Deriving Best Practices from Development Methodology Base (Part 1)

Bhushan Thakare
Assistant Professor
STES Sinhgad Academy of Engineering, Kondhwa, Pune, Maharashtra, India

Bhushan Bhokse
Assistant Professor
MAEER Maharashtra Academy of Engineering, Alandi (D), Pune, Maharashtra, India

Laxmi Thakare
Assistant Professor
MAEER Maharashtra Academy of Engineering, Alandi (D), Pune, Maharashtra, India

ABSTRACT

There are abundant methodologies for the product development. Out of which only few could have renowned because for their universal importance. But some system or process cannot follow general methodology. Hence some process follows specific methodology e.g., XP follows Agile methodology. Hence we surveyed a detailed review of existing software development methodologies in which mainly on their development processes. The descriptions of methodologies fashioned aimed to provide an abstract and structured description in a way that facilitates their elaborate analysis for the purposes of improving understanding, and making it easier to tailor, select, and evaluate the processes.

General Terms

We are providing a comprehensive review of the processes of a selection of methodologies. The methodologies reviewed in this article are:

Booch, CBD/e (Component Based Development), CRC (Class-Responsibility-Collaborator), Demeter, dX, EUP (Enterprise Unified Process), Fusion, HOOD (Hierarchical Object-Oriented Design), Octopus, OMT (Object Modeling Technique), OOBE (Object-Oriented Business Engineering), OOram, OOSE (Object-Oriented Software Engineering), OPEN (Object-Oriented Process, Environment and Notation), OSA (Object-Oriented System Analysis), RDD (Responsibility-Driven Design), ROPES (Rapid Object-Oriented Process for Embedded Systems), Scrum, Syntropy and XP (Xtreme Programming) etc.

Keywords

Methodology, Phases, Software Development, Object Oriented, Design, Management, Software Engineering.

1. INTRODUCTION

A Software Development Methodology is a framework for applying software engineering practices to develop software-intensive systems. Software development methodologies provide the means for timely and orderly execution of the various finer-grained techniques and methods of software engineering. A software development methodology can be defined as "a recommended collection of phases, procedures, rules, techniques, tools, documentation, management, and training used to develop a system".

Whereas the modelling language provides developers with a means to model the different aspects of the system, the process determines what activities should be carried out to develop the system, in what order, and how. In its most abstract form, a process is a sequence of steps—sometimes deprecatingly called a recipe—that aims to guide its users in applying the modelling language for accomplishing a set of software development tasks. The process thus acts as the dynamic, behavioural component of the methodology, governing the technical development and management sub processes, and therefore encompassing the phases, procedures, rules, techniques, and tools prescribed by the methodology, as well as the issues pertaining to documentation and project management..

And in the next paper named Deriving Best Practices from Development Methodology Base (Paper 2) will see ASD (Adaptive Software Development), BON (Business Object Notation), Catalysis, Coad and Yourdon, Crystal, Doors (Design of Object Oriented Real Time Systems), DSDM (Dynamic Systems Development Method), EROOS (Entity Relationship based Object-Oriented Specification), FDD (Feature Driven Development) , FOOM (Functional and Object Oriented Methodology) , Hodge-Mock, ICONIX, IDEA (Intelligent Database Environment for Advanced Applications), MOSES (Modeling Software and Platform Architecture in UML-2 for Simulation-based Performance Analysis), MERODE (Model Driven, Existence Dependency Relation, Object-Oriented Development), Object COMX (Object Communicating X-Machines), Objecteering, OEP (Object Engineering Process), OOIE (Object-Oriented Information Engineering), OOHDM (Object-Oriented Hypermedia Design Model),OOSC (Object-Oriented Software Construction),OOSP (Object-Oriented Software Process), OPM (Object-Process Methodology), RAD (Rapid Application Development), RUP (Rational Unified Process)/ UML, SDL (Specification and Description Language) / SOMT (SDL-oriented Object Modeling Technique), Shlaer and Mellor methodologies.

2. LITERATURE SURVEY

2.1 Booch:

Booch has modeled object-oriented design as a repeating process (referred to as "The Micro Process") within a lifecycle-level repeating process (referred to as "The Macro Process").

a) Macro Process (Booch). The macro process serves as a controlling framework for the micro process. The macro process tends to follow these steps:

(1) Conceptualization: Establish core requirements for software;

(2) Analysis: Develop a model of the system's desired behavior;

(3) Design: Create architecture for the implementation;

(4) Evolution: Evolve the implementation through successive refinements;

(5) Maintenance: Manage post-delivery evolution.

b) Micro Process (Booch). The micro process tends to follow these steps:

(1) Identify the classes and objects at a given level of abstraction.

(2) Identify the semantics of classes and objects.

(3) Identify the relationships among classes and objects.

(4) Specify the interface and implementation of classes and objects.

2.2 CBD/e: (Component Base Development)

CBD/e is a methodology for allowing organizations to begin component-based development quickly. CBD/e consists of two major parts Process and Analysis and Design (A&D) methodology.

1. **The CBD/e Process**

The CBD/e process is based on repeating plan which is follows:

a) Assessment: To decide what your organization wants from CBD.

b) Planning: Planning takes place, including selecting the project that will be executed and the participants for this particular program.

c) Education: Fundamental training that is required to go to CBD.

d) Infrastructure: Your organization's infrastructure is adjusted to enable component-based development.

e) Execution: This is where the start of implementation occurs.

f) Improvement*:* This is where to review the results.

2) **The CBD/e Analysis and Design Methodology**

A&D methodology uses behavioral analysis and design techniques to facilitate building components. Behavioral analysis and design is based on the concept that the pieces of your system are not just chunks of data that are there to be worked by whatever processes you develop.

The first steps design the specification, which describes the solution without any implementation details. The specification should work for any realization of the system. The realization of the system contains all of the details of a particular implementation, including the user interface design and the details of how the operations of an interface will work. There are three steps in CBD/e which are analysis, design and implementation.

2.3 CRC:

CRC cards characterize objects by class name, responsibilities, and collaborators, as a way of giving learners a direct experience of objects. Three dimensions which identify the role of an object in a design: class name, responsibilities, and collaborators. The class name of an object creates a vocabulary for discussing a design. Responsibilities identify problems to be solved. The solutions will exist in many versions and refinements. A responsibility serves as a handle for discussing potential solutions. Collaborators objects will send or be sent messages in the course of satisfying responsibilities.

Classes are created that describe real world objects that exist in a system. These classes are assigned responsibilities, i.e., data and actions that the class is required to support. A class may fulfill a responsibility by itself, or it may collaborate with some other class to fulfill the responsibility. The advantage is that the design team can easily move the cards around to visualize the design, and modifications to the design can be made quickly by simply replacing cards.

2.4 Demeter:

Adaptive programming deals with specifying the connections between objects as loosely as possible and this is called "structure-shy" programming. The Demeter system and tools are all about "Adaptive" programming.

Law of Demeter (LoD) for functions says that: A method "M" of an object "O" should invoke *only* the methods of the following kinds of objects:

1. Itself
2. Its parameters
3. Any objects it creates/instantiates
4. Its direct component objects

The basic idea is to avoid invoking methods of a member object that is returned by another method. When you do this, you make structural assumptions about the container object that may be likely to change. The container may later need to be modified to contain a different number of the contained objects, or it may end up being changed to contain another object which contains the original component object. If the "returned" object isn't a subpart of the object whose method was invoked, nor of some other object, then it typically is *not* a violation of LoD to invoke a method of the returned object.

Using the LoD you instead ask the container to invoke a method on its elements and return the result. The details of how the container propagates the message to its elements are encapsulated by the containing object.

A side-effect of this is that if you conform to LoD, while it may quite increase the maintainability and adaptiveness of your software system, you also end up having to write *lots* of little wrapper methods to propagate methods calls to its components. This problem is addressed by the Demeter tools, which automate the solution.

In this diagram, Object A has a dependency on Object B which composes Object C. Under the LoD, Object A is permitted to invoke methods on Object B, but is not permitted to invoke methods on Object C.

2.5 dX:

dX is an agile instance of RUP. The dX process consists of the same four phases as RUP. The dX versions of the four phases are:

a) **Inception:** This step focuses on determining the major requirements, producing a preliminary version of the project schedule, and designing a basic architecture for the system.

b) **Elaboration:** This step focuses on iterative and incremental design and coding of higher priority (higher-risk) use cases, until the architecture of the system and the project schedule are stabilized to a point that a release schedule can be reliably worked out.

c) **Construction:** This step focuses on designing and coding the remaining use cases. In dX, the construction phase is a seamless extension of the elaboration phase, with the release schedule being the only milestone signifying the transition between the two.

Transition: This step focuses on gradual introduction of the implemented releases of the system into the user environment, and the subsequent maintenance activities.

2.6 EUP: (Enterprise Unified Process)

EUP extends RUP by adding two new phases and two new disciplines, and also by extending the activities in some of the old disciplines. In EUP, several changes have been made to RUP disciplines also.

1) **New Phases.** The two new phases that EUP has added are:

a) **Production:** Its focus is on keeping the software in production until it is either replaced with a new version, or retired and removed.

b) **Retirement:** It typically involves:

—Identification of the existing system's coupling to other systems;

—Redesign and rework of other systems so that they no longer rely on the system being retired;

—Transformation of existing legacy data;

—Archiving of data previously maintained by the system;

—Configuration management of the removed software;

—System integration testing of the remaining systems.

2) **New Disciplines.** The two new disciplines that EUP has added are:

a) **Operations and support:** This is concerned with issues related to operating and supporting the system, typically associated with the maintenance phase of the generic software development lifecycle.

b) **Enterprise management:** This is concerned with the activities required to create, evolve, and maintain the organization's cross-system artifacts, such as the organization-wide models, software process, standards, guidelines, and reusable artefacts.

2.7 Fusion:

Fusion methodology is the result of the integration, unification and extension mainly of OMT, Booch, Objectory and RDD. Fusion provides consistency and completeness checks between phases to enable orderly and reliable progression. The Fusion process consists of three phases:

1) **Analysis:** This focuses is on what the system does. The models produced in this phase describe:

—Classes and objects of interest found in the application domain, and the relationships that exist among these classes and objects,

—The operations that are to be performed by the system; and

—The proper ordering of these operations.

2) **Design:** This focuses on how the system is to do what has been defined during analysis. The design phase models describe:

—Realization of system operations in terms of cooperating objects;

—How these objects are linked together;

—How the classes to which the objects belong, are specialized and refined

—Detailed particulars of each class's attributes and methods.

3) **Implementation:** This focuses on the actual coding of the system. The system design is mapped to a particular programming environment. Design classes are mapped to language-specific classes and object communications are encoded as implementation methods.

2.8 HOOD: (Hierarchical Object Oriented Design)

Object Oriented Design by Grady Booch and the concepts of Abstract Machines are integrated into a coherent design method called HOOD methodology. A HOOD design consists of a parent-child hierarchy with a root object which represents the system to be designed and a number of objects at different lower levels. An object which is not decomposed into children is called a terminal object. Intermediate objects are called non-terminal objects.

Activities which take place during the definition of a non-terminal object are like problem definition, statement of the problem, analysis and structuring of the requirement data, elaboration of an informal solution strategy, formalization of the strategy, identification of objects, identification of operations, grouping operations and objects, graphical description, justification of the design decisions, formalization of the solution - Object Description Skeleton

HOOD has its own benefits like design clarity, extensibility, mapping to manpower, integration, public domain model, maintainability and reusability. HOOD was designed special consideration: smooth integration with requirements analysis, concurrent development of independent parts, automated code generation and testing, client-server and post-partitioning support.

2.9 Octopus:

The purpose of the method is to create models which elaborate on issues such as which classes and objects exist, the structure, behavior and purpose of those objects and classes, the structure and dynamics of relationships between objects, the structural relationships between classes, the smooth transition from objects and classes to tasks and processes.

Two object-oriented software life-cycle models exist: the *incremental* and the *evolutionary* model. Octopus/UML allows any combination of the two models because real-life projects desire such flexibility.

2.10 OMT: (Object Modeling Technique)

The phases of OMT process are as follows:

1) **Analysis:** This builds a correct and comprehensible model of the real world. Requirements of the users, developers, and managers provide the information needed to develop the initial problem statement.

2) **System Design:** In this high-level structure of the system is chosen. The decisions that will be addressed during system design are organizing the system into subsystems, identifying concurrency, allocating subsystems to processors and tasks, choosing the strategy for implementing data stores in terms of data structures, files, and databases, identifying global resources and determining mechanisms for controlling access to them, choosing an approach to implementing software control, considering boundary conditions, establishing trade-off priorities.

3) **Object Design:** Object design is concerned with fully specifying the existing and remaining classes, associations, attributes, and operations necessary for implementing the system.

2.11 OOBE: (Object Oriented Business Engineering)

The aim in all this work is to make semantics precise and explicit in various OO specifications: business specifications are used to understand and describe businesses independently of any systems to be used for process automation. The hope of this approach is that a single, clear and unambiguous repository of knowledge about the business might be built in order to provide a common reference point from which relevant parties, including business domain experts, software developers, or anyone else associated with examining or changing the systems or processes of the organization, may proceed.

The goal of this approach is to achieve seamlessness across modeling phases, possibly reducing semantic dissonance by using a consistent paradigm.

2.12 OOram:

The OOram method is a frame of reference for a family of objects oriented methodologies. It captures the essence of object orientation, which is to model interesting phenomena as a structure of interacting objects. It offers the role model as a powerful abstraction that supports a very general separation of concern. The notion of role model synthesis supports the construction of complex models from simpler ones in a safe and controlled manner, and offers many opportunities for the systematic application of reusable components.

2.13 OOSE: (Object Oriented Software Engineering)

OOSE process consists of three main phases, each producing a set of models:

(1) **Analysis:** Focuses on understanding the system and creating a conceptual model of it. This phase consists of two non-sequential, iterative subphases:

(1.1) **Requirements analysis**

(1.2) **Robustness analysis:** Aims at modeling the structure of the system in terms of interface, data, and control objects, and also by specifying the subsystems.

(2) **Construction:** This phase consists of two subphases:

(2.1) **Design:** Aims at modeling the run-time structure of the system, and also the interobject as well as intraobject behavior necessary to realize the requirements.

(2.2) **Implementation**: Aims at building the software.

(3) **Testing:** Focuses on verifying and validating the implemented system.

2.14 OPEN: (Object Oriented Process, Environment and Notation)

OPF (OPEN Process Framework) is a process metamodel defining five classes of components and guidelines for constructing customized OPEN processes.

(1) **Work products**: Any significant thing of value (document, diagram, model, class, application) developed during the project;

(2) **Languages**: The media used to document work products, such as natural languages, modeling languages such as UML or OML, and implementation languages such as Java, SQL, or CORBA-IDL;

(3) **Producers**: Active entities (human or nonhuman) that develop the work products;

(4) **Work units**: Operations that are performed by producers when developing work products. Work units are activity, task, technique.

(5) **Stages**: Durations or points in time that provide a high-level organization to the work units. Stages are milestone and stage with duration.

2.15 OSA: (Object Oriented System Analysis)

In OSA, the system is modeled from three perspectives: object structure, object behavior, and object interaction. An OSA model of the system consists of three parts:

(1) **Object relationship model**: This describes objects and classes as well as their relationships with each other and with the "real world";

(2) **Object-behavior model:** This provides the dynamic view through states, transitions, events, actions, and exceptions (analogous to a state-transition diagram);

(3) **Object-interaction model:** This specifies possible interactions among objects.

2.16 RDD: (Responsibility Driven Design)

RDD process starts when a detailed requirements specification of the system has already been provided.

RDD models an application as a collection of objects that collaborate to fulfill their responsibilities. Responsibilities include two key items:

(1) The knowledge an object maintains; and

(2) The actions an object can perform.

The process is divided into two phases:

1) **Exploratory Phase:** Discover the classes, determine what behavior the system is responsible for and assign these responsibilities to specific classes; and determine what collaborations must occur among classes of objects.

2) **Analysis Phase (RDD).** The activities primarily performed are factoring the responsibilities, identifying possible subsystems of objects and modeling the collaborations among objects and determining class protocols and completing the specification of classes, subsystems of classes, and client-server contracts.

2.17 ROPES: (Rapid Object Oriented Process for Embedded Systems)

A good process is needed to:

– Improve the product quality

– Save calendar and work time

– Make development efforts more predictable, schedulable, and reliable

– Enable rapid response to looming risks

A ROPES consists of five phase which are as analysis, design, translation, testing and party.

2.18 Scrum:

The name emphasizes the importance of teamwork in the methodology and is derived from the game of rugby.

The Scrum process consists of three phases:

(1) **Pregame:** Concerned with setting the stage for the iterative-incremental development effort; this phase consists of the following subphases:

(1.1) **Planning:** Focuses on producing an initial list of prioritized requirements for the system, analyzing risks associated with the project, estimating the resources needed for implementing the requirements, obtaining the resources necessary for starting the development, and determining an overall schedule for the project;

(1.2) **Architecture/high-level design:** Determines the overall architecture so as to accommodate the realization of the requirements identified so far;

(2) **Development (game):** Focuses on iterative and incremental development of the system.

(3) **Post-game:** Focuses on integrating the increments produced and releasing the system into the user environment.

2.19 Syntropy:

Syntropy does suggest a definite process through the levels of modeling it prescribes. The three distinct, yet integrated, model levels used in Syntropy are:

(1) **Essential model**: This models the problem domain, totally disregarding software as a component of the system;

(2) **Specification model**: This abstractly models the requirements of the software system, treating the system as a stimulus-response mechanism, and assuming a computing environment with unlimited resources;

(3) **Implementation model**: This models the software system's run-time structure and behavior in detail, taking into account considerations pertaining to the computing environment, and elaborating on how the software objects should communicate.

Each model may be expressed along structural and behavioral views. There are three kinds of views in Syntropy:

—**Type view:** This provides the structural view by describing object types, their static properties and their relationships.

—**State view:** This provides the behavioral view by describing the possible states for each object type and the way it responds to stimuli by changing state and generating responses.

—**Mechanism diagram:** This is solely used in the implementation model for describing the flow of messages among objects in response to stimuli.

2.20 XP: (Xtreme Programming)

The XP lifecycle consists of six phases:

(1) **Exploration:** This focuses on developing an initial list of high-level requirements, and determining the overall design through prototyping.

(2) **Planning**: This focuses on estimating the time needed for the implementation of each requirement, prioritizing the requirements, and determining a schedule for the first release of the system.

(3) **Iterations to first release:** This focuses on iterative development of the first release of the system, using the specific rules and practices.

(4) **Productionizing:** This focuses on system-wide verification and validation of the first release, and its deployment into user environment.

(5) **Maintenance:** It is the time for system evolution, and therefore is the time when the project is considered to be in its normal state.

(6) **Death:** This focuses on closing the project and conducting post-mortem review and documentation.

2.21 MERODE: (Model Driven, Existence Dependency Relation, Object-Oriented Development)

A typical Merode analysis or conceptualization consists of three views or diagrams: a so called existence dependency graph (EDG) similarly to a UML class diagram, a propriarity concept namely an object event table (OET) and a group of finite state machines.

MERMAID is an object-oriented domain modeling tool to create enterprise models using an UML class diagram, an object-event table and finite state machines. The CASE tool checks view consistency, has XMI support and code generation abilities. The method is still incomplete

2.22 Object COMX: (Object Communicating X-Machines)

COMX is a new design methodology that contains the formal specification technique of communicating X-machines. Communicating X-machines are typed finite state machines that can communicate with each other via typed channels. Object COMX comprises five main phases, each with specific deliverables using a reachability analysis:

1) **Requirements Analysis** using soft systems analysis, use cases and traditional methods of customer and user interviews etc. to describe the proposed system in the most effective way.

2) **Object Modeling** in the external behavioral view, where the system is modeled directly as a set of interacting objects.

3) **Formal Specification** of the internal behavior of the objects, each as a communicating X-machine object.

4) **Verification** of the CXMO specification occurs via a use case-reachability analysis. Various system properties such as reachability, freedom from deadlock etc. can be gathered from the use case - reachability tree.

5) Object-oriented Design of the system for **implementation** purposes in which each CXMO is classified and any inheritance and aggregation identified.

2.23 Octopus:

The purpose of the method is to create models which elaborate on issues such as which classes and objects exist, the structure, behavior and purpose of those objects and classes, the structure and dynamics of relationships between objects, the structural relationships between classes, the smooth transition from objects and classes to tasks and processes.

Two object-oriented software life-cycle models exist: the *incremental* and the *evolutionary* model. Octopus/UML allows any combination of the two models because real-life projects desire such flexibility.

2.24 OMT: (Object Modeling Technique)

The phases of OMT process are as follows:

1) **Analysis:** This builds a correct and comprehensible model of the real world. Requirements of the users, developers, and managers provide the information needed to develop the initial problem statement.

2) **System Design:** In this high-level structure of the system is chosen. The decisions that will be addressed during system design are organizing the system into subsystems, identifying concurrency, allocating subsystems to processors and tasks, choosing the strategy for implementing data stores in terms of data structures, files, and databases, identifying global resources and determining mechanisms for controlling access to them, choosing an approach to implementing software control, considering boundary conditions, establishing trade-off priorities.

3) **Object Design:** Object design is concerned with fully specifying the existing and remaining classes, associations, attributes, and operations necessary for implementing the system.

2.25 HOOD: (Hierarchical Object Oriented Design)

Object Oriented Design by Grady Booch and the concepts of Abstract Machines are integrated into a coherent design method called HOOD methodology. A HOOD design consists of a parent-child hierarchy with a root object which represents the system to be designed and a number of objects at different lower levels. An object which is not decomposed into children is called a terminal object. Intermediate objects are called non-terminal objects.

Activities which take place during the definition of a non-terminal object are like problem definition, statement of the problem, analysis and structuring of the requirement data, elaboration of an informal solution strategy, formalization of the strategy, identification of objects, identification of operations, grouping operations and objects, graphical description, justification of the design decisions, formalization of the solution - Object Description Skeleton

HOOD has its own benefits like design clarity, extensibility, mapping to manpower, integration, public domain model, maintainability and reusability. HOOD was designed special consideration: smooth integration with requirements analysis, concurrent development of independent parts, automated code generation and testing, client-server and post-partitioning support.

2.26 ICONIX:

ICONIX is a software development methodology which predates RUP, XP and Agile software development. The ICONIX process is UML Use Case driven but more lightweight than RUP. ICONIX provides sufficient requirement and design documentation, but without analysis paralysis.

A principal distinction of ICONIX is its use of robustness analysis, a method for bridging the gap between analysis and design. This process makes the use cases much easier to design, test and estimate. Essentially, the ICONIX Process describes the core logical analysis and design modelling process. However, the process can be used without much tailoring on projects that follow different project management.

The ICONIX process is split up into four milestones. At each stage the work for the previous milestone is reviewed and updated. These are as: **Requirements** review, **Preliminary Design** Review, **Detailed Design** Review, **Deployment**.

Unit tests are written to verify the system will match up to the use case text, and sequence diagrams. Finally code is written using the class and sequence diagrams as a guide.

All material on each page should fit within a rectangle of 18 x 23.5 cm (7" x 9.25"), centered on the page, beginning 2.54 cm (1") from the top of the page and ending with 2.54 cm (1") from the bottom. The right and left margins should be 1.9 cm (.75”). The text should be in two 8.45 cm (3.33") columns with a .83 cm (.33") gutter.

3. CONCLUSION

This article presents a comprehensive review of software development methodologies, but from the perspective of process. There are existing reviews of methodologies from the perspective of modelling languages available in the literature, but there is little up-to-date synthesis material available that focuses on process.

Attempts at integration, unification, and standardization have actually aggravated the problems of complexity and inconsistency, giving rise to a new family of lightweight, agile methodologies, In every methodology, there are features to exploit and pitfalls to avoid, many of which are direct or indirect consequences of the method used in developing the methodology or the circumstances surrounding the development. Choosing the right method for developing the target methodology is therefore of utmost importance.
Macintosh, use the font named Times. Right margins should be justified, not ragged.

4. ACKNOWLEDGMENTS

Our thanks to the expert Mr. Ankush Thakare who have contributed towards development of the template.

5. REFERENCES

[1] Booch, G., Martin, R. C., and Newkirk, J. 1998. *Object Oriented Analysis and Design with Applications*, 2nd ed. (Unpublished). Addison Wesley, Reading, MA. Chapter on RUP and dX is available online at http://www.objectmentor.com/resources/articles/RUPvs XP.pdf.

[2] Cook, S. and Daniels, J. 1994. *Designing Object Systems: Object-Oriented Modeling with Syntropy*. Prentice-Hall, Englewood Cliffs, NJ.

[3] Lano, K., France, R., and Bruel, J. 2000. A semantic comparison of Fusion and Syntropy. *Comput. J. 43*, 6, 451–468.

[4] Beedle, M., Devos, M., Sharon, Y., Schwaber, K., and Sutherland, J. 2000. SCRUM: A pattern language for hyperproductive software development. In *Pattern Languages of Program Design 4*. N. Harrison, B. Foote, and H. Rohnert, Eds. Addison-Wesley, Reading, MA. 637–651.

[5] Schwaber, K. 1995. SCRUM development process. In *Proceedings of the Conference on Object-Oriented Programing Systems, Languages, and Applications (OOPSLA'95) Workshop on Business Object Design and Implementation*, available online at http://jeffsutherland.com/oopsla/schwapub.pdf.

[6] Firesmith, D. and Henderson-Sellers, B. 2001. *The OPEN Process Framework: An Introduction*. Addison-Wesley, Reading, MA.

[7] Graham, I., Henderson-Sellers, B., and Younessi, H. 1997. *The OPEN Process Specification*. ACM Press/Addison-Wesley, New York, NY.

[8] Henderson-Sellers, B. and Unhelkar, B. 2000. *OPEN Modeling with UML*. Addison-Wesley, Reading,MA. REENSKAUG, T., WOLD, P., AND LEHNE, O. 1996. *Working with Objects*: *The OOram Software Engineering Method*. Manning Publications, Greenwich, CT.

[9] D'souza, D. F. and Wills, A. C. 1995. Catalysis—practical rigor and refinement: Extending OMT, Fusion, and Objectory. Available online at http://www.catalysis.org/publications/papers/1995-catalysisfusion.pdf.

[10] Derr, K.W. 1995. *Apply OMT: A Practical Step-by-step Guide to Using the Object Modeling Technique*. Cambridge University Press, New York, NY.

[11] Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P. 1994. *Object-Oriented Development: The Fusion Method*. Prentice-Hall, Englewood Cliffs, NJ.

[12] Coleman, D., Jeremaes, P., and Dollin, C. 1992. *Fusion: A Systematic Method for Object-Oriented Development*. Hewlett Packard Laboratories.

[13] Booch, G. 1986. Object-oriented development. *IEEE Trans. Softw. Eng. 12*, 2 (February), 211–221.

[14] Booch, G. 1994. *Object Oriented Analysis and Design with Applications*. Benjamin/Cummings, Redwood City, CA.

[15] Kirby McInnis and Castek. An Overview of CBD/e.

[16] Castek Introduces CBD/e Bringing the Power of Components to Your Organization.

[17] Robert Biddle, James Noble and Ewan Tempero. Reflections on CRC cards and OO Design.

[18] Kathleen Arnold Gray and Mark Guzdial and Spencer Rugaber. Extending CRC Cards into a Complete Design Process

[19] David Bock. The Paperboy, The Wallet and The Law of Demeter

[20] Karl Lieberherr. Adaptive Object Oriented Software The Demeter Method

[21] Jagdish Bansiya and Carl Davis. A Hierarchical Model of Object Oriented Design Quality Assessment.

[22] Pawel Martenka and Bartosz Walter. Hierarchical Model for Software Design Quality.

[23] Aslett M. An Overview of the HOOD Method.

[24] Domiczi Endre, Farfarakis Rallis, Ziegler Jürgen. Release of Ocopus/UML.

[25] Octopus/UML Notation Summary. Farfarakis Rallis.

[26] Trygve Reenskaug. Working with OOram Software Engineering Method.

[27] Trygve Reenskaug. MVC and DCA.

[28] Neil Maiden, Stephen Morris, Wolfgang Emmerich. Object Oriented Analysis Design.

[29] Koichiro Ochimizu. Object Oriented Software Development.

[30] Bruce Douglass. ROPES:Rapid Object Oriented Process for Embedded Systems.

[31] Bruce Douglass. The Telelogic Harmony/ESW Process for Real Time and Embedded Development.

[32] Linda Rising and Norman Janoff. The Scrum Software Development Process for Small Teams.

[33] Jeff Sutherland, Fully Distributed Scrum: The Secret Sauce of Hyperproductive Offshored Development Teams.

[34] Mike Beeedle. Scrum: An Extension Pattren Language for Hyperproductive Software Development.