Dimensions for the Separation of Concerns in Describing Software Development Processes

Pavel Hruby

Navision Software
Frydenlunds Allé 6
DK-2950 Vedbæk, Denmark
ph@navision.com
http://www.navision.com, click *corporate info* and *methodology*

Abstract. This paper suggests a model that provides orthogonal separation of concerns for software development and management artifacts, describes a pattern for structuring units within each dimension of separation and provides a model for describing interactions between units in different dimensions. This position paper focuses on multi-dimensional separation of concerns throughout the software lifecycle and handling of concerns that span lifecycle phases. This paper can be considered as an instantiation of the model [5], [7] for software development artifacts, such as UML models, and management artifacts, such as project and team.

Introduction

One of the most influential approaches for a separation of concerns in software architecture is the "4+1 view" model [3]. The "4+1 view" model separates the static structure of software development artifacts into 5 views: logical, process, physical and development and scenario views. The logical view is the object model of the software system. The process view captures the static relationships between processes and tasks. The physical view describes the mapping of the components onto nodes. The development view describes the static relationships between components. The scenario view describes the dynamic aspects of all four views. In later versions of this model [4] the physical view has been renamed as the deployment view, the development view as the component view and the scenario view as the use case view.

The "4+1 view" model does not provide orthogonal separation of concerns because the scenario view overlaps the other four views. This paper suggests a model that provides orthogonal separation of concerns for software development artifacts, suggests additional dimensions to cover full description of a software development process and provides a model for capturing interactions between units in different dimensions.

Dimensions for the Separation of Concerns

This section will introduce an orthogonal concern space in which four Kruchten's views are some of the dimensions. Unlike the Kruchten's model, each dimension in our model contains both information about static structure and dynamic interactions. In our model, the use case dimension is similar to other dimensions; it captures both the static structure (that is, relationships between use cases and actors), and the dynamics (scenarios consisting of use case instances[1]). This approach makes the use case dimension consistent with other dimensions and orthogonal to them. To summarize, our dimensions of separations of concerns are:

- The *logical dimension* describes the logical structure of the software system in terms of the responsibilities of the system, actors, subsystems and classes and their interfaces, relationships, interactions and lifecycles.
- The *component dimension* describes the implementation of the system in terms of components, their responsibilities, interfaces, relationships, interactions and lifecycles.
- The *deployment dimension* describes the physical structure of the system in terms of hardware devices, their responsibilities, relationships, interactions and lifecycles.

[1] Description of this artifact is not within the scope of this paper. See reference [1] for details.

- The *process dimension* describes the processes, their responsibilities, relationships, interactions and lifecycles.
- The *use case dimension* identifies types of actor's interactions with the system, subsystems, classes, components and nodes in terms of use cases, their responsibilities, goals, relationships, interactions and lifecycles.

In order to capture artifacts necessary to specify all aspects of software development processes, we introduce several new dimensions in addition to the dimensions of the "4+1 views" model. The additional dimensions are:

- The *test dimension* describes tests in terms of test scripts, test cases, test algorithms, static relationships and interactions between tests.
- The *documentation dimension* describes how the artifacts in other dimensions are documented. The units of the documentation dimension are documents, document lifecycles, document relationships and interactions. The document interactions are relevant only if the documents are represented in electronic form. Documentation dimension includes the technical documentation, the user documentation and on-line help.
- The *policy dimension* describes concerns that influence the behavior of artifacts in other dimensions. For example, the security policy, the style guidelines and the artifact identification policy affect construction and quality assurance of other development and management artifacts. the policy dimension captures such cross-cutting concerns that are practical to keep in a single place.
- The *team dimension* describes the organizational structure in terms of teams, roles of team members, their responsibilities, relationships, interactions and lifecycles.
- The *project dimension* describes projects and tasks in terms of the static properties of projects and tasks, relationships and interactions between them. The project and task relationships can be represented by PERT charts that show the task and project dependencies. The project and task interactions specify a project scenario in terms of starting and finishing tasks. They are typically represented by Gantt charts.
- The *analysis dimension* describes the logical structure of the product in terms of analysis subsystems, objects and their responsibilities, relationships and interactions. The analysis dimension differs from other dimensions in the way that the software entities in the analysis dimension do not specify the software system precisely. The purpose of the analysis dimension is to record preliminary or alternative solutions to design problems. Analysis objects may – but do not always – correspond to logical or physical software entities existing in the product.
- The *business dimension* records users' views on the system. Entities in the business dimension often, but not always, correspond to the logical entities existing in the product. For example, a design that gives the best execution performance is different from one that clearly explains to the users what the system does. In such cases, the business dimension describes the entities that have meaning to the users and explains the system more clearly than entities in the logical dimension.
- The *reuse dimension* describes reusable elements, their responsibilities, relationships and interactions in terms of patterns and frameworks.

Dimensions of separation of concerns are illustrated in Fig. 1.

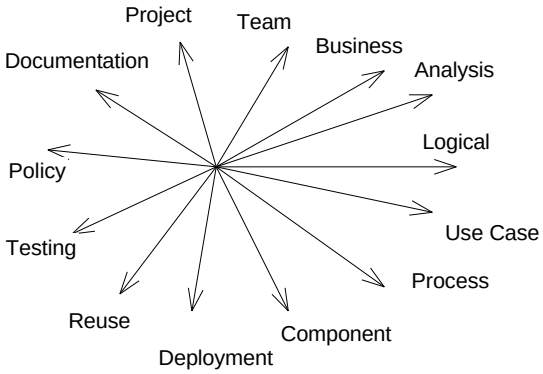

Fig. 1. *Dimensions for the separation of concerns in describing software development processes*.

Structure for each Dimension of Separation of Concerns

If we have a closer look at units in each dimension, we discover that their relationships are not random. Units in each dimension can be structured, that is, further separated, according to the following three rules:

- Separation according to level of granularity. Examples of units at different levels of granularity in the logical dimension are: programming instruction, class, subsystem and system. Examples of units at different levels of granularity in the project dimension are: task, subproject and project. In the documentation dimension, examples are: paragraph, section, chapter and book. The articles [5], [7] call units at low level of granularity *primitive units* and units at higher level of granularity *compound units* or *modules*.
- Separation into specification units and design units. Specification units capture externally visible properties and behavior. For example, the *subsystem* unit is a specification unit. Subsystem specifies subsystem responsibility, subsystem attributes that can be read and set and interface operations with preconditions and postconditions. The specification units are refined into design units [6]. For example, the subsystem unit is refined into the *class relationship model* and *object interaction model*. The class relationship model describes design of the static structure of the subsystem. The object interaction model describes design of each subsystem interface operation in terms of object interactions. Note that specification and design units alternate across the levels of granularity.
- Separation into units that describe static and dynamic information in each dimension. Examples of units describing static information are the subsystem and the subsystem relationship model. Examples of units describing dynamic information are the subsystem lifecycle and the subsystem interaction model.

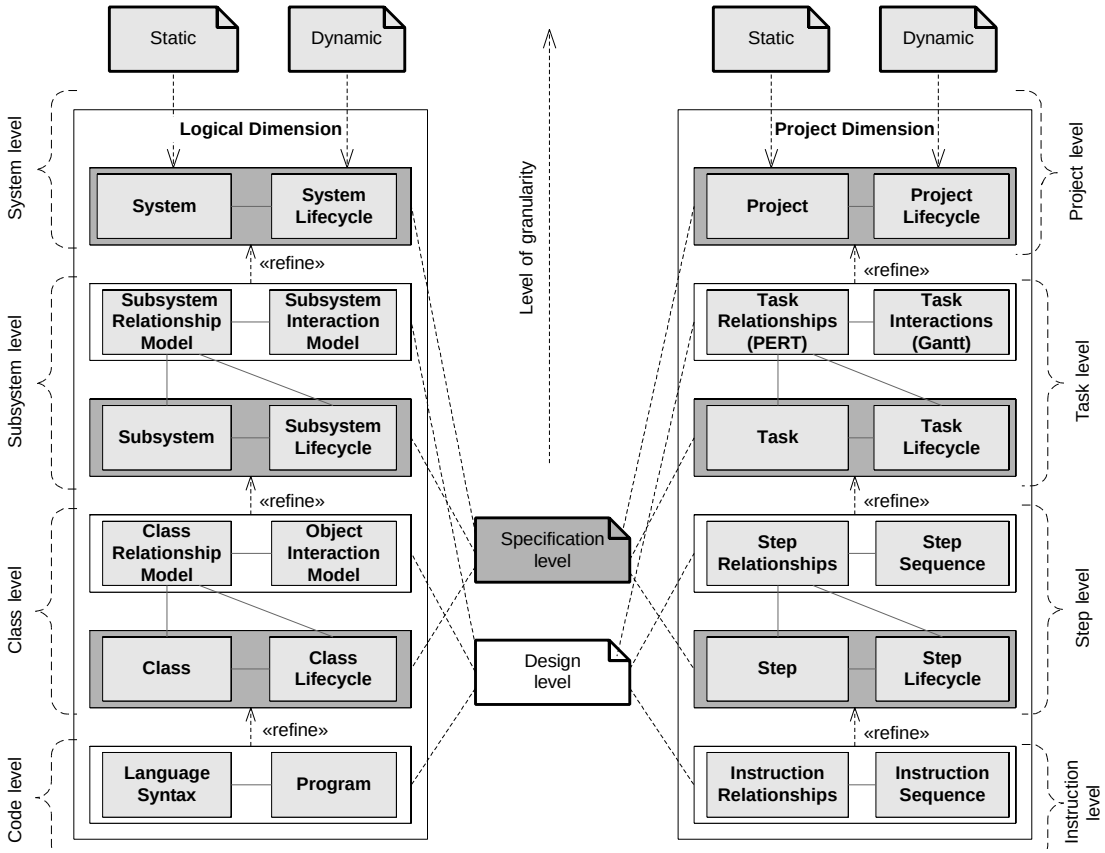

Fig. 2. *Structure of the logical dimension (left) and the project dimension (right). Additional levels of granularity can be added following the same pattern.*

Fig. 2. illustrates units in the logical and project dimensions structured according the three rules above. The logical dimension describes units at four levels of granularity: system, subsystem, class and code level. The project dimension contains four levels of granularity: project, task, step and instruction. Each level of granularity consists of specification and design sub-levels. Each sub-level contains units describing static and dynamic information about software system.

Interactions between Dimensions during Software Lifecycle

The previous sections described dimensions of separation of concerns, which are static aspects of the concern space. This section deals with dynamic aspects – interactions between units in different dimensions during software lifecycle.

In order to specify how units in different dimensions collaborate, we use object-oriented paradigm to describe the units. Units in each dimension are modeled as objects with various attributes and methods. We distinguish between the unit *type* and the unit *instance*. *Types of units* specify properties, attributes and methods of units. *Instances of units* are actual software development and management products produced during software development. An example of a unit type in the use case dimension is the use case model. An example of an instance of the use case model is a real set of use cases, actors and their relationships, represented by a use case diagram.

Units have *attributes*, such as version; representation (which typically contains a diagram, a table or a text); status (such as draft, completed, tested); and references to other units. Units have various *methods* that determine their interactions with other units. For example, the *team* unit has a constructor (a method how to form a team) and quality assurance method (a method how to evaluate whether the team is formed well enough to fulfill its purpose).

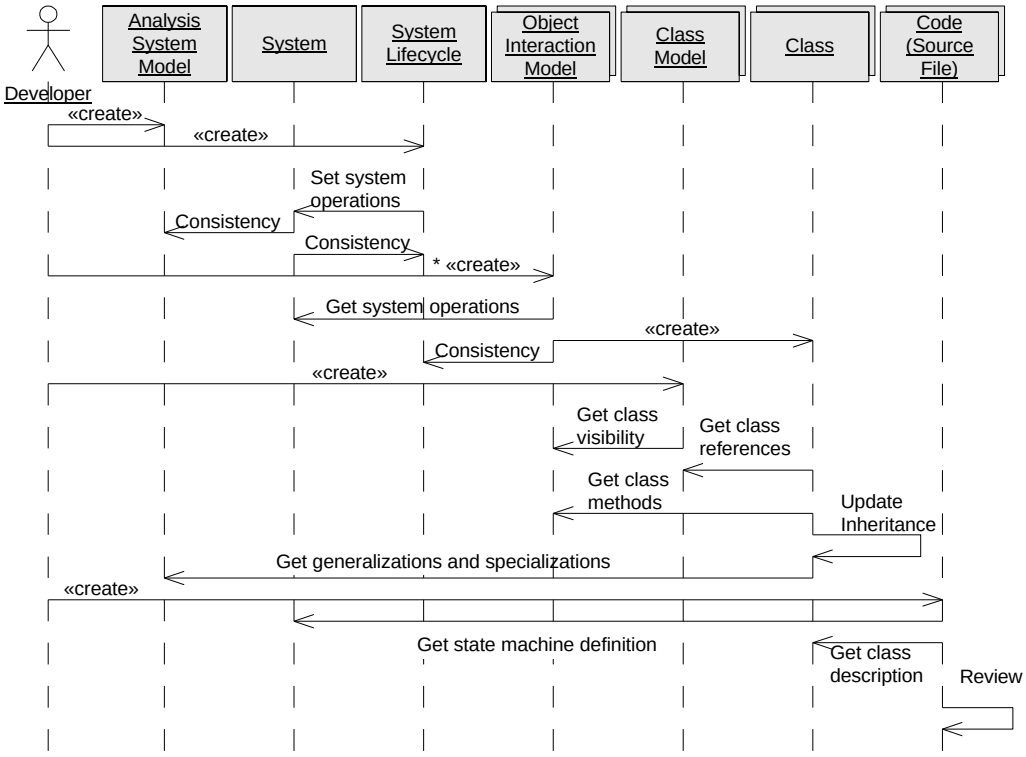

Fig. 3. *Development process of the Fusion method illustrated as interactions between the units in the team dimension and the logical dimension.* (*After ref.* [2])

The software development process is modeled as interactions between units in different dimensions. As an example, Fig. 3 illustrates the development process of the Fusion method. The process is shown as interactions between the developer (a unit in the team dimension) and software development artifacts (units in the logical dimension). A developer who uses the Fusion method is responsible for calling the constructors of the units in the order illustrated in the Fig. 3. Constructors and quality assurance methods generate various messages between the units. Note that Fig. 3 is meant as an illustration of the idea of interactions between units, not as a complete description of the entire Fusion method.

Summary

This position paper introduced several orthogonal dimensions of separation of concerns in describing software development processes. The paper discussed the structure of units within each dimension and introduced the pattern for structuring the units. The paper suggested a model for describing the interactions of units in different dimensions, based on the concept of considering units as objects with responsibilities, methods and attributes.

References

[1] Hruby, P.: Structuring Specification of Business Systems with UML, OOPSLA 98 Workshop on Business Object Design and Implementation IV, Vancouver, British Columbia, October 18-22, 1998, available at http://www.navision.com, click *corporate info* and *methodology*.
[2] Hruby, P.: Framework for describing UML compatible Development Processes, <<UML>>'99, Fort Collins, CO, 1999, available at http://www.navision.com, click *corporate info* and *methodology*.
[3] Kruchten, P.: Architectural Blueprints . The "4+1" View Model of Software Architecture, IEEE Software 12(6), 1995, pp.42-50.
[4] Kruchten, P.: The Rational Unified Process, Addison-Wesley, 1998.
[5] H. Ossher and P. Tarr. "Multi-Dimensional Separation of Concerns using Hyperspaces." IBM Research Report 21452, 1999.
[6] D'Souza, D., Wills, A.: Object, Components and Frameworks with UML – the Catalysis Approach, Addison-Wesley Longman, Inc. 1998.
[7] P. Tarr, H. Ossher, W. Harrison and S.M. Sutton, Jr. "N Degrees of Separation: Multi-Dimensional Separation of Concerns." Proceedings of the International Conference on Software Engineering (ICSE'99), 1999