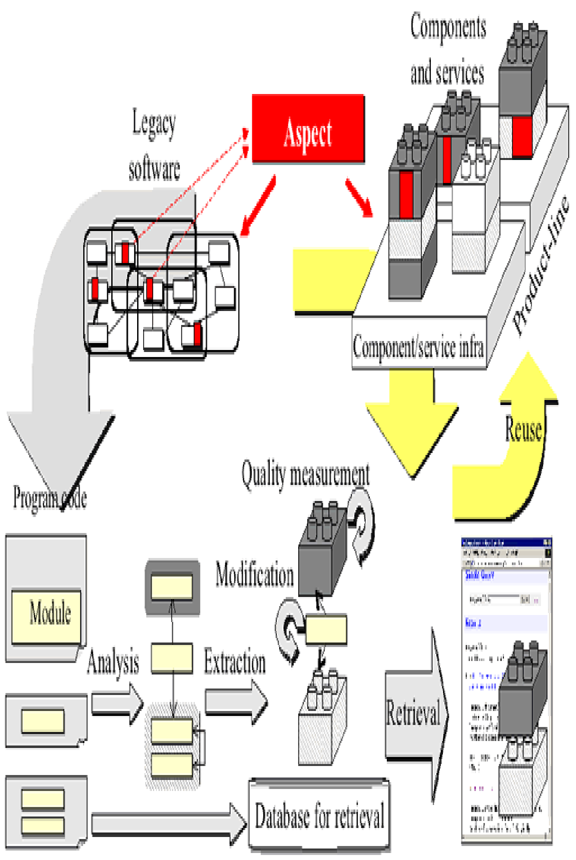

Fig.2.1. Model Driven Approach to View the Quality

It is important to emphasize the works of Rick on the analysis of software architecture properties, the SAAM and ATAM methods. The SAAM method introduces three perspectives to analyze software architecture specifications: functionality, structure, and allocation. Functionality is the activity that the system performs; structure refers to the components and connections; and allocation describes how the functionality is reflected on the structure.

3. PROPOSED METHODOLOGY

A number of difficulties for aspect identification, either at requirements or at other stages of software development stem from a definition of aspect that needs to be made more complete and precise. Let us for instance consider the proposal on early aspect identification as in. Their approach towards aspect identification relies on use cases, when a use case extends more than one use case or when a use case is included by one or more viewpoints then it is considered an aspectual use case. There are a number of difficulties associated with aspect identification by doing so, for instance, prioritization of conflicts that stem from different viewpoints is done by hand, and by hand is made also the decision of what is an aspect and what is not an aspect once they identify candidate aspects. It is nevertheless a valuable approach that gives an important insight towards aspect identification. Moreover, the problem of aspect identification relates to the fact that we need to have an integral view of the problem of concern cross-cutting and consider its context as well. As authors like have already outlined, the problem AOSD solves is one of complexity in today's software applications.

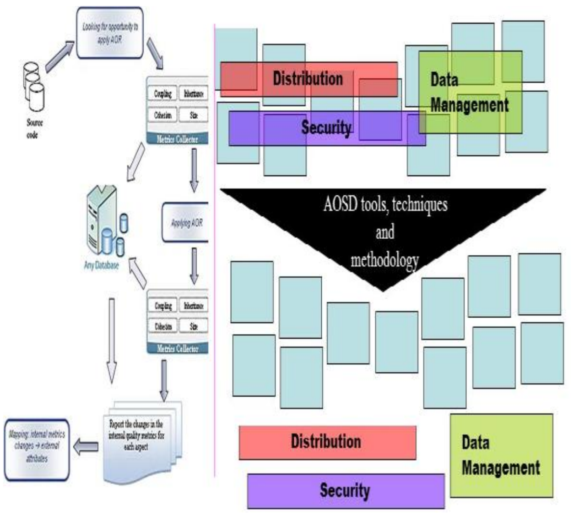
Looking for opportunity to apply AOR
Source code
Metrics Collector
Any Database
Applying AOR
Metrics Collector
Mapping internal metrics changes -> external attributes
Report the changes in the internal quality metrics for each aspect
Distribution
Data Management
Security
AOSD tools, techniques and methodology
Distribution
Data Management
Security

Fig.3.1.1 Architectural Design Model of the AOSD Model

This method is divided into five steps: the canonical functional partition, the mapping of the functional partition on structure, the selection of quality attributes, the selection of testing tasks, and the evaluation of results. The ATAM method is based on the analysis of scenarios, which are obtained as a refinement of software architecture descriptions. The result of this analysis is a set of risks, non-risks, sensitivity points, and trade-off points in the architecture. In addition, the ARID method emerges to complete the proposal of ATAM with a technique for insuring quality detailed designs in software.

$$\mathsf{weaveC}^{\mathsf{pj}} : Prog^{\mathsf{pj}} \times Class^{\mathsf{pj}} \longrightarrow Class^{\mathsf{j}}$$

$$\begin{aligned}
&\mathsf{weaveC}^{\mathsf{pj}}(P, \mathtt{class}\ C\ \mathsf{ext}\ C'\ \mathtt{imp}\ C_1..C_p\{D_1..D_j I_1..I_n\}) \\
&\qquad = \mathtt{class}\ C\ \mathsf{ext}\ C''\ \mathtt{imp}\ C'_1..C'_p\{D'_1..D'_k\} \\
&\text{where}\quad D'_1..D'_k = \mathsf{GetDecs}^{\mathsf{pj}}(P, C) \\
&\qquad\quad C'' = \mathsf{super}^{\mathsf{pj}}(P, C) \\
&\qquad\quad S = \{C''' \mid P \vdash_{\mathsf{pj}} C''' \overset{\triangle}{it}, C \sqsubseteq C'''\} \\
&\qquad\quad C'_1..C'_q \in \{C_1..C_{\#S} \mid \{C_1..C_{\#S}\} = S\}
\end{aligned}$$

$$\begin{aligned}
&\mathsf{weaveC}^{\mathsf{pj}}(P, \mathtt{interface}\ C\ \mathsf{ext}\ C'\ \{H_1..H_n\}) \\
&\quad = \mathtt{class}\ C\ \mathsf{ext}\ C''\ \mathtt{imp}\ C'_1..C'_p\{D'_1..D'_k\} \\
&\text{where}\quad S = \{C''' \mid C \sqsubseteq C'''\} \setminus \{C\} \\
&\qquad\quad C'_1..C'_q \in \{C_1..C_{\#S} \mid \{C_1..C_{\#S}\} = S\}
\end{aligned}$$

Another work that offers an interesting perspective on the properties that should be analyzed in a software architecture specification is the method. In all these rewrite rules; the target expression is evaluated first to give an address. The type of this address is obtained using function type. This gives the target object's dynamic class Ct. Due to polymorphism; this might be a subclass of the class in which the member defined. Caj def (P,Ct, id) is used to give the defining class Cd. The defining class, together with the member name, is used to look up the member definition and obtain the signature. This information is used to form a join point designator to use as the argument of advices which gives the set of all valid sequences of applicable advice to execute the join point. Any member of this set, i.e. any valid sequence of applicable advice may be executed.

4. EVOLUATION AND ANALYSIS

It allows the designer to capture the essential interactions between objects that are present in the system without requiring him to make unnecessary decisions about which objects will be involved in those interactions. Since the focus is on the interactions and not on the objects themselves, the main unit of modularity is the role model. Since interactions take place between objects all interesting role models include more than one role. Thus the modularity cuts across class based decomposition. Roles describe the interaction behavior of objects and not their identity so many roles in a role model might serve to specify complex interactions between one object and itself.

5. CONCLUSION AND FUTURE WORK

It suggests three properties or dimensions to analyze software architecture descriptions: abstraction level, dynamism, and the aggregation level. The abstraction level dimension determines if the software architecture description is more conceptual (analysis) or realization. The dynamism dimension determines whether the architecture is static or dynamic. Finally, the aggregation dimension establishes to what extent a structure is made from other structures. These three dimensions are represented as a matrix, and the result of the evaluation method is the position of a specific architecture inside the matrix.

6. REFERENCES

[1] E. Baniassad, P. C. Clements, J. Ara_ujo, A. Moreira, A. Rashid, and B. Tekinerdo_gan, “Discovering early aspects,” IEEE Softw., vol. 23, no. 1, pp. 61–70, Jan.-Feb. 2006.

[2] A. Rashid, A. Moreira, and J. Ara_ujo, “Modularisation and composition of aspectual requirements,” in Proc. 2nd Aspect-Oriented Softw. Develop. Conf., 2003, pp. 11–21.

[3] M. Mortensen, S. Ghosh, and J. M. Bieman, “Aspect-oriented refactoring of legacy applications: An evaluation,” IEEE Trans. Softw. Eng., vol. 38, no. 1, pp. 118–140, Jan./Feb. 2012.

[4] S. Miller, “Aspect-oriented programming takes aim at software complexity,” Comput., vol. 34, no. 4, pp. 18–21, Apr. 2001.

[5] N. Noda and T. Kishi, “On aspect-oriented design-an approach to designing quality attributes,” in Proc. 6th Asia Pac. Softw. Eng. Conf., 1999, pp. 230–237.

[6] M. Shomrat and A. Yehudai, “Obvious or not? regulating architectural decisions using aspect-oriented programming,” in Proc. 1st Int. Conf. Aspect-Oriented Softw. Develop., Apr. 2002, pp. 3–9.

[7] J. Viega and J. Voas, “Can aspect-oriented programming lead to more reliable software?” IEEE Softw., vol. 17, no. 6, pp. 19–21, Nov./Dec. 2000.

[8] A. Rashid, A. Moreira, and B. Tekinerdogan, “Early aspects— Aspect-oriented requirements engineering and architecture design,” IEEE Proc. Softw., vol. 151, no. 4, pp. 153–155, Aug. 2004.

[9] S. M. Sutton Jr. and I. Rouvellou, “Modeling of software concerns in cosmos,” in Proc. 1st Int. Conf. Aspect-Oriented Softw. Develop., 2002, pp. 127–133.

[10] A. Dardenne, A. van Lamsweerde, and S. Fickas, “Goal-directed requirements acquisition,” Sci. Comput. Programm., vol. 20, no. 1–2, pp. 3–50, 1993.

[11] J. Mylopoulos, L. Chung, and B. Nixon, “Representing and using nonfunctional requirements: A process-oriented approach,” IEEE Trans. Softw. Eng., vol. 18, no. 6, pp. 483–497, Jun. 1992.

[12] A. van Lamsweerde, R. Darimont, and E. Leitier, “Managing conflicts in goal-driven requirements engineering,” IEEE Trans. Softw. Eng., vol. 24, no. 11, pp. 908–926, Nov. 1998.

[13] J. Lee and Y. Fanjiang, “Modeling imprecise requirements with xml,” Inform. Softw. Technol., vol. 45, no. 7, pp. 445–460, May 2003. [14] W. Lee, W. Deng, J. Lee, and S. Lee, “Change impact analysis with a goal-driven traceability-based approach,” Int. J. Intell. Syst., vol. 25, pp. 878–908, Aug. 2010.

[14] F. Steimann, “Domain models are aspect free,” in Model Driven Engineering Languages and Systems, New York, NY, USA: Springer-Verlag, 2005, pp. 171–185.

[15] A. Moreira, A. Rashid, and J. Ara_ujo, “Multi-dimensional separation of concerns in requirements engineering,” in Proc. 13th IEEE Int. Conf. Requirements Eng., 2005, pp. 285–296.

[16] Rashid and A. Moreira, “Domain models are not aspect free,” in Proc. 9th Int. Conf. Model Driven Eng. Lang. Syst., Springer, 2006, pp. 155–169.

[17] L. Constantine and L. Lockwood, Software for Use. Reading, MA, USA: Addison-Wesley, 1999.