Res. Lett. Inf. Math. Sci., 2011, Vol. **15**, pp. 1–11
Available online at http://iims.massey.ac.nz/research/letters/

An Ontology of Agile Aspect Oriented Software Development

DAVID PARSONS
Institute of Information & Mathematical Sciences
Massey University at Albany,
Auckland, New Zealand
D.P.Parsons@massey.ac.nz

Both agile methods and aspect oriented programming (AOP) have emerged in recent years as new paradigms in software development. Both promise to free the process of building software systems from some of the constraints of more traditional approaches. As a software engineering approach on the one hand, and a software development tool on the other, there is the potential for them to be used in conjunction. However, thus far, there has been little interplay between the two. Nevertheless, there is some evidence that there may be untapped synergies that may be exploited, if the appropriate approach is taken to integrating AOP with agile methods. This paper takes an ontological approach to supporting this integration, proposing ontology enabled development based on an analysis of existing ontologies of aspect oriented programming, a proposed ontology of agile methods, and a derived ontology of agile aspect oriented development.

Keywords: agile methods; aspect oriented programming; ontology.

1 Introduction

In recent years we have seen a number of different approaches to bringing software development forward by applying both new technologies and new engineering and lifecycle management practices. In particular, we have seen the emergence of agile methods as a major new paradigm for the management of software development projects. At the same time, aspect oriented programming has emerged from the lab and seen significant practical application. A few researchers have suggested that agile methods and aspect oriented programming may have the potential to be synergistic. However much of the published research into aspect oriented software engineering takes a highly formalised approach to specification that does not integrate well with the informality of agile methods. This paper takes an ontological approach to an analysis of both agile methods and aspect oriented programming, and attempts to identify how these ontologies may be successfully integrated. It is hoped that such an analysis may help to formalise the viable relationships between agile methods and AOP so that the benefits of AOP can be integrated into development projects that are managed using an agile approach.

The basis of the proposal in this paper is a general ontology-based integration of agile methods and Aspect Oriented Software Development (AOSD). This is distinct from AOP insofar as AOSD is concerned with the higher level conceptual aspects of aspect development, not on the minutia of programming. The AOSD ontology described here is derived from an ontology of AOP based on existing proposals and a constructed ontology for agile methods. These two ontologies are merged to create a new ontology intended to provide an analytical model to represent the core relationships between agile methods and AOSD. The following section introduces the known relationships between agile methods and AOSD. Section 3 considers ontologies for software development and introduces an ontology of AOSD based on previous work. Section 4 introduces a new ontology for agile methods, based on an analysis of a number of published agile methods. Section 5 discusses ontology mapping, and introduces an ontology that merges agile methods and AOSD. Section 6 considers some related work, and section 7 contains the summary and conclusions.

2 Agile Methods and Aspect Oriented Programming

The mid 1990s saw the emergence of a new set of informal analysis and design approaches known as agile methods [1]. These methods place emphasis on being flexible to changes in requirements and working in collaboration with customers and other stakeholders. However, agile projects can encounter problems with features of development that cut across user stories (for example usability), which may cause severe disruption to an emergent architecture, and/or require specialist skills not available within the development team, particularly late in the project lifecycle.

Aspect oriented programming (AOP) grew primarily out of the work of Kiczales et al [2] at Xerox Parc, though similar work has been done elsewhere. Its primary value as a software development tool is its ability to treat cross cutting concerns as first class programming components and to enable flexible weaving of these concerns across a domain model. These concerns are often characterised as being primarily non-functional requirements such as caching, security or persistence, though they are also often used for supporting design level functional requirements such as logging and exception handling. There is also a school of thought that maintains that aspects should not be relegated to the margins of development but become the primary mechanism for implementing behaviour and generating attributes, by regarding all system features as aspects derived from the use case model [3]. The general concept of treating aspects as first order constructs in the software development model is known as ‘early aspects’ [4]. Despite its potential, AOP has not (apart from a few notable exceptions such as the JBoss application server and optional features of the Spring framework) been widely adopted in mainstream software development. One possible contributory factor to this situation is that there may be some reluctance to use AOP in agile projects, if there is a lack of published knowledge about the issues involved.

It appears that there may be some synergy between aspect oriented development and agile methods, since late arising requirements changes can be cross cutting concerns. Handling such changed requirements may prove easier if these concerns are implemented using aspects. A number of authors have previously suggested that AOP and agile methods are not incompatible. Many of these take a somewhat general approach, suggesting for example that required system behaviours (i.e. aspects) are identified in cooperation with stakeholders and that aspects may assist in the process of coping with changing requirements [5]. Indeed, the issue of changing requirements is important, since AOP may provide the ideal solution to non-functional requirements that arise late in an agile project lifecycle, as they surely will with agile’s ‘Embrace Change’ philosophy [6]. Others have pointed to the integration of early aspects with an evolutionary model and refactoring of requirements, working with a code centric iterative domain model [7]. However, previous work on the relationship between these two approaches to software development has been limited and tends not to be comprehensive. For example, late arising requirements may be implemented using an aspect layer over a non-aspect core model. This may be a pragmatic solution but does not address a possible deeper relationship between AOSD and agile methods.

3 Ontologies for Software Development

In attempting to provide a more formalised way of analysing the relationships between AOP and agile methods, one approach that may prove fruitful is to consider the use of ontologies as an analytical tool, and perform some ontology mapping in order to increase our understanding of the two objects of our investigation. AOP already has a published ontology, and others have suggested some further development of this ontology. Agile methods too, have an ontology, though as yet this has not been formally published so is largely implicit.

The benefits of an ontology include the ability to categorise the key components of the entities of interest and the relationships between them. Ontologies have been widely explored in software engineering. e.g. [8], [9], [10], [11]. There has also been extensive work on developing and enhancing a public ontology of aspect orientation [12], [13]. Grundy and Hosking [14] stress the role of ontology in specifying aspect oriented software components. There have also been a number of papers relating to application ontologies within specific agile projects. Mishali and Katz [15] explicitly refer to an ontology of XP in driving the architecture of their Eclipse plug-in, though this ontology is not formally expressed. This work is particularly interesting in that it is implementing an aspect oriented approach, supporting XP from the perspective of software process aspects. The aspects are seen as a way of implementing an ontology that is semantically congruent with the various practices of XP. The prototype targets certain

practices that can easily be integrated into an AOP architecture without ambiguity, such as enforcing a test first policy. It would be interesting therefore to explore further how higher level features of agile methods might by supported by an aspect oriented conceptual approach.

Clearly there is a relevant body of work that may contribute to an understanding of how AOP and agile method ontologies may be congruent. So far, however, a more general ontology of agile methods has not been proposed. The motivation for this paper, therefore, is to propose some underpinnings for such an ontology and attempt to map it to subsets of existing AOP and software engineering ontologies.

It is important to specify why one is attempting to build, modify or apply an ontology and what kind of ontology is therefore required. Happel and Seedorf [11] categorise ontologies using two dimensions, one that distinguishes development time from run time, and another that differentiates software and infrastructure. In this paper we are concerning ourselves with development time infrastructure, so are focused on what Happel and Seedorf [11] call 'ontology-enabled development' which uses ontologies at development time to support developers with their tasks. In attempting to propose an ontology that may map aspect oriented software into an agile infrastructure, we are therefore concerning ourselves with a subset of the possible ontologies that might apply to AOP on the one hand and software engineering on the other.

3.1 Ontologies of Aspect Oriented Programming

The primary ontology for AOP is by van den Berg et al [12]. This consists of fifteen ontology views, of which two are of particular interest in identifying ontology-enabled development from an AOSD perspective. These are the Software System View and the Weaving View, which between them encompass the high level view of the system and the core engineering perspective of managing aspects. In addition, Black and Harman [13] propose some extensions to this model, in particular the integration of spatial, lingual and social aspects. These views and extensions have been integrated together to provide a single ontology of aspect oriented software development, as opposed to aspect oriented programming (Figure 1.) Of course the mapping of ontologies is not a trivial matter, but at this level of abstraction we are not concerned with application specific ontologies so in theory, at least, a generic ontology is possible, even if its final form may be a matter for debate.

In summary, an aspect oriented software system is composed of concerns that are composed together using some kind of scheme that assumes composability. Significantly, these concerns, once composed, are tangled with each other (i.e., they are cross cutting concerns.) The system is realized by woven components of base code that instantiate implementation aspects. However there are additional aspects that encompass the wider considerations of software development, namely spatial (the physical nature of the development, which may be distributed), lingual (the woven code will rely on a set of languages and tools) and social (the software will in most cases be developed by a team rather than an individual, and have external stakeholders.)

4. An Ontology of Agile Methods

The relationship between ontologies and agile methods appears in the literature from time to time. For example Knublauch [16] suggests that ontology driven development should be more applied in the agile domain, and asserts that, with the correct tool support, ontologies can be a powerful support for agile methods, in particular for generating test methods and supporting stakeholder involvement. Thus far, however, no single generic ontology of agile methods has been proposed. Therefore we have begun to propose such an ontology, based on an analysis of a number of commonly used agile methods. We took seven agile methods and attempted to summarise their terminology, illustrated with some key examples (Table 1.)

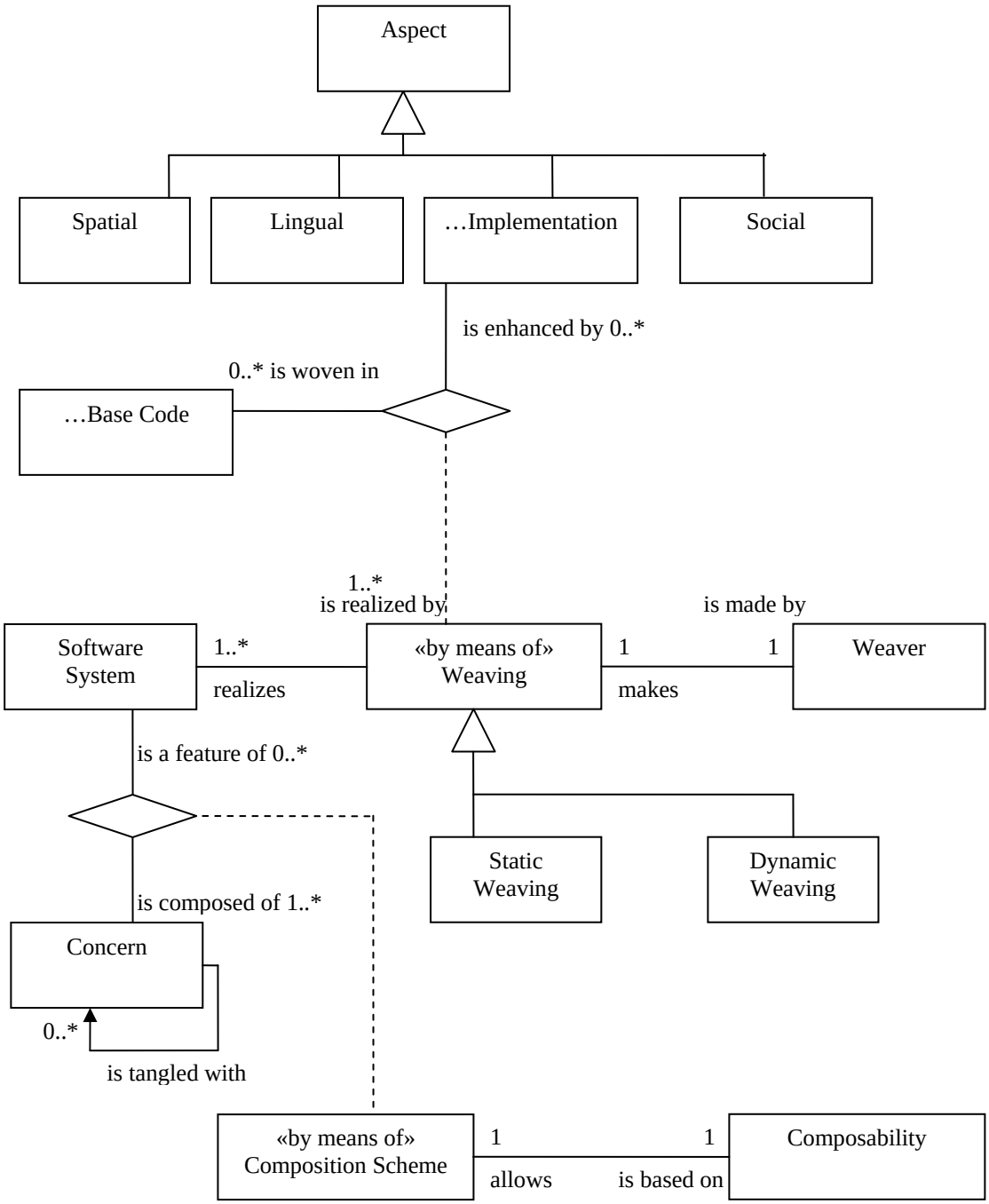

Figure 1: A single ontology of aspect oriented software development

Table 1: Key concerns of agile methods

Agile Method	Term	Examples
Agile Microsoft Solutions Framework	Principles	Foster open communications, empower team members, establish clear accountability and shared responsibility
	Mindsets	Focus on Business Value, Foster a Team of Peers, Internalize Qualities of Service
Agile UP	Phases	Inception, elaboration, construction, transition
	Disciplines	Model, implementation, test, project management
	Philosophies	Simplicity, tool independence
Crystal Clear	Properties	Frequent delivery of usable code, reflective improvement, osmotic communication
	Strategies	Incremental Rearchitecture, Information Radiators.
	Techniques	Daily Stand-up Meetings, Side-by-Side Programming, Burn Charts.
DSDM	Principles	User involvement, empowered project team, frequent delivery of products, testing throughout the project life-cycle
	Techniques	Timeboxing, MoSCoW, testing, workshop
eXtreme Programming (XP)	Values	Communication, simplicity, feedback, courage, respect
	Activities	Coding, testing, listening, designing
	Techniques	Pair programming, test driven development, continuous integration, collective code ownership
Feature Driven Development	Activities	Plan by feature, design by feature, build by feature
	Best practices	Domain object modelling, developing by feature, individual class (code) ownership, visibility of progress and results
Scrum	Techniques	Team creation, backlog creation, project segmentation, scrum meetings, burn down charts
	Phases	Review release plans, sprint, sprint review, closure

The purpose of this exercise was to identify commonality (or otherwise) of a representative number of agile methods to explore the viability of building an ontology that might apply across all agile methods. The purpose of the examples was to enable us to filter the various terms used in the seven methods so that we could identify synonyms. This approach follows Happel and Seedorf [11], where their ontology classification is illustrated by exemplars. Where synonyms were identified, one term was chosen to subsume the others. Where possible, the chosen term was the most commonly used of the synonyms. The chosen terms were;

- Technique
- Phase
- Principle
    - Subsumes property, value
- Activity
    - Subsumes discipline
- Practice
    - Subsumes mindset, philosophy, strategy

In general it seems that an agile method will have some guiding set of principles that underpins its approach. It will also have high level activities, supported by management and engineering techniques. These will be organised under the umbrella of a set of practices. There may also be the concept of phases within the overall process. Within the detail of the various methods, the instantiation of techniques, for example, may vary widely. Engineering focused methods like eXtreme Programming (XP) will promote a specific set of techniques, whereas other methods, such as Scrum, do not concern themselves so much

with engineering practice as with project management processes. Common ideas emerge from many methods, including testing, communication and visibility of progress. Incompatibilities are few and far between, with individual code ownership in Feature Driven Development being one of the few examples, contrasting with the common code ownership promoted by most other methods. This however has no impact on the overall ontology, since these are simply different instantiations of technique.

From this analysis an initial ontology of agile methods has been derived that attempts to encompass the various characteristics of commonly used methods. This ontology is shown in Figure 2.

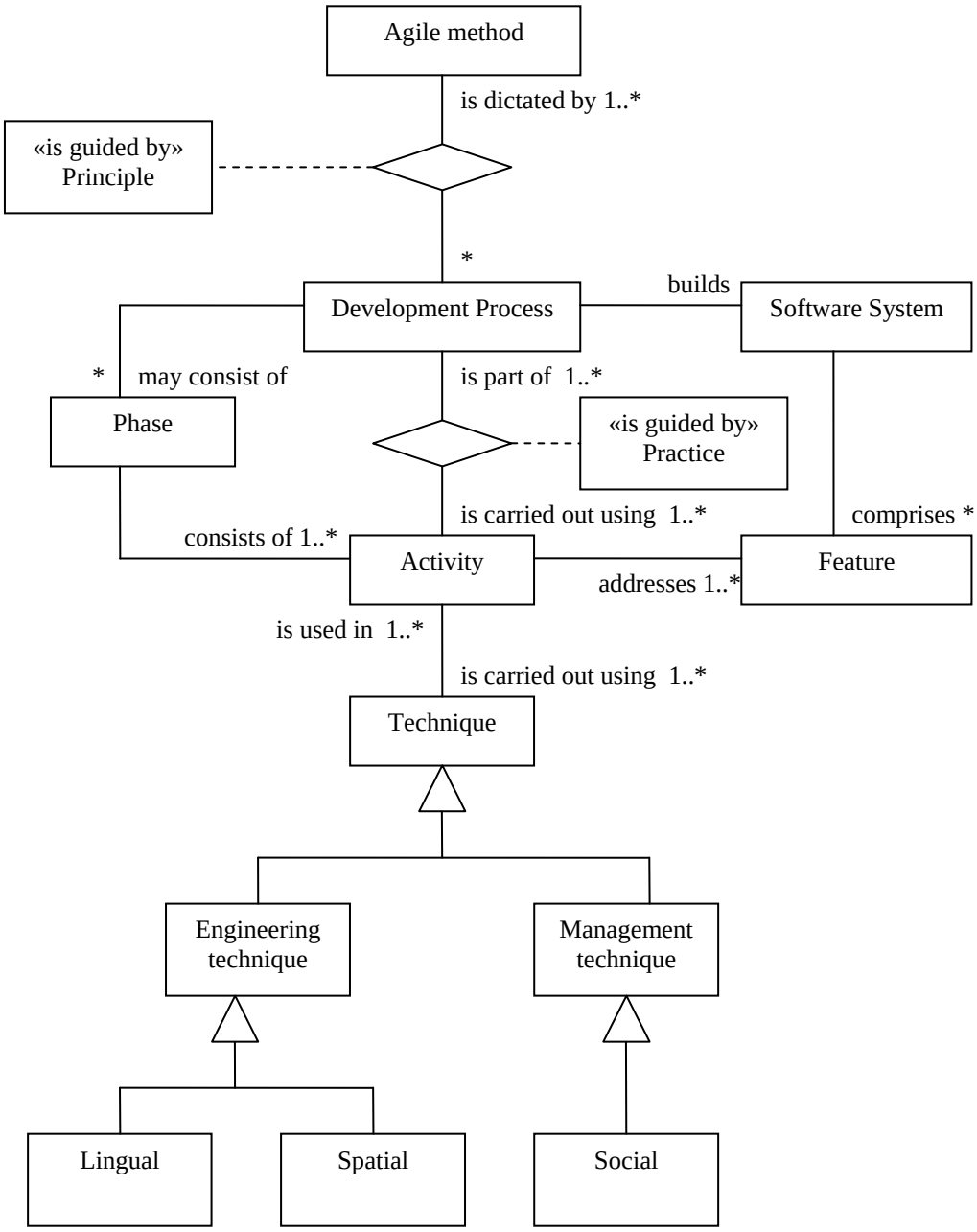

Figure 2: A generic ontology of agile methods

In this generic ontology for agile methods, a software system consists of a set of features built within activities that are part of a development process. That process will be guided by the principles of a particular method. Various techniques are used to carry out the activities (they will vary between methods) but these techniques will be either engineering or management oriented. The engineering techniques will include spatial considerations (co-location, pair programming etc.) and lingual issues (languages and tools). The management technique will address social issues such as active stakeholder involvement, sustainable pace and activities such as stand up meetings and retrospectives.

5. A Unifying Ontology for Agile Aspect Oriented Software Development

So far this paper has outlined an ontology for AOSD, based on filtering and extending an existing ontology for AOP. It has also proposed a new ontology for agile methods, based on an analysis of the core features of a number of published agile methods. The next stage is to consider how these ontologies may usefully be mapped together. Ontology mapping may be done in many ways, but these various approaches can be analysed in terms of logical theories [17]. These theories may be based on a range of approaches, including a mapping from one ontology to another in a directional way, the expression of two ontologies in a single language representation, and the derivation of a third ontology from two original ontologies. Much ontological work, such as that related to the semantic web, is based on the translation of formal schemas. In this paper, such formal schemas do not exist, so we are working with somewhat looser and more incomplete translation [18]. In such a case we can only attempt to draw out a best fit mapping from disparate domain models and attempt to create a third ontology that acts as a unification of shared concepts and themes. At the level of abstraction of the schemas we are working with here, only a level of conceptual bridging is required. Table 2 summarises the concepts in the two ontologies and indicates where semantic mapping may occur (leaving aside for the moment the context of relations between concepts).

Table 2: Mapping AOSD and Agile ontology concepts

AOSD Ontology	**Agile Ontology**
These ontology components are outside the scope of the AOSD ontology	Agile method
	Principle
	Phase
	Practice
	Management technique
	Activity
	Technique
Implementation, Composition scheme	Development process
Software system, Base code, Weaver	Software system
Aspect, Concern	Feature
Weaving , Static weaving, Dynamic weaving	Engineering technique
Lingual, Composability	Lingual
Spatial	Spatial
Social	Social

Table 2 suggests that although there are few direct mappings between the ontologies, there are a number of indirect mappings and sub-mappings which together imply that an agile method may subsume AOSD as an integrated set of practices, which provide further support for software development processes that are not inherent in the AOSD ontology.

There are, however, some issues to be resolved regarding the relations within the two ontologies. If we examine the AOSD ontology, the relation between the software system and the implementation of aspects is the woven base code, while the separation of concerns and aspects is quite striking. These characteristics may be accounted for by the derivation of this ontology from a view of aspect orientation that is based on programming models. We can also see from Table 2 that some components of the AOSD

ontology tend to be spread across the Agile ontology, in particular the implementation, base code and weaving.

Figure 3 shows a proposed unified ontology for agile aspect oriented software development. Some elements from the AOSD ontology have been subsumed into a general concept of ‘implementation’ that can be seen as the work done using engineering techniques to implement the features of the software system. What then, does this ontology tell us? Perhaps the key features are the central role of the composition model within the relationship between the software system and the concerns that are implemented with aspect oriented features. An agile aspect oriented development method must have some way of specifying composability in terms of the features that it derives from the concerns of the domain. This underlines the issue that a method that does not apply early aspects does not inherently assume composability criteria. However this is an important concept, as it implies that aspects should be first order constructs in such a method.

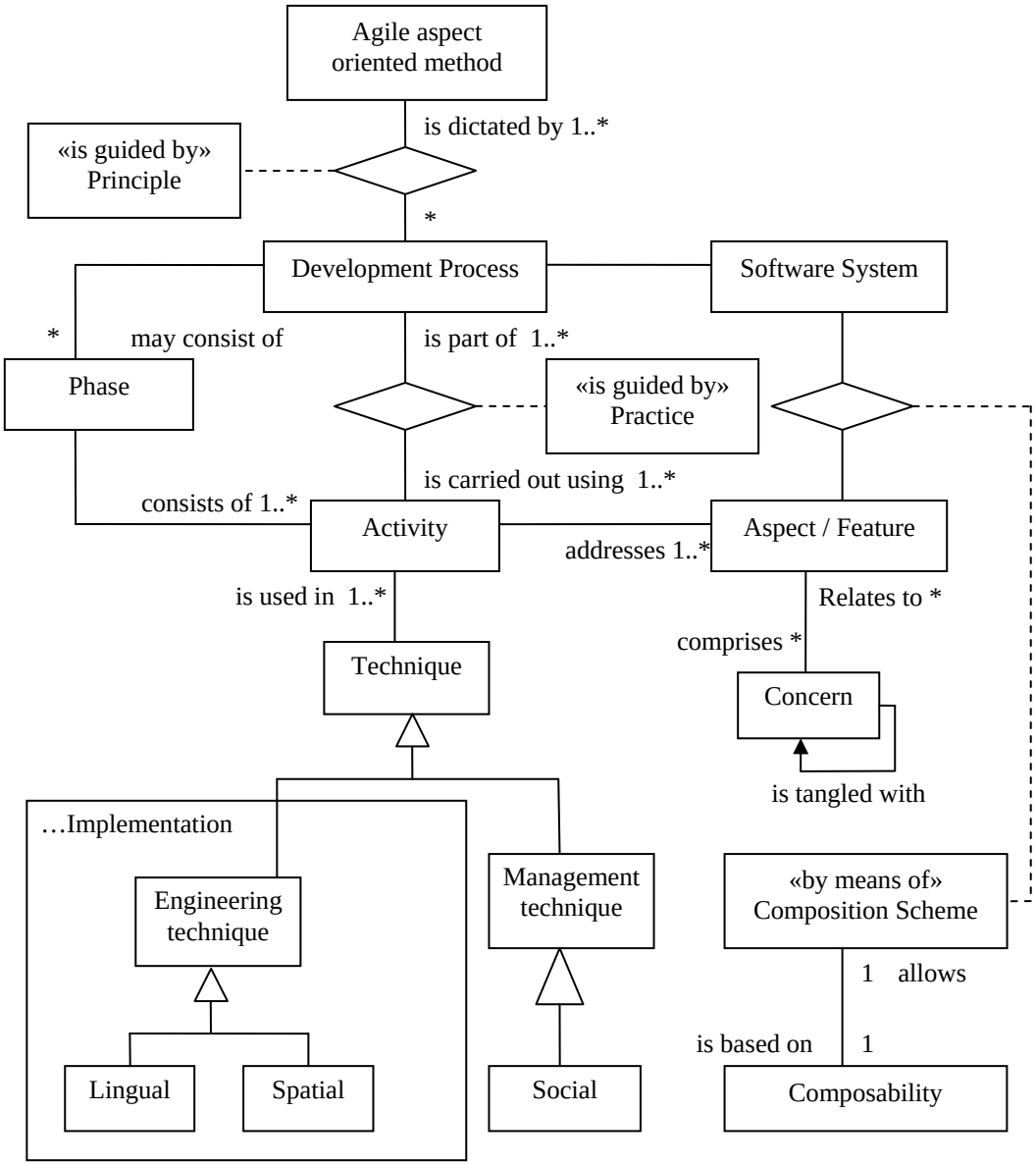

Figure 3. A unified ontology for agile aspect oriented software development.

6. Related Work

Other authors have proposed specific mappings between AOP and individual agile methods. For example Pang and Blair [19] integrate early aspects into Feature Driven Development. One of the key concepts is that an aspect can be modelled as a feature, overcoming some limitations of other approaches where an aspect may not be modelled as a first class component. Singh et al [20] combine AOP with another agile method, eXtreme programming (XP). They suggest that Aspect Oriented Component Engineering (AOCE) on its own does not provide any mechanism to support team management, communication issues and coping with change, and that these features might be gained by integrating with XP. Their paper outlines 'eXtreme AOCE' as an extension of the 12 principles of XP, modified to incorporate the characteristics of AOCE. For example, user stories need to be broken down into aspects and the implementing components. Another feature is the re-casting of the 'metaphor' to include aspects. Included in this concept is the use of naming conventions which may be important for the weaver. The authors also stress the importance of testing in agile environments, and a hierarchy of tests is proposed, which comprises (from the bottom up) aspect testing, component testing, component integration testing and customer acceptance testing. Kircher, Jain and Corsaro [21] also address the integration of AOP and XP, suggesting that AOP can reduce the amount of refactoring required, and systems can be more naturally extensible, though this is assuming that early aspects are applied rather than being refactored out of existing code. The authors also suggest testing being implemented as an aspect so unit tests can be seen as aspects. They also warn against partial usage of AOP because it can make continuous integration more difficult and should not be used for a small number of tasks or it just becomes an overhead.

Yet another agile method, Scrum, is integrated with AOP by Wiese [22]. This may be significant in the sense that Scrum does not really impinge on engineering practice in the same way that, for example, XP does. In Weise's study, AOP impacted on the management of the agile project, because a cross module team was established to take responsibility for cross cutting concerns, supported by common code ownership. Module teams however could 'spike' prototype AOP solutions. Published AOP adoption patterns such as Aspect Team pattern and Magic Build System [23] were used. The important role of the AOP evangelist is stressed in the paper, along with strategies for ensuring that aspects do not impact excessively on the overhead of continuous integration. AOP modularizes crosscutting concerns so cross module issues can be handled centrally, supporting Scrum goals. However AOP goes one step further than Scrum. Not only are cross module issues reflected on and designed in a central team but they are also implemented in a central place.

A significant feature of AOP is that it can support the agile concept of simplicity. Elgar [24] reports metrics from a study that showed cyclomatic complexity reduced by 35% and Halsted effort reduced by 41% by the introduction of AOP in parts of a pre-existing framework.

7. Summary and Conclusions

In this paper we have described ontologies for AOSD and agile methods and used ontology mapping to propose an analytical framework for understanding how agile methods and AOSD might usefully be integrated into a software engineering approach. This work is preliminary in nature and has yet to be exercised by empirical study. However the ontology mapping has helped to clarify the main issues in attempting to integrate AOSD and agile. The semantic challenge is the relationship between aspects, features and concerns. To some extent this challenge is the result of attempting to map an ontology derived from a programming perspective and an ontology derived from software development methods. On the one hand we have aspects that are code components woven into an application structure. On the other we have features, which are the user story driven units of project management for an agile development team. The unifying concept between these is the (cross cutting) concern, but it is important to clarify the meaning of these terms if we are to fully understand how the ontology mapping may support software development processes. There is a cognitive leap between aspects as orthogonal cross cutting concerns and aspects as core business features that can make it difficult to embrace early aspects. At the heart of this problem is the issue of composability. Our ontological analysis suggests that it is necessary to have a software development process that integrates composability throughout the lifecycle if early aspects are to be used in an agile project that emphasises flexibility in the face of changing requirements.

References

[1] J. Highsmith, *Agile software development ecosystem.* Boston: Addison-Wesley 2002.

[2] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J-M. Loingtier, and J. Irwin, "Aspect-Oriented Programming," in *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Finland, Springer-Verlag, LNCS 1241, 1997

[3] I. Jacobson, and P-W. Ng, *Aspect-Oriented Software Development with Use Cases*, Addison-Wesley, 2003

[4] E. Baniassad, P. Clements, J. Araujo, A. Moreira, A. Rashid, and B. Tekinerdogan, "Discovering Early Aspects," *IEEE Software* Vol. 23, No. 1 pp.61-70 2006.

[5] J. Araújo, and J. C. Ribeiro, "Towards an Aspect-Oriented Agile Requirements Approach," in *Proceedings of the Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, Lisbon, Portugal 2005.

[6] G. Boström, M. Henkel, and J. Wäyrynen, "Aspects in the Agile Toolbox," in Proceedings of *Workshop on Software-engineering Properties of Languages and Aspect Technologies, Fourth International Conference on Aspect-Oriented Software Development (AOSD'05)*, Chicago, Illinois, USA 2005.

[7] J. Araujo, D. Zowghi, and A. Moreira, "An Evolutionary Model of Requirements Correctness with Early Aspects," in *Proceedings of the Ninth International Workshop on Principles of Software Evolution (IWPSE'07)*, Dubrovnik, Croatia 2007.

[8] B. Marick, "Methodology Work Is Ontology Work," *ACM SIGPLAN Notices*, Vol. 39, No. 12 pp. 64–72 2004.

[9] P. Wongthongtham, E. Chang, T. Dillon, and I. Sommerville, "Software engineering ontologies and their implementation," in Kobol, P. (ed), *Proceedings of IASTED International Conference on Software Engineering*, pp. 208-213, Innsbruck, Austria, ACTA Publishing 2005.

[10] M. Leppänen, "Towards an Ontology for Information Systems Development," *Via Nova Architecturs* 2006.

[11] H-J. Happel, and S. Seedorf, "Applications of Ontologies in Software Engineering," in *Proceedings of the 1st international conference on Theory and practice of electronic governance*, Macao, China, Pages 5-11, 2007

[12] K. van den Berg, J. M. Conejero, and R. Chitchyan, "AOSD Ontology 1.0 - Public Ontology of Aspect-Orientation," *AOSD Europe*, 2005

[13] S. Black, and M. Harman, "Aspect Oriented Software Development: Towards A Philosophical Basis," *Technical Report TR-06-01*, Department of Computer Science, King's College London, 2006

[14] J. Grundy, and J. Hosking, "Developing Software Components with Aspects: Some Issues and Experiences" in R. Filman, T. Elrad, S. Clarke and M. Aksit (eds) *Aspect-Oriented Software Development* (Chapter 24), Addison Wesley Professional, 2004

[15] O. Mishali, and S. Katz, "Using Aspects to Support the Software Process: XP over Eclipse," in *Proceedings of AOSD 06*, Bonn, Germany 2006.

[16] H. Knublauch, “Ramblings on Agile Methodologies and Ontology-Driven Software Development,” in Proceedings of the *Workshop on Semantic Web Enabled Software Engineering*, Galway, Ireland 2005.

[17] Y. Kalfogloui, and M. Schorlemmer, “Ontology mapping: the state of the art,” *The Knowledge Engineering Review*, Vol. 18, No. 1 pp. 1–31 2003.

[18] J. Madhavan, P. Bernstein, P. Domingos, and A. Halevy, “Representing and Reasoning about Mappings between Domain Models,” in *Proceedings of the Eighteenth national conference on Artificial intelligence*, Edmonton, Alberta, Canada 80-86, 2002.

[19] J. Pang, and L. Blair, “Refining Feature Driven Development - A methodology for early aspects,” in Proceedings of *Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design Workshop*, Lancaster, UK 2004.

[20] S. Singh, H-C. Chen, O. Hunter, J. Grundy, and J. Hosking, “Improving Agile Software Development using eXtreme AOCE and Aspect-Oriented CVS, “ in *Proceedings of the 12th Asia-Pacific Software Engineering Conference*, 2005.

[21] M. Kircher, P. Jain, and A. Corsaro, “XP + AOP = Better Software?” in *Proceedings of XP2002 - eXtreme Programming and Agile Processes in Software Engineering*, Alghero, Sardinia, Italy 2002.

[22] D. Wiese, “Large Scale Application of AOP in the Healthcare Domain,” in *Proceedings of the Seventh International Conference on Aspect-Oriented Software Development*, Brussels, Belgium 2008.

[23] A. Schmidmeier , “Aspect Team and other patterns for an adoption of AOP,” in *Proceedings of the 12th European Conference on Pattern Languages of Programs*, Bavaria, Germany 2007.

[24] C. Elgar, “Real World Evaluation of Aspect-Oriented Software Development,” MSc thesis, Massey University 2006.