Chapter 5
Application of the Process Analysis Method

5.1 Goals

The primary objective of this chapter is to apply the method developed in the previous chapter to a variety of software process models and to provide a comparative analysis of the models studied. The analysis involves the steps of *elaboration*, *normalization* and *abstraction* as described in the previous chapter. We have developed a collection of metrics to quantify the process characteristics. These metrics are applied on the outcomes of the abstraction step for each process model in order to observe the differences and similarities between them. Six of the popular software process models are selected for the analysis. The analysis topics are based on the measurement provided in Section 5.2 and process tailoring guidance provided in the joint standard IEEE/EIA 12207.0-1996 (IEEE/EIA 1998).

This chapter is organized as follows. Section 5.2 describes the metrics developed to characterize process models. Section 5.3 prepares six illustrative case studies for the analysis steps. Measurement results are analyzed and interpreted in Section 5.4. Finally, Section 5.5 gives the conclusions.

5.2 Measurement

For comparison purposes, a numerical measurement is required to quantify software process model characteristics. Five metrics are defined here.

1) **Details Index** (DI) of a node in the abstract model is a measure of the number of occurrences of the corresponding root activity in the normalized model. It is measured by the thickness of the border of a node in the abstract diagram. A thicker

node border designates a higher number of occurrences of the phase[1]. For instance, DI for the Planning activity in the Waterfall model is 2 because the Planning node in the abstract model of Waterfall (Figure 4-11) has a thickness of 2 deriving from the two planning activities in the normalized diagram (Figure 4-9). The details index demonstrates how elaborately the process model has defined activities and artifacts related to the corresponding root activity.

2) **Expanse** (E) of a model is measured as the sum of the details index of all nodes in the abstract diagram. This is a measure of how elaborately the whole process model is defined. For instance, the Expanse of the Waterfall model (Table 5-1) is 19 given by the sum of details index of specifying (4), planning (2), designing (6), coding and testing (4) and delivering (3).

3) **Weighted Interaction Index** (WII) of a node in the abstract diagram is the weighted sum of incoming and outgoing edges. The weight is simply the frequency of interaction. WII is a measure of how intensively the corresponding root activity interacts with other root activities.

4) **Interaction Index** (II) of a node in the abstract model is the sum of the number of incoming and outgoing edges without considering the weight of each edge. Interaction index is a measure of how the corresponding root activity interacts with other root activities. The sum of the number of edges in the abstract model is half of the Interaction Index.

5) **Dynamism** (D) of a process model is measured as the sum of the weighted interaction index of all nodes. This is a measure of how much interaction occurs between different root activities within a process model. A higher value indicates a more dynamic process model.

6) **Interaction Complexity** (IC) of a process model is measured by McCabe metric (McCabe 1976) adapted to measure the complexity of the interaction for the whole model.

---

[1] Unit thickness is the thickness of the line used in drawing a node in the abstract diagram which represents one occurrence of the activity in the normalized diagram.

There can be other metrics which can be used for further analysis. Examples are:

- Inward Weighted Interaction Index (IWI) - the weighted sum of incoming edges of all modes
- Outward Weighted Interaction Index (OWI) - the weighted sum of outgoing edges of all modes
- Inward Interaction Index (III) - the sum of the number of incoming edges
- Outward Interaction Index (OII) - the sum of the number of outgoing edges
- Connectivity Index (CI) - the sum of the number of edges that point to, and start from the corresponding node

It may be noted that these are not further used in this thesis but can be part of future work.

5.3 Case studies

The software process analysis method is applied to six software process models corresponding to six software processes: (1) Waterfall, (2) Spiral, (3) Extreme Programming, (4) Scrum, (5) Crystal Clear and (6) Feature-Driven Development.

A process model represents possible activities in a life cycle of software development process. It also represents the structural aspect of a process. By studying their structures, we will be able to identify their structural and interaction relations.

5.3.1 Background

Before we go into the results of analysis of each process, in this subsection, we will briefly describe some of the challenges faced. Unfortunately, there are no conventions and standards for writing process descriptions. Different process designers take different approaches. Mostly they follow the top-down approach. Generally, for each process, a graphical high-level view of processes is given along with narrative texts. A general problem with graphical representations in process description literature is that they are not given in a uniform manner and, often, without sufficient explanation of the pictures. Another problem we have found during process modeling is that the level of detail available in the description varies considerably. Some provide process decomposition in

a straightforward manner (Boehm, B.W. 1988; Cockburn 2005; Palmer & Felsing 2002). Others describe the process model at a high-level and give its details based on the actual practices comprising the process (Beck 2000; Cockburn 2005; Palmer & Felsing 2002; Schwaber & Beedle 2002). Yet others provide a modular decomposition to process description (Palmer & Felsing 2002; Schwaber & Beedle 2002; Wake 2002; Wells 2002). Frequently, processes are explained without explicitly giving details of the sequence of activities. One reason is that a process specification is often provided as a guideline rather than as a complete specification that is ready to be implemented. The attempt to create a complete model for each process is easier said than done.

After the process elaboration step, process specific terms are normalized based on comparisons of English phrases. Although comparing in the same language, measuring similarity of process terms have proved to be difficult. Even though two terms, say t_a and t_b, are considered close in meaning but not identical, two other terms; say t_c and t_d, each made by joining a new term with one of the first pair may be considered identical. For example, consider *write story* (Extreme Programming (XP) process) and *write user requirement*. In XP, a story is a high-level definition of requirements and is informal (Beck 2000). If *user requirement* is used to refer to a high-level requirement, then *user story* is considered a synonym to *user requirement*. These two terms are identical in practice but not in meaning. Unless process terms are defined rigorously, automated comparison of terms will not be able to identify such relationships.

As for our experiment with automated identification of normalized terms in Chapter 4, we found that the tool is able to pick up correct similar terms in most cases when the definition given to the process term is rich enough. Increasingly, however, use of process specific terms in term definitions requires more intervention from process engineers. Examples are Extreme Programming and Scrum processes. In these processes, most practices are given new names instead of using the traditional process terminology. This can be seen as an obstacle in analyzing multiple processes. In contrast to these processes, most terms used in Waterfall and Spiral processes are more general and the automatic identification of normalized terms yields more positive results.

During the abstraction step, ideally each normalized term is transformed into a single root activity. However, some normalized terms fit into more than one root

activities. For instance, *spike story* in XP refers to a process of making a throwaway prototype to answer questions regarding planning or designing. The developer writes the smallest possible code to answer such questions. Its normalized term is *prototyping*. When it comes to identification of a corresponding root activity, considering the purpose and activities performed during *prototyping*, *spike story* can fit into three root activities: planning, designing and coding & testing. In this case, all three root activities are selected.

Once root activities are identified, creating an abstract model is straightforward. By grouping root activities, software process models are ready for further analysis.

In the following sections, we give a brief description of the process including a diagram from a base reference first. Then, we show the result of the analysis as an abstraction diagram and the corresponding values of the metrics. The elaboration diagram and the normalization step are not shown here; they are available in the Appendix.

5.3.2 The Waterfall model

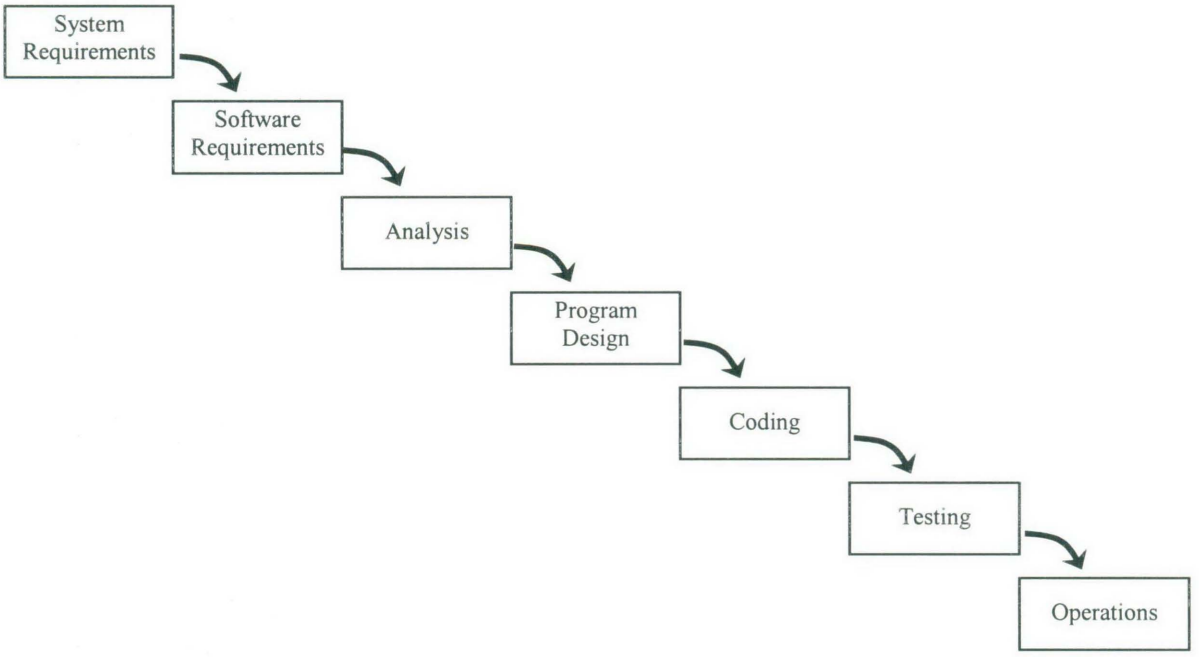

Figure 5-1 The conventional Waterfall model (Royce 1970)

The base reference used to model the Waterfall process is Royce (1970). Supporting information is derived from the later publications by Boehm (1996) and Sommerville (2001).

The Waterfall model is considered a traditional heavyweight process. This well-known model is originally proposed as a concept model. Although Royce stated the flaws in his initial model and later proposed alternatives, the Waterfall model has already been recognized as a classic sequential software development model. Since then, several variations were proposed to improve the model. We selected the conventional linear Waterfall model (see diagram in Figure 5-1) to represent software processes belonging to the same group. The elaboration of the linear Waterfall model is demonstrated in Section 4.4 of Chapter 4.

5.3.2.1 Abstraction

Figure 5-2 reproduces the abstraction diagram of the waterfall model that we derived in the previous chapter. The numbers adjacent to the links and nodes indicate their thickness. The figure illustrates the intensity of the root activities and their interaction. Based on this diagram, the values of Details Index for the root activities are as shown in Table 5-1. Table 5-2 shows the Interaction Index and Table 5-3 the Interaction

Complexity. In addition, the Bar Chart in Figure 5-3 gives a visualization of the intensity of elaboration and interaction of individual root activities in the Waterfall model.

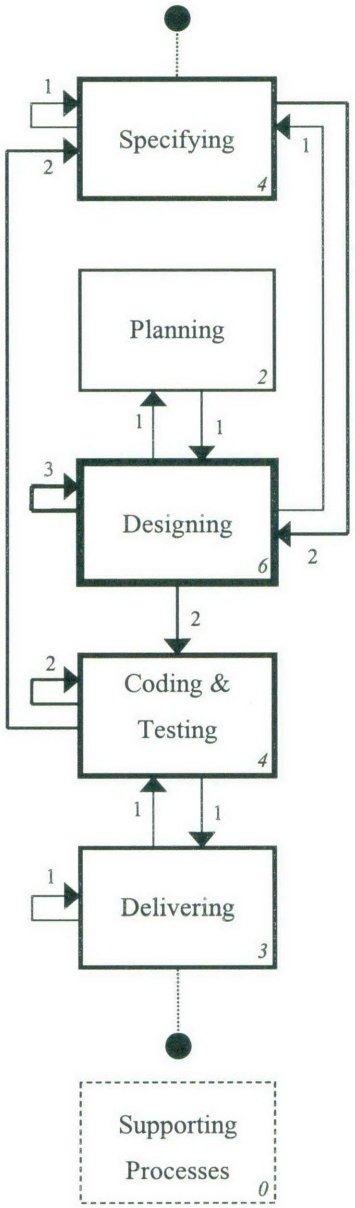

Figure 5-2 Abstraction diagram for the Waterfall model

Phase	Details Index (Node)	Details Index (%)
Specifying	4.00	21.05%
Planning	2.00	10.53%
Designing	6.00	31.58%
Coding & Testing	4.00	21.05%
Delivering	3.00	15.79%
Supporting Processes	0.00	0.00%
Expanse (E)	**19.00**	**100%**

Table 5-1 Details Index of the Waterfall model

Phase	Weighted Interaction index	Weighted Interaction index (%)
Specifying	7.00	19.44%
Planning	2.00	5.56%
Designing	13.00	36.11%
Coding & Testing	10.00	27.78%
Delivering	4.00	11.11%
Supporting Processes	0.00	0.00%
Dynamism (D)	**36.00**	**100%**

Table 5-2 Weighted Interaction Index of the Waterfall model

Process	Expanse	Dynamism	Interaction complexity (IC)
Waterfall	19.00	36.00	18

Table 5-3 Interaction Complexity of Waterfall model

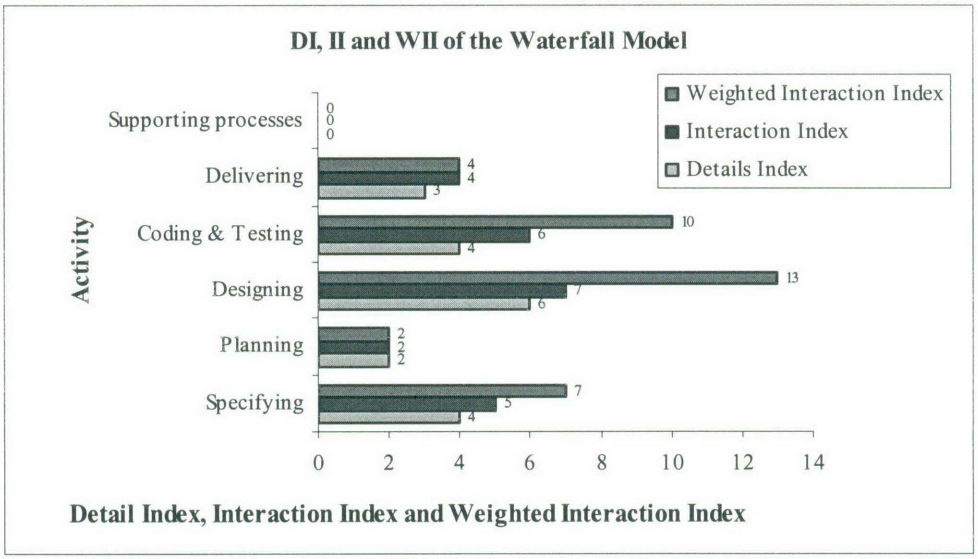

Figure 5-3 Information indexes of the Waterfall model

5.3.2.2 Observations and Analysis

Based on the abstraction diagram and measured values of the Waterfall model, the following observations can be made.

Specifying phase

- Information indexes presented in Table 5-1 and Table 5-2 suggest that specifying activities are a significant part of the Waterfall process. It occupies around 21% of the process activities with only designing having a greater emphasis (see Table 5-1). The waterfall model begins with specifying activity. The high ceremony nature of the model is explicitly illustrated by the intensity of its coverage and interactions to and from designing phase. The interaction out of specifying back to itself either explains that the specifying activity is immediately followed by another instance of specifying activity, for example, defining system specification followed by software specification. While there are interactions between Specifying and other activities such as Designing, the link is not very strong. This shows the sequential nature of Waterfall where a complete specification is needed before going to the next phase. A feedback from the coding and testing activity may trigger the specifying activity to restart; however, this link is also very thin.

Planning phase

- The abstract model shows that Waterfall does not give much emphasis to planning activities compared to other activities such as specifying and designing. Some of the planning activities may occur as part of specifying and designing activities, however.

Designing phase

- The Waterfall model puts great emphasis on designing activities. As seen in Figure 5-2, the designing activity is the thickest one. This explains that the Waterfall model emphasises design over other phases by describing designing in more detail than other activities. In other words, the Waterfall model can be described as a design oriented model. Furthermore, there is a link between specifying and designing as illustrated by the bidirectional interaction between them. It is there because the design activity starts after the specification is done and the team may have to revisit the specification after or during the design activity. However, as explained before, this is not a thick link unlike in some of the other process models discussed later.

Coding and Testing phase

- In Waterfall, the coding and testing activity begins after the design is finalized. According to the abstraction diagram, this activity's emphasis is at the same level as specifying.

Delivering phase

- The Waterfall model ends with the delivering activity which is scheduled to begin after implementation. The coding and testing activity is performed at one point in the delivery phase to verify or validate the software product.

Emphasis in Waterfall Process

- The Waterfall model emphasizes designing activities. It defines preliminary design and detailed design activities separately (Royce 1970). Similarly, it separates system specification and software specification, also coding and testing. *Unlike the Waterfall model, contemporary software processes describe coding and testing together.*

Limitations

- According to Table 5-1, more than 50% of process steps are dedicated to specifying and designing activities; these happen before the actual implementation can begin. That means a large part of the time spent in the software project may be wasted if the implementation phase fails to deliver the product according to the requirements and designs specified earlier.
- The delivery occurs only after the whole project is implemented. The links out of the delivering activity are thinner resulting, for instance, from the lack of incremental delivery and lesser customer involvement during implementation in Waterfall.
- In the Waterfall model where complete specification and design are required before proceeding to the implementation, adding or modifying requirements during development is not part of the model.

5.3.3 Spiral model

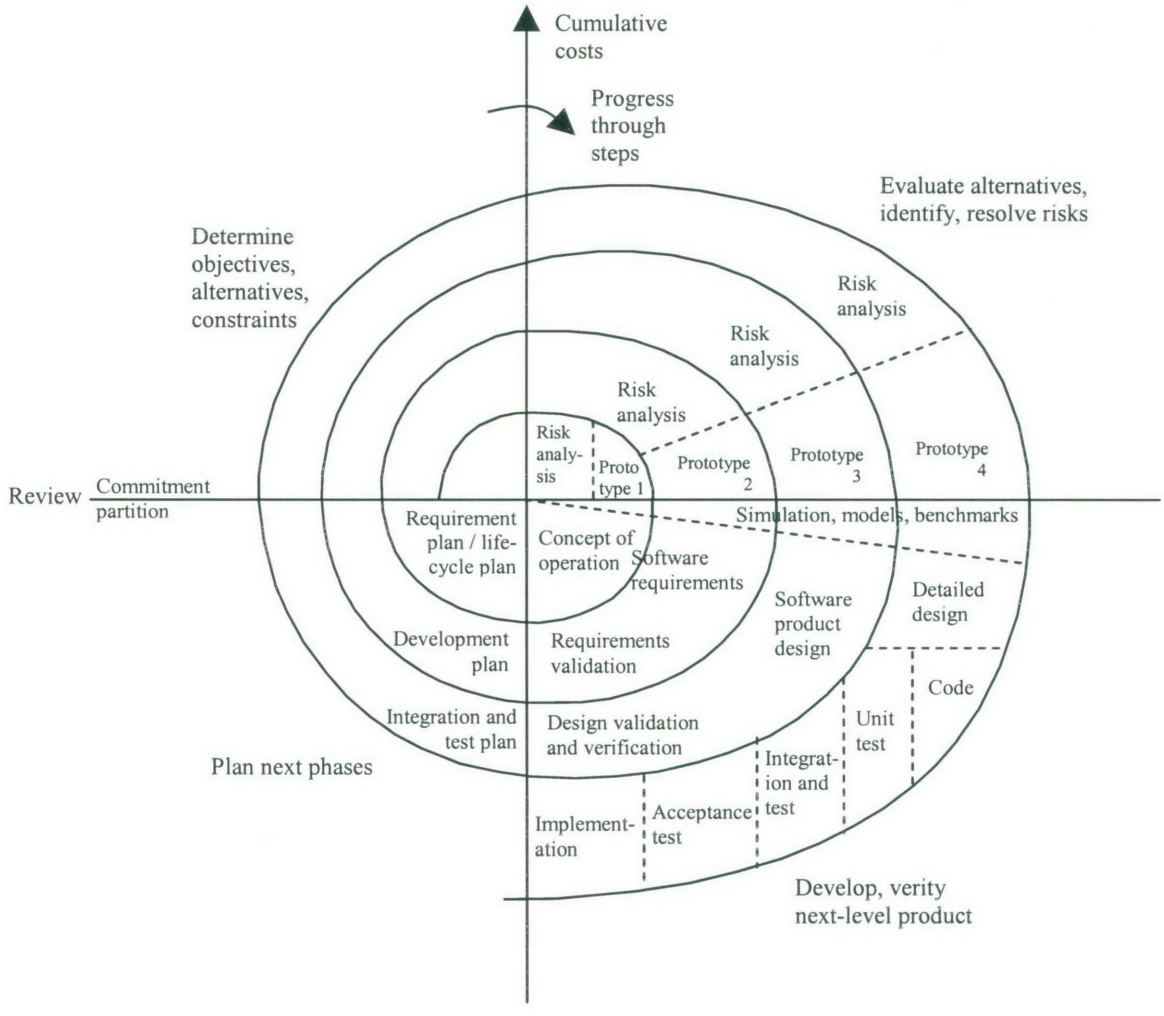

Figure 5-4 The original Spiral model. From (Boehm, B.W. 1988)

The base reference used to model the Spiral process is Boehm (1996; 1988).

The Spiral model is recognized as one of the first iterative development models. In fact, it incorporates evolutionary prototyping into the phases in the Waterfall model. Boehm stated in his paper that the advantage of the model is accommodating good features of existing software process models. An implementation of the model may follow the classic Waterfall phases, a series of evolutionary prototyping, or a combination of both. The life cycle of the model is based on risk analysis and finding solutions. When risks are resolved, the prototyping may be addressed but not implemented and the model sequentially follows the traditional Waterfall model. On the other hand, when relative risks exist, the model suggests a development of the prototype to reduce risks. In the latter case, the Waterfall specification phases may be addressed

but not implemented. The Spiral model is, therefore, a risk-driven software development model.

5.3.3.1 Abstraction

The abstraction of the spiral model, shown in Figure 5-5, is derived based on the publications by Boehm (1996; 1988). The values measured from the abstraction are shown in Table 5-4, Table 5-5, Table 5-6, and visualized in Figure 5-6. Observations made based on the abstraction are then presented in Section 5.3.3.2.

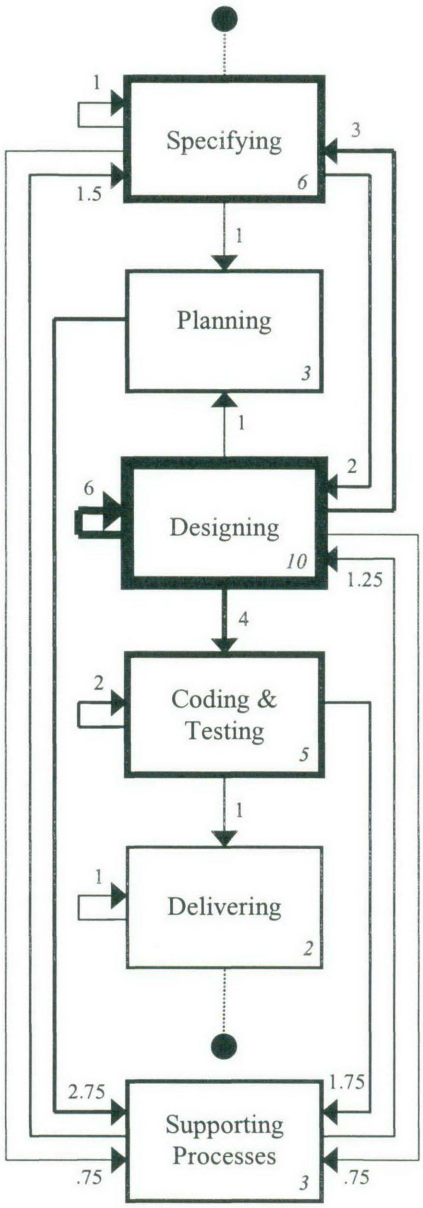

Figure 5-5 Abstraction diagram for the Spiral model

Phase	Details Index (Node)	Details Index (%)
Specifying	6.00	20.69%
Planning	3.00	10.34%
Designing	10.00	34.48%
Coding & Testing	5.00	17.24%
Delivering	2.00	6.90%
Supporting Processes	3.00	10.34%
Expanse	**29.00**	**100%**

Table 5-4 Details Index of the Spiral model

Phase	Weighted Interaction Index	Weighted Interaction index (%)
Specifying	10.25	16.67%
Planning	4.75	7.72%
Designing	24.00	39.02%
Coding & Testing	10.75	17.48%
Delivering	3.00	4.88%
Supporting Processes	8.75	14.23%
Dynamism	**61.50**	**100%**

Table 5-5 Weighted Interaction Index of the Spiral model

Process	Expanse	Dynamism	Interaction complexity
Spiral	29	61.50	33.50

Table 5-6 Interaction Complexity of the Spiral model

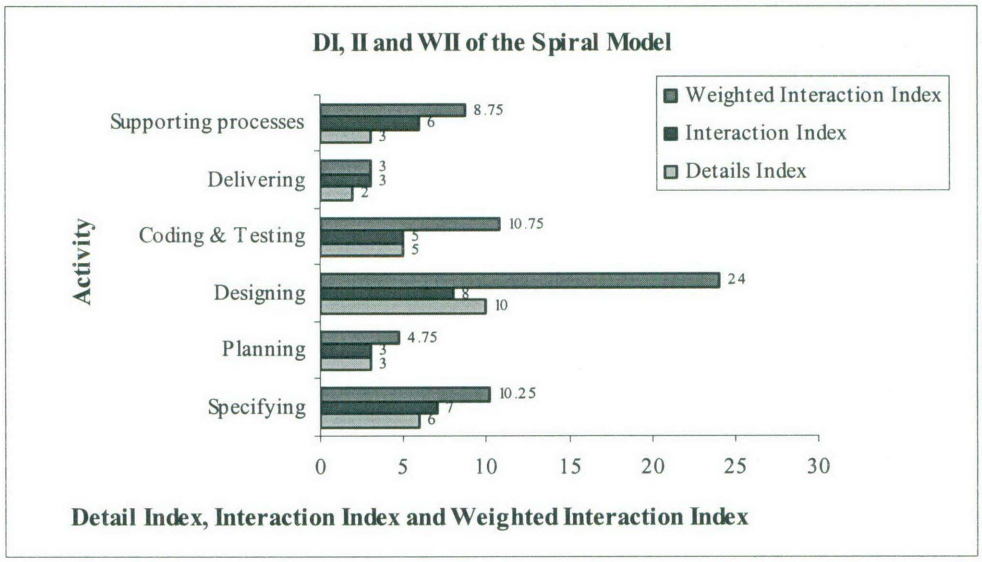

Figure 5-6 Information indexes of the Spiral model

5.3.3.2 Observation and Analysis

Specifying phase

- The Spiral model begins with making a specification. This activity is repeatedly addressed and, at least, elaborately described once. (In the latter case, software requirements specification is followed by requirements validation.) Strong

connections between specifying and designing, indicated by thick arrows, imply that the specifying-designing pair is frequently executed. After the specifying activity, a plan is crafted. Some supporting processes may be also performed to assist in specification phase. Specification review and evaluation are example activities performed between specifying and performing support processes.

Planning phase

- The Spiral model gives reasonable attention to planning. Planning starts immediately after the specification or design is created or updated. Following a planning activity, some supporting processes are performed. This implies that the Spiral model encourages the use of planning to support or to assist the rest of the development process.

Designing phase

- The Spiral model certainly puts focus on designing as illustrated by the thickest node in the abstraction diagram.. Initially, the designing activity commences after receiving input from the specifying activity. The design is later refined as specification is added or updated. The plan is also created or updated. The main input to designing comes from specifying. During designing, some supporting processes may be carried out.

Coding and Testing phase

- Coding and testing are scheduled to start after an appropriate amount of design and specification is accomplished. This phase occurs several times during the life cycle, but is moderately elaborated in the specification. Some kind of supporting processes is involved after a portion of coding and testing is done. This indicates that coding and testing in the Spiral model are not performed in a high ceremony fashion as in the Waterfall model.

Delivering phase

- The Spiral model ends at this phase. The delivering activity occurs at least twice during the development process, but poorly elaborated. This phase begins after the complete product is achieved from a series of coding and testing activities.

Supporting Processes

- Supporting processes play an important role in the Spiral model. Every phase, except delivering, is engaged with some supporting processes at some point. The activity categorized in this phase may involve enforcing the plan, approving the design, reviewing the prototype and so on.

Emphasis in Spiral Process

- The model is risk-driven. Thus, the model is centered on risk analysis and risk resolution.
- No matter which subset of activities is chosen, specifying and designing will be significantly addressed. When the model follows the evolutionary prototyping steps, these activities are repetitively visited and produced portions of specification and design. When the model follows the basic waterfall approach, the specification and design have to be completed before proceeding to the implementation.
- Specifying, designing, coding and testing and supporting processes are iterated until the software product is finalized. The model describes the delivery as a final phase and does not describe what happens after the delivery.

Limitations

- Spiral requires the development team in exploring various design alternatives and making decisions. These steps could put an extra load on the team. Moreover, repetitively working on specifications and designs is time-consuming. A project with tight schedule and an inexperienced project manger could find it hard to manage the Spiral process. When no risk related to the product is presented, the team can omit risk analysis and resolution steps and the Spiral model can become a Waterfall-like model.
- Similar to traditional software development models, the Spiral model's big focus is laid on producing specification and design. Unlike them, it provides a choice of creating a full specification in one go or iteratively adding the specification and design based on evaluation of risks. Spending much time on design does not necessarily lead to project failure, but it surely leaves less time for other phases of the development process. Also, too much design does not necessarily guarantee customer satisfaction of the final product.

5.3.4 Extreme programming

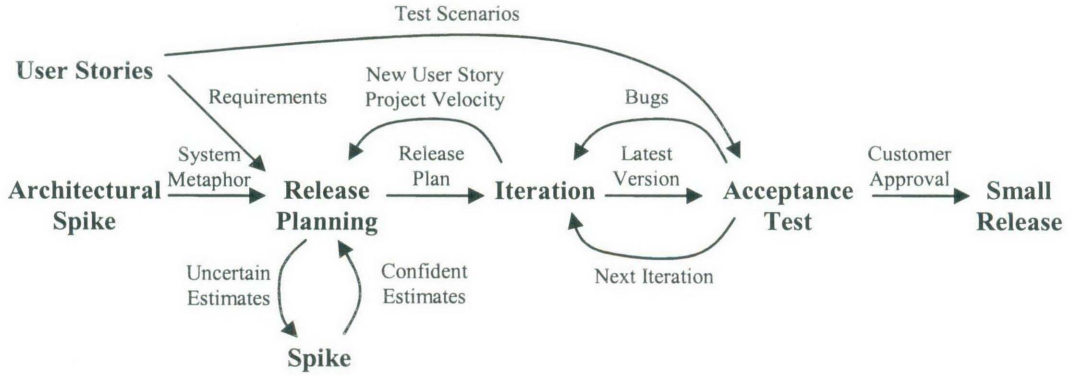

Figure 5-7 Extreme Programming flow chart. From (Wells 2002)

The base references used for modeling the XP process were: (Beck 2000; Cockburn 2002; Wake 2002; Wells 2002).

Several software development methodologies have been developed to counter some of the drawbacks of heavyweight software processes. Extreme programming (or XP in short) is one of these methodologies which addresses itself as a lightweight software development methodology for small projects. Initially, XP was intended for a team of up to 14 people. (A project may contain multiple teams.) Later, there were attempts to involve XP in larger projects (Reifer 2003). Similar to other methodologies of the same genre, XP offers a few rules to follow, leaves room for changes, and aims to shorten the development cycles. The main idea is to spend more time on building the product than on doing complete specification and design before the implementation can begin.

An XP process starts from identifying features the customer needs in their products (Wells 2002). The features with the highest business value are selected for the first release. Then the team creates just enough specification and design for the selected features rather than wasting time to create a complex and complete specification which it may fail to deliver later on. The product is built and integrated daily to make sure errors are discovered and fixed early. XP emphasizes frequent delivery of small releases and encourages the customer to participate in all phases.

5.3.4.1 Abstraction

Figure 5-8 presents the abstraction of XP derived from above base references. Table 5-7, Table 5-8 and Table 5-9 give the values measured from the abstract diagram. Figure 5-9 shows a bar chart of the Details and Interaction Indexes for XP.

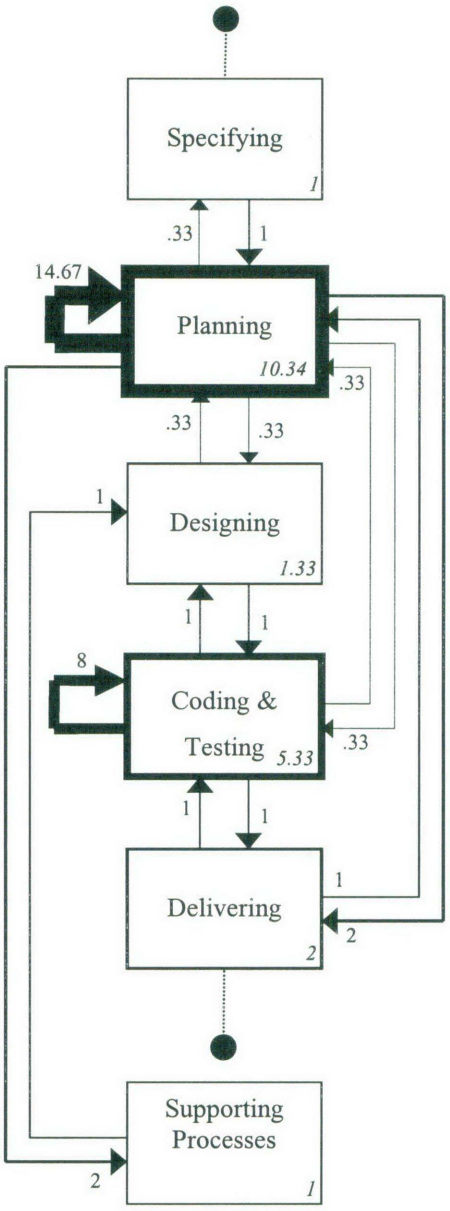

Figure 5-8 The Extreme Programming abstraction

Phase	Details Index (Node)	Details Index (%)
Specifying	1.00	4.76%
Planning	10.34	49.24%
Designing	1.33	6.33%
Coding & Testing	5.33	25.38%
Delivering	2.00	9.52%
supporting processes	1.00	4.76%
Expanse	**21.00**	**100%**

Table 5-7 Details Indexes of a single iteration Extreme Programming

Phase	Weighted Interaction index	Weighted Interaction index (%)
Specifying	1.00	1.47%
Planning	34.68	51.00%
Designing	3.66	5.38%
Coding & Testing	20.66	30.38%
Delivering	5.00	7.35%
supporting processes	3.00	4.41%
Dynamism	**68.00**	**100%**

Table 5-8 Weighted Interaction Index of a single iteration Extreme Programming

Process	Expanse	Dynamism	Interaction complexity
Extreme Programming	21.00	68.00	48.00

Table 5-9 Interaction Complexity of a single iteration Extreme Programming

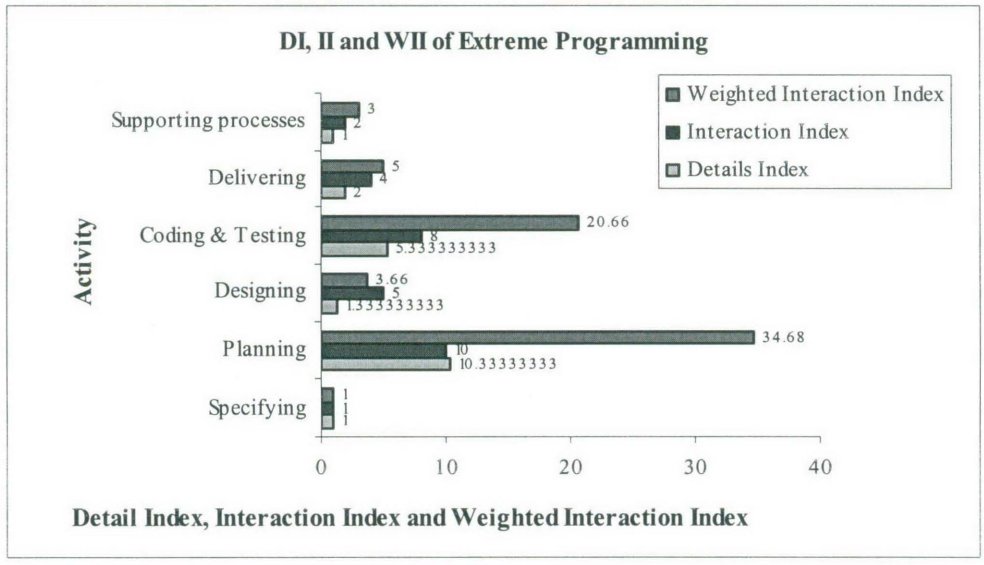

Figure 5-9 Information indexes of the Extreme Programming process

5.3.4.2 Observations and Analysis

Specifying

- An Extreme Programming process begins by specifying the overall functionality of the software product. This is not a complete software requirements specification. The interaction to and from planning indicates that some sort of planning activity occurs during specifying to assist in specification. In addition, it indicates that specification in XP is not done in one go but rather be added or modified.

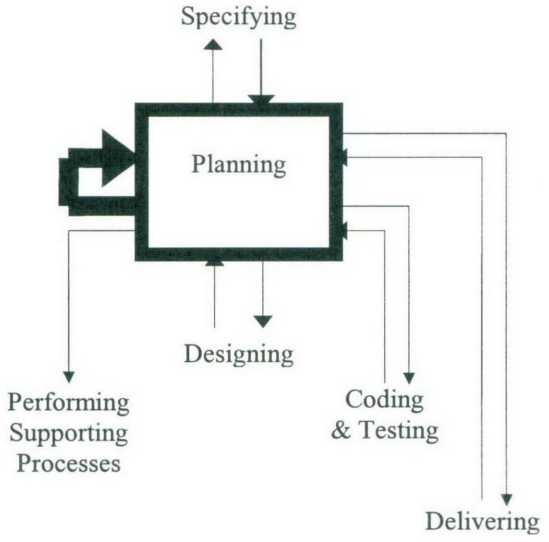

Figure 5-10 XP's Planning activity

Planning phase

- The plan is developed after identification of a set of functionality. During planning, designing or coding & testing may occur. One way to look at this is to consider that activities such as designing are conducted to assist in the planning step, or the plan is progressively created or modified. Planning also involves the use of some kind of supporting processes. The thickness of the planning node in the abstract diagram (Figure 5-8) shows more emphasis the XP process puts on planning activities. The links from and to planning also illustrates the close interaction between planning and other activities in XP.

Designing phase

- XP does not concentrate on designing unlike Waterfall or Spiral models. The design occurs after a series of planning activities and it is relatively quick and iterative.

Coding and Testing phase

- After a design session, developers write code and test it. The interaction to and from delivering indicates that the version of the software product may be delivered to the customer often, or it may indicate that this phase has to be repeated if the result from delivering is not satisfied.

Delivering phase

- Delivering starts early in the XP process during planning. The customer is asked to participate very early. The delivering phase is revisited again when the product is ready to be delivered. Also, it is revisited when the next development cycle begins. This reflects one of XP's main philosophies of "frequent delivery of working software".

Supporting Processes

- Similar to delivering, XP provides supporting processes early in the development process for planning and designing phases. Planning is like a system blueprint. Designing and the rest are like a manufacturing factory with machines and operators. By scheduling some kind of supporting processes before starting the actual manufacturing, operators may inform their progress to other departments or management. They may report problems found during the manufacturing process. Other operators may come up with an idea to fix the problems or they may want to know who can help them with the problem. However, this has to be a quick step as no one would want the product to be delayed.

Emphasis in XP Process

- Although most XP practices are put together around coding and testing activities, planning is what drives a team. Planning is essential in XP. Details indexes of a one-iteration XP process shown in Table 5-7 reflect this perception. XP should be used when the customer requires versions of the system early. A typical XP process uses the planning game to create a simple plan which can be refined later. The planning game establishes the overall project scope and the date the team needs to deliver. Based on the initial plan, the team can optimize it along the way and track the progress to decide what should be done next.
- The second highest Details Index according to Table 5-7 belongs to coding and testing practices. An XP process includes 12 core practices:

TestDrivenDevelopment, PlanningGame, OnsiteCustomer, PairProgramming, ContinuousIntegration, DesignImprovement, SmallReleases, SimpleDesign, SystemMetaphor, CollectiveCodeOwnership, CodingStandard and SustainablePace (Beck 2000). Eight out of the 12 practices are directly associated with coding and testing activities. This methodology with a high emphasis on programming activities certainly provides as many as necessary principles for software projects that require rapid delivery of software product.

Limitations

- XP is a sound process which emphasises early and frequent delivery. It promotes a simple and quick design. XP does not encourage design for future. Developers only do the design before coding. The design phase takes up around 6% of the process. Similarly, supporting processes are not emphasized in XP. The lesser emphasis on designing activity in XP seems to be a weakness of the process model.

5.3.5 Scrum

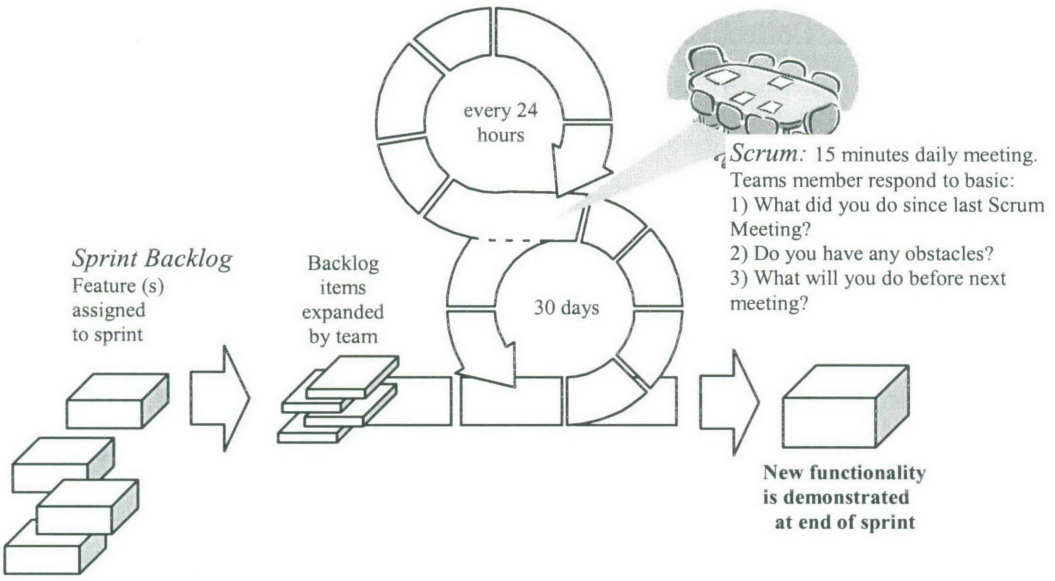

Figure 5-11 Scrum process from (Schwaber & Beedle 2002)

The publication by Schwaber & Beedle (2002) is used as the base reference for modeling Scrum.

Scrum is defined as a "knowledge creating process." It is assumed that process requirements are complicated, unpredictable and cannot be completely identified at one time. In addition, repeating the same processes in a success story might not ensure the same level of achievement in another circumstance. Therefore, although its main goal of delivering quality software products is not different from other software development approaches, Scrum expects unpredictability during software development and encourages self-organizing knowledge creation and sharing.

Scrum process maps the five traditional software development phases (requirement, analysis, design, evolution and delivery) into a 'Sprint' or 'series of Sprints'. A Sprint is a short development interval that typically lasts about a month. Each Sprint is driven by the list of work items called 'Backlog'. At the end of a Sprint, the product built during a Sprint is demonstrated and assessed.

5.3.5.1 Abstraction

The abstraction diagram for the Scrum process is given in Figure 5-12 followed by the Details Indexes and Interaction Indexes.

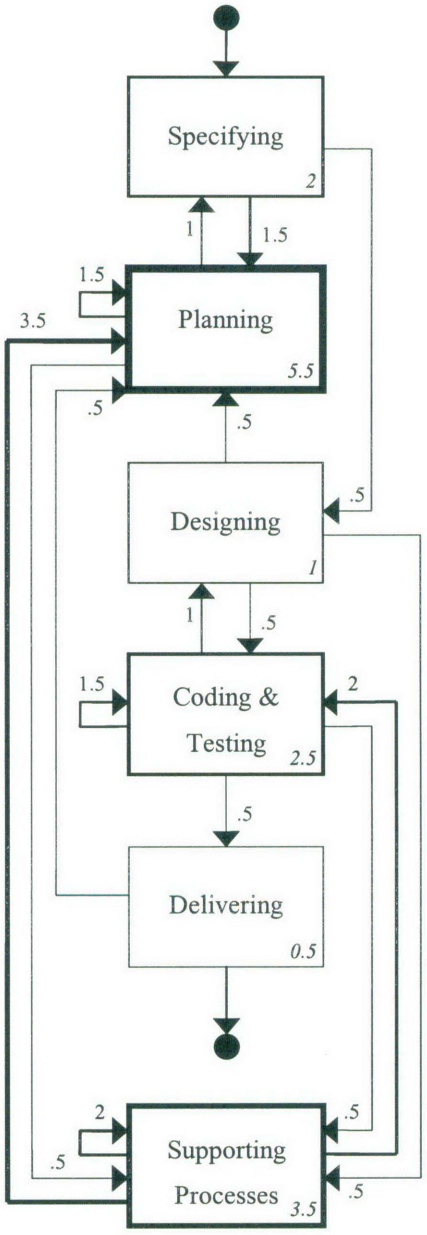

Figure 5-12 The abstraction diagram for Scrum model

Phase	Details Index (Node)	Details Index (%)
Specifying	2.00	13.33%
Planning	5.50	36.67%
Designing	1.00	6.67%
Coding & Testing	2.50	16.67%
Delivering	0.50	3.33%
Supporting processes	3.50	23.33%
Expanse	**15.00**	**100%**

Table 5-10 Details Indexes of a single iteration Scrum

Phase	Weighted Interaction index	Weighted Interaction index (%)
Specifying	3.00	8.33%
Planning	10.50	29.17%
Designing	3.00	8.33%
Coding & Testing	7.50	20.83%
Delivering	1.00	2.78%
Supporting Processes	11.00	30.56%
Dynamism	**36.00**	**100%**

Table 5-11 Weighted Interaction Index of a single iteration Scrum

Process	Expanse	Dynamism	Interaction complexity
Scrum	15.00	36.00	22

Table 5-12 Interaction Complexity of a single iteration Scrum

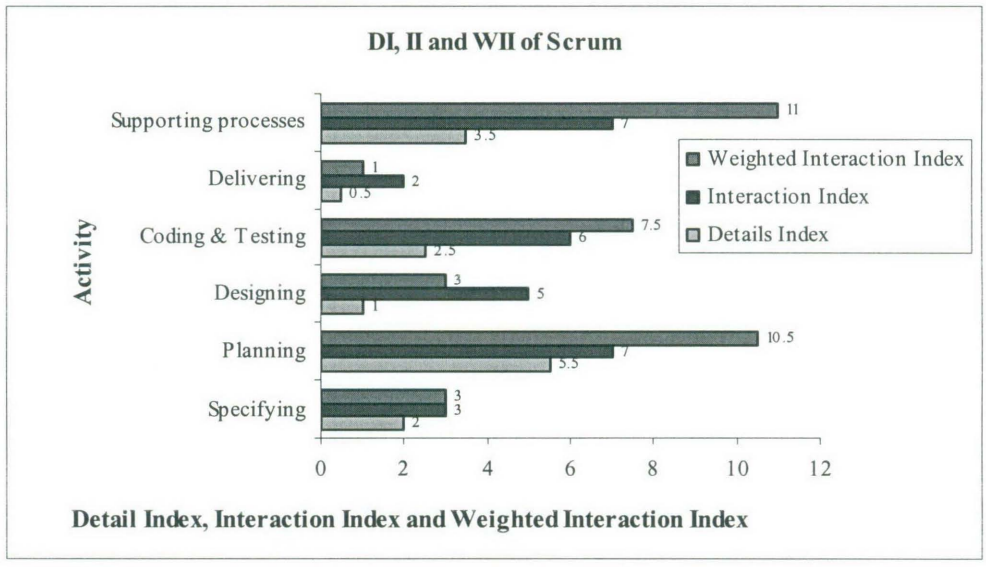

Figure 5-13 Information indexes of the Scrum process

5.3.5.2 Observations and Analysis

Specifying phase

- A process begins by creating a product specification. Similar to other lightweight process models, Scrum does not encourage the development team to spend too much time on creating specifications.

Planning phase

- Planning is the most emphasized phase in Scrum. The planning phase can be started as soon as the first specification is achieved. A sequence of planning activities is performed to establish the overall scope of the project and the important dates. After the design is proposed, it may result in a plan adjustment. The adjusted plan may result in additions or changes to the specification.

Designing phase

- Similar to XP, designing in Scrum is as simple and as quick as possible. Developers design prior to coding.

Coding & Testing phase

- Scrum does focus on coding and testing as much as other lightweight processes and it does show in the process model. Unlike other processes, Scrum provides supporting processes to coding and test phase.

Delivering phase

- The delivery in Scrum starts late in the development cycle. After delivering, the plan may be adjusted for the next development cycle.

Supporting Processes

- Scrum is certainly different from other lightweight processes in the sense that it provides more details on supporting processes. Most other lightweight processes supply more process details on the actual development activities such as coding and testing.

Limitations

- Scrum pays much attention to supporting processes rather than the specification or implementation. This is indicated by low values for specifying, designing, delivering, and coding and testing activities compared to the high values for

planning and performing support processes activities. In this way, Scrum may not be suitable for some projects that need more direction on technical side of software development.

5.3.6 Crystal Clear

Cockburn (2000; 2002; 2005) is used as the base reference for Crystal Clear.

Crystal is a family of methodologies developed by Alistair to reflect the idea that each project needs its own methodology. Crystal methodologies recognize different types of software projects based on key project aspects such as team size, system criticality and project priority. Crystal allows developers to utilize the methodology that fit their requirements and needs. The methodologies are indexed by color; Clear (1-6), Yellow (-20), Red (-40), Magenta (-100), blue (-200), violet (-500), and so on, to indicate their hardness dimension.

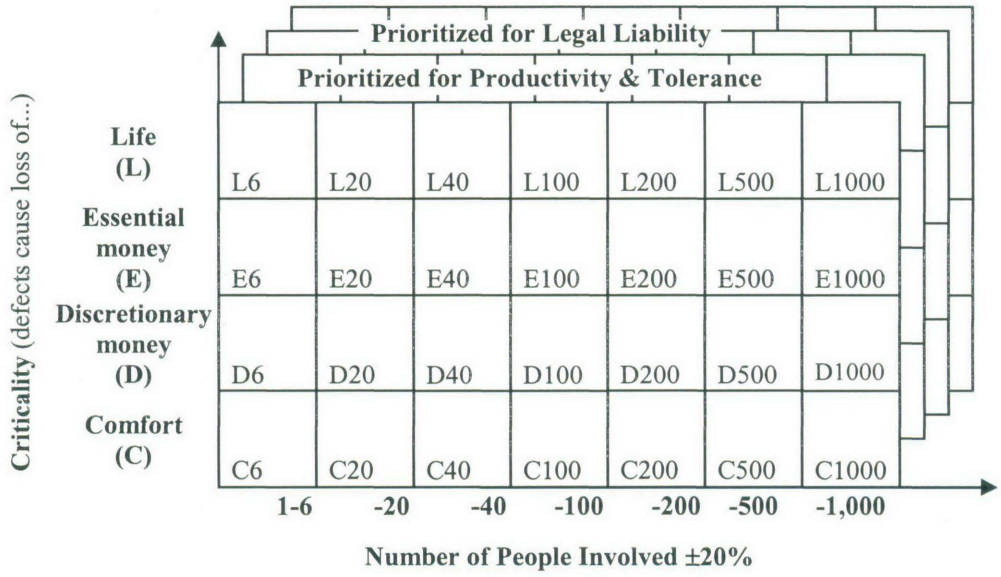

Figure 5-14 Crystal methodologies organized as people x criticality x priority[2]

Crystal Clear is a light process developed for projects that fell into D6 category (Discretionary money with one to six people involved). It encourages a team to sit in the same room with some workstations and whiteboards and requires direct user participation in the project. Software is delivered incrementally with a short cycle of two or three months.

---

[2] http://alistair.cockburn.us/index.php/Crystal_light_methods

Crystal Orange is for larger teams (up to 40 developers) and, thus, involves more ceremony than Crystal Clear. It also calls for frequent delivery with the cycle period that may be extended to three or four months.

Crystal Orange Web is a modification of Crystal Orange shaped for web development with up to 50 people. It deals with a "continuous stream" of initiatives requiring programming.

One of the main principles of Crystal methodologies is that the team is allowed to choose the practices that suit their process and environment. Developers may incorporate some development practices from other methodologies such as Scrum, XP, etc. Nevertheless, there are two restrictions on Crystal methodologies. Firstly, the authors point out that it is not tested on life-critical systems. (This could be true for some of the other agile process models as well.). Another restriction is their focus on collocated teams.

In this chapter, we select Crystal Clear as a representation the Crystal methodology for a light process.

5.3.6.1 Abstraction

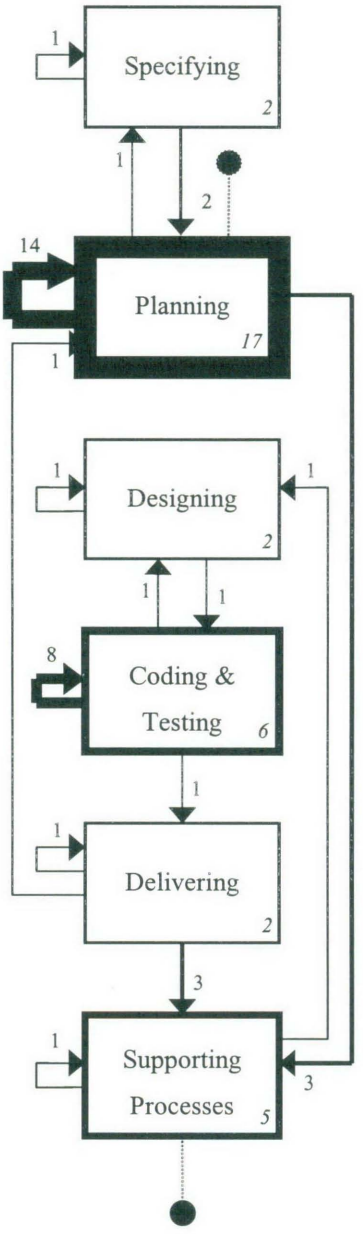

Figure 5-15 The abstraction diagram for Crystal Clear model

Phase	Details Index (Node)	Details Index (%)
Specifying	2.00	5.88%
Planning	17.00	50.00%
Designing	2.00	5.88%
Coding & Testing	6.00	17.65%
Delivering	2.00	5.88%
supporting processes	5.00	14.71%
Expanse	**34.00**	**100%**

Table 5-13 Details Indexes of the single iteration Crystal Clear process

Phase	Weighted Interaction index	Weighted Interaction index (%)
Specifying	5.00	6.25%
Planning	35.00	43.75%
Designing	5.00	6.25%
Coding & Testing	19.00	23.75%
Delivering	7.00	8.75%
Supporting Processes	9.00	11.25%
Dynamism	**80.00**	**100%**

Table 5-14 Weighted Interaction Index of a single iteration Crystal Clear

Process	Expanse	Dynamism	Interaction complexity
Crystal Clear	34	80.00	47

Table 5-15 Interaction Complexity of a single iteration Crystal Clear

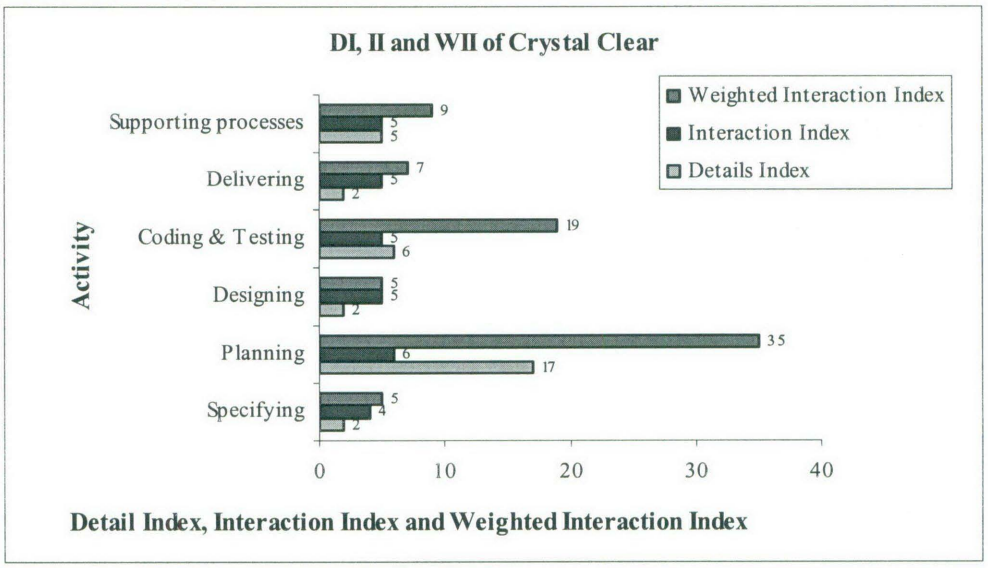

Figure 5-16 Details Index of the Crystal Clear process when a number of iteration is ranged from 1 to 10

5.3.6.2 Observations and Analysis

Specifying phase

- Specification is started after the plan is initiated. An interaction from the specifying node to itself in Figure 5-15 is an indication of Crystal Clear specification providing more details of the iterative nature of specifying activity.

Planning phase

- Crystal Clear starts the process from planning phase. It begins with building the team that will take part in its software development project. Then the selected team proceeds to create the plan. A Crystal Clear process does provide much detail regarding the planning activities.

Designing phase

- Designing in Crystal Clear starts after an initial plan is created. Designing is a simple and quick session as in other typical lightweight processes.

Coding & Testing phase

- After a design session, coding and testing occur. The next design session begins after developers finish with integration. A sequence of design-coding-testing activities is performed until the product is ready for delivery.

Delivering phase

- After a series of designing-coding-testing activities, the product is delivered. Crystal Clear provides some detail on how the delivery phase is done. After delivery, the project plan may be adjusted for the next development cycle.

Supporting Processes

- Even though Figures 5-13 and 5-16 indicate that both Scrum and Crystal Clear provide supporting processes, Crystal Clear provides supporting processes in a different way. It encourages the use of supporting processes to assist in planning and delivering. After each delivery, the team performs a supporting process to discuss about progress, problems and issues regarding the product and development process.

Emphasis in Crystal Clear

- Planning is the most emphasized phase from every single perspective. Similarly, the majority of the metrics indicate that Coding & Testing phase is the second most focused phase in the Crystal Clear model. However, very interestingly, based on the abstract model, there are no interconnections between these two critical phases. The only channel allowing Planning and Coding & Testing to connect is though a sequential set of Supporting processes, Designing and Delivering phases. The lack of direct interactions between the most critical phases might result in a delay of the entire cycle.

Limitations

- Crystal Clear encourages software development to embrace more of supporting processes. It introduces a supporting process after a delivery. The process could be one that aims to support individual development phases or the whole development process, or support the team in other ways than technical issues. A highly experienced team may find this kind of supporting processes useful. Others may not.
- Since there are not many interactions between phases, the interaction based metrics cannot be efficiently used for depicting the model.

5.3.7 Feature-Driven Development (FDD)

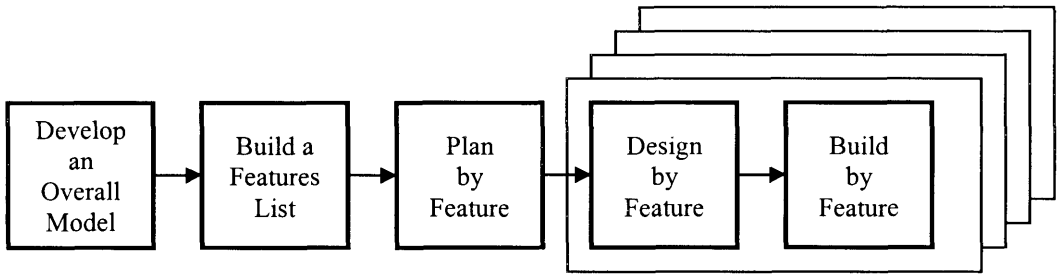

Figure 5-17 FDD process

The publication by Palmer & Felsing (2002) is used as the base reference used for modeling Feature-Driven Development (FDD).

FDD (Palmer & Felsing 2002) is a process that aims to satisfy user requirements by identifying them as features and implement a set of features in short cycle time of two weeks or less. FDD consists of five processes. The first three processes are done sequentially, starting from developing an overall model to building a features list and to planning by feature.

The first phase of FDD is to develop an overall model by initially presenting a scope in high-level, creating a common model and placing more details on the model. After that a feature list, a list used for identifying needed features, is built and prioritized. Features with similar business-value are grouped. The plans and milestones for later processes are proposed based on the result of the earlier activities. Then, after this stage, the overall completion date can be estimated. FDD cycle begins with a design by feature (DBF) process and follows by a build by feature (BBF) process. In features implementation, team leaders are appointed as a chief programmer and class owners. The chief programmer leads both DBF and BBF processes, and coordinates on identifying classes and assigns features to class owners. The class owners are responsible for coding. FDD rules a class owner to hold a class to gain a sense of ownership and for consistency reason. This is different from XP that relies on the pair programming concept. FDD also provides process tracking using schedule-percentage guidelines to let teams track progress with precision.

5.3.7.1 Abstraction

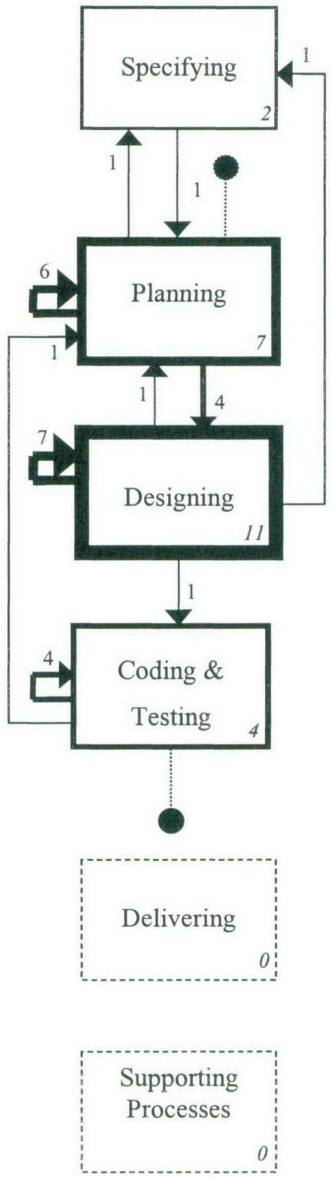

Figure 5-18 The Abstraction diagram for Feature-Driven Development model

Phase	Details Index (Node)	Details Index (%)
Specifying	2.00	8.33%
Planning	7.00	29.17%
Designing	11.00	45.83%
Coding & Testing	4.00	16.67%
Delivering	0.00	0.00%
Supporting processes	0.00	0.00%
Expanse	**24.00**	**100%**

Table 5-16 Details Indexes of a single iteration FDD

Phase	Weighted Interaction index	Weighted Interaction index (%)
Specifying	3.00	5.56%
Planning	20.00	37.04%
Designing	21.00	38.89%
Coding & Testing	10.00	18.52%
Delivering	0.00	0.00%
Supporting Processes	0.00	0.00%
Dynamism	**54.00**	**100%**

Table 5-17 Weighted Interaction Index of a single iteration FDD

Process	Expanse	Dynamism	Interaction complexity
FDD	24.00	54.00	31

Table 5-18 Interaction Complexity of a single iteration FDD

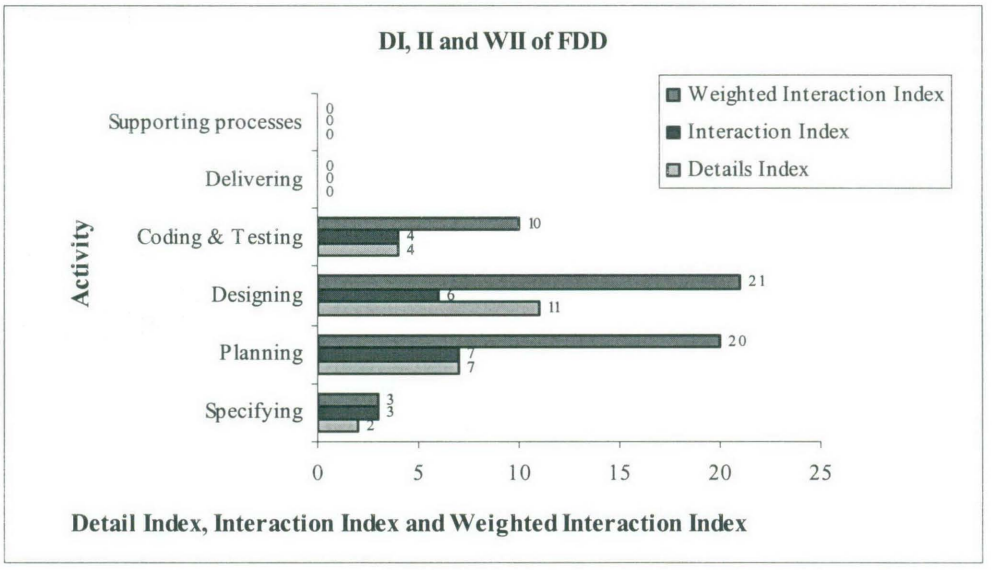

Figure 5-19 Information indexes of the Feature-Driven Development process

5.3.7.2 Observation and Analysis

Specifying phase

- The specification is created following the initial project plan. The specification may be added or changed after a series of design steps, and that may result in plan adjustment.

Planning phase

- Similar to Crystal Clear, an FDD process begins with planning. FDD values planning like other processes of the same kind but the connection between planning and designing appears to be stronger.

Designing phase

- FDD has a strong focus on designing. It provides a lot more detail on designing than other lightweight processes. FDD distinctively defines designing as a single phase and provide elaboration of the phase.

Coding & testing phase

- Coding and testing in FDD are similar to other processes. It is driven by design. Then the resulting product drives the planning of the next development cycle.

Delivering phase

- FDD process specification does not explicitly describe delivery. Therefore, there is no elaboration for the delivery phase in the FDD model.

Supporting Processes

- Similar to delivering, FDD does not elaborate any supporting processes.

Limitations

- The main limitation is that FDD does not provide enough detail on delivering and any supporting process that may assist a team in following its process. Thus, FDD may be a practical approach for a highly experienced software development team. For some team that lacks expertise, FDD may not be a good start.

5.4 Comparative analysis

Characteristics of selected software processes are discussed in previous sections. In this section, these processes are compared side by side. Each development phase is compared.

Specifying phase

Heavyweight processes pay more attention to specification development. (See Figure 5-20) and, therefore, could be costly to implement in projects where specifications change during the development life-cycle. Lightweight processes recognize this downfall and focuses relatively less on specification as an activity.

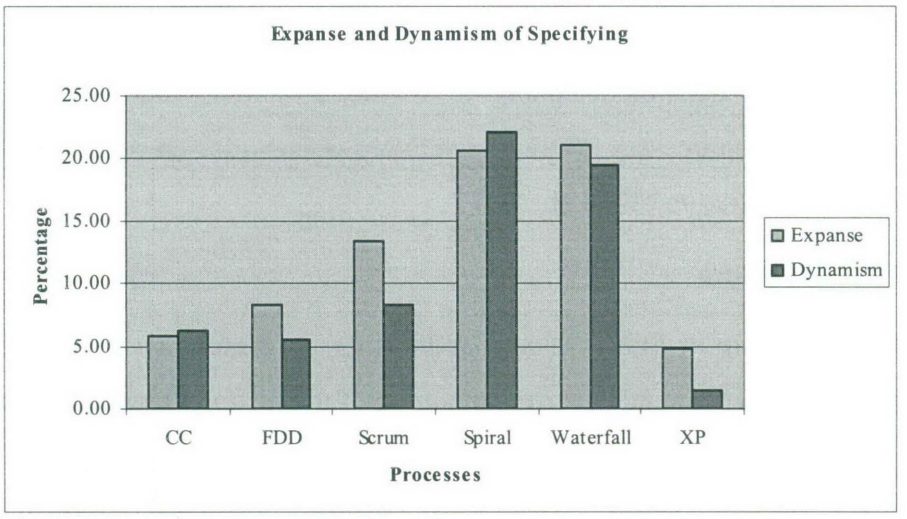

Figure 5-20 Comparison of Specifying

Planning phase

Figure Figure 5-21 shows the value of planning activity in each of the six processes investigated. It shows that lightweight software development processes emphasise planning significantly more than their heavy weight counterparts. Considering that agile processes are a more recent development, it demonstrates the recognition by process designers of the importance of planning in modern software projects.

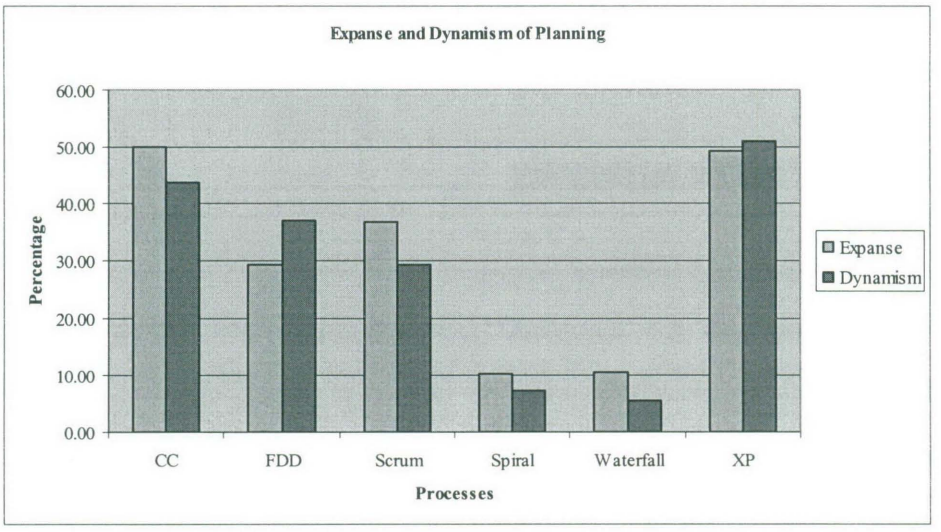

Figure 5-21 Comparison of Planning

Designing phase

Lightweight processes generally provide fewer details on the design activity compared to heavy weight processes (See Figure 5-22). In such processes, design could happen as a prelude to coding. An exception is FDD which does focus on designing. Thus FDD could be a candidate for design-focused projects which need a light-weight process.

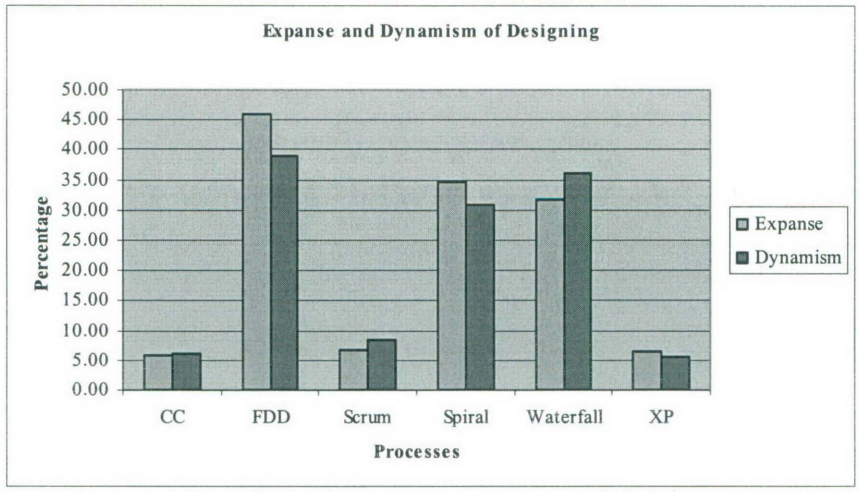

Figure 5-22 Comparison of Designing

Coding & Testing phase

Irrespective of the type of process—lightweight or heavyweight—coding and testing activities are still an important focus for any software development process. According to Figure 5-23, XP, an agile process, along with heavy-weights Spiral and Waterfall provide the highest emphasis on coding and testing activities.

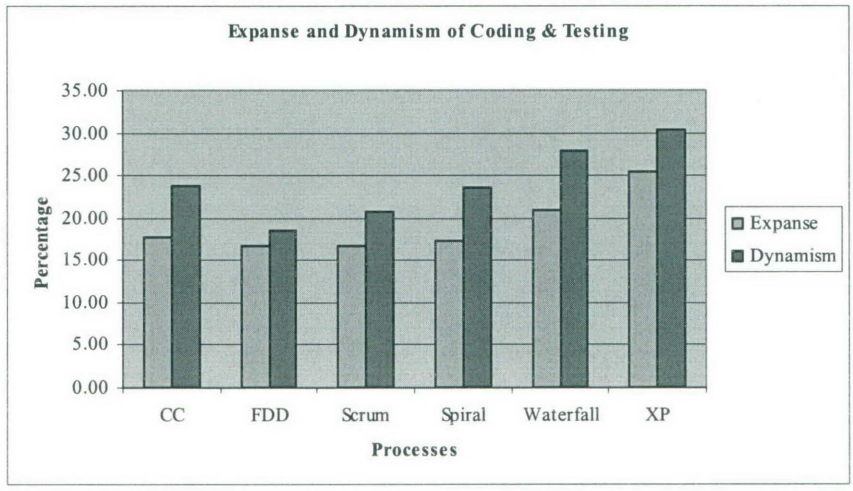

Figure 5-23 Comparison of Coding & Testing

Delivering and Supporting processes

Not every software process has a focus on the delivering activity (Figure 5-24) or in providing supporting processes (Figure 5-25). Waterfall, Spiral and XP, and to a lesser extent, Crystal Clear consider delivering as a significant process activity whereas FDD and Scrum specifications do not. Support processes to support the main activities of a software process are provided by Scrum and to a less extent by Crystal Clear. Waterfall and FDD, on the other hand, are silent on support processes (Figure 5-25).

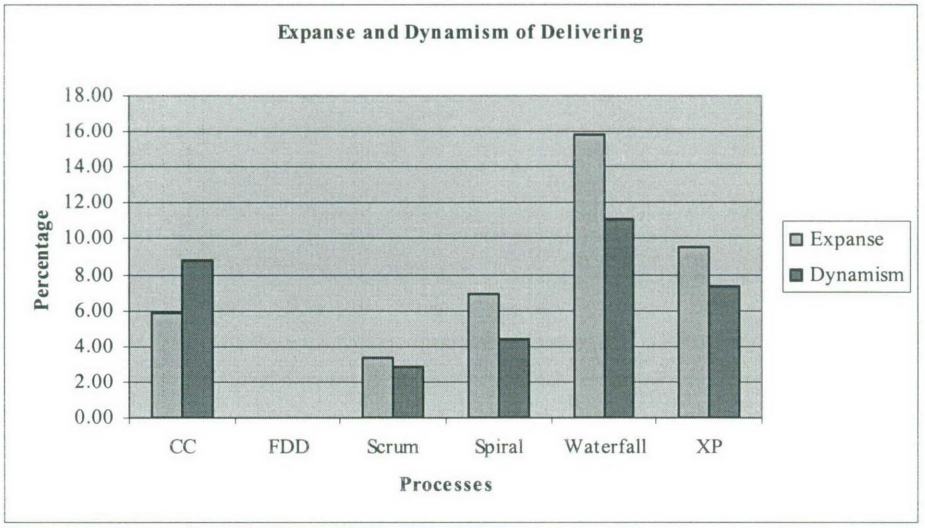

Figure 5-24 Comparison of Delivering

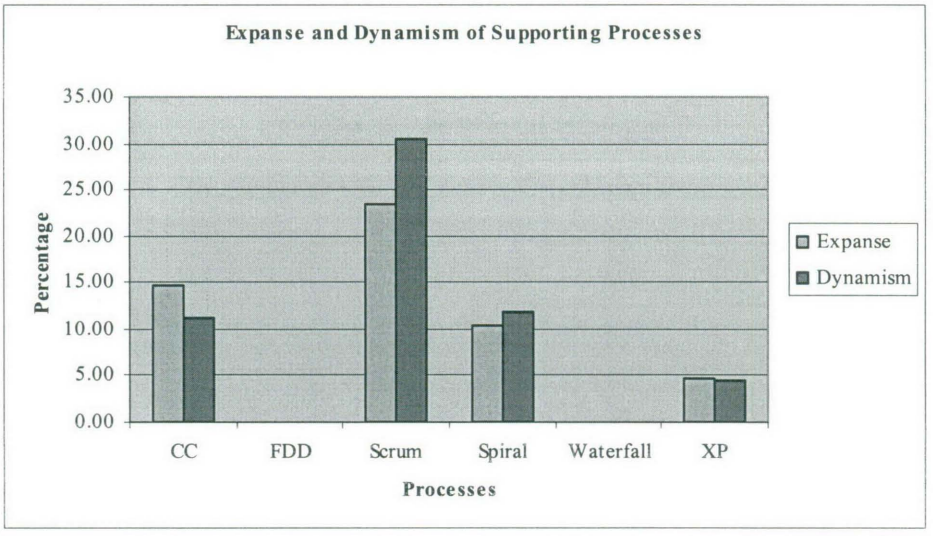

Figure 5-25 Comparison of Supporting processes

Interaction analysis

Lightweight processes tend to interact the same way. So do heavyweight processes. Figure 5-26 illustrates the difference. Heavy weight models—Waterfall and Spiral—follow more or less the same interaction paths. Similarly interactions of light weight processes (XP, FDD, Crystal and Scrum) follow a similar pattern. The difference in the patterns between light weight and heavy weight are significant in planning, specifying and designing activities. It is also worth noting the high interaction in Scrum for performing support-processes.

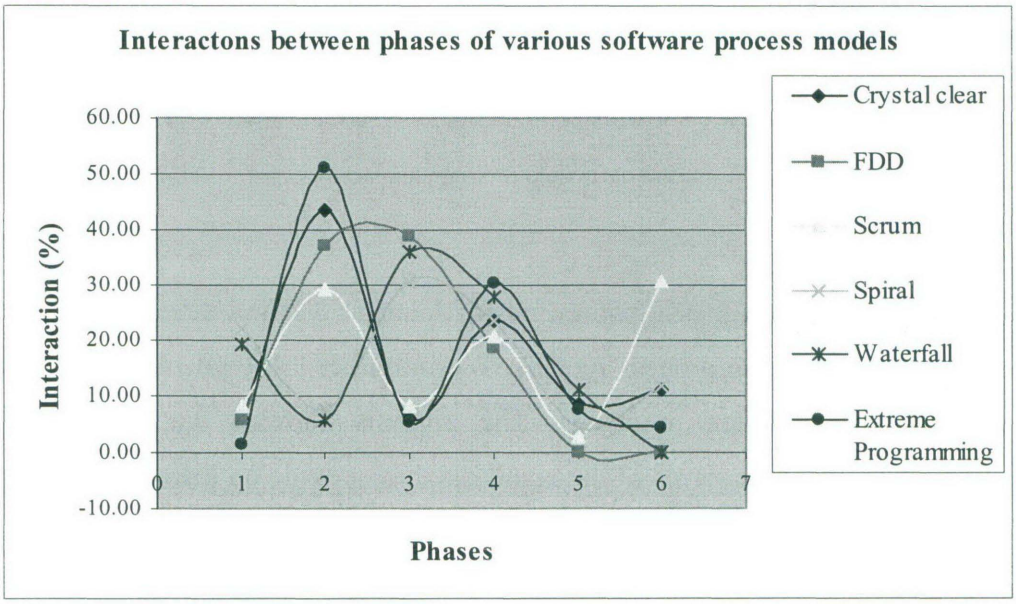

Figure 5-26 Comparison of difference phase's Details Indexes of various processes

Interaction Complexity is a measure of the complex nature of the interactions calculated using the McCabe metric. Waterfall and Scrum seem to have less complex interactions whereas XP and Crystal Clear have higher values for Interaction Complexity. While the high levels of interaction in XP are considered a positive for providing process agility, the high complexity value is an indication that the interactions may be difficult to manage especially in large projects involving a large number of developers.

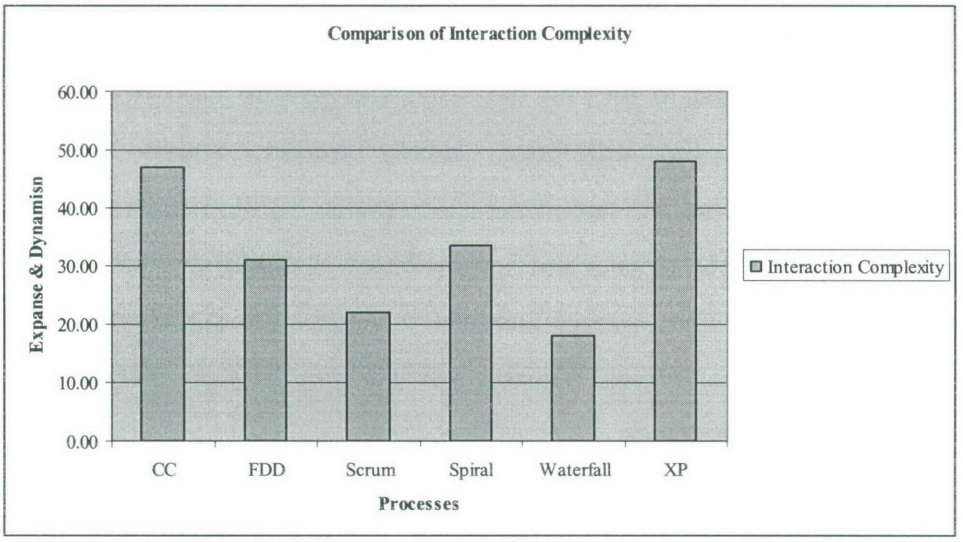

Figure 5-27 Comparison of Interaction Complexity

5.5 Summary

In this chapter we applied the Software Process Analysis Method developed in Chapter 4 to six different software process models. We introduced several metrics to analyze and compare the software processes. The analysis showed the similarities and differences of processes in terms of the emphasis they provide on different core process activities as well as interactions among them. We will further analyse the results in the next chapter.