Comparing Software Development Models Using CDM

Luciano Rodrigues Guimarães
UNIMEP – Methodist University of Piracicaba
Rodovia do Açúcar, Km 156 – CEP 13.400-911
Piracicaba, SP - Brazil
E-mail: guimaraes_luciano_r@cat.com

Dr. Plínio Roberto Souza Vilela
UNIMEP – Methodist University of Piracicaba
Rodovia do Açúcar, Km 156 – CEP 13.400-911
Piracicaba, SP - Brazil
E-mail: prvilela@unimep.br

ABSTRACT
During the last few decades a number of software development models have been proposed and discussed within the Software Engineering community. Examples of such models are Waterfall, Spiral, V Model, and Prototyping. The introduction of different models and their subsequent adoption by practitioners motivates the need to compare them. Comparisons are defined to i) find the best fit for a particular software development project; ii) improve the models themselves; iii) facilitate dissemination and education on development best practices and iv) find relevant information to define new models. However, existing comparisons are often largely based on the experience of practitioners and the intuitive understandings of the authors. Consequently, they tend to be subjective and inaccurate. The lack of a systematic way of comparing development models reduces understanding of such models and their overall acceptance by practitioners. We propose a systematic way of comparing software development models based on a formal technique originally proposed to compare design methodologies. The use of one well-defined formal technique provides a more consistent and efficient way to compare software development models and could be used to fine-tune the software development process of a software development organization. We present the results of a case study conducted to compare two well known and largely used development models. Results of such comparisons could be used by Information Technology educators to show their students the advantages and disadvantages of each model and instruct them on how and when to apply each model or variations of them.

Categories and Subject Descriptors
I.6.5 [Model Development]: Model Development Comparison – *life cycle, documentation, risk analysis.*

General Terms:
Design, Standardization.

Keywords
Comparison of Development Models, Software Engineering, IT Education.

SIGITE'05, October 20–22, 2005, Newark, New Jersey, USA.

1. INTRODUCTION
The overall cost of computer-based systems is associated to the hardware where the system will be deployed and to the cost of software development and maintenance. The cost of hardware has systematically decreased over the last few decades. Moreover it represents an initial and well-defined fixed cost. This scenario indicates that the main restrictive factor for the development of computer-based systems tends to be the cost of software. Software represents a great share in the total cost of systems development. Thus, the production of cost-effective quality software is essential for the future of computer-based systems.

Software Engineering discipline has the cost-effective planning and development of software as a goal [11]. Software is abstract and intangible. It is not composed of materials, governed by physical laws or constructed within manufacturing processes. Software processes are complex and, like every intellectual process, dependent on human judgment.

Software Engineering embraces a group of three fundamental elements: *methods, tools and procedures* [7]. Together they allow the software manager to control the software development process and provide the practitioner with a base for the construction of high quality productive software. The sequence of phases or stages that the software has to go through from its initial conceptualization to its final deployment and subsequent maintenance life is called Software Paradigm or *Software Development Model*.

The human aspect of software development emphasizes the importance teaching the appropriate techniques, methods, models and tools to future practitioners. Software development models are especially important in this scenario since they provide the structural framework where all the other activities involved in developing professional software unleash [2].

During the last few decades several software development models have been discussed in the context of Software Engineering. Several comparisons have been performed and published by different specialists in order to identify weaknesses and strengths of different models. According to Song, et. al [12], the existing comparisons are often largely based upon the experiences of the practitioners and the intuitive understandings of the authors. Consequently these comparisons tend to be subjective and inaccurate. The use of a formal and well-defined technique provides an effective way to compare development models.

Currently an immense diversity of software development models exists. There is no ideal model, and different organizations have developed approaches that are entirely different from one another.

The objective of this work is to identify the main characteristics of development models and compare them. We propose to adapt the technique called CDM [12] (*Compare Design Methodologies*) to compare models. The adapted technique is called CDMod

(*Compare Development Models*). We emphasize model characteristics and consider similar activities between models, such as, requirements, analysis, design, coding, implementation and testing. CDM is a systematic and defined process, using a classification technique as a comparison tool. It is similarly applied to software development models, facilitating reasoning, communication, customization and education.

Differently from most of the previously published comparisons of software development models, CDMod allow us to identify characteristics of the models, highlighting qualities and drawbacks of each one and following a logical sequence for the comparison.

Two models are compared in our case study. We present the results of such comparison and discuss how CDMod could be used in fine-tuning the software development process currently used in an organization.

2. RELATED WORK

Several software development models [7, 5, 11] are available in the literature; some are more thoroughly discussed, such as, the Waterfall Model [9] and the Spiral Model [2]. For that reason we have selected these two models to use in our case study.

- The Waterfall Model – "Classic Life Cycle", presents the following sequence of development phases: requirements, analysis, program design, coding, testing and operations. It is the oldest model [5] and is still thoroughly used [7].
- The Spiral Model appeared as a model that had a greater flexibility to the developer. It encompasses requirements gathering, design planning, implementation and deployment in an interactive and incremental fashion. It also incorporates a risk analysis phase.

2.1 Waterfall Life Cycle

The Classic Life Cycle or Waterfall, also known as the "top-down" approach, was proposed by Royce [9]. Up until the mid 80's it was the only model with a level of general acceptance. It was derived from models used in traditional engineering activities with the objective of establishing an order in the development of large software products. Compared with other software development models, it is more rigid and less manageable.

The Waterfall Model is one of the most important models ever published. It is a reference to others, and serves as the basis for many modern projects. Its original version was improved over time and is still frequently used today [5].

A great part of the success of the Waterfall Model is due to the baseline management, which identifies a fixed group of documents produced as a result of each phase of the life cycle [5]. The produced documentation includes more than text files, it has graphical representations of the software and even simulations. In this model the phases are executed systematically in a sequential order as shown in Figure 1 and it usually has the following phases: Analysis, Design, Construction, Evaluation and Maintenance.

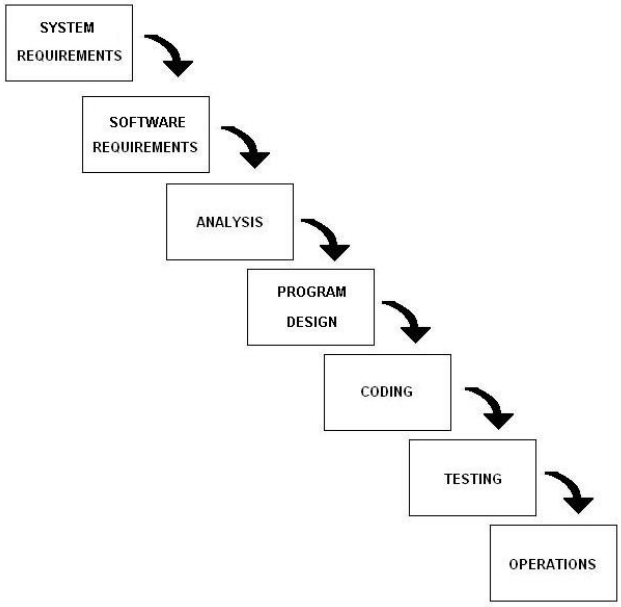

Figure 1: Original Waterfall Model

The Waterfall Model is an engineering model designed to be applied in software development. The main idea that drives it is that the different development stages follow a sequence: the output of the first stage is an input to the second, the output of second stage is an input to the third and so on. The activities to be done are contained in tasks, executed sequentially, so that a task will only begin when the previous one has finished.

2.2 Spiral Model

The Spiral Model was originally proposed by Boehm [2]. A simplistic way to analyze it is considering it as a Waterfall Model in which each phase or each cycle is preceded by a risk analysis and its execution is incremental.

The goal of the Spiral Model is to incorporate qualities of other models and solve some of their problems [2]. This model includes prototyping, evolutionary and cyclical development, and the main activities of the Waterfall Model.

The major innovation is to guide the development process with strong basis in risk analysis and development planning. Risks are adverse circumstances that can appear during the software development, impeding the process or reducing the quality of the product. Examples of risks are: people that abandon the development team, tools that cannot be used, flaws in equipments used in the development or those they will be used in the final product, etc. The identification and management of risks is an important activity in software development today.

The Spiral Model describes a cyclical and evolutionary flow of activities composed of four Stages.

- *Stage 1*: Determination of objectives, alternatives and limitations.
- *Stage 2*: Evaluation of the alternatives and analysis of risks: risks associated to decisions of the previous stage should be analyzed. During this Stage prototypes can be built or simulations of the software can be done.

- *Stage 3*: Development: consists of the activities of the development phase, including design, specification, code and verification.
- *Stage 4*: Evaluation of the cycle and planning for the next phases: understand the revision of the previous stages and the planning of the next phase. Depending on the results obtained in the previous Stages (decisions, analysis of risks and verification) the development could proceed in a Waterfall (linear) model or on a Evolutionary model. For instance, if the first cycle is already done, and the requirements are completely specified and validated, the Waterfall model could be chosen. Otherwise, it might be necessary to construct new prototypes, improving them, evaluate new risks and re-plan the project, following an Evolutionary model.

The Spiral Model is illustrated in the Figure 2. The radial dimension represents the updated cost and the angular dimension represents the progress through the Spiral. Each section of the Spiral corresponds to one Stage of the development.

The Spiral Model is one the most practical approach for the development of software and systems in large scale. It uses an evolutionary approach, qualifying system analysts and customers to understand and react to risks.

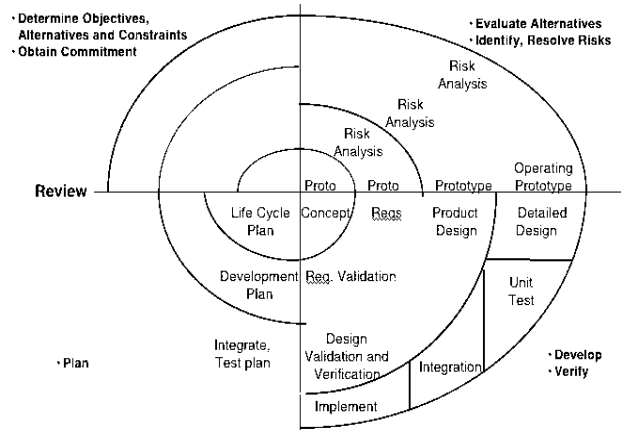

Figure 2: Spiral Life Cycle

Prototyping is used as a mechanism of risk reduction. However, it is difficult to convince customers that the evolutionary approach is manageable. If a great risk is not discovered, problems will certainly occur.

2.3 Compare Design Methods - The Original Methodology

The methodology to compare development models used in this work is based on a methodology proposed to compare design methods called CDM, in this section we describe some of the details of that work.

The methodology of comparison of development methods CDM (Compare Design Methodologies) proposed by Song and Osterweil [12] is used as a support to the identification of qualities and problems in the studied methodologies. CDM uses techniques of modeling processes for software methods classifying the components and analyzing the functionalities in a hierarchical decomposition. The similar components of the methods are classified for an addressed comparison, allowing the identification of the differences in the same procedure of the development phase. Figure 3 illustrates the structuring of CDM.

The first step in CDM is to develop a model, a formal description of each software methods compared. Considering that most of the technologies have the same development characteristics [10, 7, 11], similar components are identified and later described informally and in high level.

To apply CDM it is necessary to use a formal language to describe the activities of the compared [12]. This formalism is necessary to consistently represent the details of each activity, extracting the common characteristics to compare. Two formalisms are usually used: HFSP described by Katayama [4], and used in [12], and SLANG described by Bandinelli et al. and used by Podorozhny [6].

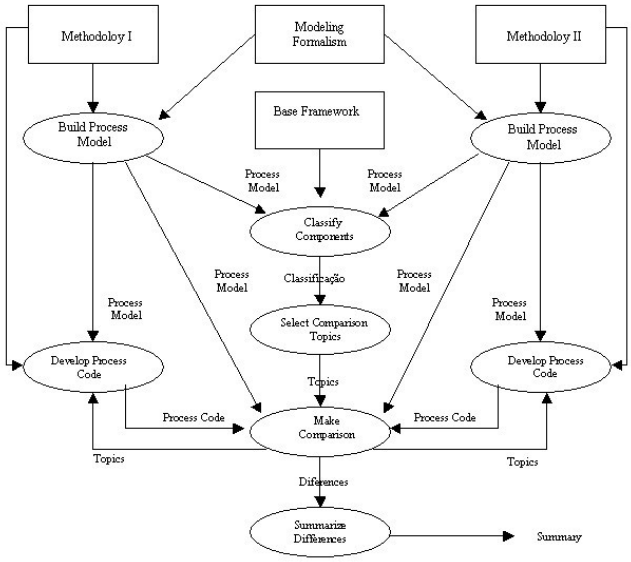

Figure 3: CDM – Compare Design Methods

An experiment evaluating the sensibility of the classification and comparison of different formalisms has been published [6]. That work used CDM with HFSP and CDM with SLANG to compare JSD (*Jackson System Development*) and BOOD (*Booch Object Oriented*).

3. APPLICATION OF CDMod

3.1 Approach

The following hypotheses are identified as the basis for our study: *i*) Software development models are based on the same principles, having common components; *ii*) It is possible to systematically compare development models; and *iii*) It is possible to make those comparisons with a methodology originally proposed to compare Design Methods.

This study was developed to collect evidences to support hypothesis *iii*. We analyze two different development models identifying similar activities and extracting evaluation criteria. The instantiation of CDM to make such comparison is called CDMod.

3.2 Application of CDMod (Instantiation)

With CDMod it is possible to identify qualities and drawbacks of the compared models. The first step for the application of CDM is to establish two SDM's (*Software Design Methods*) to be compared; in our case we will select two *Software Development Models* - SDMod.

3.2.1 Construction of the Process Model

The first step is to build the Process Model, which identify the components of the models. At this point the components are extracted in a high level of abstraction, with no evaluation of comparable components. The main extracted components of the Waterfall Model [7] are presented in Table 1. The Spiral Model is composed of four different Stages; the main extracted components of the Spiral Model are presented in the Table 2. A formalism is used to classify the components and build the process model. The SLANG formalism for the Waterfall and Spiral models are shown in the Figure 4 and Figure 5.

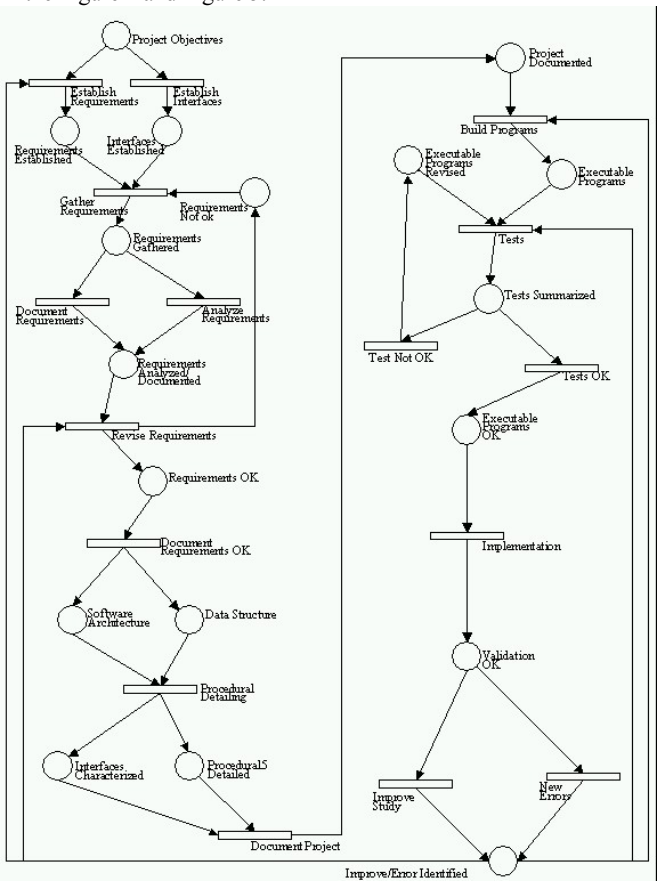

Figure 4: Kernel Slang of the Waterfall Model

The use of Slang allows for the decomposition of the activities in events as is shown in Figure 6. These events help in the detailed comparison of the components.

Figure 6 shows two events used in the kernel SLANG of the Spiral Model: "Establish Objectives" and "Establish Solutions".

3.2.2 Classification of the Components

The third step of CDMod is classifying the components to help the identification of what is comparable.

This stage consists of listing all the components of the models to identify the comparable ones, in other words, to classify in a way that allows us to identify them as having the same, or similar, objectives. This process generates a Base Framework used for comparison. We use the models MSDL (*Model of the Software Design Life Cycle*) and MCTH [12] (*Method Component Type Hierarchy*) for the composition of Base Framework.

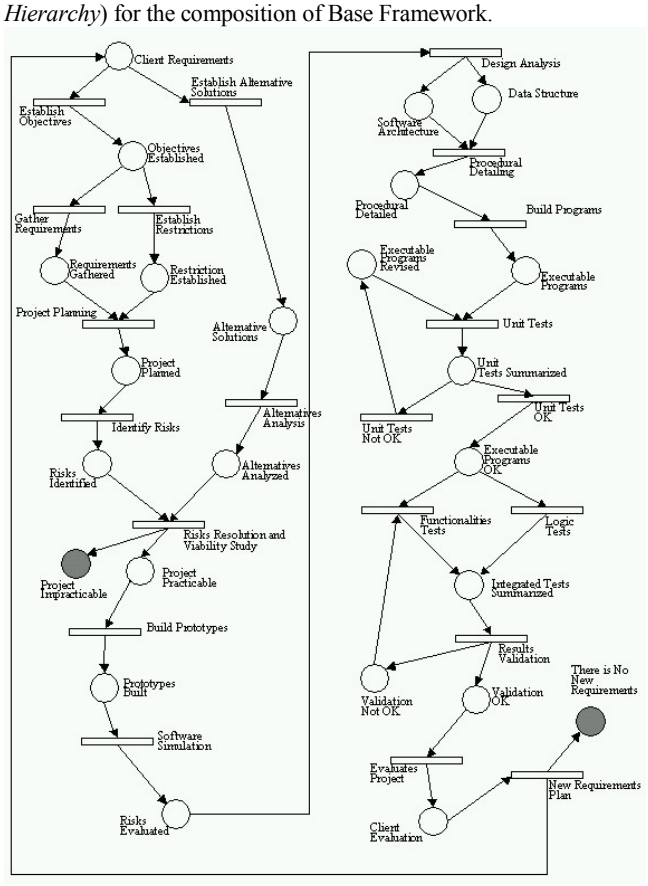

Figure 5: Kernel Slang of the Spiral Model

The MSDL is based on domains and allows the identification of three different functionalities:

- Modeling of the Problem.
- Modeling of the Solution.
- Design Documentation.

Thus, we can apply Level 1 of MSDL starting from the stages of the Waterfall and Spiral models, as it is shown in Figure 7 and Figure 8.

MSDL contributes to identification of the phases that deal with the problem and the ones that deal with the solution of the project. The functionality documentation shows the number of documents used.

After the separation of the stages in the three different domains, a table is created in three levels with the detail of the components separate for the three functionalities shown in the Figure 7 and Figure 8. The first level treats the functionality type, the second the stages of the models and the third the detailed components in the Kernel SLANG as display the Table 3.

Table 1: Extracted Components from the Water Model

Analysis	Project	Coding	Tests	Operations
-Requirement Establishment -Software Project Objective - Establish Interfaces -Requirement Gathering -Requirement Analysis -Requirement Documentation -Requirement Revision	-Data Structure -Software Design -Procedural Details - Interface Characteristics -Project Documentation	-Program Building -Unit Tests	-Internal Logic Tests -External Functionalities Tests -Behavior Errors Decision -Validations of Results According to Requirements	-Software Improvement -Fix Post-release Errors

Table 2: Extracted Components from the Spiral Model

Stage 1 – Establish Objectives	Stage 2 – Risks Analysis	Stage 3 – Engineering	Stage 4 – Evaluation
-Establish Objectives -Alternative Solutions -Project Constraints -Requirements Gathering -Project Planning	-Evaluate Alternatives -Risks Identification -Risks Resolution -Prototypes Building -Software Simulation	-Design Data Structure Software Architecture -Specification Procedural Details -Implementation Program Building Unit Tests -Verification Internal Logic Tests External Functionalities Tests -Behavior Errors Decision -Validations of Results According to Requirements	-Early Steps Revision -Next Cycle Plan

```
event Establish Objective ( CR:Client Requirements ;
                            OD: Objective Defined)
   guard
     CR->necessities_pending > 0
   action  {
     OD->objectives_accumulated = OD->objectives_ accumulated + 1
     CR->necessities_pending = CR-> necessities_pending -1
           }
end
event Establish Alternative Solutions (CR:Client Requirements ;
                                       AS: Alternatives Solution)
   guard
     CR-> necessities_pending > 0
   action  {
     SA->solution_accumulated = SA->solution_accumulated + 1
     CR-> necessities_pending = CR-> necessities_pending -1
           }
end
```

Figure 6: Definition of Types and Hierarchy

3.2.3 Selecting Topics to Compare

The fourth step of the CDMod is the selection of the comparison topics, which identify the comparable components for the stage of comparison. According to MSDL presented in Table 3, it is possible to identify the items that have the same characteristic in a certain stage of each model, for instance, Software_Project_Objectives of the Waterfall Model and Establish_Objectives of the Spiral Model, are considered comparable because they are under the same domain, in other words, on the same stage – "Objectives".

The Waterfall and Spiral models have the same basic foundation to software development - Definition of Requirements, Analysis, Project, Tests and
Implementation – which leads to the comparable items shown in the Table 4.

The models have different stages such as Risks Analysis, Prototyping, Simulation, etc, that are not considered comparable.

3.2.4 Comparison of the Models

After the classification of the components and selection of the similar and comparable ones, the next stage is the comparison of the models.

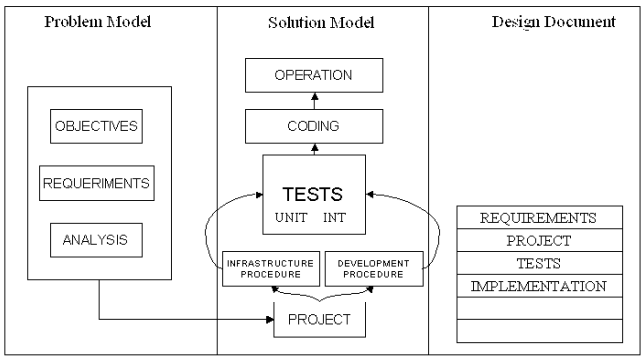

Figure 7: MSDL of the Waterfall Model Level 1

The comparison is done pairwise, focusing on understanding the differences between each component in detail. The comparison is performed in a high level description of the components. If it is not possible to compare all the components at this level, a more detailed version of the models is created; this version is called the *Process Code*.

The process code aids the identification of differences in the following aspects:

- Inter-component dependency: What is the dependency among components. It may illustrate different uses and characteristics of components
- Degree of human involvement: The need for human intelligence in performing actions, which may indicate how much of the action could be automated or systematically applied in practice;
- Development procedure: The order in witch the actions are to be performed;
- Scope of issues: The Scope of the design issues that the SDModel addresses.

Figure 6 shows the types that each component has, identifying its inputs, outputs and actions. Following we show the detailed comparison of each phase of the models.

Comparison Between Objectives Phase

Differences in the inter-component dependency: Comparing the inputs of Software_Project_Objectives of the Waterfall Model and Establish_Objectives of the Spiral Model we verified that an input does not exist in the Waterfall Model. That shows that the model does not have a specific phase that treats the customer's needs. Therefore, it suggests that the inputs for the event to Software_Project_Objectives are collected spontaneously.

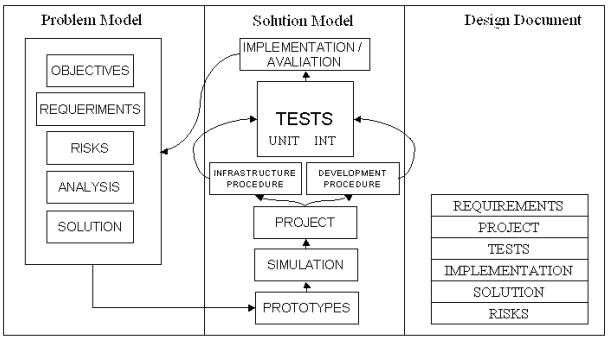

Figure 8: MSDL of the Spiral Model Level 1

Differences in the need for human involvement: As in the system objectives phase human involvement is essential for their definition, there is no way to differentiating the human involvement in this phase.

Differences in the development procedure: The order in both models are quite similar, actually when they refer to the development model many stages do not exist before the phase of definition of objectives. However, the Spiral Model stands out as having some additional components that differentiate it in the quality of the stage: Risks associated to Cost and Alternative Solutions. These components are important and fundamental in the definition of objectives and they should be pointed out.

Table 3: Classification of the Components using MSDL

MSDL		COMPONENTS
LEVEL 1	LEVEL 2	
Problem Model	Objectives	Define_ Objectives (Waterfall) Determine_ Objectives (Spiral)
	Requirements	Establish_ Requirements (Waterfall) Collect_ Requirements (Waterfall) Collect_ Requirements (Spiral)
	Analysis	Analyze_ Requirements (Waterfall) Revise_ Requirements (Waterfall) Project_ Plan (Spiral) Evaluate_ Alternatives (Spiral) Viability_ Study (Spiral)
	Risks	Determine_ Restriction (Spiral) Identify_ Risks (Spiral) Resolve_ Risks (Spiral)
	Solution	Establish _ Alternatives_ Solution (Spiral)
Solution Model	Prototypes	Build_ Prototypes (Spiral)
	Simulation	Software_ Simulation (Spiral)
	Project	Design_ Analysis (Spiral) Build_ Programs (Waterfall) Build_ Programs (Spiral)
	Procedures	Procedure_ Details (Waterfall) Procedure_ Details (Spiral)
	Tests	Tests (Waterfall) Unit_ Tests (Spiral) Functionalities_ Tests (Spiral) Logic_ Tests (Spiral) Results_ Validation (Spiral)
	Operation	Improves (Waterfall) Identify_ New_ Errors (Waterfall) New_ Requirements_ Plan (Spiral)
	Evaluation	Evaluate_ Project (Spiral)
Design Document	Requirements	Document_ Requirements (Waterfall)
	Project	Document_ Project (Waterfall)
	Tests	Document_ Tests (Waterfall)
	Implementation	Document_ Implementation (Waterfall)
	Solution	
	Risks	

Differences in scope of issues: Figure 4 shows that the stage of project objectives provides information for the establishment of requirements. The same happens in the Figure 5 in the Spiral Model. However it is realized that the project objectives contributes to the phase of establishing interfaces that shows the importance of analyzing the relationship between different parts of the system or the relationship between different systems.

Comparison between Requirements Phase

Differences in the inter-component dependency: For the requirements phase the Waterfall Model has two types: Establish Requirements and Requirements Gathering while the Spiral Model has just one: Requirements Gathering. The Waterfall Model presents a specific type to establish requirements before collecting it while the Spiral omits the information, it could be interpreted that the establishment of requirements depends on the level of the analyst's knowledge. The Waterfall Model also specifically treats the revision of requirements, while and the Spiral does not.

Differences in the need for human involvement: both models need the same level of human involvement in the requirements phase.

Differences in the development procedure: In the Waterfall Model a stage of establishment of requirements is performed before the actual collection, which does not happen in the Spiral Model. It also has a procedure for definition of interfaces for the requirements gathering. Thus, the Waterfall Model presents a definition of requirements with more procedures than the Spiral Model.

Differences in scope of issues: Figure 2 of the Spiral Model presented in the Section 2 shows in the third quadrant a stage for validation of the requirements; however, it does not specify a revision stage. In Figure 1 of the Waterfall Model there is an input indication for the requirements phase indicating the validation and revision of the requirements.

Comparison between Analysis Phase

Differences in the inter-component dependency: The Waterfall Model just presents dependences of the requirements for the analysis phase. In the other hand, the Spiral Model presents the phase of analysis, besides the requirements gathering, in the analysis of alternatives established through the customer's needs. The Spiral Model shows an analysis more focused in the risk of the project where its execution could be interrupted in case of the non-feasibility are greater than the feasibilities. The Waterfall Model just presents the revision of the requirements.

Differences in the need for human involvement: For both models the human involvement is fundamental in the analysis criteria.

Differences in the development procedure: The Spiral Model presents the analysis based on alternative solutions and feasibility study. The project could be cancelled depending on the appraised risks. The Waterfall Model presents a flow of requirements revision, addressing in a looping between the phases, until all the requirements have been reviewed. The continuity of the project is not approached in the model.

Differences in scope of issues: The Waterfall Model presents the analysis phase based on requirements while in the Spiral Model analysis based on requirements, risks and alternative solutions.

Comparison between Design Phase

Differences in the inter-component dependency: There are no differences between dependences assuming that the inputs Detailed Design of the Waterfall Model and Procedure Detailed of the Spiral Model have the same necessary inputs for the construction of the programs.

Differences in the need for human involvement: For both models the human involvement is fundamental in the implementation phase.

Differences in the development procedure: There are no differences between the models in the procedure requirement.

Differences in scope of issues: Figure 5 shows that the Spiral Model has a simulation phase and prototyping for construction of programs. These types contribute with the evaluation of risks and less maintenance and code improvement. Prototyping influences directly in the construction of the programs based on the simulations accomplished in the prototype by the customer. The Waterfall Model does not specifically treat this technique.

Table 4: Comparable Components Waterfall and Spiral

WATERFALL		SPIRAL	
Objectives	Define_ Objectives	Objectives	Determine_ Objectives
Requirements	Establish_ Requirements Gathering_ Requirements	Requirements	Gathering_ Requirements
Analysis	Analyze_ Requirements Revise_ Requirements	Analysis	Project_ Plan Evaluate_ Alternatives Feasibility_ Analysis
Procedures	Procedure_ Details	Procedures	Procedure_ Details
Project	Build_ Programs	Project	Build_ Programs
Tests	Tests	Tests	Unit_ Tests Functionalities_ Tests Logic_ Tests Results_ Validation
Operation	Improves Identify_ New_ Errors	Operation	New_ Requirements_ Plan

Comparison between Testing Phases

Differences in the inter-component dependency: For both models the inputs are the executable programs indicating the similarity of the dependence at the beginning of the phase.

Differences in the need for human involvement: For both models the human involvement is fundamental in the analysis of the results of the execution of the programs. However the Spiral Model demands more human involvement since the testing phase is separated in several interactions.

Differences in the development procedure: The Waterfall Model has just one task for execution of tests in spite of [9] suggesting that specialists execute the tests and that all the logics of the program are tested. In the other hand, the Spiral Model presents a considerable exploration of the testing phase, being incrementally conducted in the software for each new interaction.

Differences in scope of issues: Figure 5 shows the detail of the Spiral Model phases and Figure 2 displays in the third quadrant the validation through the unit tests, integration and acceptance for implementation and end of the spiral.

Comparison between Maintenance Phases

Differences in the inter-component dependency: At end of spiral cycle there is the validation and the customer's evaluation for the next stage. In this case the Spiral Model presents the dependence of the customer's evaluation before maintenance what is not clearly exposed in the Waterfall Model.

Differences in the need for human involvement: The Spiral Model presents the customer's involvement in the maintenance phase. That involvement identifies the need for new improvements or the existence of mistakes.

Table 5: Summarizing the differences between Waterfall and Spiral

	WATERFALL	SPIRAL
Objectives	× There is no Phase to treat client necessity; × The Model suggests that the inputs are random, there is no procedure; ✓ Contributes to Interfaces Management.	✓ There is Phase to treat client necessity; ✓ There is a Cost Risk Phase and Alternative Solutions Steps; × There is no step to Interface management.
Requirements	✓ The model has a phase to establish requirements before Gathering Phase; ✓ Special treatment for interface requirements; ✓ The Model specifically treats requirement revision.	× Requirements Establishment depends on designer knowledge; × Does not treat the interfaces requirements; × Does not treat specifically the requirement revision, just the validation.
Analysis	× There is no phase of Alternative Solutions before analysis phase; × The Model does not support project feasibility analysis, just the requirements revision; × Analysis based on requirements.	✓There is a phase of Alternative Solutions before analysis phase; ✓ Has project viability analysis and possibility to stop it depend on the risks associated; ✓ Analysis based on requirements, risks and Alternative Solutions.
Project	× Does not have a simulation and prototyping phase before code development.	✓ There is a phase to simulate and prototype before of code development.
Tests	× Does not have separation and treatment of test phase.	✓ There are different test phases to evolve to the next phase.
Operation	× There is no special treatment to client validation. × The identified maintenance can begins in any place depending on the necessity.	✓ For each end of spiral cycle, there is a client validation before move on. ✓ In the Spiral Model the maintenance begins after the implementation, it is assumed that the whole spiral cycle must end before maintenance.

Differences in the development procedure: There are no differences between the models during maintenance.

Differences in scope of issues: In the Waterfall Model the maintenance phase can send the development back to any phase of the development process. In the Spiral Model the beginning of the maintenance phase is not clearly shown, the understanding is that the end of the quadrant develops for the next stage.

3.2.5 Summarizing of the Differences

Table 3 presents the model MSDL with the separation of the components for process levels, and Table 4 presents the items considered in the scope as comparable or similar. Like this after the comparison Table 5 is presented with the summary of differences found in the models.

In a broader sense, it can be pointed out that the Waterfall Model is less efficient than the Spiral Model, however the exposed differences are of comparable or similar items. Other important items such as documentation, risk analysis and project evaluation, for instance, were not considered comparable.

3.2.6 Integration of the Models

One of the collateral results of the comparison using CDMod is the identification of characteristics that could be combined to form a new improved model. Figure 9 provides a broad view of the main characteristics of the Waterfall and Spiral that could be used for integrating the models.

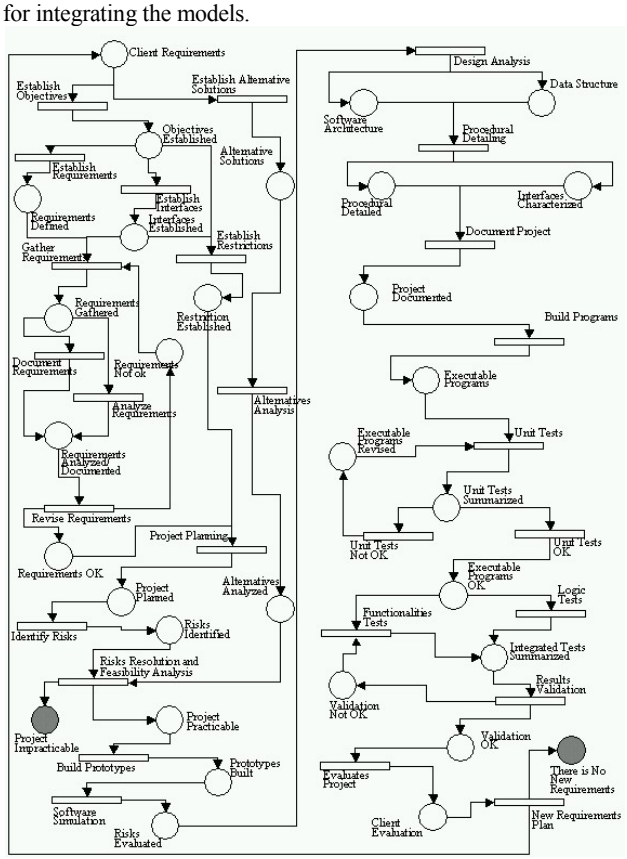

Figure 9: Kernel Slang built for the Integration Waterfall and Spiral

4. CONCLUSIONS

CDM is a systematic and well-defined process, using a classification of methods with modeling and science as tool of comparison. The CDMod has a similar approach to compare software development models, facilitating reasoning and communication on the development model problems [12].

Differently from most of the comparisons of software development models, the comparison using CDMod allow us to identify the characteristics of the models as well as the qualities

and drawbacks of each one obeying an important logical sequence for the comparison:

- Construction of the Process Model - Identify the components of the models in high level, without the concern if they will be comparable or not.
- Classification of the Components - Where through a Structure Base Framework of the formal specification it is possible to separate the tasks in order to allow the identification of comparable components.
- Selection of the comparison topics - In which comparable components are identified for the comparison stage.
- Comparison of the Models - the comparison identifying the qualities and drawbacks through the analysis of the formalism.
- Summarizing of the Differences - Presenting a table or a summary with the qualities and identified defects
- Integration of the Models - Combination of the best characteristics of the models. It is demonstrated that the comparisons using CDMod provide results that can be used in the integration of models.

This paper provided a comparison between the Waterfall and the Spiral models, which have similar development characteristics. They are thoroughly discussed in the software Engineering literature. The Spiral Model is a model that composes the Waterfall and Prototyping Model characteristics, and most of the previous comparisons identify the Spiral Model as the most complete, however the Waterfall Model is still more thoroughly used even today.

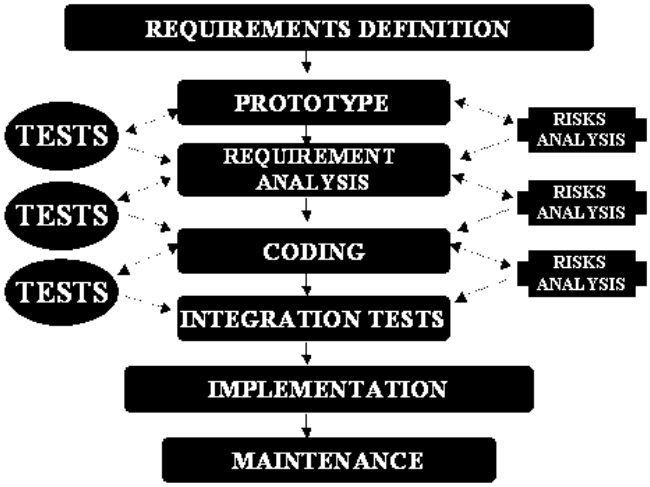

Figure 10: Development Model extracted of the Waterfall and Spiral Models characteristics

4.1 Future Work

From the current experience other comparisons can be conducted in order to identify potential points for process improvement. Comparisons like V Model and W Model, Evolutionary Model and Waterfall Model, RUP and Extreme Programming, Prototyping and Spiral Model, and others could be accomplished.

Figure 10 presents a sketch of a process model derived from the observations collected from the CDMod comparison. It emphasizes the concerns with risk analysis and defect prevention. We intend to compare other software development models using CDMod. The goal is to build a body of knowledge on model comparisons to help on the definition of new, robust, models and contribute to the evolution of Software Engineering.

REFERENCES

[1] S. Bandinelli, A. Fuggetta, C. Ghezzi, and L. Lavasa. *SPADE: An Environment for Software Process Analysis, Design, and Enactment.* Software Process Modeling and technology. Research Studies Press, Tauton, Somerset, England, 1994

[2] B. W. Boehm: A Spiral Model of Software Development and Enhancement, *IEEE Computer*, pages 61-72, May 1988.

[3] G. J. Hidding. Reinventing Methodology: Who Reads It and Why? *Comm. of the ACM*, 40(11):102-109, 1997.

[4] T. A. Katayama. A Hierarchical and Functional Software Process Description and Its Enaction. In 11th Conference on Software Engineering, May 1989.

[5] J. F. Peters and W. Pedrycz. *Software Engineering: An Engineering Approach*. John Wiley & Sons, 2000.

[6] R. Podorozhny and L. Osterweil. The Criticality of Modeling Formalisms in Software Design Method Comparison. *In 19th International Conference on Software Engineering*, pages 303-313, Boston, MA, May 1996.

[7] R. S. Pressman. *Software Engineering A Practitioner's Approach.* McGraw-Hill, 5th Edition., 2003.

[8] S.A. Raghavn and D. R. Chand. Diffusing Software Engineering Methods. *IEEE Software*, pages 81-90, July 1989.

[9] W. W. Royce. Managing the Development of Large Software Systems. In *IEEE Western Conference (Wescon)*, pages 1-9.

[10] J. Smith. A Comparison of RUP and XP. Technical Report, IBM Library, 2003

[11] I. Sommerville. *Software Engineering*. Addison-Wesley, 6th Edition, 2000

[12] X. Song and L. J. Osterweil. Experience with an Approach to Comparing Software Design Methodologies. *IEEE Transactions on Software Engineering*. 20(5), May 1994.

[13] E. Yourdon. What Ever Happened to Structure Analysis? *Datamation*, pages 133-138, June 1986.