Does object-oriented domain analysis work?

FAILY, S.

2005

RE-Papers

Does Object-Oriented Domain Analysis Work?

Shamal Faily, LogicaCMG Space & Defence Division, shamal.faily@logicacmg.com

The Rational Unified Process (RUP) [5], the Rapid Object-Oriented Process for Embedded Systems (ROPES) [2] and other OO approaches propose Use Case driven analysis as a mechanism for capturing requirements and deriving object models.

B. Douglass' *Real Time UML* [3] describes how "later analysis decomposes the system into … objects". Beyond discussion of a number of common object identification strategies, such as noun-phrase underlining and key concept identification, only lip-service is paid to what has been described as the Fundamental Difficulty (FD) of Object Oriented Domain Analysis (OODA).

The Fundamental Difficulty was defined by Svetinovic et al (at RE'05 in Paris, see the reports in this issue) [7] as the difficulty of identifying system domain concepts as Objects (which some might think pretty central to the OODA enterprise - Ed.). A study of undergraduate projects, documented by Svetinovic using these techniques, found that object models of the same system often differed drastically in terms of concepts identified, while software concepts were often specified at inconsistent abstraction levels.

These observations raise the concern that OODA may be incorrectly applied by many practitioners. Microsoft's Steve McConnell [6] argues that most practitioners neither have the benefit of a Software Engineering education, nor do they have ready access to evaluations of the myriad of available tools and techniques.

Managing the abstraction problems

Software Engineers will appreciate the difficulty of working at multiple levels of abstraction whilst evaluating or developing Object Oriented software, even for solutions within a well-known problem domain. Could the results presented by Svetinovic indicate cognitive overload, caused by the combination of trying to understand the problem space and conceptualising this within the object metaphor?

A cost-effective remedy might be to focus on key Object-Oriented concepts, in practice, using an OO Programming Language such as Python or Smalltalk. Many people, myself included, commonly use Python to prototype concepts and ideas which can be eventually implemented in other languages, such as C++.

By using OO Programming to test one's own ideas and explore how idioms and patterns can work

together, one can appreciate some of the beauty and elegance of solving problems with the object metaphor. Students and practitioners alike should then be more confident and better prepared to abstract the problem space more accurately.

A Reality Check

If transforming domain concepts to objects were purely a visualisation task, a good argument would still exist for OODA. Unfortunately, the more one looks at the grey area between the problem and solution spaces, the less good the argument seems.

For example, Concept Identification by noun-phrase analysis, using scenarios, depends on the quality of the scenario text. But even if the scenarios were initially well-written, what may they be like after repeated and possibly biased editing?

Tool-support can help; promising approaches such as Metamorphosis [1] are likely to evolve to support OODA.

But Open-Source tool support in this area is scarce; COTS tool support is costly; and research approaches have had only a limited outing within industry. Worse, the combination of domain and engineering knowledge necessary to make OODA work remains scarce.

Since few people in industry will want to volunteer their own projects as a proving ground even for suitably risk-mitigated OODA approaches, we may have to accept that progress will be slow.

A way forward?

Fortunately, there are other ways to explore the problem space besides OODA, including traditional approaches like context analysis, and modern ones like goal modelling.

Also, because of the increasing ubiquity and maturity of OO and its applications within the solution space, Software Engineers are well positioned to apply Objects in the problem space.

Hawthorne [4] has suggested that continuing trends such as the move of software development off-shore to the lowest-cost location of the day and growing reliance on COTS/Open Source components willforce Software Engineers to apply their architectural expertise within problem-space areas, ie in the domain of the Requirements Engineer.

We should not necessarily abandon OODA; many industrial projects have used it successfully. But perhaps we should drop OODA for now, until we can better mitigate the concept identification risk?

And, what about a head-to-head comparison between OODA and other approaches? Such a study could help determine where the Fundamental Difficulty lies.

In conclusion

I hope that this short discussion of OODA will provoke reflection on the value of applying the object metaphor within Requirements Engineering.

At the least, OODA needs a health-warning on its cellophane wrapper.

References

[1] Diaz, I., Pastor, O., Matteo, A., *Modeling Interactions using Role-Driven Patterns*, Proceedings of RE'05, Paris, IEEE

[2] Douglass, B., *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*, 1999, Addison-Wesley

[3] Douglass, B., *Real Time UML*, 2004, Addison-Wesley

[4] Hawthorne, M. J., Perry, D. E., *Software Engineering Education in the Era of Outsourcing, Distributed Development, and Open Source Software: Challenges and Opportunities*, Proceedings of ICSE 2005, ACM

[5] Kruchten, P., *The Rational Unified Process*, 1999, Addison-Wesley

[6] McConnell, S., *Code Complete*, 2004, Microsoft Press

[7] Svetinovic, D., Berry, D, M., Godfrey, M., *Concept Identification in Object-Oriented Domain Analysis: Why Some Students Just Don't Get It*, Proceedings of RE'05, Paris, IEEE