OO SYSTEMS DEVELOPMENT BARRIERS FOR STRUCTURAL DEVELOPERS

Making the most of your systems engineering methodology

Elsabé Cloete, Aurona Gerber
School of Computing, University of South Africa, Pretoria, South Africa

Keywords: Systems Engineering Methodologies, System Development Paradigms, Object Oriented Development, Structural Development

Abstract: Paradigm contamination occurs where methods from different system development (SD) paradigms are integrated or combined. We investigate the OO and structural SD approaches and concern ourselves with the question of how paradigm contamination can be avoided, especially when developers were initially exposed to structural programming techniques and are now expected to apply an OO approach. By comparing the techniques associated with specific SD approaches, an outline is given of the particular differences and commonalities that regularly cause paradigm contamination. Guidelines for avoiding contamination traps are then provided. This is significant for practitioners enabling them to be aware of the possible contamination pitfalls as well as how to avoid them, and as a result to reap the intended benefits of the chosen SD method.

1 INTRODUCTION

Since the formal introduction of *object-oriented systems development* (OOSD) it has been rapidly adopted and chosen by industry. However, many IT educators, students and practitioners were fundamentally trained in the *structured*[1] SD paradigm.

Owing to the many advantages (Bahrami 1999, Brown 2002, Coad & Yourdon, 1990) OOSD is claimed to have, development teams are increasingly required to follow an OOSD approach. This has created a demand for the teaching of OOSD, implying that students are likely to be introduced to both paradigms during training. Sadly, these students frequently do not master any of the approaches completely since they are not yet experienced enough to make a successful switch between the two schools of thought, and the result is *paradigm contamination*[2].

Paradigm contamination occurs when methods from different methodologies are mixed or combined and as a result some of the unique functionalities and benefits of a specific approach are lost. Many people have attempted to reconcile structural systems development (SSD) and OOSD processes (Alabiso 1988, Brown & Dobbs 1989, Gray 1988, Khalsa 1989) but none of these attempts produce clean results that maintain the benefits of either approach because the underlying philosophies of the approaches differ so substantially. According to Berard (2003) experience has shown that the integration of OO thinking into structural methodologies is a mistake as it results in various spin-off problems. (We recognize the position of, and sometimes the necessity for hybrid systems (for example, see (Ambler, Keller 2002)). True hybrid systems are not the result of contaminated efforts, but were planned that way to reap very specific benefits that could be offered by the different approaches at the expense of some of the other benefits.

[1] Also called *functional* systems development

[2] Note that we use *contamination* and not *confusion*. Contamination implies an approach that is *not pure*, which is what this paper is dealing with. Confusion has a narrower definition.

In industry many individuals have been through OOSD courses only to find that the lack of experience plunges them into paradigm contamination. It seems that (re)training in OO techniques does not necessarily produce the essential paradigm shift, but a clear outline of the peculiarities and possible contamination pitfalls is required to reap the intended benefits of the SD approaches. Literature addressing these peculiarities and contamination pitfalls to assist novice OO developers is limited. Furthermore, many authors actually encourage the integration methods from the two paradigms (Post 2001, Wesson 1997) which leads to confusion and paradigm contamination.

The question that we concern ourselves with is how paradigm contamination can be avoided. We address this question by comparing the specific approaches and techniques associated with the aforementioned SD approaches to outline the particularities that commonly cause their integration and combination. We have also developed guidelines to identify areas where contamination commonly occurs in order to avoid contamination traps. The benefit of our work is found in the strategies provided to detect and prevent paradigm contamination and as a result be able to retain the advantages of the chosen SD approach. This is significant to practitioners as it enables them to be aware of the possible contamination pitfalls as well as how to avoid them. As a result it facilitates an SD environment that can be optimised to improve the quality of the end product. In the paper the discussion is focused around structural contamination that occurs in the development of OO systems.

Section 2 uses the theoretical essentials of the two SD paradigms to highlight the differences between them. These differences are used in Section 3 to compile the abovementioned guidelines, whilst conclusions are drawn in Section 4.

2 PARADIGM CONTRASTS

The term *paradigm* originated from Greek and was originally only used scientifically. Today it is used to indicate a perception, approach, theory or frame of reference. Covey (1989) describes paradigms as mental maps through which we interpret the world around us. Within the software domain we use *paradigms* to create mental models of systems, and these paradigms influence the approach and techniques used to analyse and develop systems. Covey (1989) describes a *paradigm shift* as the "Aha!" experience when someone finally 'sees' something in another way.

The notion of a paradigm as a frame of reference is applicable within the systems development domain where the *structured paradigm* in essence views a system as a collection of processes operating on data and this approach uses functional decomposition and entity models to identify the central functions and processes required in the system. The *object-oriented paradigm* in contrast views systems as a collection of interacting objects and models interactions between objects to achieve the required systems functionality. We do not have central control in OO as in the structured paradigm.

Many comparisons and discussions of the two approaches fail to take notice of the *paradigm* difference.

In their paper, Shah et al. (1997) analyse potential pitfalls of OOSD from different viewpoints including the conceptual and political ones, analysis and design, environment, language and tools, implementation, class and object, and re-use. Although a useful discussion, it does not focus on in-depth analysis and design issues but rather on implementation issues. It also seems to favour the mature software engineering team as the target audience. Our discussion concentrates on analysis and design issues that commonly cause the integration and contamination of the two SD approaches. Our target audience includes (1) the novice developer or student who might be expected to work in both structural and OO environments, (2) the novice developer or student who has been trained in the structural paradigm and is expected to be trained for, or to work in, the OO environment, and (3) the instructor responsible for teaching OOSD principles to structurally exposed students

We use five categories to broadly classify the issues that form the basis of our discussion. These include the *General Approach*, which embodies all other categories; *Analysis,* dealing with contamination issues at the outset of an SD project; *Modelling,* which explains how models and their uses are easily misapplied; *Coding,* dealing with implementation contamination issues and, finally, *Consistency*, which discusses consistency issues across other categories.

2.1 General Approach

An important difference between the two SD approaches is the inclusion of an iterative and incremental development style (OOSD) versus the conclusion of phases (SSD). Although the difference in approach is in essence due to the historical development of software development methodologies, the object-oriented *paradigm* includes the notion that the requirements of a system changes throughout development and it therefore requires an iterative and incremental method. The iterative style prescribes the re-analysis/ re-design of a portion of the system, recognizing the possibility that completed parts might be flawed. Incremental development implies a piecemeal development of the whole application.

The structural paradigm requires the conclusion of each step within each phase before moving to the next step or phase. For example, the requirements-gathering phase is seldom revisited because of the basic assumption that the first attempt was correct and complete. Accommodating late requirement changes is complex due to the complexity of accurately establishing the affected parts. Some variations of the traditional SDLC such as the spiral SD model attempted to address this matter, but these approaches still require the completion of a phase before the initiation of the subsequent phase.

2.2 Analysis

A fundamental paradigm shift required in the move to OOSD expands conceptual modelling at the beginning of the analysis process. The SSD approach analyses the problem in terms of the solution domain when it segregates the data and processes/functions. A conceptual model in SSD is therefore often includes entity models and information flow diagrams. OOSD, in contrast, analyses the problem in terms of the problem domain when it first models problem domain objects and their interactions before translating the analysed information to the solution domain.

A second analysis area where a paradigm shift is required is on the level of abstraction. Closely related to conceptual modelling, the level of abstraction refers to the tools that are used to perform analysis. OOSD approaches use use-cases, interaction diagrams, activity diagrams and the resulting conceptual class diagrams to analyse and fully understand the problem domain. Structural approaches, on the other hand, use solution-based diagrams from the commencement of the analysis. Paradigm-contaminated analysts often start their analysis with use cases and then follow this up by using class models and entity models interchangeably. In this way the analysis of the problem domain is fused with an analysis of the solution domain. Paradigm-contaminated developers often do not realize that entity models are already at the design level of system development, while use-cases, activity and interaction diagrams are at the analysis level, which makes the application of entity models to OO analysis inappropriate in problem domain-based analysis.

2.3 Modelling

We distinguish between two modelling issues, namely the *systems model* and *diagrammatic representation*. As mentioned above, the interchangeable use of the entity models and specifically ER-diagrams, and class diagrams points to paradigm contamination. Although the different diagrams have elements of commonality, their semantics differ completely. Entity identification is commonly considered as central to design specification (Bulman 1998). But, as Sha et al. (2001) put it: "OOSD does not simply imply the definition of classes, objects and methods, in the same way that structural programming does not simply imply the removal of GOTO statements from spaghetti programs". Bulman (1998) correctly summarizes the OO design phase as the establishment of the system architecture (class diagrams) as well as the definitions of their interactions and interrelationships. However, this is not what the ER-diagram represents.

Structurally contaminated developers are frequently impatient to get to the solution domain. Instead of focusing on objects, they concentrate on data entities when analysing requirements. They often incorrectly assume that objects are the same as data entities – often because some OOSD methodologies suggest noun identification as a first level of object identification (Booch 1982, 1983a, 1983b).

The second modelling contamination aspect is found in *diagrammatic representation* where it is often accepted that similar diagrammatic representations have identical semantics. For example, both paradigms use *rectangles,* but in the structured paradigm, a rectangle represents entities (data without functionality) and in the OO paradigm, classes (data and functionality). In order to model the functionality of the system, an ER-diagram needs an (additional) accompanying *data-flow diagram*.

Another example of confusion in diagrammatic representation is found in the link[3] between two objects/entities as well as the *multiplicity* or *cardinality* in such a link. In an ER-diagram, a link represents a time-independent *relationship* between entities, while it represents a time-dependent *association* in a class diagram where a message is typically passed from one object to another, indicating behaviour or interaction. *Cardinality* models a general truth between entities, while

[3] adjoining lines between rectangles

multiplicity models behaviour at a specific point in time. For example, when modelling the marriage relationship between a man and a woman for system, a structural approach would depict the *cardinality* between the two entities as many-to-many to capture the possibility of multiple partners during a lifetime. In the OOSD approach the *multiplicity* between the same two objects is one-to-one, indicating the marriage of one man to one woman at any particular time. The history of the man's (or woman's) previous marriages could be captured in a different persistent object depicting marital history.

2.4 Consistency Level

Phases in the life cycle of an OO methodology tend to be much more consistent with each other (Berard 2003), than those that follow a structured approach. Concepts are reused and tend to retain their meaning throughout a life cycle instead of new concepts being introduced for each phase.

Structured approaches, on the other hand, commonly use different techniques for each life cycle phase. In these approaches, different styles of modelling in the different life cycle phases require different styles of thinking, while in the OO paradigm the same representation, notation and style of thinking from inception to final management are used.

2.5 Coding Level

Some developers claim that they do not need any thorough SD, because owing to their experience, they can code the problem into the required system straight away. Although this is commonly true for small scale systems, this boldness is very often a giveaway of paradigm contamination as structural programming can be done this way through functional decomposition, but it is very difficult to do an OO implementation where every object executes a small part of the overall system functionality without modelling. In an OO system a responsibility is shared by different objects, while in structured systems a responsibility is captured in a single system process.

3 AVOIDING PARADIGM CONTAMINATION

Paradigm contamination can be avoided through awareness. According to Covey (1989) "The more aware we are of our basic paradigms, maps or assumptions, and the extent to which we have been influenced by our experience, the more we can take responsibility for those paradigms, examine them, test them against reality, listen to others and be open to their perceptions, thereby getting a larger picture and a far more objective view."

In the next section we develop a set of guidelines to create awareness of paradigm contamination. We will use the comparison categories used in the previous sections to describe guidelines for the detection and possible avoidance of paradigm contamination pitfalls.

3.1 General approach contamination

OO methodologies generally subscribe to iterative and incremental SD, while structural methodologies advocate models such as the waterfall model where each phase forms a conclusive unit. The following are questions that can be asked to establish whether structural contamination occurs:

– Does the developer insist on having all requirements on all facets of the problem before moving on?
– Does the developer insist on completing all facets of the problem during each development phase?
– Does the developer insist on having all design models before being able to commence with implementation?

An affirmative answer to any of the above questions points to structural contamination. At this point the rationale behind incremental design and development to deal with incomplete or inconsistent requirements and to partition the development into several increments can be explained to the developer (Rowlet 2001). In this way developers can avoid endangering critical code, increments can be added as the development process proceeds, and productivity might be improved by working with more manageable pieces.

3.2 Analysis contamination

Structural contamination on the analysis level is not always easy to spot, since it is necessary to understand the thought processes of the student. The following guiding questions might reveal analysis contamination:
– Is the problem expressed in terms of data entities or database fields?
– Is the problem partitioned into functional units to capture a possible solution (functional decomposition)?

– Is a type of flow diagram drawn to capture a possible solution?
– Is the analysis process going ahead with class diagrams that are developed from the identification of data entities?

Structural developers often claim that their experience leads them to understand the requirements to such an extent that they see the solution immediately and do not have to analyse the problem in the OO way. This type of thinking is a common source of paradigm contamination that can be rectified through examples and practice.

3.3 Modelling contamination

The issues raised in 3.2 are also relevant to establishing modelling contamination because structurally contaminated developers commonly begin with identification of data entities or ER diagrams. In doing so, they fail to incorporate the behaviour of the system, which would have been revealed if a detailed analysis had been performed. Pertinent questions include the following:
– Is a class or ER diagram used in the analysis process?
– Does the multiplicity reveal general truths or does it model a truth at a specific time?
– Is elaboration through data flow diagrams required to interpret the behaviour of class objects?
– Does the link between two class objects imply a relationship rather than an interaction?

Paradigm contamination could be rectified by pointing out the specific differences as well as the impact of each on the different paradigms.

3.4 Consistency contamination

Structurally contaminated developers tend to introduce new concepts between the different life cycle phases because different structural models work on different objects. Two simple questions to establish structural contamination at this level are as follows:
– Do new objects appear in the different models that portray the problem or the solution?
– Is there a clear translation from one phase into the next?

An affirmative answer to the first question, and a negative answer to the second one points to a possible contamination struggle. A way to overcome this is to introduce the use of UML, which retains semantics between the different phases and produces different views on the same objects.

3.5 Coding contamination

Coding contamination is always a result of contamination at earlier levels. An affirmative answer to any of the following questions might indicate structural coding contamination:
– Do some objects only contain data or are there elements that are not encapsulated within any object?
– Are any of the following used: exceptions, parameterised classes, meta-classes and concurrency?
– Inheritance: Does there seem to be confusion between interface inheritance and implementation inheritance? Does the use of inheritance violate encapsulation? Are multiple inheritances used in any way?
– Is there confusion between is-a, has-a, and is-implemented-using relationships?

If structural contamination occurs in this category it might be better to take the developer back to the analysis and design stages of the project, which prescribes the implementation.

4 CONCLUSIONS

There is a demand for the application of different paradigms in software development projects, but the OO-approach is nowadays often favoured or preferred by industry. However, many developers have been trained in structural or even both paradigms, which often lead to paradigm contamination when they are working in the OO paradigm. It would assist developers and instructors if they were aware of the implications of the different paradigms and the impact of paradigm contamination.

In this paper we discussed the problem of structural paradigm contamination in the OOSD and outlined categories where structural contamination usually occurs. We concluded each category by providing guidelines for detecting and avoiding contamination. We foresee (and have experienced) that the most difficult problem in eradicating structural paradigm contamination is to address the human side of it, encouraging developers and instructors, students and developers to do a paradigm shift, and perceive contamination pitfalls and the benefits of applying a pure development approach.

REFERENCES

Alabiso B. Transformation of Data Flow Analysis Models to Object -Oriented Design. 1998. *In: OOPSLA '88. Special Issue of SIGPLAN* Notices, Vol. 23 (11). pp. 335 - 353.

Ambler S.W. 2000. *Mapping objects to relational databases*. An AmbySoft Inc. White Paper. Retrieved 11 February 2002 from http://www.AmbySoft.com/mappingObjects.pdf.

Baharami A. 1999. Object oriented systems development using the unified modelling language. Irwin McGraw-Hill.

Berard E.V. Object Oriented Design. Retrieved February 2003 from http://www.toa.com/pub/ood_article.txt.

Booch G. 1982. *Object Oriented Design*. Ada Letters, Vol. I, No. 3, pp. 64 - 76.

Booch G. 1983a. Software Engineering with Ada. Benjamin/Cummings, Menlo Park, California, 1983.

Booch G. *Object Oriented Design*. 1983b IEEE Tutorial on Software Design Techniques, 4th Ed., P. Freeman and A.I. Wasserman, Editors, IEEE Computer Society Press, pp. 420 - 436.

Brown D.W. 2002. An Introduction to Object-Oriented Analysis - objects and UML in plain English. 2nd Ed. John Wiley & Sons, Inc. New York. pp 668.

Brown R.J. AND Dobbs V. 1989. A Method for Translating Functional Requirements for Object-Oriented Design. *In The Seventh Annual National Conference on Ada Technology*. pp. 589- 599.

Bulman D. 1998. *Objects Don't replace design.* Computer Language. Vol 6. No. 8. pp 151-152

Coad P. AND Yourdon E. 1990. Object-Oriented Analysis. Computing Series. Yourdon Press, Englewood Cliffs, NJ.

Covey S.R. 1989. The 7 Habits of Highly Effective People. Simon & Schuster UK Ltd.

Gray, L. 1988. Transitioning from Structured Analysis to Object-Oriented Design. *In The Fifth Washington Ada Symposium, Association for Computing Machinery,* New York, New York, pp. 151 - 162.

Khalsa K. 1989. Using Object Modelling to Transform Structured Analysis Into Object-Oriented Design, *In The Sixth Washington Ada Symposium*. pp. 201- 212.

Keller W. *Mapping objects to tables, a pattern language.* Retrieved 11 February 2002 available from http://www.objectarchitects.de/ObjectArchitects/Papers/Published/ZippedPapers/mappings04.pdf.

Post G. 2001. Database Management Systems: Designing and Building Business Applications, 2nd edition. McGraw-Hill.

Rowlet T. 2001. The Object-Oriented Development Process. Upper Saddle River, NJ, USA. Prentice Hall. pp 420.

Sha V., Sivitanides M. & Martin R. 1997.. *Pitfalls of Object Oriented Development*. Retrieved January 2003 from http://www.westga.edu/~bquest/1997/object.html

Wesson J.L. 1997. *An Investigation into Design Methodologies for Usability: A Case Study Approach.* Doctoral Thesis, University of Port Elizabeth