

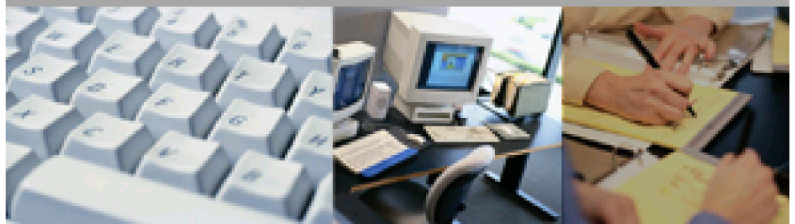
The Web Production



ST. PETERSBURG - RUSSIA

TWP Software Development Framework

White Paper



38, 11 th line of Vasilievsky
Island , 199178
Saint Petersburg, Russia



Contents

CONTENTS	2
THE WEB PRODUCTION SOFTWARE DEVELOPMENT FRAMEWORK	4
1 ABSTRACT	4
2 INTRODUCTION	4
3 EXISTING SPECIFICATIONS OF SOFTWARE DEVELOPMENT PROCESS	5
3.1 MICROSOFT SOLUTION FRAMEWORK (MSF).....	6
3.2 RATIONAL UNIFIED PROCESS (RUP).....	7
3.3 OBJECT-ORIENTED FRAMEWORK (OOF).....	8
4 STATIC STRUCTURE OF THE SOFTWARE DEVELOPMENT PROCESS	8
4.1 BASIC FEATURES OF THE PRODUCT-FOCUSED OBJECT-ORIENTED PROCESS SPECIFICATION	8
4.2 STATIC STRUCTURE OF SOFTWARE DEVELOPMENT PROCESS AND MANAGEMENT ARTEFACTS.....	10
5 DYNAMICS OF SOFTWARE DEVELOPMENT ARTEFACT- DEVELOPMENT PROCESSES	14
6 SOFTWARE DEVELOPMENT PROCESS	14
6.1 PROJECT PHASES AND MILESTONES	14
6.2 ANALYSIS.....	15
6.3 DESIGN.....	17
6.4 DEVELOPMENT (CODING)	20
6.5 STABILIZATION	21
6.6 DEPLOYMENT (OPERATION TESTING).....	23
6.7 SPIRAL STRUCTURE	24

7	REFERENCES.....	25
7.1	THE OBJECT-ORIENTED SPECIFICATION OF DEVELOPMENT	25
7.2	ISO 9001 COMPATIBILITY	26
8	SUMMARY	26
9	AUTHOR	27
10	APPENDIX A	28
11	APPENDIX B	29
12	APPENDIX C	30
13	APPENDIX D	31
14	APPENDIX E	32
15	APPENDIX F.....	33
16	APPENDIX G.....	34
17	APPENDIX H.....	35
18	APPENDIX I.....	36

***The Web Production*[®] Software Development Framework**

1 Abstract

The present paper is intended for TWP potential and existing partners, as well as the whole personnel of the Company. It is aimed at acquainting the Reader with TWP Software Development Framework (TWPPSDF) and showing that the products of the Company are of the highest quality and meet all the requirements of such world quality standards as ISO9000 and ISO15504 SPICE (*Software Process Improvement and Capability Determination*). The Reader will also be given the possibility of ascertaining that all our products fully comply with the quality standards of third-party producers, whose software are used in our systems to provide custom solutions. In other words, it will be proved that not a single product of ours is ever under the standards of those producers.

The present paper is only an introductory document that does not give a detailed description of all the aspects of the software development process in TWP. It discusses only the general questions of structuring and ordering the development process that reflect the policy of the Company in matter of software development and quality assurance. A technical documentation and guidelines that thoroughly specify the development process for whole divisions and single workstations of the Company are available in TWP repository in a more detailed form.

2 Introduction

The software development process, as actually performed, is so complex that, if an accurate description of it were provided, it would be difficult and tedious to read, understand and follow out. To go round this problem, TWP invites the Reader to have a look at the specification of a process framework that describes all allowable processes, instead of trying to analyze a concrete process development scenario. Such a framework is abstract, yet precise. To meet the demands of specific development problems, this framework is reused each time for creating specific development processes. The concrete

development processes being analysed can be represented at the required level of granularity. This solution significantly simplifies the description of concrete development processes, because their complexity is localized in the abstract form, i.e. the main process framework.

This paper is structured in the following way. The first section explains the existing specifications of software development processes. The next sections explain the main ideas of the TWP Software Development Process Specification. The last section outlines the structure of the TWP Software Development Framework.

3 Existing specifications of Software Development Process

The software development model in use in TWP relies on well-known, tested and approved software developing methods such as MSF Application Development and RUP. Nevertheless, it has its own particularities.

As a big number of small projects (that is, projects, duration of which does not exceed one month) involving one or two people in the development phase are simultaneously in progress in the Company, *good labour coordination* and *strictly specified project gradual development* (stage by stage) are required. On another side, relatively big products requiring more than 100 men-hours are developed too in the company. This fact obliges keeping a very careful watch over the analysis, design and planning of projects, in order to minimize the gap between the expected project results and the factual ones.

The following conditions and specificities are taken into account to ensure an efficient control of the development process.

- *A fixed project timeframe.* For all projects practically, the development and deployment deadlines are strictly specified by the clauses of the contract related to them.
- *Special Agreements.* Continuation of a project after the deadlines specified in its contract is not the usual rule. When it happens, it takes place only within the frame of a new agreement.

- *Novelty of the data-domain.* Projects are developed in different domains and, for this reason, often comport elements of substantial novelty for the developers. In such cases, the risk of getting an incomplete or inadequate first version of a considered system is very high.
- *Guaranteed final success.* To enlarge and stabilize the circle of Customers, a guaranteed final success is required for each project individually, even if it implies internal extra outlays for the complete development of the project.

The model of software development process in use in TWP combines the traditional multi-stage model with the iterative milestone-based model proposed by MSF.

In the waterfall model (also named conveyer model), the development process (that is, the lifecycle of a software product) is divided into phases that successively substitute one the other.

3.1 Microsoft Solution Framework (MSF)

In MSF model the development process is divided into cycles (loops, rounds or turns as one may call them), each of which consists of four phases:

- Analysis,
- Planning,
- Development,
- Stabilization.

A milestone ends each phase. A milestone can result in any of the following:

- a project vision,
- a project requirements specification,
- a programme code,
- a product release.

After release of an intermediate version, the process goes into a new loop of the spiral. An illustration of this is provided on Figure 1.

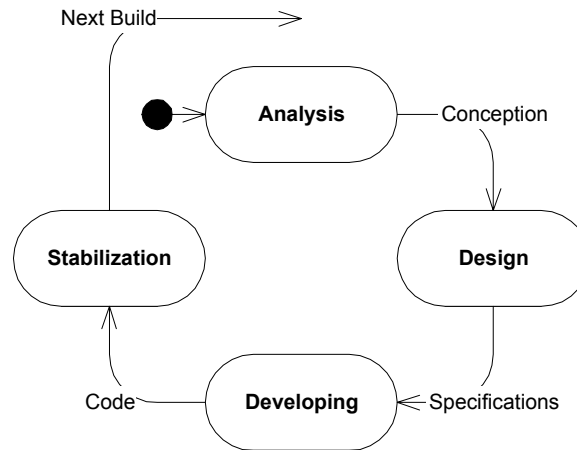


Figure 1. MSF model of development process

3.2 Rational Unified Process (RUP)

The object-oriented specification of the *Rational Unified Process* is illustrated in [Figure 2](#). The figure illustrates the scenario of the requirements, analysis and design workflows of the Rational Unified Process. The figure shows the evolution of the software as a series of interactions between the worker (an operator of the system such as the system analyst, the use-case specifier, the architect or the designer) and the software development artefacts of the Rational Unified Process.

A worker who uses the Rational Unified Process is responsible for calling the constructors of the software development artefacts in the order illustrated in [Figure 2](#).

Constructors and quality-assurance methods generate various messages between the objects.

3.3 Object-Oriented Framework (OOF)

The traditional specification of a development process is typically illustrated with a graph of tasks, techniques, software development artefacts and activities.

Tasks are small behavioural units that usually result in a software development artefact. Examples of tasks are construction of a Use-Case model, construction of a class model and coding.

Techniques are formulas for performing tasks, for example, object design using CRC cards, functional decomposition and programming in Visual Basic.

Software development artefacts are final or intermediate products resulting from software development, for example, a Use-Case model, a class model, or source code.

Activities are units that are larger than task units. Activities typically include several tasks and software development artefacts. Examples of activities are requirement analysis, logical design and implementation.

In general, the traditional specification of a development process cannot cover all the possible combinations of activities and software development artefacts without becoming overly complex. This is similar to experience from within software development, where the object-oriented approach can deal with the complexity of large software solutions better than the traditional structured approach.

4 Static structure of the Software Development Process

4.1 Basic features of the Product-focused Object-oriented Process Specification

Software development and management artefacts produced during a software development process are considered objects with various methods and attributes. Evolution during software development is represented as collaborations between software development artefacts, management artefacts

and users of the method. The object-oriented specification of the software development process distinguishes between artefacts (that is, artefact types) and their representations (instances of the considered artefacts). A software development artefact determines information about a software product. Examples of software development artefacts are Use Cases, software architecture, object collaborations and class descriptions. A management artefact determines information about a management product, such as a project and a team. Software development artefacts can be very abstract, such as the vision for the software system, or very concrete, such as the source code. The representation determines how the artefact is presented. For example, a Use-Case model is represented by a Use-Case diagram; a state model can be represented by a state-chart diagram, an activity diagram or a state transition table. The object interaction model can be represented by a set of sequence diagrams or a set of collaboration diagrams. Various design methods typically recommend a suitable representation of each software development artefact. However, the choice often depends on the concrete situation, and it is sometimes advisable to leave the final decision about the representation to a user of the method. An artefact has a representation, properties, responsibilities, methods, relationships to other artefacts and attributes.

Artefact types have two kinds of methods:

- *Constructors*, which are methods describing how to create an artefact.
- *Quality-assurance methods*, such as completeness and consistency checks.

Artefacts have instance-specific attributes:

- Name;
- version;
- representation, which typically contains a diagram, a table or a text;

- status, such as draft, completed, tested;
- references to other software development and management artefacts;
- and attributes such as who created and modified the artefact and when.

In addition, artefact types have type-specific attributes:

- the purpose,
- the recommended representation
- the owner of the artefact type.

Artefacts may have other additional attributes and methods than those mentioned above. [Figure 3](#) illustrates the object-oriented specification of a software development artefact type with attributes and methods. The inheritance diagram of the artefact types is in [figure 5](#).

The fact that the inheritance tree is incomplete allows for flexibility throughout the process and for creating new artefacts to match different kinds of development processes.

4.2 Static structure of Software Development Process and Management Artefacts

This section specifies the static relationships between software development and management artefacts. The static structure is based on the pattern for structuring project repositories with UML-design artefacts. This section outlines how the pattern is applied to create a static structure of the framework for describing UML-compatible development processes. A software system can be represented from various viewpoints and at various levels of granularity [Figure 5](#). In each view and at each level of granularity, a UML-compatible system can be described by four artefacts:

- static relationships between classifiers 2,
- dynamic interactions between classifiers,

- classifier responsibilities,
- classifier lifecycles.

The artefact called *classifier relationships* specifies static relationships between classifiers. The artefact called *classifier interactions* specifies interactions (an instance of a scenario) between classifiers. The artefact *classifier* identifies the classifier and specifies classifier responsibilities and other static classifier properties, for example, a list of classifier operations with preconditions and postconditions, and a list of classifier attributes that can be read and set. The classifier lifecycle specifies classifier state machine and dynamic properties of classifier interfaces, for example, the allowable order of operations and events.

Examples of views are the logical view, the collaboration view, the deployment view and the analysis view. The logical view describes the logical structure of the product in terms of subsystems and classes and their responsibilities, relationships and interactions. The collaboration view identifies types of collaborations with actors of the system, subsystems, classes, components and nodes. The deployment view describes the physical structure of the system in terms of hardware devices and their responsibilities, relationships and interactions. The analysis view describes the logical structure of the product in terms of analysis subsystems, objects and their responsibilities, relationships and interactions. The analysis view differs from other views in the way that the software entities in the analysis view do not specify the software system precisely. The purpose of the analysis view is to record preliminary or alternative solutions to design problems, and to record requirements or user's view of the system. Analysis objects may - but do not always - correspond to logical or physical software entities existing in the product. Examples of levels of granularity are the system level, the subsystem level and the class level. The system level of granularity describes the context of the software system. The system level specifies the responsibilities of the system being designed and the responsibilities of the other systems that collaborate with it, responsibilities of physical devices and software modules

outside the system, and static relationships, along with the dynamic interactions between them and the system being designed. The subsystem level of granularity describes subsystems, software modules and physical devices inside the system, along with their static relationships and dynamic interactions. The class level of granularity describes the detailed design of the subsystems in terms of classes and objects, and their relationships and interactions. The software product can be represented by additional views, such as the testing view and the view of reusable elements. The software product can be specified at additional levels of granularity, such as the tier level for systems with layered architecture and the organizational level for business systems. At each additional level of granularity and in each additional view, the software product is specified by static relationships between classifiers, dynamic interactions between classifiers, classifier responsibilities and classifier lifecycles. The semantics of these additional software development artefacts are out of the scope of this paper.

It is recalled that the object-oriented model distinguishes between the information itself (called software development or management artefact) and its representation. Software development artefacts can be represented by UML diagrams, tables or text [Figure 6](#). The artefact classifier relationships is represented by a UML static structure diagram, a Use-Case diagram, a deployment diagram and a component diagram, if classifiers are classes, Use Cases, nodes and components, respectively. The artefact classifier interactions are represented by a UML sequence diagram and by a collaboration diagram. The UML Notation Guide describes only interaction diagrams in which classifiers are objects; it does not describe interaction diagrams in which classifiers are Use Cases, subsystems, nodes or components. The artefact classifier is represented by a CRC card, Use-Case template, structured text or table. The artefact classifier lifecycle is represented by a UML statechart diagram, activity diagram, a state table, a Backus-Naur form and a Nassi-Schneidermann diagram. *Management artefacts* can be represented by project diagrams, tables or text. The artefact classifier relationships is represented by an organizational chart, if the classifiers are roles or teams; or by a PERT chart,

if the classifiers are tasks and projects. Overeager UML users can use class diagrams, in which the classes have a stereotype «task» and «project». The artefact classifier interactions is represented by a Gantt chart if the classifiers are projects or by text if the classifiers are roles and teams. The artefacts classifier and classifier lifecycle are represented by table or text.

[Figure 7](#) specifies the static structure of software development artefact types at three levels of granularity and in the logical and collaboration views. Each software development artefact type specifies certain information about the software system. The structure uses the pattern described above, and the result is a regular structure, which allows consistent customization of the design specification. In simple cases, the specification consists of only a small subset of the software development artefacts identified in [Figure 5](#) and [Figure 6](#). Conversely, if software designers have to specify something unusual or unexpected, such as the things not covered by the method, the specification is extended by adding additional views and additional levels of granularity.

[Figure 8](#) illustrates the application of the pattern for structuring management artefacts. The pattern can be used to structure two kinds of *management product*: teams and projects. Management artefacts can be shown as stereotyped classes in UML, such as *team*, *role*, *project* and *task*. The artefacts *team relationships* and *role relationships* specify the organizational structure at two levels of granularity. The artefact team specifies the responsibility of the team and the artefact role specifies the role of the team member. Examples of roles are developer, programme manager, product manager, user education and logistics. The artefacts team interactions and role interactions specify scenarios -interactions between teams and team members, which are responses to various events. The artefacts project and task specify static properties of projects and tasks. The artefact project relationships and task relationships specify static relationships between projects and tasks. These artefacts can be represented by a PERT chart. The *PERT chart* shows the task dependencies, which are the most important static relationships between tasks. The artefact *task interactions* specifies a project scenario in terms of starting and finishing tasks. Accordingly, the artefact *project interactions*

specifies the project scenario in terms of starting and finishing projects. These artefacts are typically represented by Gantt charts, but they might be represented by UML sequence diagrams as well. *Gantt charts* show the *task constructors*, which are the most important messages between tasks. It can be noted that every project generates a number of artefacts not captured by the pattern. Examples of such artefacts are a glossary, minutes of meetings, reviews, comments and notes. These artefacts do not describe a product, and therefore they cannot be structured using the pattern. These artefacts can be related to any other software development or management artefact. For example, the glossary is related to the artefact system, the minutes of meetings are related to the artefact project. In order to reuse them in a consistent way, they have specified types in the process repository. They are illustrated, for example, in the inheritance diagram in

5 Dynamics of Software development Artefact-Development Processes

The previous section described the static structure of software development and management artefacts. In the object-oriented specification of a development process, evolution is seen as collaborations between artefacts, and between artefacts and members of a development team. Software development artefacts can be created and completed in various orders depending on the features of various design methods.

6 Software Development Process

6.1 Project Phases and Milestones

TWP Software Development Process divides into five phases:

- Analysis,
- Design,
- Development
- Stabilization
- Deployment (and Operation testing)

MSF model comprises only the first four phases for the following reasons. The deployment phase is considered as a separate process and thus not included in the development one. TWP keeps to the principle of specifying the development and the deployment of a project by a same agreement. Both phases are therefore considered as parts of the elaboration process. A sequence of the five above-mentioned phases constitutes a cycle and is called a turn or a loop. The phases of a turn always go in the specified order.

The succession of phases in a loop is logical in the sense that a subsequent phase depends on the previous one. This does not mean that the phases in a turn are executed in time strictly one after the other. On the contrary, phases in a loop can be executed in parallel and partially or fully cumulated in time.

Each phase ends with its main milestone, which is a characteristic token of that phase. A *milestone* is an identifiable, instant event that goes along with the appearance and the record of some alienable material, i.e. an artefact. This can be a document, a programme code or a protocol. Apart from the main milestones, which are visible for the external world, phases can also have internal milestones, which can be visible or invisible for the outside world. The required set of milestones and their sequence depend on a concrete project. This set is defined during the planning of the development process. Visibility means here that the artefacts and the milestones are discussed with the Customer for agreement. Therefore, the main milestones and the externally visible internal milestones of a phase are checkpoints that are attached to the workflow appended to the project requirements specification.

Further in the present paper, by *external* item it will be assumed, an item that is *to be submitted or suggested to the Customer for approval*.

6.2 Analysis

The *analysis phase* is the first one in each loop. The main milestone of this phase is named ***Project Vision Approving*** or simply *Project Vision*. It consists in registering a document or a series of documents titled *Project Vision*. A project vision must include the next compulsory parts:

1. **Introduction.** This section specifies the technical justification and the status of the project.
2. **Formulation of the main problem.** This section gives a description of the data-domain, clearly formulates the central problem the project is aimed at solving, a brief survey of the alternative solutions to it if any and the main particularities of the project.
3. **Objective of the project.** This section provides a clear, concise laconic and easily verifiable statement of the goals of the project. Usually, a project has one main goal and several auxiliary or intermediate goals.
4. **Categories of user.** The section provides a classification of the end-users and enumerates the project results they expect.
5. **Used approaches.** This section enumerates and briefly describes the methods, means, techniques and procedures that are to be used to reach the project goals.
6. **Success criteria.** This section specifies some quantitative, efficiently computable criteria that serve for estimation of the level of completion of the project.
7. **Typical examples.** Few informal examples (one to three) that illustrate the use of the expected solutions.
8. **Artefacts.** This section enumerates and briefly describes the alienable material that constitutes the result of the project.

The main goal of the project vision is to formulate a common understanding of the problem and the ways to its solution that would be approved by all the involved parties: the Customer, the Development Team and TWP Management.

An internal milestone of the Analysis phase may be any of the following:

- **Investigation.** A report of the preliminarily performed investigation of the business process, the issues of which the project is aimed at dealing with.
- **Requirements specification.** A formalized document reflecting in a standardized form the content of the project vision.

In some cases the requirements specification is elaborated at the first preparation stage instead of the second stage of the work on the project.

For some types of project, the requirements specification coincides with the project vision.

6.3 Design

The Design phase is the decisive stage that determines the success of the entire project. The main milestone of this phase is the approving of the project specifications. **Project specifications** are a set of documents of different kind and other material, among which the next ones are compulsory and of major importance:

- **Functionalities Specification.** A thorough description of all the functions of the chosen solution. In all cases where it is possible, it is strongly recommended to use modelling tools that support the Unified Modelling Language (UML). The use of other means such as natural language, formal languages, computer modelling tools etc., is allowed only in cases, where it is quite impossible to use UML for composing the functionalities specification. Internal and external project specifications are distinguished. **External** specifications need to be submitted to the Customer for approval, while **internal** ones do not. All the internal specifications are necessarily elaborated during the design phase; exception should be made for some of them that may be carried over to subsequent phases.

- **Project workflow.** A list of the phases and their constituent parts, which comports the main and intermediate milestones attached to concrete dates, executors and alienable material of each milestone. It is strongly recommended to use *Microsoft Project* (a software tool) to compose the project workflow (See example at Appendix I). Any other tool of the same kind is allowed only if the Customer makes objection against the use of *Microsoft Project*.

A document titled *Project Workflow* is composed at the first preparation stage, appended to the Agreement and is submitted to the Customer for approval. The workflow for the design phase can coincide with the project workflow that is visible for the Customer. This happens when all the milestones of the phase workflow are external and coincide with the stages of the requirements specification. More often, the project workflow is an internal document, which specifies in detail all the phases, milestones and involved workers. In such cases the external project workflow, a less detailed document, is composed in accordance to the wishes of the Customer while the internal project workflow is composed with *Microsoft Project*.

The general planning of the whole project during the design phase does not exclude the planning the of each phase in particular.

Apart from the above-mentioned obligatory parts, the Project Specification can include these facultative ones:

- **Prototype.** A computer model of the application if the project implies the elaboration of an application. The prototype can be operational. It can also be just a prototype of the application interface or the skeleton of the future product without its full interface in the case, where the application has an interface. If an operational prototype is implemented, it takes the place of the Functionalities Specification.

- **Internal Language.** A description of an internal command language is recommended for all multi-function applications that permit various scenarios for their use. Such a description is extremely desirable for applications that have a programme interface (API).
- **Supplementary Requirements.** The enumeration and description of quantitative restrictions and (or) special requirements, with which the adopted solution should be consistent. This could be, for example, the stability requirement for a data-security system. Supplemental requirements can be presented as an addendum of the Project Vision or the Requirements Specification as well.
- **Testing Plan.** A plan that comprises a set of tests and the description of how to apply them in order to measure such important characteristics of the application being developed in the frame of the project as reliability, functional fullness and usability. More information on the testing procedure is provided further in [section 6.5](#). In some cases, the Testing Plan is not included in the Project Specifications but becomes instead the first intermediate milestone of the Stabilization phase.
- **Deployment Plan.** A description of the deployment procedure of and the procedure of demonstrating the serviceability of the application being developed. Besides, in cases where this is stated in the general plan of the project, the Deployment Plan can include a plan for training the users of the application and the Customer's specialists (operating staff). On that same basis it can also include a project plan for composition of user documentation. The Deployment Plan, in some cases, is not included in the project specifications but becomes instead the first internal milestone of the operation-testing phase.

Each component of the project specification can be a separate intermediate milestone of the design phase. The functionalities specification, Project Workflow and Deployment Plan are external material. The Testing Plan, on the contrary, is an internal document in general.

6.4 Development (Coding)

The main milestone of the Development phase is named **Ready-to-use Code**. Although the main artefact of the project, the application programme code, is created during this phase, it represents not more than 20% of the whole resources required by the project. The main milestone *Ready-to-use Code* implies that all the functionalities that were specified in corresponding documents are fully implemented and the serviceability of the entire application has been proved on several examples.

Many research reports show that the bigger a project is, the smaller becomes the time (in percentage) required for pure coding. The value of 20% mentioned above is an average value for typical projects.

Intermediate milestones of the Development phase depend on the project type and the development software used. They include the next parts:

- **Logical Project.** It consists of a model of the application objects (services), which is a specification of the interfaces for the facilities (methods and properties) provided by the services (that is, the objects).
- **Database General Structure.** Fixes the data presentation used in the project. This part defines the table structures and the relationships between them if a standard relational Database Management System is used.
- **Interface Design.** Fixes the look (the appearance) of the static part of the interface (forms and dialogue windows). Fixes the set of menu items and the user's language if it is so stated by the Contract.

- **Physical Project.** This part wraps the services into components and describes the protocols used for interaction of the components one with the other in distributed applications.

Those of the intermediate milestones of the Development phase that are external (for instance, the Logical Project) are related to external specifications and are performed at the planning stage.

TWP model of Software Development Process recommend to use Case tools, which are suitable for visual design and automatic code generation. The better is to use tools that support UML. Such tools make the result of many milestones (such as the functionalities specification, logical project, database general structure, physical project and even the Ready-to-use code) a model of the application or parts of the application. These model or parts are composed by means of the Case tool being used.

6.5 Stabilization

The main milestone ending of the Stabilization (testing) phase is named **Release**. This milestone is considered fulfilled if a complete, ready-for-replication set of project artifacts that were specified during the planning phase has been made and is available.

The main activities in the Stabilization phase are all-side testing and bug fixing. In other words, the main activity in this phase consists in making sure that all errors have been detected and corrected. The following complex testing methods are distinguished:

- **Reliability testing.** The main object of this type of test is to discover *uncaught runtime exceptions* if any. This is to say that reliability tests are aimed at “hacking” the application. Typical examples of the methods used for such tests consist in entering data that are out of the allowed range of values, altering the right order of operations in a scenario and creating situation, where some quantitative restrictions are not observed.

- **Functionality testing.** This type of test makes sure that allowed input data produce the expected correct output data that fully complies with the specifications. The expected correct output corresponding to an allowed input should be obtained earlier by other means than the application being tested application independent means). A typical technique used for functionality testing consists in entering the boundary values of the allowed input range.
- **Usability Testing.** This type of test allow for making sure that the suggested way in which the application should be used are satisfying for a user, that performs, for his work, a sequence of operations defined in the scenarios. For instance, during the usability testing the following parameters are tested: application reactivity, data security, recovery ability after error occurrence, how easily the interface can be understood etc...
- **Programme source code verification.** This step makes sure the source code complies with the adopted corporative coding standards.
- **Documentation check.** This step allows verifying if the documentation for the product is ready for duplication and delivery to the Client in cases, where the product is developed on contract with a Customer.

The Stabilization phase includes intermediate milestones that are obligatory if the project implies that the corresponding material should be issued. These are:

- **Error absence status confirmation.** No errors detected. This implies that none of the tests specified by the testing plan detects errors. All previously detected and corrected errors are reflected in the database as fixed.

- **Source Code Readiness.** All programme source codes are verified for compliance with the adopted coding standards (comments, identifiers, text structure).
- **Documentation readiness.** The entire documentation for the product is ready for duplication. This means that it has been brought to concordance with the corrections and recommendations of the technical writer. Useful advices can be found in the document *Guidelines for composing user documentation*.

Composition of the user documentation can be carried over to the operation-testing phase, provided the Customer agrees with that.

The technical tasks related to the documentation prepress and its duplication (if the latter was specified in the plan) can be performed by specialists that are not members of the team responsible for the project development.

6.6 Deployment (Operation testing)

The Deployment phase is the final one. Its main milestone is named **Project Completed**. This milestone is characterized and conditioned by the next tokens:

- The Customer (or end-users) is effectively using the developed solution in practice;
- Collaboration relationship with the Customer goes on in full agreement, without discordance;
- All the results of the project are documented, archived and prepared for future reuse.

The operation-testing phase is similar to the Deployment phase but is not the final one. These two phases are different in the part of the documents describing the level of satisfaction of the Customer with the results of the project. By the end of the Deployment phase all the recorded complaints and

recommendations of the Customer are entirely satisfied. In the case of the Operation Testing phase, by the end of it the Customer's complaints, remarks and recommendations are just recorded and filed in special remark reports.

In the deployment phase the Customer can request intermediate milestones.

Typical milestone of this phase are listed below:

- **Application Deployed (Solution Provided).** The elaborated application (solution) has been deployed on the customer side and tested. It has successfully passed the specified testing procedure.
- **Database Loaded.** Operation of the solution is carried out with the Customer's real data.
- **Users Trained.** Users are now working with the developed application without the permanent control of the developers. Intervention of the developers and help from them is provided exclusively upon receiving the corresponding request issued by the users.

Recommendations and guidelines for performing an operation testing can be found in the document *ASDH Infrastructure Deployment*.

6.7 Spiral structure

In TWP model of Development Process, a full process usually consists of one or several loops, each of which includes five phases. This assures the interactive character of the whole development, which is indispensable for making the final success guaranteed. It means that a considered project, predictably, will be completed within the strictly specified timeframe. The spiral loops of the structure are not performed in a rigorous sequence in time; they are partially superposed instead. This assures a reduction of the project overall development time but at the cost of a slight increment of the required number of men-hours. Different spiral structures are allowed. They can differ by their

number of loops, the superposition of some of their phases or a shift of their loops in time.

7 References

7.1 *The Object-Oriented Specification of Development*

This section describes the framework for an object-oriented specification of development processes using the pattern of four artifacts. The structure of the process specification framework is illustrated in [Figure 9](#). The artifact called artifact relationships specifies the static relationships between software development and management artifacts. It can be represented by a UML static structure diagram, where classes and objects have stereotypes, such as *class relationships*, *class lifecycle*, *project* and *team*. The artifact called artifact interactions specifies the development scenario. This artifact can be represented by a UML Sequence diagram or a UML Collaboration diagram. In the Rational Unified Process these scenarios are called workflows. The artifact called artifact type specifies the purpose, constructor, quality assurance, and specific attributes of software development and management artifacts. The artifact called artifact lifecycle specifies the artifact states and the events that change the artifact state, such as creation, completion and approval. The lifecycle shown in [Figure 9](#) is an illustrative example; different software development and management artifacts have different and often more complex lifecycles.

This is not the so-called document-based process. Document-based processes focus on the documents delivered. The documents often have a form, which is well defined by templates and checklists. Document-based processes have a bad reputation, because the success of a project has often been measured in terms of the documents being delivered on time. For software projects, this encourages problems, such as bureaucracy, slow delivery and requirements drift. Many software development and management artifacts in the framework, such as class lifecycle, are too small to be useful documents delivered

separately, for example, at a project milestone. For pragmatic reasons, software development and management artifacts can be combined together to make useful documents. The framework specified in this article can be characterized as information-focused, rather than document-focused. Software development and management artifacts are essentially pieces of information. Sometimes the information is not represented physically and might exist only as a mental model. Even for such artifacts, the framework concepts can be applied.

7.2 ISO 9001 compatibility

ISO 9000 defines quality criteria and the key practices that concrete development processes should meet. The object-oriented specification of a development process is directly evaluated against the standards simply by comparing the quality-assurance methods of the software development and management artifacts with the key practices and quality criteria defined by the process definition standard.

8 Summary

TWP model of Software Development Process comports the following particularities:

- From the beginning, an intermediate version of the future product, a prototype, is planned. All the required phases are performed for the prototype but, by definition, it will not take the place of the final product.
- In cases, where a portfolio of light projects is available, a continuous work cycle can be organized for the main developer roles in order to minimize their downtime by a more efficient management of their workload. This concerns the analysts, programmers and specialists in logistics.
- The Customer regularly verifies the evolvement of his project and discusses the alienable material of the main and external intermediate milestones.

- The Customer takes really part in testing the provided solution starting from the Operation Testing phase.

9 Author

Ivan Tkatchenko

cto@thewebproduction.com

10 Appendix A

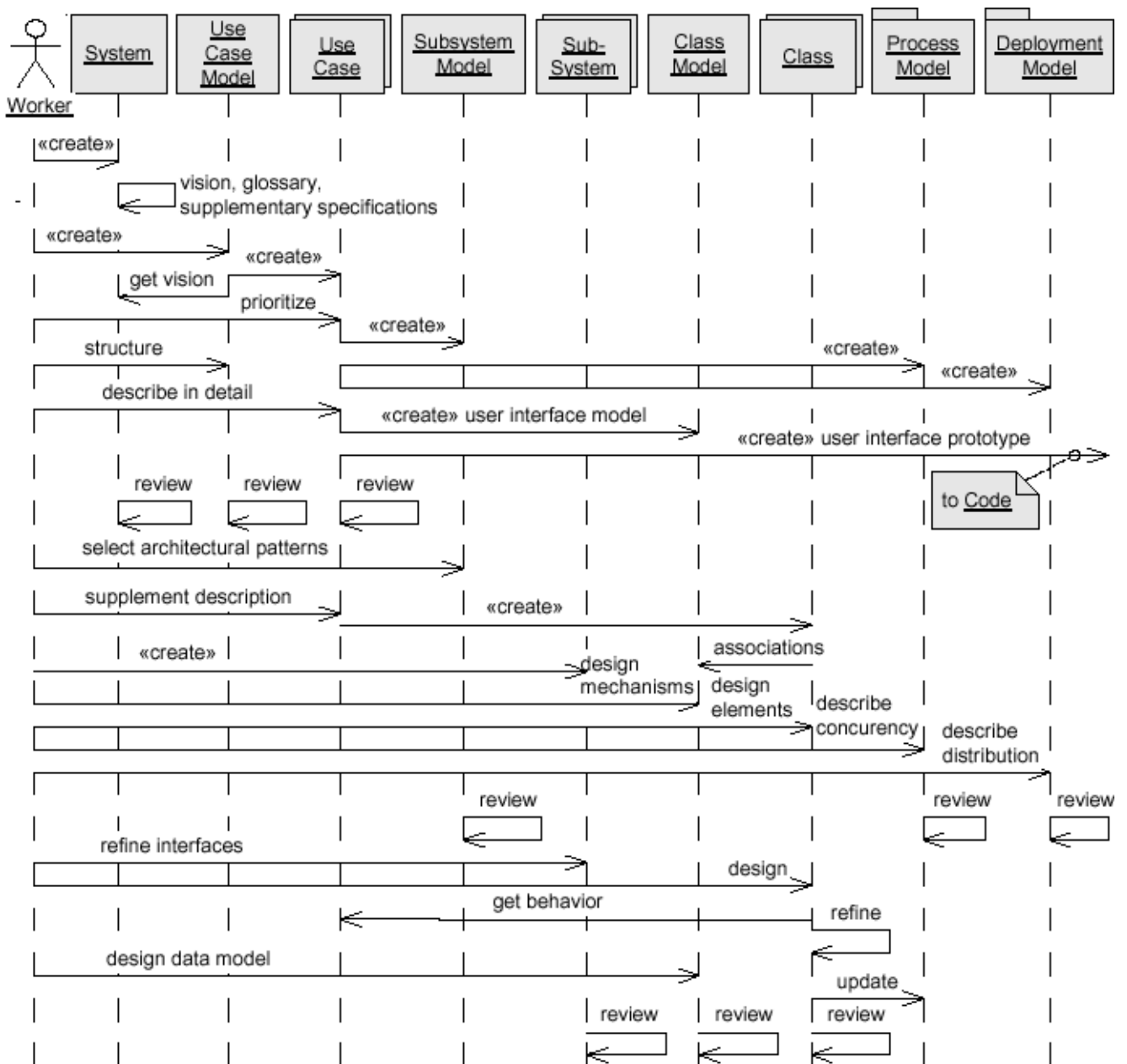


Figure 2. RUP model of development process

11 Appendix B

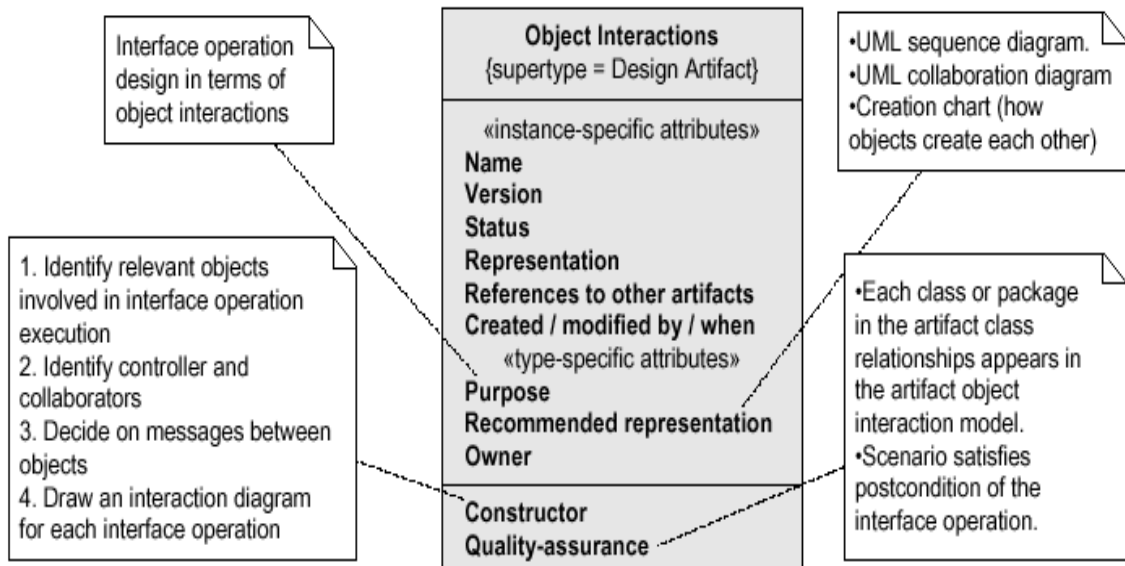


Figure 3. Object-oriented model of artifact type

12 Appendix C

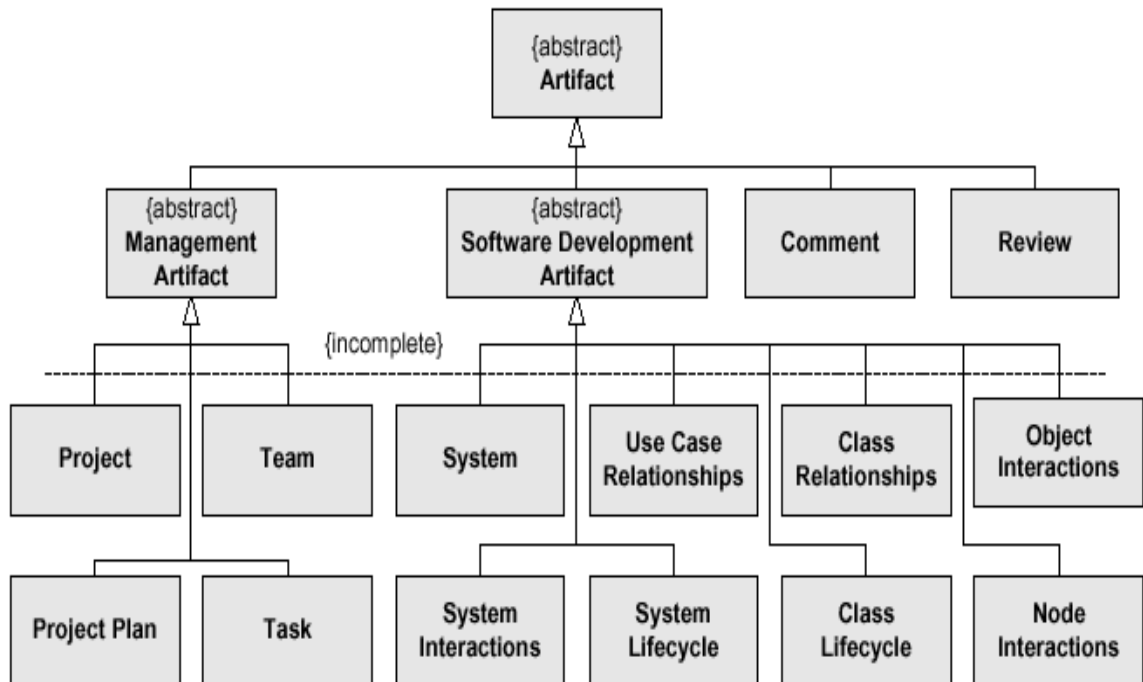


Figure 4. Inheritance diagram of the artifact types

13 Appendix D

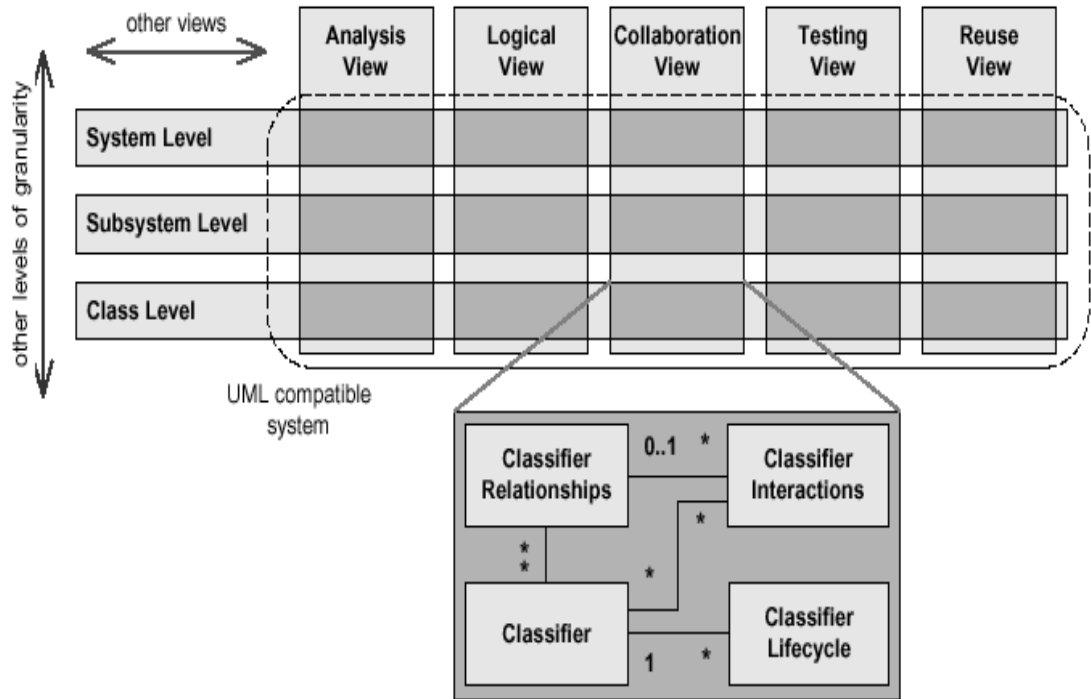


Figure 5. Static relationships between artefacts

14 Appendix E

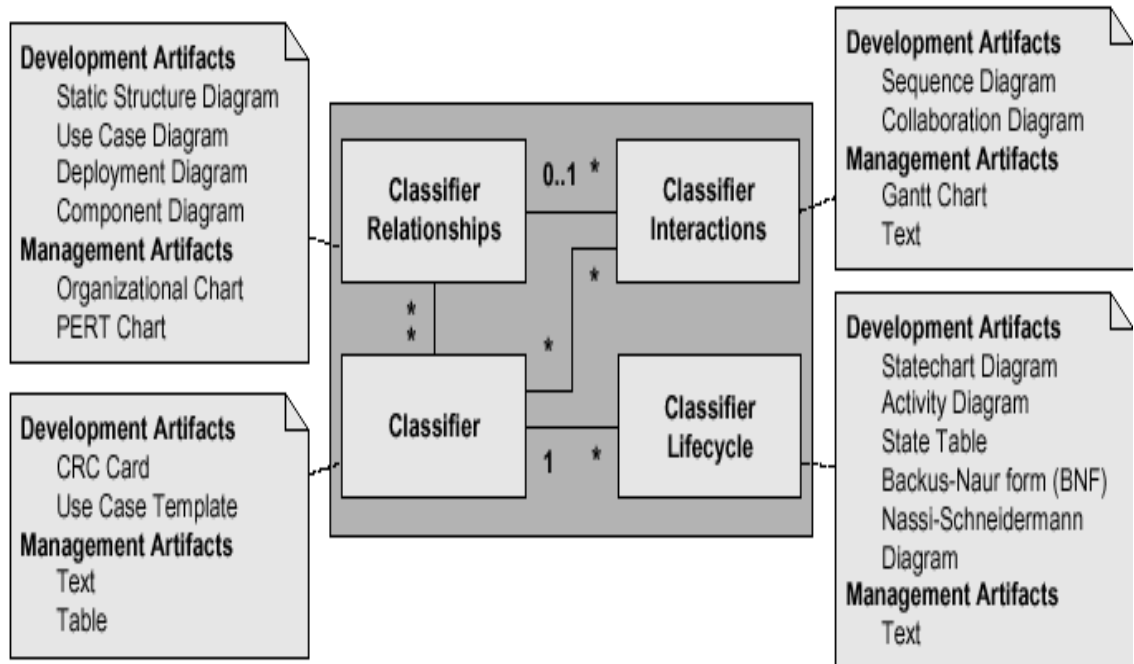


Figure 6. Software development artifacts

15 Appendix F

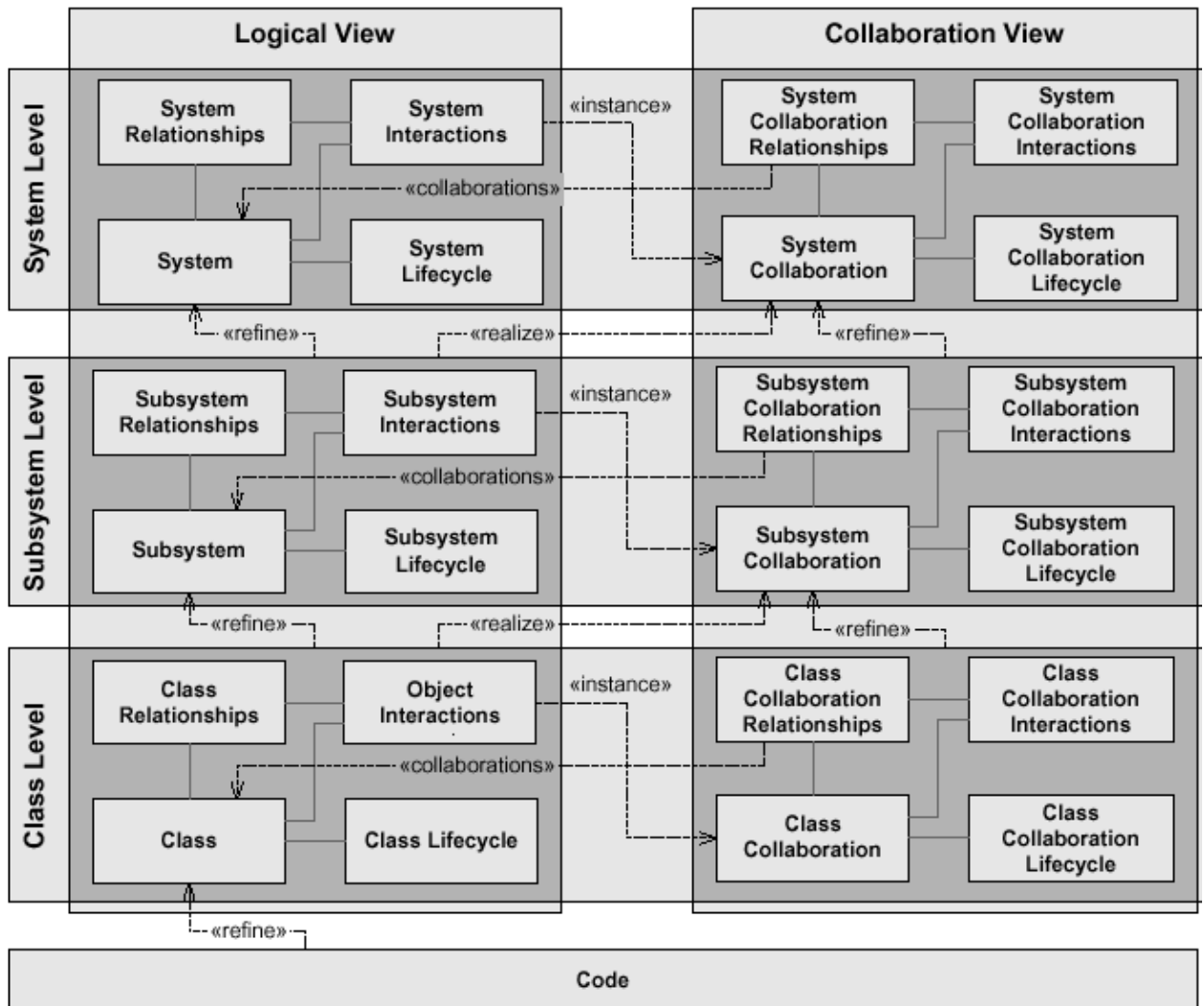


Figure 7. Static structure of software development artifact types

16 Appendix G

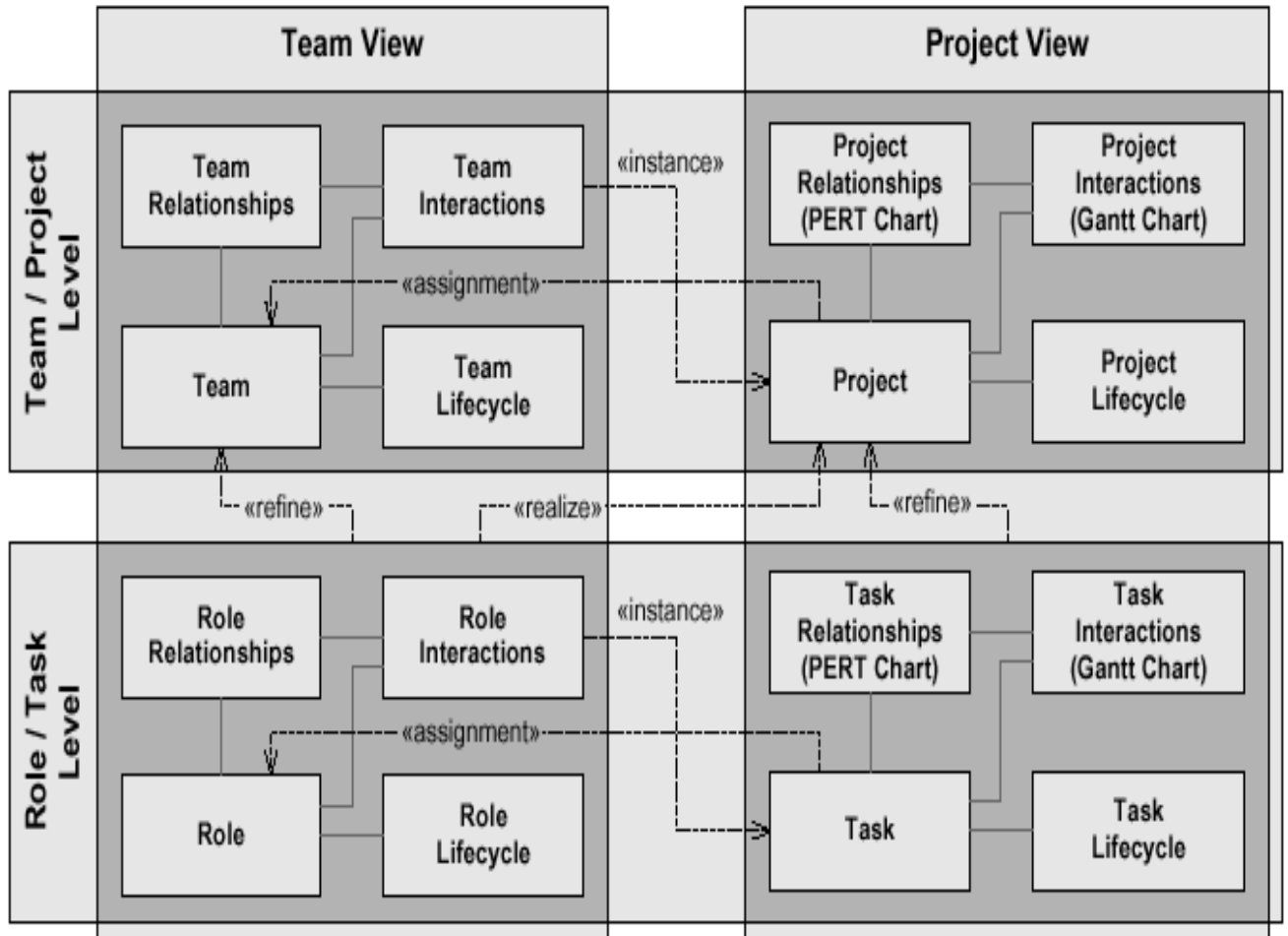


Figure 8. Static structure of management artifact types

17 Appendix H

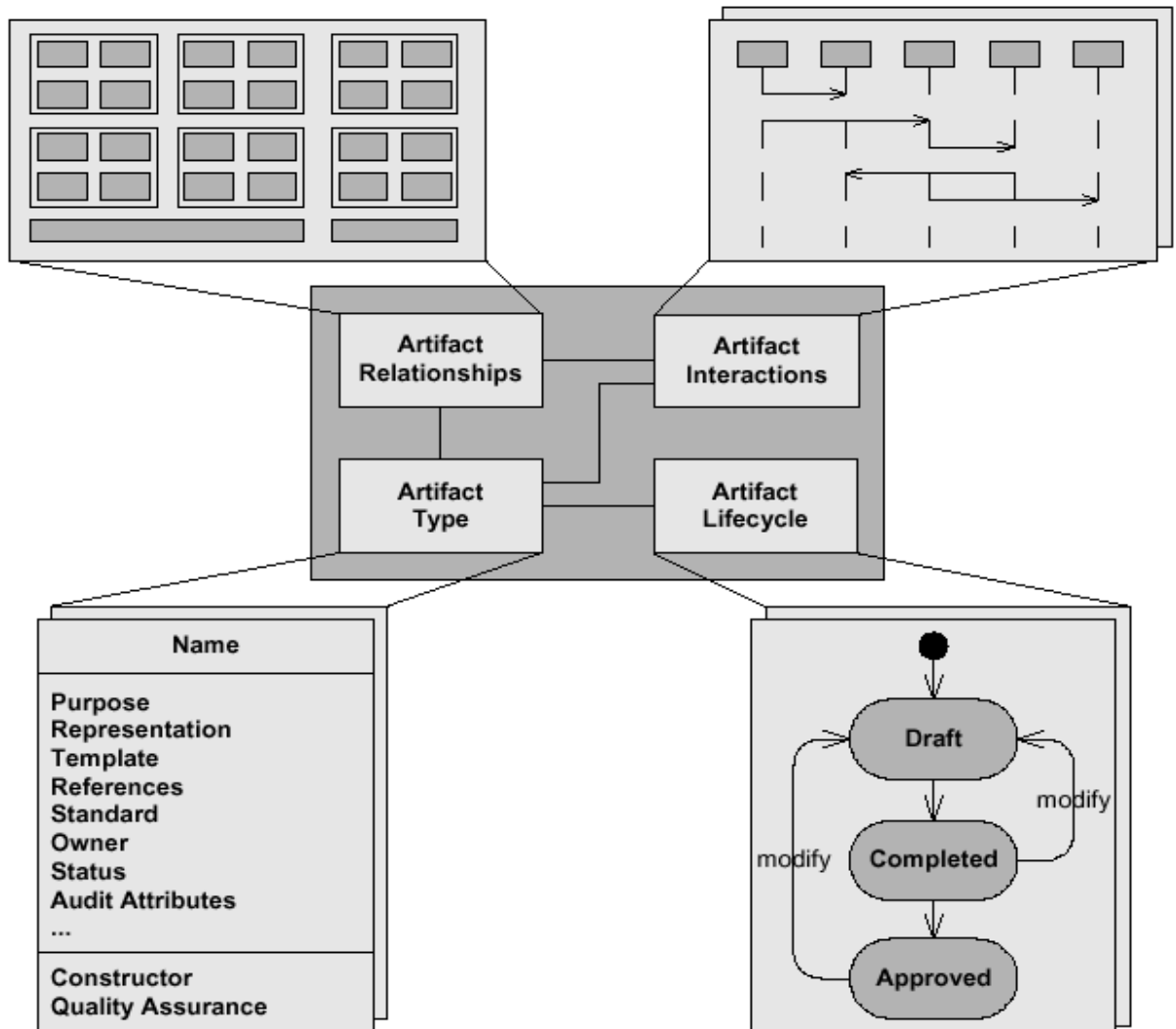


Figure 9. Structure of the process specification framework

18 Appendix I

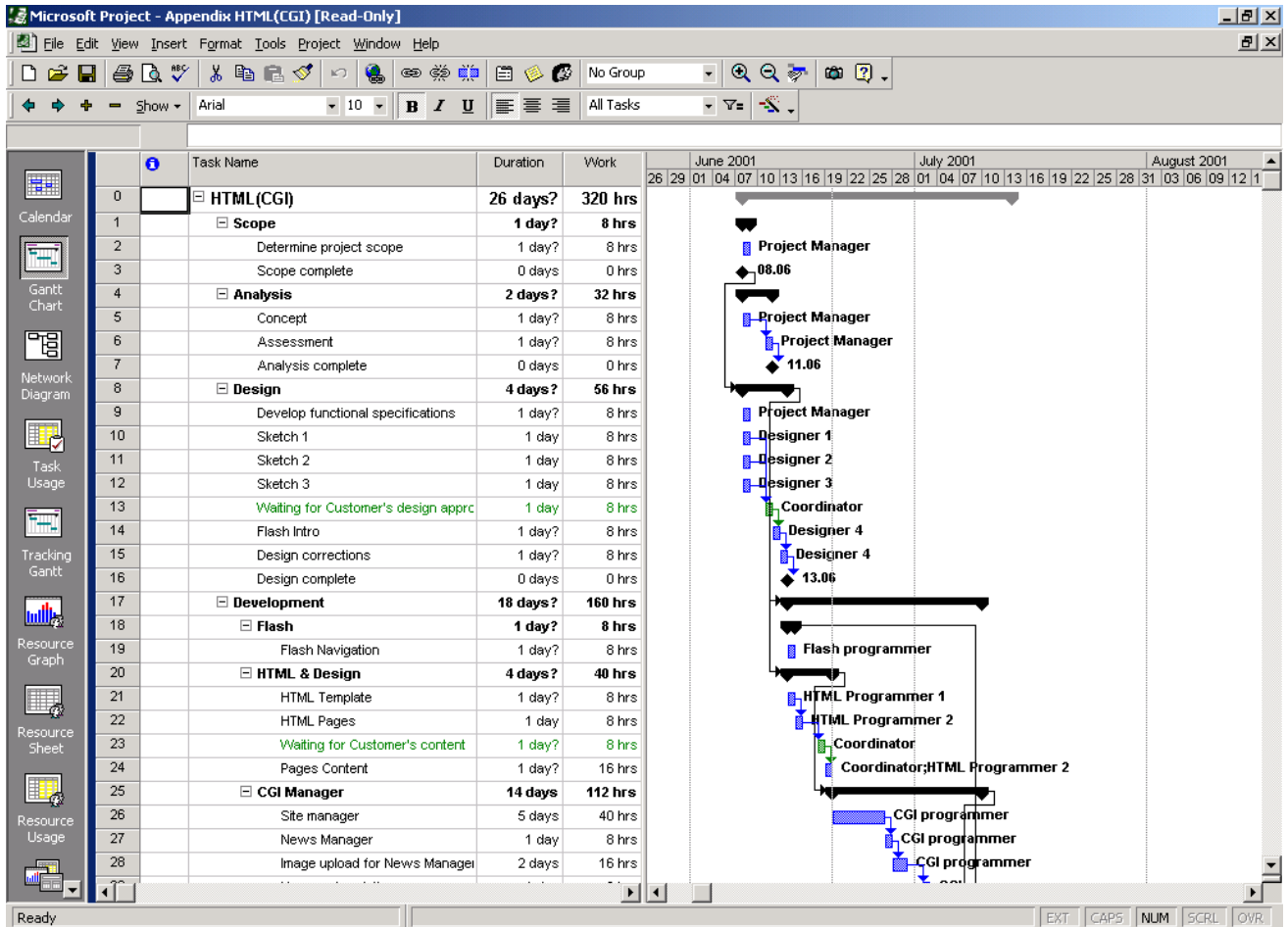


Figure 10. Typical view of a project workflow in Microsoft Project