Deriving Best Practices from Development Methodology Base (Part 2)

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 commence with basic definitions software development methodologies. We then provide a comprehensive review of the processes of a selection of methodologies. The methodologies reviewed in this article are:

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 etc.

Keywords

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

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 subprocesses, 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 last paper named Deriving Best Practices from Development Methodology Base (Paper 1) we have seen already 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) methodologies.

2. LITERATURE SURVEY

2.1 ASD: (Adaptive Software Development)

The ASD constitutes five phases process which are as follows:

a) Project initiation: This step focuses on understanding the project's objectives and estimating its size and scope, exploring the constraints.

b) Iterative development phases:

a) **Adaptive cycle planning:** This step focuses on setting time frames for the project and the development cycles, defining the components that should be developed, assigning the components to cycles, and scheduling the iterations.

b) **Concurrent component engineering:** This step focuses on concurrent design and implementation of the components assigned to individual cycles.

c) **Quality review:** This step focuses on conducting group reviews of the components produced and rectifying the problems confronted.

c) **Final Q/A and release:** This step focuses on validating the produced system and deploying it into the working environment.

2.2 BON: (Business Object Notation)

Basically the BON process is used to build the deliverables, which provide static and dynamic descriptions of the system. Description of each task is as follows:

a) **Delineating System Borderline.** The scope of the system and its subsystems is identified, user metaphors are compiled, and the system functionality is defined as typical usage scenarios.

b) **Listing Candidate Classes.** This task is mainly concerned with extracting a list of candidate classes from the problem domain.

c) **Selecting Classes and Grouping into Clusters**. Concepts will then be modeled as classes, which are then grouped into clusters. This task also involves the identification of relationships: inheritance, client-server, and aggregation, among the classes in a cluster, and among the clusters themselves.

d) **Defining Classes.** Define each class in terms of its state, its behavior, and the general rules that must be obeyed by the class and its clients.

e) **Sketching System Behavior.** The dynamic model of the system is elaborated.

f) **Defining Public Features.** The informal class descriptions filled into the class charts during defining classes are translated into formal class interfaces (features).

g) **Refining the System.** This task begins the design part of the BON process, and therefore includes a repetition of many activities already performed for the analysis classes, now applied to new design classes.

h) **Generalizing.** This task concerns improving the inheritance hierarchy of the classes by factoring common state and behavior into deferred (abstract) superclasses.

i) **Completing and Reviewing the System.** This typically involves reviewing and perfecting the static and dynamic models, syntactic verification of the classes, and checking the consistency of class invariants and the pre- and post-conditions of routines.

2.3 Catalysis:

Catalysis proposes a set of process patterns to be selected and applied according to the characteristics of the project at hand. It consists of the following activities:

(1) Identify and model the requirements.

(2) Develop the system specification.

(3) Develop the architectural design.

(4) Develop the component internal design.

Analysis usually starts by modeling the problem domain as a collection of classes, with their own inter-relationships and interactions. Then the system is added to the context, treated like another problem domain type, whose state, operations (functionality), and behavior are carefully modeled. The focus is then shifted onto the system itself, modeling it as a collection of components, again with their own interrelationships and interactions. Finally, each component is modeled as a collection of implementation-level classes, interfaces, and off-the-shelf components; yet again with their own interrelationships and interactions.

2.4 Coad-Yourdon :

Coad-Yourdon methodology suggests techniques for translating the design models into code.

a) **Analysis:** This step consists of five principal activities:

(1) Finding abstract classes and concrete classes;

(2) Identifying generalization-specialization and whole-part relationships among classes;

(3) Identifying subjects (partitions/subsystems);

(4) Defining attributes, and instance-connections;

(5) Defining class operations and invocations of operations.

Results of these activities are reflected in a special Class-and-Object Diagram that is the pivotal model of the system.

b) **Design:**

Components, provides certain functionality needed to realize the requirements and implement the system, are listed below:

(1) **Problem Domain Component:** During OOD, result of the analysis phase is improved and enriched with implementation detail.

(2) **Human Interaction Component:** This handles sending and receiving messages to and from the user.

(3) **Task Management Component:** This is needed to implement multiple threads of control.

(4) **Data Management Component:** This provides the infrastructure to store and retrieve objects.

2.5 Crystal:

In Crystal, projects are categorized according to their size and the criticality of the system being produced. Four levels of criticality have been defined, based on what might be lost because of a failure in the produced system: comfort (C), discretionary money (D), essential money (E), or life (L). Crystal methodologies put heavy emphasis on communication among people involved in the project.

The project lifecycle in Crystal Clear consists of the following phases:

a) **Chartering:** It involves forming the development team, performing a preliminary feasibility analysis, shaping and fine-tuning the development methodology, and developing an initial plan for the project.

b) Cyclic delivery: It involves team updates and refines the release plan, implements a subset of the requirements through one or more program-test-integrate iterations, delivers the integrated product to real users, and reviews the methodology adopted and the project plans.

c) Wrap-up: This involves the software product is deployed into the user environment, and post deployment reviews and reflections are performed.

2.6 DOORS: (Design of Object Oriented Real Time Systems)

The objective of DOORS is to develop a method for the design of real-time (RT) systems that: is practical for systems of large scale and complexity; includes support of object-oriented techniques for achieving reusability in implementations; helps with visualizing and testing design concepts during the early stages when the big decisions are being made.

This output of DOORS will be useful to designers of real-time systems to help them do their job better, and to designers of future tools to help them identify the capabilities required.

Approach is to develop a loosely-coupled set of (1) modified versions of existing OO and RT methods and (2) new methods. Important tasks are: development of a new design method that focuses on a concept time threads, the method is called Timethread-Driven design; visualization techniques for OO frameworks and design guidelines for frameworks; research into integrating the ideas of our design method into CASE tools; and new visualization techniques for displaying the structure and behavior of a large system in a limited screen area.

2.7 DSDM: (Dynamic Systems Development Method)

DSDM is referred to as a configurable process framework, rather than a methodology. In customizing the process framework, the development team also has to set up a strict time-constrained plan for the development. In DSDM, stringent constraints are set on time and resources, leaving the requirements (functionality) as the only variable parameter of the project.

DSDM is thus deemed especially suitable for projects with highly volatile requirements. The DSDM process consists of 7 phases which are:

a) Pre-project: This focuses on providing the necessary resources for starting the project, along with a plan for the feasibility study.

b) Project-proper: The five main phases of the DSDM are applied:

- **Sequential phases:** This primarily is concerned with studying the business domain and performing a preliminary analysis of the system i.e., feasibility studies.
- **Iterative phases:** The functional model, design and build, and implementation iterative phases iteratively and incrementally analyze, design, code, and deploy the system through evolutionary prototyping:
- **Post-project:** The focus is on system maintenance.

2.8 EROOS: (Entity-Relationship based Object Oriented Specification)

The power of EROOS is based on three important aspects:

a) A combination of behavior modeling and structure modeling;

A full integration of these two modeling techniques will provide the specification method with more powerful composition and decomposition properties. This will lead to better structured, changeable, maintainable and reusable software systems.

b) Declarative rather than operational description of the software system;

The description of a software system at the beginning will be focused on what the system should do, whereupon a gradual transformation will be achieved during the design phase to describe *how the system should do it*.

c) And separation of the system kernel from the system functionality.

This will provide a flexible and changeable system the heart of which is a solid kernel. This will allow to adapt the existing system to the ever changing user demands instead of restarting from scratch. By its layered approach, the EROOS method aims at increasing modularity, extendibility, reusability and maintainability of software systems.

2.9 FDD: (Feature Driven Development)

FDD is based on expressing and realizing the requirements in terms of small user-valued pieces of functionality called features. Each feature is a relatively fine-grained function of the system. The subprocesses of the FDD process are:

9.1. Sequential subprocesses: In this the problem domain is modeled, requirements are identified as hierarchical lists of features, and development planning is performed. The subprocesses are as follows:

a) Develop an overall model: This focuses on building a mainly structural model of the problem domain, called the object model. This model mainly consists of full-featured class diagrams.

b) Build a features list: This focuses on identifying the required functionality of the system.

c) Plan by feature: This focuses on scheduling the features for development, and then assigning the feature sets (activities), and the classes in the object model, to developers.

9.2. Iterative subprocesses: During this phase, strands of design and- build iterations start off as each chief programmer selects the set of features that should be developed in each of the iterations performed under his supervision. The subprocesses are as follows:

a) Design by feature: This focuses on determining how the features in the work package should be realized at run-time by interactions among objects.

b) Build by feature: This focuses on coding and unit-testing the necessary items for realization of the features in the work package.

2.10 FOOM: (Functional and Object Oriented Methodology)

The FOOM process consists of the following phases:

10.1.Analysis: Analysis is concerned with requirements elicitation and problem-domain modeling, this phase consists of two activities which are:

a) **Data modeling**: This focuses on identifying and modeling the class structure of the problem domain.

b) **Functional analysis**: This focuses on identifying and modeling the functional requirements of the system;

10.2.Design:

Design is concerned with designing implementation-specific classes and adding structural and behavioral detail to the models, this phase consists of the following stages:

a) **Defining basic methods:** This focuses on specifying primitive operations for the classes.

b) **Top-level design of application transactions:** This focuses on identifying transactions; each transaction is in fact a unit of functionality performed by the system in realization of its functional requirements.

c) **Interface design:** This focuses on designing a menu-based user interface for the system; suitable classes are then defined in order to implement these menus.

d) **Input/output design:** This focuses on designing the input forms/screens and the output reports/screens of the system, and defining classes for implementing them.

e) **Design of system behavior:** This focuses on providing detailed specifications for the transactions, and elaborating on object interactions and operations of the classes;

10.3. Implementation: FOOM is mainly targeted at data-intensive information systems. Targeting data intensive systems has also resulted in a slack attitude towards behavioral design of the system.

2.11 Hodge-Mock:

Hodge-Mock methodology is for use in a simulation and prototyping laboratory, the sole purpose of which was to explore the feasibility of introducing higher levels of automation into air traffic control systems. The Hodge-Mock process consists of five phases:

11.1.Analysis: This focuses on refining the requirements and identifying the scope, structure and behavior of the system. This phase in turn consists of four subphases:

a) **Requirements analysis:** This focuses on eliciting the requirements of the system.

b) **Information analysis:** This focuses on determining the classes in the problem domain, their interrelationships, and the collaborations among their instances.

c) **Event analysis:** This focuses on identifying the behavior of the system through viewing the system as a stimulus-response machine.

d) **Transition to system design:** This focuses on providing a more detailed view of the collaborations among objects.

11.2.System design: This focuses on adding design classes to the class structure of the system and refining the external behavior of each of the classes.

11.3.**Software design:** This focuses on adding implementation-specific classes and details to the class structure of the system, and specifying the internal structure and behavior of each class.

11.4.**Implementation:** This focuses on coding and unit testing.

11.5.**Testing***:* This focuses on system-level verification and validation.

2.12 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.

2.13 IDEA: (Intelligent Database Environment for Advanced Applications)

The goal of the IDEA methodology is to produce a coherent body of concepts, languages and tools, together with an execution environment, suitable for the design and development of database applications requiring intelligent features.

The IDEA addresses the analysis, design, prototyping, and implementation of modern database systems applications, taking benefit of modern approaches developed in the context of database design, but also in the broader area of object-oriented software engineering. IDEA emphasis on both deductive rules and active rules, which significantly enrich the semantics supported within database applications.

2.14 MOSES: (Modeling Software and Platform Architecture in UML-2 for Simulation-based Performance Analysis)

MOSES is a full-lifecycle, OO software development methodology which encompasses not only technical aspects of OOA/D but also project management, business planning, maintenance and product enhancements. MOSES is the new implementation of our methodology based on the UML 2 metamodel which allows the modeling of software and platform architecture for simulation-based performance analysis within the same modeling environment.

2.15 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 similarly to a UML class diagram, a proprietarily concept namely an object event table 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.

2.16 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.

2.17 Objecteering:

Objecteering/UML has just added dynamic behavior rules to UML Profiles. This technique has provided Objecteering/UML with the qualities for which it is renowned: the benefit of interactive model control, the power of automatic transformation of analysis models into design models, the performance of its code generators and the flexibility of its parameterization.

Objecteering/UML is already realizing all the promise of MDA such as truly reusable models, reusable technical rules, progressiveness of applications, through the productivity gains, thanks to the availability of off the shelf generators, simple exchange of generation rules, quality gains, through the systematizing of model and code writing rules, reactivity, through the flexibility of parameterization, complete traceability throughout all the development phases.

2.18 OEP: (Object Engineering Process)

The OEP has been developed to create a usable process guideline for all important project stakeholders. The main addressees are:

- Persons who are involved in the project leading
- Persons who are involved in quality assurance and project controlling
- Project members

The process model is an important planning document for the project management. They are able to deduce sensible und useful activities. The optimal profit will be achieved by the project management, if they checking all including activities, results, mile stones, etc. The project management can also use or deduce important coherences and dependencies to define the order and urgency of the project activities. In spite of the support the project management has the responsibility for the project planning and execution.

For persons who are involved in quality assurance and project controlling the process model is a useful guideline to check planning and execution. Numerous descriptions of activities and results are contained as well as also hint how the quality can be valued.

2.19 OOIE: (Object Oriented Information Engineering)

It aims to enable an enterprise to improve the management of its resources, including capital, people and information systems, to support the achievement of its business vision. Information engineering has many purposes, including organization planning, business re-engineering, application development, information systems planning and systems re-engineering.

There are two variants of information engineering.

1) **DP-driven**: The DP-driven variant of Information engineering was designed to enable IS Departments to develop information systems.

2) **Business-driven**: Information Engineering was extended into strategic business planning and developed the business-driven variant of IE.

2.20 OOHDM: (Object Oriented Hypermedia Design Model)

OOHDM uses abstraction and composition mechanisms in an object oriented framework to, on one hand, allow a concise description of complex information items, and on the other hand, allow the specification of complex navigation patterns and interface transformations.

1) **Domain Analysis:** This describes the collection of classes, with their own inter-relationships and interactions.

2) **Navigational Design:** This describes the navigational structure of a hypermedia application in terms of navigational contexts such as Nodes, Links, Indices, and Guided Tours.

3) **Abstract Interface Design:** The abstract interface model is built by defining perceptible objects in terms of interface classes. Interface objects provide navigational objects with a perceptible appearance.

4) **Implementation:** This focuses on the actual coding of the system.

2.21 OOSC: (Object Oriented Software Construction)

"OOSC", presents object technology with a special emphasis on addressing the software quality factors of correctness, robustness, extendibility and reusability. It starts with an examination of the issues of software quality, then introduces abstract data types as the theoretical basis for object technology and proceeds with the main object-oriented techniques: classes, objects, generosity, inheritance, Design by Contract, concurrency, and persistence.

The structured paradigm focuses on decomposing behaviors. The OO paradigm focuses on objects, classes, and inheritance. The two paradigms do not mix well. While the OO paradigm tightly integrates the development phases of analysis, design and implementation, intrinsic differences between these phases should not be blurred. OO methods are compatible with prototyping efforts, especially those constructed in order to elucidate otherwise unknown requirement fragments.

2.22 OOSP: (Object Oriented Software Process)

A Process Pattern provides guidance on how to effectively carry out discrete tasks within the development process. That is, a Process Pattern represents a structured approach to a given task that has been proven to yield effective results. Process Pattern includes describing the Forces, Initial Context and Solution.

Types of Process Patterns

1. **Task process patterns:** This depicts the detailed steps to perform a specific task, such as the Technical Review and Reuse First process patterns

2. **Stage process patterns:** This depicts the steps, which are often performed iteratively, of a single project stage.

3. **Phase process patterns:** This depicts the interactions between the stage process patterns for a single project phase, such as the Initiate and Delivery phases.

2.23 OPM: (Object Process Methodology)

OPM's modeling strength lies in the fact that only one type of diagram is used for modeling the structure, function, and behavior of the system. The single diagram type is called the object-process diagram (OPD). The OPM process consists of three high-level subprocesses:

(1) Initiating focuses on preliminary analysis of the system, determining the scope of the system, the required resources, and high-level requirements.

(2) Developing focuses on detailed analysis, design, and implementation of the system.

(3) Deploying focuses on the introduction of the system into the user environment, and the subsequent maintenance activities.

2.24 RAD: (Rapid Application Development)

RAD proposes that products can be developed faster and of higher quality by using workshops or focus groups to gather requirements, prototyping and user testing of designs, re-using software components, following a schedule that defers design improvements to the next product version, keeping review meetings and other team communication informal. RAD usually embraces object-oriented programming methodology, which inherently fosters software re-use.

Advantages of RAD are such as buying may save money compared to building, deliverables sometimes easier to port , development conducted at a higher level of abstraction, early visibility, greater flexibility, greatly reduced manual coding, increased user involvement, possibly fewer defects, possibly reduced cost, shorter development cycles, standardized look and feel.

2.25 RUP: (Rational Unified Process)

RUP is use-case-driven, a feature inherited from OOSE. UML is used as the modeling language in RUP; therefore RUP has also been mistakenly called the UML Methodology. The overall RUP development *cycle* consists of four *phases*:

(1) **Inception**: This focuses on defining the objectives of the project, especially the business case;

(2) **Elaboration**: This focuses on capturing the crucial requirements, developing and validating the architecture of the software system, and planning the remaining phases of the project;

(3) **Construction**: This focuses on implementing the system in an iterative and incremental fashion based on the architecture developed in the previous phase;

(4) **Transition**: This focuses on beta-testing the system and preparing for releasing the system.

2.26 SDL: (Specification and Description Language)

SDL is an object-oriented, formal language defined by ITU-T. SDL provides graphical symbols to represent flow lines, input, output, tasks and decisions. SDL is specification and description of the behavior of telecommunications systems. An SDL specification defines system behavior in a stimulus/response fashion, assuming that both stimuli and responses are discrete and carry information.

The SOMT (SDL Object Modeling Technique) method provides a framework that shows how to use object oriented analysis and SDL-based design together in a coherent way. The framework is based on describing the analysis and design of a system as a number of activities. Each activity deals with some specific aspects of the development process.

SOMT consists of five major activities:

• Requirements analysis

• System analysis

• System design

• Object design: The purpose of the object design is to define in detail the functionality including the behavior of all objects.

• Implementation

2.27 Shlaer and Mellor Methodology:

This process covers the analysis, design, and implementation phases of the software development lifecycle. It can be broken down into eight steps:

(1) Partition the system into domains according to the four domain types: problem domain, application independent services, physical architecture, and physical implementation;

(2) Analyze the application (problem) domain;

(3) Confirm the analysis through static and dynamic verification;

(4) Extract the requirements for the application-independent service domains;

(5) Analyze the service domains;

(6) Specify the components of the architectural domain;

(7) Build the architectural components;

(8) Translate (implement) the analysis models of relevant domains into the architectural components.

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 S. Thakare who have contributed towards development of the template.

5. REFERENCES

[1] 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.

[2] D'souza, D. F. and Wills, A. C. 1998. *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison-Wesley, Reading, MA.

[3] Coad, P. and Yourdon, E. 1991a. *Object-Oriented Analysis*, 2nd ed. Yourdon Press/Prentice-Hall, Englewood Cliffs, NJ.

[4] Coad, P. and Yourdon, E. 1991b. *Object-Oriented Design*. Yourdon Press/Prentice-Hall, Englewood Cliffs, NJ.

[5] Cockburn, A. 2004. *Crystal Clear: A Human-Powered Methodology for Small Teams*. Addison-Wesley, Reading, MA.

[6] DSDM Consortium. 2003. *DSDM: Business Focused Development*, 2nd ed. J. Stapleton, Ed. Addison-Wesley, Reading, MA.

[7] Palmer, S. R. and Felsing, J. M. 2002. *A Practical Guide to Feature-Driven Development*. Prentice-Hall, Englewood Cliffs, NJ.

[8] Kabeli, J., and Shoval, P. 2003. Software analysis process—which order of activities is preferred? An experimental

[9] Comparison using FOOM methodology. In *Proceedings of the IEEE International Conference on Software-Science, Technology and Engineering*, 111–122.

[10] Shoval, P. and Kabeli, J. 2001. FOOM: Functional- and object-oriented analysis and design of information systems: An integrated methodology. *J. Database Manage. 12*, 1 (January–March), 15–25.

[11] 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.

[12] Stefan Baelen, Johan Lewi, Eric Steegmans and Helena Riel. EROOS: An Entity-Relationship Based Object-Oriented Specification Method.

[13] Stefano Ceri, Piero Fraternali and Stefano Paraboschi. The IDEA Tool Set.

[14] Mark Birkin, Martin Clarke, Phil Rees. MOSES: Modelling and Simulation for e-Social Science.

[15] Aaron McDaid and Neil Hurley Model-based Overlapping Seed ExpanSion.

[16] Edwards, J.M., MOSES: Methodology for Object-oriented Software Engineering of Systems| Henderson-Sellers, 1994a, BOOKTWO of Object-Oriented Knowledge: The Working Object, Prentice Hall, Sydney, 616pp.

[17] Scott W. Ambler An Introduction to Process Patterns.

[18] Martin and Odell. Object-Oriented Information Engineering.

[19] Daniel Schwabe, Gustavo Rossi and Simone D.J. Barbosa Systematic Hypermedia Application Design with OOHDM.

[20] Rubén Tous. Updating Hypermedia Object Oriented Design Method (OOHDM) to systematic the process of designing Web applications.

[21] Daniel Schwabe and Gustavo Rossi. The Object-Oriented Hypermedia Design Model (OOHDM).

[22] Bertrand Meyer. Object Oriented Software Construction.

[23] SDL by Telelogic Specification and description language (SDL). Telecommunication Standardization Sector of ITU.

[24] Intelligent Database Environment for Advanced Applications http://www.iai.uni-bonn.de/~idea/.

[25] Agile Software Requirement. Retrieved from http://www.agiledata.org/essays/evolutionaryDevelopment.html.

[26] Rapid Application Development http://www.mariosalexandrou.com/methodologies/rapid-application-development.asp.

[27] Rapid Application Development http://www.sce.carleton.ca/rads/doors.html.

[28] Objectneering http://www.objecteering.com/.

[29] Object Oriented Software Engineering http://www.oose.de/oep/main/eng_zielgruppe.htm.