Information and Software Technology 55 (2013) 795–821

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A taxonomy of design methods process models

Eric Céret [a,*], Sophie Dupuy-Chessa [b], Gaëlle Calvary [c], Agnès Front [b], Dominique Rieu [b]

[a] Joseph Fourier University, Grenoble Informatics Laboratory (LIG), 41 rue des Mathématiques, 38041 Grenoble Cedex 9, France
[b] Pierre Mendès France University, Grenoble Informatics Laboratory (LIG), 41 rue des Mathématiques, 38041 Grenoble Cedex 9, France
[c] Grenoble Institute of Technology, Grenoble Informatics Laboratory (LIG), 41 rue des Mathématiques, 38041 Grenoble Cedex 9, France

ARTICLE INFO

Article history:
Received 22 March 2012
Received in revised form 20 October 2012
Accepted 13 November 2012
Available online 10 December 2012

Keywords:
Process models
Taxonomy
Design methods
Characterization

ABSTRACT

Context: Designers and developers are increasingly expected to deliver high quality systems, i.e. systems that are usable, robust, consistent as well as evolutionary, and that fulfill users' needs. To produce such systems, Design Methods suggest many approaches. However, the important number of existing approaches makes the choice of a method among the others particularly difficult. In addition to this, and because of the time required for understanding (and then operationalizing) new methods, designers tend to use already known methods, even though those which sometimes may not really be adapted to their needs.

Objective: This paper proposes a classification of characteristics of design methods process models. In other terms, it proposes a taxonomy that aims to facilitate the discovery and the choice of methods for designers and developers.

Method: From a study of process models of several design methods, we identify six main axes, namely Cycle, Collaboration, Artifacts, Recommended Use, Maturity and Flexibility, which are in turn divided into 34 characteristics.

Results: This paper provides a deep theorical insight. For each characteristic identified from relevant literature, a definition and a gradation, illustrated using examples, are given. Moreover, it presents a web site that offers various tools for exploring the axes of our taxonomy. This web site provides an overview of process models as well as means for comparing them, textually or graphically. Finally, the paper relates the first evaluation conducted in order to estimate designers' adhesion to the taxonomy in terms of easiness of learning, completeness and intention to use.

Conclusion: We show, based on evaluation results, that our taxonomy of process models facilitates the discovery of new methods and helps designers in choosing suitable methods, really adapted to their needs. Therefore, it enhances chances to conduct high quality projects.

1. Introduction

Several works, such as Garzotto and Perrone [1], Barry and Lang [2] or Fitzgerald [3], analyzing the actual use of system development methods, pointed that designers and developers are poorly satisfied by methods. As a consequence, they tend to modify methods when applying them to practical cases.

The diversity of types of projects (information system, web site, embedded computing, cloud computing, etc.) explains this phenomenon. Indeed, designers' and developers' needs may vary considerably from one project to another: the appropriate method for a specific project depends on the system nature. Yet, the large number of existing design methods [4] does not let the opportunity to practitioners to discover and compare even a small amount of them. Consequently, designers and developers often use already known methods that they eventually tailor in order to meet their needs. Fitzgerald [3] noticed: "*a unique methodology-in-action is enacted for each development project*".

In order to overcome the issues discussed above, we propose a taxonomy, a conceptual tool which analyzes and evaluates the main characteristics of design methods. This taxonomy provides system development specialists with an overview of existing methods and helps them identify the relevant ones according to their needs.

To elaborate such a taxonomy, the first step is to identify elements to be classified. Booch [5] defines a method as "*a disciplined process for generating a set of models that describe various aspects of a software system under development, using some well-defined notation*". According to Harmsen [6], a method is "*an integrated collection of procedures, techniques, product descriptions, and tools, for effective, efficient, and consistent support of the engineering process*". Both of these definitions highlight the need for detailed descriptions of a method process as well as detailed descriptions of the

* Corresponding author. Tel.: +33 6 19 26 16 06.
E-mail address: Eric.Ceret@imag.fr (E. Céret).

0950-5849/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2012.11.002

product to be developed. Harmsen argues for tools whilst Booch requires a set of models that can be considered as a language supporting the method.

Based on the two abovementioned definitions, we define a method[1] as a triplet made of a process model, a product model and a collection of tools. In this paper, we focus on process models.

Based on Humphrey's [7], Scacchi's [8] and Sommerville's [9] definitions, we define a design process model as an abstract representation of a design process structured around a set of activities required to conceive a system, fully or partially.

We propose a taxonomy that aims to identify characteristics of process models and associate a gradation to each of them. In order to achieve this goal, we compare several process models. Thus, according to Beaudoin-Lafon [10], our taxonomy embeds the descriptive and evaluative powers of a model. In addition, such a taxonomy is expected to highlight positive as well as negative or missing points in process models. Therefore, it eases the choice of methods for designers and offers to authors opportunities for identifying potential improvements. Thus, according to Beaudoin-Lafon [10], our taxonomy also embeds the generative power of a model.

We identify characteristics of process models. For each characteristic, a definition and a gradation are given. Definitions suggest a unified vocabulary for process models description and gradations represent a means of comparison. Definitions and gradations are illustrated. In addition, their learnability and usability are evaluated.

We introduce Promote (**Pro**cess **Mo**dels **T**axonomy for **E**nlightening choices), a multidimensional taxonomy. Promote is applied to three existing methods in order to show that methods characteristics are underlined and to enlighten to which extent the taxonomy makes the choice of methods easier for designers. We also describe the evaluation of Promote and discuss future work.

2. Related works

Several works describe process models. Such papers are typically organized as follows. Methods are first described and commented before being, possibly, refined and completed. However, to our knowledge, only few studies deal with process models classification and comparison.

Some works comparing methods exist. These works generally focus on approaches underlying methods or on profits from using them. However, some of them compare somehow processes. They can be classified in two categories. In the first category, comparison is possible thanks to informal descriptions of methods. In the second category, comparision is possible thanks to different criteria that describe methods in a more formal manner. Here, our goal is to propose a method that falls into the latter category.

2.1. Methods and process models descriptions

Sommerville describes characteristics of a few methods in a paper divided into three parts [11]. The first part presents limitations due to linear lifecycle. The second part focuses on evolutionary development relying on iterations and increments. Finally, the third part presents a set of recommendations for methods adaption according to a project size and maturity. Later on, the author dedicates a whole book chapter [9] to software processes. In different editions of the same book, he discusses strengths and weaknesses of five process models: Waterfall, evolutionary development, component-based software engineering, and Agile methods. These publications provide an overview of the aforementioned models and intesting insights regarding their applicability. However, Sommerville does not analyze the models using the same criteria, making it difficult to compare them systematically. For instance, the author states that all generic models support iteration. However, he only demonstrates how the Spiral model and the Extreme Programming (XP) are iterative and does not report whether component-based SE supports any kind of interation or not.

The limitation of Sommerville's work lies on the lack of systematic comparison of models.

Recently, Hug proposes a new approach that suggests a combination of fragments of methods [12]. In this approach, the design and development team members are guided at each step of the project lifecycle. Moreover, a set of recommendations help them identifying the method fragments that are mostly adapted to their needs. In order to identify the right fragment, this process compares two main characteristics of methods: the elements the process is focusing on (activities, product, goals, and so on) and the level of abstraction of these elements. However, even if these two characteristics are very interesting, they do not cover all the facets of a process model.

Seffah studies several User Interface (UI) design methods in [13]. Carter et al. present the Putting Usability First methodology in [14] which divides the development process into four main cycles: possibilities analysis, requirements analysis, design and implementation. Ferre et al. [15] focus on usability methods. They study four approaches: Star Life Cycle, ISO 13407, Usage-Centered Design and Usability Engineering Lifecycle. These comparisons enlighten various UI design techniques.

2.2. Methods and process models categorizations

In 1988, Boehm, in his well-known paper that introduced the Spiral model [16], compares four process models in regard to expected maturity of requirements and quality of resulting code.

In [17], Alexander and Davis suggest 20 criteria and apply them to a project, through a process model, in order to evaluate the appropriateness between the project needs and the process model. For each of the 20 criteria, the authors propose a gradation with three possible values. For instance, '*Problem Complexity*' can be '*Simple*', '*Difficult*' or '*Complex*; '*Users' Ability to Express Requirements*' can be '*Silent*', '*Communicative*' or '*Expressive*'. However, Alexander and Davis do not detail the extent of criteria. As a consequence, evaluating their scale is difficult. For instance, one may wonder: what does problem complexity exactly include? Should the designer consider the very narrowly-specialized requirements, the hardness of technical matters, the financial context, all of that, or something else? When is a problem no more simple and when does it become difficult? What is a very difficult problem: is it a problem that is always difficult, or an already complex one? These unanswered questions show that a taxonomy should use explicit terms in order to be really usable. Moreover, conversely to what our process model definition suggests, many of the criteria proposed by Alexander and Davis concern the result (product size, product complexity) or the context (users' experience, frequency of requirement changes, funds availability, etc.) rather than the process model itself. Our work focuses on how the process model manages these different elements and what it recommends in each situation.

Pérez et al. [18] also suggest 20 criteria. However, while Alexander and Davis [17] measure the appropriateness between a process model and a project needs, Pérez et al. identify the properties that are needed for elaborating a process model adapted to a project characteristics.

[1] This paper focuses on design methods, i.e. methods that guide designers in the creation of an application. To avoid any confusion, methods that do not match our definition (e.g. that only cover one aspect of a method) will no longer be named methods here. They will be referred to as techniques or procedures.

One should note that the criteria proposed by Pérez and his colleagues are difficult to use due to the lack of definitions and explicit gradations.

Sharon et al. [19] discuss characteristics of software development processes and link process suitability to projects particularities. The authors describe projects characteristics using a numeric gradation. In order to identify the appropriate process for a given project, the project characteristics are summed up and compared to the same kind of sum for the process models. The best suitable process is the process with the maximum score. As each characteristic has its own score, the development team can refine these results.

Sharon's and colleagues' work helped us reinforce our selection of characteristics. However, it is worth noting that this work focuses on description of projects characteristics far more than on processes, which are the heart of our endeavors.

Card [20] lists common characteristics of UIs that he considers as being successful. First, the author lists methods that have made these successes possible and classifies them using three characteristics: (1) *problem identification*, i.e. the extent to which the method help identify problems, (2) *artifact construction*, i.e. the stages that the process model suggest in order to build artifacts, and (3) *evaluation*, i.e. the tools that the process model suggest for evaluation implementation. Then, the author refines these three characteristics: he divides them into 36 techniques coming from four design methods. For instance, *problem identification* measures use of techniques like "*Cut and Try*" or "*Field Interview*". A few words describe each technique, e.g. *participatory design* is explained as follows: "*the users are encouraged to make suggestions and to evaluate the prototypes*". Each technique is also classified either analytical (information is extracted by analysis) or synthetical (information is deducted by building something).

Card proposes a classification of techniques and reveals a lot about UI design methods comparison. However, his work presents several limitations. First, definitions are not very precise, conveying only a general idea regarding the what (to do) while neglecting the how (of doing the what). Second, Card's definitions are sometimes ambiguous. For instance, *cut and try* or *participatory design* are both part of two categories (artifact building and problem identification). The author does not clarify whether it is about analyzing twice the same techniques or achieving the same task in both situations. Thus, the process is not fully clarified. Third, Card does not study any scale for techniques: in his classification, a method fully recommends something or does not mention it at all.

None of the works discussed above proposes a complete characterization system. This motivated us to elaborate Promote, a taxonomy for process models. Promote is described in next section.

3. Global overview on promote

Taxonomy is the process of naming and classifying entities into groups within a larger system, according to their similarities and differences. This definition guided our approach. First, we have studied 15 surveys comparing or classifying methods, and analyzed 41 methods or partial process models. Thanks to this review of literature, we have identified main similarities and differences between process models that were abstracted to define the categories of entities we called axes. This analytical work has been confronted to around 450 scientific publications and books about methods and discussed with researchers in method engineering that also contributed to its enrichment.

One of the key points that emerged from our review of literature is the gap between process models in theory and process models in practice. Actually, as clearly mentioned by Fitzgerald [3], process models are usually modified during their operationalization.

Our taxonomy focuses on classifying recommendations expressed in process models. These recommendations have been grouped into six main axes, namely ***cycle***, ***collaboration***, ***artifacts***, ***recommended use***, ***maturity***, and ***flexibility***, which have been in turn divided into 34 sub-axes (see Fig. 1).

- The ***cycle*** concerns method recommendations regarding iteration granularity, lifecycle duration, increments, backlashes management, top-down or bottom-up approach.
- The ***collaboration*** categorizes the types of relationship that may exist in between the design/development team members and between them and the other stakeholders; it lists the characteristics of user-centered and usage-centered approaches.
- The ***artifacts*** classify the types of prototypes and documents to be produced (internal/ deliverable, executable vs. not executable, formalized or not).
- The method ***recommended use*** identifies the characteristics inherent to method context of such as project type(s), team sizes, required knowledge, requirements maturity, managed situations (UI plasticity, multi-core, cloud computing).
- The ***maturity*** measures evolutions of methods that result from their actual use.
- The ***flexibility*** measures the adaptability of the process model; for instance, it explicits possible choices or extensions.

4. Illustrations

We follow an illustrative approach, associating our proposals with examples coming from three methods: Rapid Application Development (RAD) [21], the Spiral Model [16] and XP [22].

In order to choose the methods that would illustrate Promote, the taxonomy was applied to several methods. Analyses of outcomes from these applications elicited the three above mentioned methods as being the most representative, the ones that make obvious the differences we wanted to enlighten. To show that Promotes is able to clearly surface differences, we have chosen two methods that are very close to each other (RAD and XP) and a method that is very different from these two ones: the Spiral Model. Others process models, not detailed in this paper, will be used to enlighten some axis. A presentation and a classification of each method mentioned here can be found on the Promote Web site.[2]

1. Rapid Application Development (RAD)

The RAD is a software development method created in 1991 [21]. This method includes a preparation phase of learning and personalization (customization of tools and stages) of the method. This preliminary phase is followed by an iterative development in four steps with cycles of 60 days optimally, but that can be increased up to 120 days. The four steps of the RAD are defined as follows.

- *Requirements*: users and design/development team members analyze needs, constraints and system requirements; this step ends as soon as users and design/development team members both agree on the main issues.
- *User design*: designers and users detail functions (inputs and outputs) and illustrate them using a model and a prototype.
- *Construction*: includes programming and application development involving users who can suggest improvements.
- *Cutover*: includes testing and deployment of the current version as well as user training and collection of user feedbacks.

[2] http://www.design-methods.net.

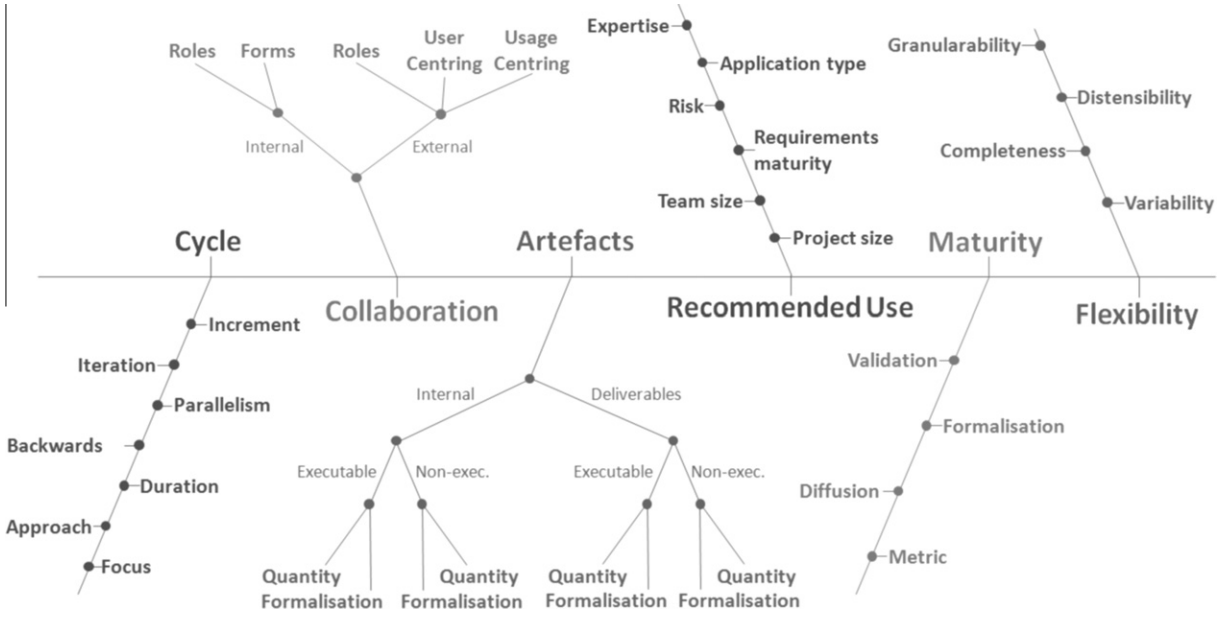

Fig. 1. Global overview on Promote axes and sub-axes.

2. Spiral Model

The Spiral Model [16] is a software development process model developed in 1986. This development model combines both elements of design and elements of prototyping-in-stage in order to take advantage of top-down and bottom-up approaches. The development model also combines features from the prototyping model and features from the waterfall model. The Spiral model cycle is divided into four large stages corresponding to:

- *determining objectives, constraints and alternatives;*
- *evaluating alternatives and identifying and resolving risks;*
- *developing and verifying;*
- *planning next phase.*

Each stage in the Spiral model includes sub-steps that are more or less detailed depending on the project state. For instance, the development stage involves *software requirements and requirements validation* during the first iteration and into *simulation, models, benchmarks, detailed design code, unit tests* and so on during the following iterations.

The Spiral model is intended for large, expensive and complicated projects.

3. Extreme Programming (XP)

The XP was elaborated in 1999 [22]. It is an Agile method and this belongs to a family of methods which, according to Anderson [23], rely on two core ideas: "*delivering value in the form of working software*" and "*responding to change*".

The Agile Manifesto [24] defines twelve fundamental principles such as "*customer satisfaction through early and continuous delivery of valuable software*" or "*welcome changing requirements, even late in development*". XP focuses on user needs and suggests a short iterative development cycle that can be adapted according to changes occurring in the customer needs. The XP is based on simplicity, communication, feedback, respect and courage. Its cycle relies on six main steps. First, *User Stories* and *Architectural Spike*, are produced. Then, the team is involved into *Release Planning*, develops functions in Iterations and produces a new version of the product. This version is submitted to *Acceptance Tests*. Finally, after being approved by the customer, the version is deployed as a *Small Release*.

These three methods will be used to enlighten all Promote axis and sub-axis, which are detailed in the following sections.

5. Axis #1: Cycle of the process model

The lifecycle of process models is certainly the most studied characteristic in literature. We have divided this axis into some very classical sub-axes like iteration, increment or duration, and some less usual ones, like parallelism or focus. Each of these sub-axes is detailed below.

5.1. Increment

Lots of methods claim to be incremental but this term is rarely defined by authors. Booch [5] says that an incremental process is a gradual refinement of strategy and decisions through an analysis/design/evolution cycle. RAD [21] suggests developing a set of functionalities during 3–6 months and recommends delivering the result at the end of each period.

Booch's vision of an increment appears to be very generic and can be applied to many kinds of projects: unlike the other examples, it does not imply that the code is part of an increment. Sommerville [11] or Jacobson et al. [25] define an increment as a delivered part of the system functionality. Cockburn's definition [26] does not mention any delivery, increments are "*pieces of the system [...] developed at different times or rates and integrated as they are completed*". These definitions contain some uncertainty, because they do not specify the size of the increment nor its scope. Especially they do not require that the increment is a consistent unit: it can be some random subset of incomplete functionalities. The definition of Benediktsson et al. [27] or Graham [28] is more precise about this point. According to them, an increment is "*a self-contained functional unit of software with all supporting material such as requirements and design documentation, user manuals and training*". This definition appearing as the most precise one, we retain it as our definition of an increment.

Benediktsson et al. make a clear difference between incremental development, where the system is completed increment by increment, and incremental delivery, where the system is delivered after the development of each increment.

When we studied methods, it appeared clearly that the number of deliveries can vary from one method to another but that incremental development process model do not specify any duration for the increments. This is why we suggest several grades for incremental delivery but only one for incremental development.

We suggest a scale with seven values:

- The process model suggests no increment.
- The process model suggests **incremental development**, there is only one final delivery.
- The process model suggests **a very small number of deliveries** (more than 12 months between two deliveries).
- The process model suggests **a small number of deliveries** (6–12 months between two deliveries).
- It defines a **medium number of deliveries** (3–6 months between two deliveries).
- It suggests a **large number of deliveries** (1–3 months between two deliveries).
- It suggests a **very large number of deliveries** (less than 1 month between two deliveries).

Fig. 2 shows how the three illustrative methods are classified: RAD [21] is considered as having a medium number of increments, due to its deliveries every 3–6 months. XP [22] considers that there should be many *short release cycles*, each of them including some *customer-requested features* and being created in 1–4 weeks. XP is graduated as having a very large number of increments. The Spiral model [16] suggests developing the system incrementally over four stages (planning, evaluation, risk analysis and engineering) but does not specify the duration of an increment. It only suggests one delivery at the end of the project. This corresponds to an incremental development.

5.2. Iteration

Iteration clearly appears to be an important aspect of methods and several process models claim to be iterative, like do the Rational Unified Process [29], Scrum [30], Feature Driven Development [31] and many others. But it is also quite hard to find a precise definition for this term. Graf [32] defines an iterative process as: "*a process for calculating a desired result by means of a repeated cycle of operations*". This definition could lead to a never ending cycle. This is why Graf specifies that "*an iterative process should be convergent, i.e., it should come closer to the desired result as the number of iterations increases*". For instance, the Spiral model [16] suggests a cycle based on a first stage including a risk analysis, a requirement plan, several prototypes and simulations, design, coding, tests, implementation and ending with an evaluation stage.

According to Graf's definition, we consider that an iterative process model defines repetitions of a set of design and development tasks and an evaluation phase that leads to converge to the expected application. Iteration is thus clearly independent from incrementation: it is possible to iterate so as to refine a system without incrementing it because there is not delivery.

A process model can be iterative at different levels. Some cycles may have very small iterations or more general iterations. Using names coming from the signal analysis [33], and more specifically from the recognition of repetitive sub-graphs in a graph, we suggest to name granularity the measure of iterations sizes in a process model and to evaluate this granularity with the following gradation:

- The process model does **not recommend explicitly** iterations.
- It suggests to **iterate without giving a precise definition** of the stages involved in it.
- It defines a **global** iteration on the whole process.
- It specifies a **regional** iteration on an important subset of the process.
- It defines a local **iteration** involving a few tasks.

A process model can suggest more than one form of iteration. This is why we added four complementary values:

- The process model specifies both **local and regional** iterations.
- It suggests **local and global** iterations.
- It defines **regional and global** iterations.
- It includes **local, regional and global** iterations.

The Spiral model [16] divides the whole cycle into large stages. This corresponds to a global iteration. Methodology charts of RAD [21] explain that the construction is divided into releases, made with iterative detailed design of the system, transactions implementation, unit test, usability test and integration. An iteration is also suggested within each of these stages. The test and integration stages can be considered as the validation phase. RAD releases suggest iterating over the complete development cycle and in every stage and is therefore classified as proposing global and regional iterations. XP [22] details that a release is built in 1–4 weeks and is divided into tasks achieved in 1–3 days. Validation is made by the customer, who is in charge of testing the release and giving a feedback. XP division of a 1-month cycle into small tasks corresponds to both regional and local iterations.

5.3. Cycle duration

In her general overview, Messager-Rota [34] explains that Agile methods are based on a cutting of the project in "*several stages*",

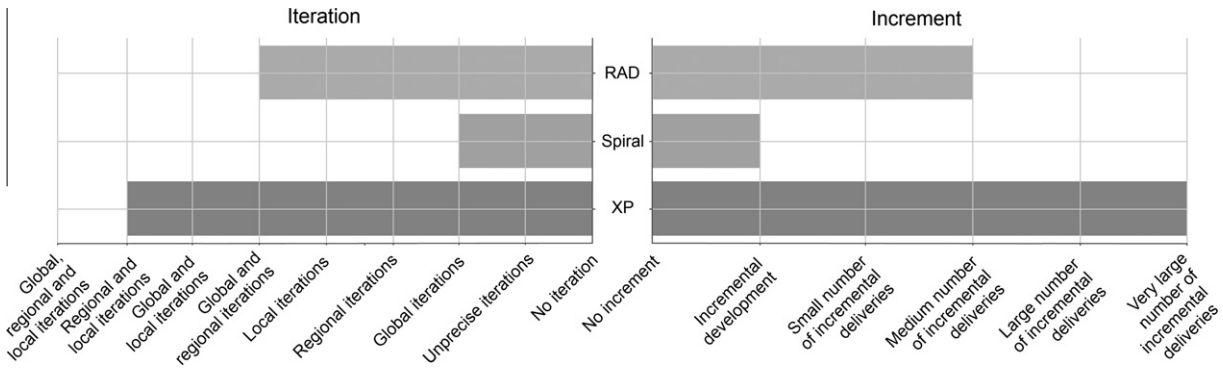

Fig. 2. Examples of increment and iteration classifications.

whose durations are "*a few weeks*". Booch [5] does not give any time limit to a cycle. XP [22] defines very clearly that a cycle should last 1–3 weeks. RAD [21] explicits that its ideal cycle lasts 60 days, but offers a variation called timebox development in which the duration can be between 90 and 120 days. These examples show that some methods recommend more or less precise durations for more or less precise stages of a cycle, when some others do not give any indication about these elements.

Methods can therefore be compared to each other on the cycle duration they recommend. When they suggest a limit, they often define minimum and maximum durations. We suggest classifying them according to the average recommended duration. For instance, Rapid Application Development suggests a 60-days lifecycle, with a variant named timebox development, the duration of which being between 90 and 120 days. We would synthesize that RAD has a 60 days cycle, and timebox development has a $(90 + 120)/2 = 105$ days cycle.

When a method does not suggest any duration for its cycles, we have chosen to associate no default value to it, because we consider that every value would be a bias in the comparison with other methods. We could imagine giving a value of 0, but this does not represent a possible duration for an activity. We also could imagine an infinite value, interpreting that the lack of time limit in the method enables activities to never end. But if the method lets it be possible, it does not recommend it to be so. Thus, associating the method with an infinite cycle would be an interpretation and not the transcription of the method recommendations. Therefore, we decide to notice the lack without interpreting it.

Considering all these remarks, the gradation of the cycle axis is:

- The process model suggests **no explicit duration**.
- It suggests a **very short cycle** (less than 1 month).
- It defines a **short cycle** (1–3 months).
- It is based on a **medium-long cycle** (3–6 months).
- It defines a **long cycle** (6 months to 1 year).
- It suggests a **very long cycle** (more than 1 year).

The 60 days of a basic cycle duration of RAD – without the timebox option – is considered as a short cycle, while the timebox alternative has a medium-long cycle. XP [22] recommends iterating every 1–3 weeks, a very short cycle. The Spiral model [16] expects the first cycle to last two or three months and the others up to 1 year. This leads this model to be graduated as having long cycles. These values are illustrated in Fig. 3.

5.4. Parallelism

This axis measures if a process model suggests realizing in the same time many stages of the design or the implementation. We are not interested here in the parallelism of small tasks, like the development of a few functions by developers at the same time, an organization that has been recommended very early: as Boehm recalls [16], the Waterfall model has been extended in the seventies to integrate parallel development. We are here interested into the divisions of important stages of the process and into their realization at the same moment. For instance, according to Roques and Vallée [35], the Two Track Unified Process (2TUP) recommends implementing a Y lifecycle, which separates and parallelizes the study of the requirements in one branch and the study of the technical aspects in the other one, both of them being merged into the lower branch. The Symphony approach [36] has the same structure. RAD [21], the Spiral model [16] and XP [22] do not suggest such a structure.

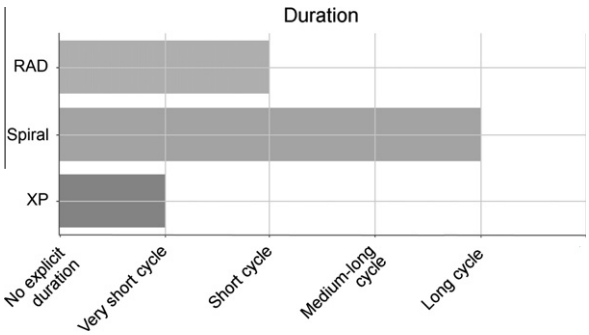

Fig. 3. Classification examples on cycle duration.

According to the parallelisms we have found in process models, we suggest considering them as

- **Not parallelized**.
- Including a **suggested parallelism**, without concrete definition of activities in each parallel branches.
- Defining a **parallelism with clearly identified activities** in each branch.

RAD, XP and the Spiral Model are classified as proposing no parallel stages.

5.5. Backwards

Humphrey and Kellner [37] notice that "frequently, during development, it is necessary to go back to some earlier entity, such as requirements, to recheck or clarify something, or to consider modifying the design as a result of something uncovered during development". Some authors consider that the procedure needed to achieve this go-back is often unclearly defined, constituting an important lack in a process model. For instance, Boehm [16] or Messager-Rota [34] point out that the Waterfall model requires that one stage is fully achieved before the next stage can start. They consider that this model leads therefore to a rigid process and is unable to manage changes in requirements or re-work of a previous stage. Going back remains possible in the actual process, but it contradicts the Waterfall process model and implies a rupture with the method.

XP [22] suggests that each release is evaluated by the customer which can decide to modify realizations of the previous releases. The customer can thus require a modification of what has just been done. The Spiral model [16] does not mention any such procedure: previous stages have to be validated before going forward and are not sensed to be questioned again. RAD [21] mentions validation procedures (like unit tests, usability tests and integration), but does not specify what has to be done when the stage is not validated.

We suggest a gradation with three values:

- The process model does **not define a backwards procedure**.
- It suggests **some vague backwards procedure**, without any concrete definition of validation points or of validation techniques.
- It suggests a **well defined backwards procedure**, with precise chronology, scope and mechanisms.

Customers' evaluation in XP [22] constitutes a continuous change management including backwarding. This process model is rated as having a well defined backwards procedure. The Spiral model and RAD are classified as not proposing go-back, as shown in Fig. 4.

5.6. Approach

According to Morley [38], most of information systems design methods suggest a cutting of the project into "*quite autonomic*"

subsets. This cutting relies on temporal and structural criteria, enabling to chain the subsets and guarantying the internal consistency of each subset. Dividing a problem into smaller ones corresponds to the second principle of the logical treatment of problems, as formulated by Descartes [39]: "*Divide each difficulty into as many parts as is feasible and necessary to resolve it*". Applying this, a problem is successively divided into less complex problems, most often decreasing the abstraction level. This approach is named Top-Down, or descending approach.

Nevertheless, Feynman [40] estimates that splitting of a problem implies early design choices and leads to a consistency lack between the different parts: "*When troubles are found* [...] *it is more expansive and difficult to discover the causes and make changes.* [...] *If an understanding or a fault is obtained, a simple fix* [...] *may be impossible to implement without a redesign of the entire engine*". So, the opposite procedure, named Bottom-Up or ascending approach, recommends to reuse well known components as much as possible, and to assembly or complete them. The approach here is to start from pre-existing components and not to identify required components at the end of the decomposition process.

According to Pizka's comparison [41] of the two approaches, bottom-up approach seems to avoid the problems identified by Feynman. Indeed, the final system relies on already tested components and this approach makes it possible to create quickly an embryonic system, which can be in turn tested and validated. But Pizka noticed that bottom-up approach can lead to increase the risk that the final system would not answer the needs, because requirements are taken into consideration lately in the process and represent some faraway goal. Some methods have tried to compose the two approaches. For instance, when describing the identification of data structures, Martin [21] lists top-down activities (e.g. identify objects, then relationship, then attributes) and bottom-up activities (capture existing view of data, capture inputs and outputs, decompose into items) merging into a set of common activities (synthesize both representations, check the model, etc.).

We suggest the following possible values for this axis:

- **Top-down**: the process model proceeds by successive decompositions, starting from the requirements up to the implementation, while each cutting can decrease the abstraction level.
- **Mostly top-down**: the process model suggests a top-down approach combined with a small number of bottom-up activities or stages when relevant
- **Mixed**: the process model composes both approaches.
- **Mostly bottom-up**: the process model suggests most of the times a bottom-up approach with some top-down activities when needed.
- **Bottom-up**: the process model suggests to aggregate concrete and existing solutions to try solving a complex problem, eventually while increasing the abstraction level.
- **Other approach:** corresponding to another proposition, an unclear approach or a lack of analyzable explanations.

Fig. 5 illustrates how the three example methods are classified. In the Spiral model [16], problems are successively divided into smaller ones and there is no mention about the reuse of pre-existing components. This corresponds to a full top-down approach. Beck [22] explicitly avoids top-down planning from XP but does not mention bottom-up practices. He writes "*I don't think I work either top-down or bottom-up. I start with an end-to-end slice through the system; then add functionality where it adds the most value, promotes morale, demonstrates trustworthiness, and enables early deployment*". XP claims to reject the two approaches, but does not clearly define its own approach. Therefore, it is impossible to say which approach it uses, and this process model is classified as having another approach. RAD [21] details that complex system are created by successfully filling in details and linking together blocks of low-level detail. It includes bottom-up activities in data analysis, and recommends a study for reusing code at the lowest level of the decomposition. This method suggests a mainly top-down approach, with a few bottom-up activities. It is classified as mostly top-down.

5.7. Focus

Boehm wrote [16] that "*the primary functions of a software process model are to determine the order of the stages involved in software development*". Ordering stages requires identifying them first. Therefore, the process models have to define the activities to be realized. The process model authors can identify these activities by analyzing various facets of the design and development process. Dowson [42] identified three of these facets: "*is the primary focus in characterizing the process on the activities involved, on the objects that result from those activities or on the decisions that give rise to process activities and objects?*". In other words, he noticed that the process model can describe directly the activities, or it can infer these activities either by analyzing the successive states of the resulting product either by studying the decisions made by the design team.

Rolland [43] completed Dowson's list with two other facets: according to her, authors can also focus on the context (the situations causing successive product transformations) and the goals (the choices of strategies in a set of possibilities leading to reach a goal).

According to these two studies, we named these facets the ***focus*** of the process model and we classified this focus into five categories: process models can be activity, product, decision, context or goal oriented. Literature suggests many other categories, like agent-oriented in Tropos method [44] or conversation-oriented

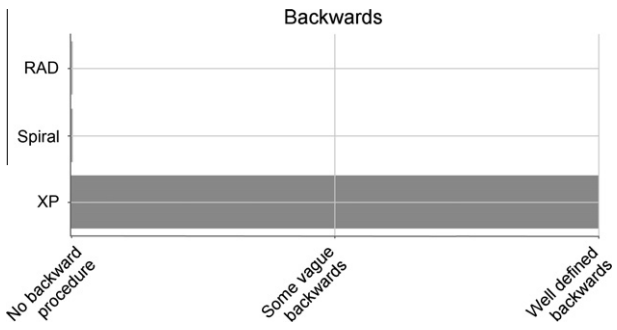

Fig. 4. Classification of the three illustrative process models on backwards.

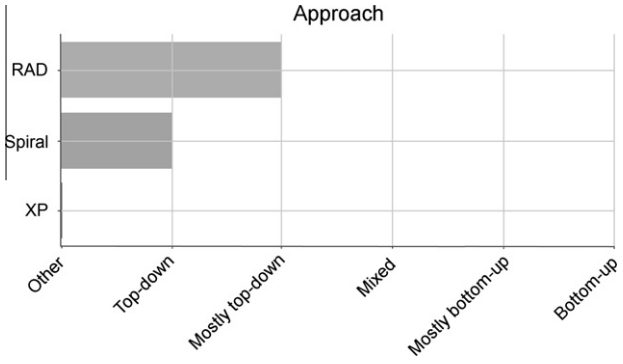

Fig. 5. Example of approach classifications.

[45]. But a deeper analysis of these possibilities shows that the focus of the process model is in fact one of our five categories. For example, conversation-oriented models aim to identify the intentions and commitment of the project stakeholders. Capturing intentions is the process of a goal-oriented model. On the other hand, Tropos cares about future possibilities of evolutions for the project and thus recommends to create a structure based on agents, and to embed this notion all over the design and development process. To achieve this, Tropos suggests a metamodel to represent the actors in order to identify the "*goals to be achieved, plans to be performed, and resources to be furnished*". This also classifies this model in goal-oriented models.

5.7.1. Activities oriented models

Rolland [43,46] refines and completes the three categories listed in Dowson's question. According to her, activity oriented models suggest a set of activities, decomposed when needed into simpler ones, and linearly ordered. For instance, such a process model could propose the following activities: study the users' needs, describe the use cases, identify the objects and so on.

Such a model would for example recommend that a designer makes a behavior analysis which includes three stages: use case analysis, design elements identification and design refinement. The Waterfall model [47], the Spiral model [16] and the Fountain model [48] are examples of activity oriented models.

5.7.2. Product oriented models

Product-oriented models do not focus on the activities of a process but on the result of these activities. They aim to model the evolution of the product and to link the state of the product to the activities that led to this state. Benabdellatif [49] explains that product oriented models "*represent the development process through the evolution of the product*", and Rolland represents them with a statechart diagram [43], focusing on the state of the product at the end of each task. An example could be the study of the successive states of a document: first, it could be "to be done", then "prepared", eventually "refined" and finally "validated". In this example, the process model would have to define the activities leading to each state.

Rolland considers that, beyond their different points of view and formalism, activity and product oriented models have the same power of expression, and thus, the same limits, especially the difficulty to write a realistic state-transition diagram describing all what can happen. However, Nurcan [45] and Börstler [50] point out some advantages in using product-oriented models: according to them, this kind of models can trace the transformations performed between two stages and their results.

5.7.3. Decision oriented models

In decision oriented models [42], activities appear to be the consequences of the decisions made to manage the project evolution. These models care about the context of the decision, i.e. the arguments and the alternatives. They try to identify and memorize the rationale of the decision process all along the project, and thus to integrate the semantics attached to the evolutionary aspects of the design [51]. For instance, a part of the process model could contain a stage where questions are asked ("should we model the business context?") from which two decisions can be made according to the situation ("Yes because the business process is large" and "No because the design team is experimented in this domain"), leading to further questions and decisions.

5.7.4. Context oriented models

Other models are focusing on the context during the development process. Rolland [52] defines a context as a situation causing successive product transformations under the influence of a decision, while, according to Pohl [53], context is the combination of a situation and a decision. For instance, a context could be "the entity Person has been identified and is not represented in the class diagram" and the corresponding decision could be "describe this entity". The next step could correspond to the couple ("The attribute Name has been identified", "Discuss if it has to be attached to the entity").

5.7.5. Goal oriented models

Goal oriented models focus on the choice of a strategy in a set of possibilities, in order to achieve an intention. Different possible strategies might refer to different approaches. Rolland [54] expresses that "*an intention captures in it the notion of a task that the application engineer intends to perform whereas the strategy is the manner in which the intention can be achieved*". In such models, the process is built dynamically. It is possible at every moment to add alternatives, choice criteria or intentions. For instance, various strategies could be implemented to achieve the goal "specify an entity", like "using specialization", "using generalization", "verify completeness", "using patterns" and so on.

Works like Brinkkemper et al. [55], Ralyté et al. [56] or Guzelian [57] suggest to combine methods fragments in order to implement the most adapted approach for every stage of the project. This kind of composition defines a new way to realize a concrete process, but does not change the categorization of each fragment. Each fragment may have its own point of view on the process and so the method itself may have several focuses. This makes that the five focus sub-axis are not totally independent. If most of the fragments are focusing on activities, only a few can focus on another element. This is also true for a classical method, which is usually oriented to only one focus, leading to the fact that the method will not focus on other elements.

5.7.6. Gradation

An ideal measure of the focuses of a method would then be the percent of the method corresponding to each focus. A classical method could then be for instance 100% oriented on activities or on decision. But the proportion corresponding to each focus is hard to evaluate in a composed method, for it is hard to specify what has to be measured so that the results are representative of the method. It is possible to measure each orientation with various dimensions, like the number of stages, the average time spent on them or the relative importance of the result and so on. Therefore, we suggest giving an equal rate to each focus of the method with the following degrees:

- The process model is **not focused** on [activities/strategies/product...]
- The process model is **very lightly focused** on [activities/strategies/product...]
- The process model is **lightly focused** on [activities/strategies/product...]
- The process model is **strongly focused** on [activities/strategies/product...]
- The process model is **very strongly focused** on [activities/strategies/product...]
- The process model is **fully focused** on [activities/strategies/product...]

This gradation let it possible that a process model is for instance strongly focused on activities and very lightly focused on decisions.

RAD [21], XP [22] and the Spiral model [16] are all fully activities oriented: all of them specify directly the activities to be performed. Humphrey [58] focuses on difficulties related to the increasing size and complexity of the product under construction. This process is rated as context-oriented. Potts [59] presents a

'*paradigm for constructing method-defining models*' in which he focuses on issues, arguments and positions leading to choose an action applied to an artifact. Therefore, this approach can be considered as focusing on decisions. Rolland [60] defines a requirement engineering model in which the context of each action is analyzed and recorded, including the description of the decisions, the situations and the arguments. This corresponds to a decision oriented model. The same author [61] presents a metamodel named the MAP which focuses on the goals of the designer when he has to decide what to do next and presenting to him all known alternatives. This metamodel is graduated as goal oriented.

6. Axis #2: Collaboration

Methods can define who is in charge of an activity or a set of activities and how people are working together. The first dimension corresponds to the role played by someone, defined as a set of related skills, competencies, and responsibilities of an individual or individuals. The second dimension is the form of the collaboration between one role and the others, i.e. the way by which two (or more) actors are interacting with each other. Each of these two dimensions can be subdivided: we have identified possible recommendations about internal and external collaboration. Internal collaboration refers to the collaboration between the members of designers and developers team – which is named a feature team by Eckstein [62]. External collaboration deals with the relationships between this feature team and the other stakeholders.

6.1. Internal collaboration

Internal collaboration describes the roles within the feature team and the form of collaboration between these roles. We are focusing on design and development process models and therefore, we are not interested here in the management aspects.

6.1.1. Roles

This axis looks at how – if ever – the method suggests a functional and/or hierarchical structure within the feature team. The various methods suggest many different roles. For instance, Rochfeld et al. [63] list a project leader, an archivist, a librarian, a tests leader and a system analyst; Ecksteing [62] mentions a coach, an architect, a database administrator, a designer, a technical writer, a domain expert, an infrastructure specialist, an integration expert and an UI designer; Williams and Ferguson [64] identify a tracker, a project manager, an iteration manager and a quality assurance specialist. Rapid Application Development [21] suggests seven roles "from the Information System Community" (project manager, RAD workshop leader, scribe, data modeling expert, etc.). XP [22] defines five roles (programmers, tester, tracker, coach, big boss) and the Spiral model [16] does not define any role.

Phalp and Shepperd [65] notice that a small number of roles implies that each role contains unrelated activities, which reduces the cohesiveness of each role. However, we could not find a study expliciting the efficient minimum number of roles. Conversely, Harrison [66] analyses that a high number of roles may be a handicap. Indeed, the communication between roles, required to ensure the consistency of the project, becomes more and more complex when the number of role increases. Several roles have to be informed and the risk grows up that critical information does not reach the right person at the right time. This can be measured by the communication saturation ratio that Harrison defines as the ratio of actual collaborations to the possible collaborations. When the number of roles doubles, the number of possible collaborations paths increases quadratically but the number of actual collaborations grows only linearly and the saturation ratio decreases. This is shown in Fig. 6, which presents a scatter plot of communication saturation as a function of roles. The author considers that a saturation ratio of 60% (16 roles) is very good and that a ratio of 30% (25 roles) is low. Considering these recommendations, Cain et al. [67] recommend that organizations should simplify their structure by reducing the number of roles, expecting communication to become easier and less expensive, leading to higher productivity.

We suggest the following gradation for the internal roles:

- The process model does **not define roles**.
- It suggests **roles but does not define them explicitly**.
- It defines **more than 30 roles**.
- It defines **21 up to 29 roles**.
- It defines **17 up to 20 roles**.
- It defines **16 roles or less**.

This gradation is ordered on an increasing communication saturation ration. The limits between two grades come from Harrison's indications: less than 16 roles leads to a good saturation ratio, a number 21 roles corresponds to the average number for all organizations and 30 roles appears to be a upper limit in Fig. 6. RAD and XP define less than 16 roles, and the Spiral model defines no role.

6.1.2. Form of internal collaboration

Amblard [68], a researcher from the sociological field, remarks that "exchange, coordination and cooperation between agents require conventions between involved people, i.e. a system in which each one waits some behaviors from the others". We measure here at which point a process model suggests such a system within the feature team and what kind(s) of collaboration it suggests.

Process models show various forms of expected collaboration between team members. For instance, the Spiral model [16] does not define roles and therefore does not define the collaboration form between roles. RAD [21] suggests a coordinating model which is focused on the identification of the number of parallel persons or teams and the verification of their non-redundancy without presenting in detail how people work together. Anyway, an analysis of this model shows that strong communication between people is not expected: the system relies on the use of tools. In XP [22], developers work in pairs and there are daily stand up meetings involving the whole team.

These different forms of collaboration have been studied and modeled. Some authors study the collaboration forms in specific

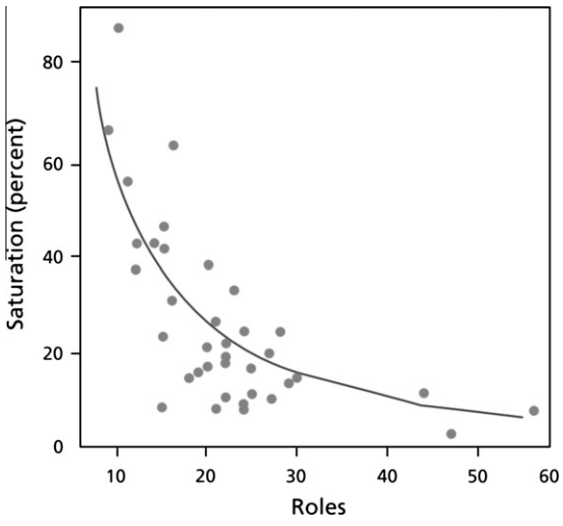

Fig. 6. Communication saturation as a function of roles [62].

domains, like Little [69] in educational field. She identifies four forms of collaboration (scanning and storytelling, help and assistance, sharing and joint work). Scanning and storytelling do not appear to be possibly recommended by a design/development method: it treats more of daily communication than of work organization. The three other forms can be boiled down to Amblard's categories. Other studies are more generic, like Handy [70], where a chapter defines roles and interactions between people, but the definition of roles is very generic, coming from daily life, and focuses on problems in interactions and their solutions.

Amblard's gradation appears to be generic enough for categorizing various methods and specific enough for identifying the different types of collaboration. The gradation that we suggest is directly inspired from Amblard's three types:

- **Undefined**: the process model does not explicit how actors are supposed to collaborate.
- **Exchange**: (or moderate collaboration): the actors inform each other about the progression of their individual tasks.
- **Coordination**: (or average collaboration): the actors interact while achieving tasks.
- **Cooperation**: (or strong collaboration): several actors realize together one task.

The Spiral model is classified as having an undefined collaboration form. RAD is evaluated as expecting collaboration at exchange level, because it reduces collaboration to the use of communication tools. XP, with its pairs of developers and its daily meetings, is rated as proposing a cooperation form of collaboration. Fig. 7 illustrates these classifications.

6.2. External collaboration

Methods can also define how the feature team interacts with external stakeholders. Similar to internal collaboration, this can be divided into roles and forms of collaboration. The stakeholders mentioned in process models are not numerous: we have found specific indications about the interactions with the customer but only very general considerations about other stakeholders like the company to which the feature team belongs. This is why we focus here on the collaboration with the customer. Two main models theorize this collaboration: User Centered Design, where the user is involved into the team and Usage Centered Design when he is considered as an external stakeholder.

6.2.1. Roles

Symmetrically with the roles evaluated in internal collaboration, we want here to measure how the process models define customers' roles.

The various process models explicit many roles. For instance, Eckstein lists [62] the Product Owner and the On-site-customer, Cerón et al. [71] mention the Executive Sponsor, the Visionary, the Facilitator and the Ambassador User. RAD [21] defines eight roles "*from the User Community*" (Executive Owner, User Coordinator, User Design Team, User Review Board, etc.). Booch's method [5] does not explicitly define any external role, even if some of them are mentioned (third-party vendor, customer, final user, etc.).

Our study of methods and process models shows that the number of external roles is by far lower than the number of internal roles. The average proportion that we computed from the analyzed models is one external role for six internal roles. Thus, we used this ratio to define the external roles gradation:

- The process model **does not define roles**.
- It **suggests roles but does not define them explicitly**.
- It defines **five roles or more**.
- It defines **four roles**.
- It defines **three roles**.
- It defines **one or two roles**.

The eight roles defined by RAD graduated this process model as proposing more than five roles. XP [22] suggests two external roles: customer and consultant. The Spiral model [16] does not mention any role. These classifications are shown in Fig. 8.

6.2.2. User centering

Several authors and many methods recommend nowadays a "user-centered" approach. According to Gulliksen [72], this approach was originally defined by Norman [73] with these words: "*user-centered design (UCD) emphasizes that the purpose of the system is to serve the user, not to use a specific technology, not to be an elegant piece of programming. The needs of the users should dominate the design of the interface, and the needs of the interface should dominate the design of the rest of the system*".

Other definitions exist. ISO 13407 [74] says that "human-centered design is an approach to interactive system development that focuses specifically on making systems usable", and Karat [75] specifies that "UCD is an iterative process whose goal is the development of usable systems, achieved through involvement of potential users of a system in system design".

These definitions do not indicate concretely what has to be done in UCD nor how to identify if a process model is user-centered, and even less how much user-centered it is. Many papers, like ISO 13407 [74], Gould et al. [76], Greenbaum and Kyng [77], Nielsen [78] or Maguire [79], suggest principles on which a user-centered process, or more specifically user-centered activities, can be based. As shown in Table 1, Gulliksen [72] establishes a set of key principles that lead to the participation of all (internal and external) stakeholders in the design and the realization of

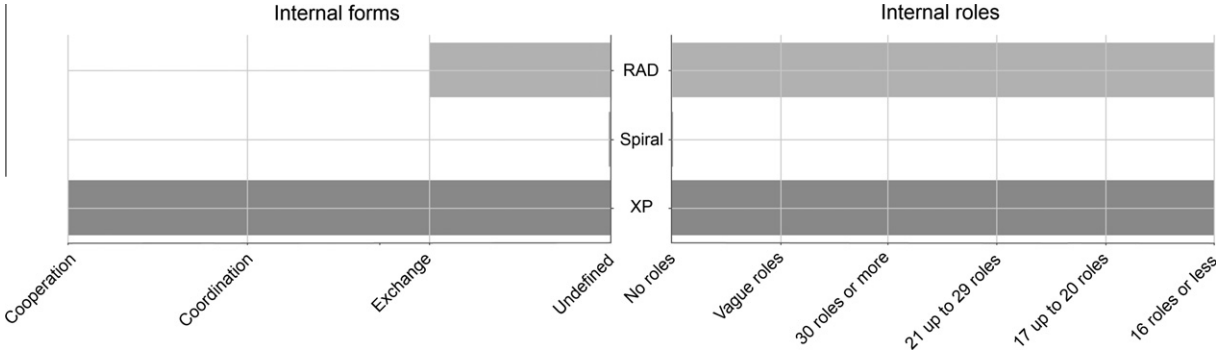

Fig. 7. Example of classification on internal roles and internal form of collaboration.

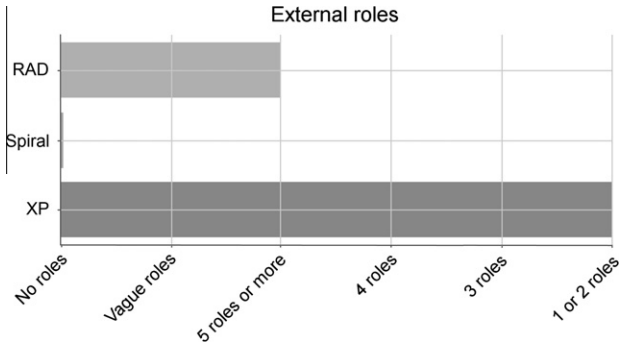

Fig. 8. Example of classification on external roles.

the system. These principles can be suggested by process models and will therefore be part of this axis gradation.

But principles are quite abstract and designers may need to know which activities have to be achieved concretely. Mao et al. [80] list the most common practices, defined by industrial UCD experts. This list indicates the importance that these experts assign to each activity and their frequency of use, i.e. the number of experts who have mentioned the activity. For instance, the activity named "*task analysis*" has an importance of 2.61 and a frequency of 34, and "*focus group*" is scaled respectively at 2.79 and 16. We have completed the activities list to suggest a definition of each activity, this can be seen on the promote website (see chapter 12 p. 37). Task analysis is defined as "*the study of what a user is required to do in terms of actions and/or cognitive processes to achieve a task*" [81] and focus groups as "*interviews in which the participants are encouraged to explore and share in depth to share their thoughts, feelings, attitudes and ideas on a certain subject*" [82].

Gulliksen's principles and Mao's lists of activities constitute the base of this axis gradation. We first consider the principles suggested by the process model and complete this information by comparing the set of activities defined in the process model to the set of activities listed by Mao. This comparison relies on the relative importance of the two sets of activities and not on the relative quantity of activities. To make this comparison possible and simple, we synthesize Mao's importance and frequency of use in a number that we call the popularity and which is calculated as the product of the two values. The importance of a set of activities is defined as the sum of the popularities of its activities.

It seems not possible to reduce our evaluation to Mao's list, because other practices have been mentioned or defined by other authors: for instance, the questionnaires [83], the "logging actual use" [78] (a detailed record of the use of the system during the test stage) and the Stakeholders analysis [84] – which consists in identifying, for each group of users or stakeholders, the main roles, responsibilities and task goals in relation to the system. When a process model suggests activities that are not in Mao's list, we consider that these activities are not very popular as they are not in the top ten activities mentioned by Mao. As there is no other survey giving a value of the popularity for these activities, we associate to them an arbitrary value equal to the lowest known popularity – which means 14, associated to "user requirement analysis" that has an importance of 2 and a frequency of 7.

Using the results of these calculations, a process model will be said

- **Not user-centered** when it suggests neither user-centered principles nor activities.
- **Lightly user-centered** when it suggests principles without defining any concrete activity.
- **User-centered** if its global popularity is lower than 50% of Mao's activities total popularity.
- **Strongly user-centered** if its global popularity exceeds 51% of Mao's activities total popularity.

Analyzing RAD [21], this gradation gives a rate of "strongly user-centered". Indeed, RAD suggests a Usability Laboratory focusing on users' behavior when facing the system and measuring reactions in context; it suggests several roles involving users; the construction is iterative and incremental, lying on prototypes; the design is a specific stage; there is a usability champion and usability evaluation; the process is customizable and includes Formal Heuristic Evaluation, User interviews and participatory design. Therefore, RAD global UCD importance is evaluated as 56% of the possible maximum. XP [22] suggests a daily 15-min meeting with the customer; it focuses on users who define the stories to be carried out. According to XP principles, the customer is always here, but XP mentions just a few user-centered principles and activities. Its calculated UCD importance is 24%, and this process model is classified lightly user-centered. There are no User-Centered principles or activities mentioned in the Spiral model [16] which is therefore rated as not user-centered. Fig. 9 illustrates these results.

6.2.3. Usage centering

Usage-centered design is defined by Constantine et al. [85] as "a systematic, model-driven approach to visual and interaction design for user interfaces in software and Web-based applications" where "the center of attention is not users per se but usage, that is, the tasks intended by users and how these are accomplished".

Table 1
Gulliksen's User-Centred Key Principles.

Principle	Description
User focus	The users' goals, tasks and needs should early guide the development
Active user involvement	Representative users should actively participate, early and continuously throughout the entire development process and throughout the system lifecycle
Evolutionary systems development	The system development should be both iterative and incremental
Simple design representations	The design must be represented in such ways that it can be easily understood by users and all other stakeholders
Prototyping	Early and continuously, prototypes should be used to visualize and evaluate ideas and design solutions in cooperation with the end users
Evaluate use in context	Baselined usability goals and design criteria should control the development
Explicit and conscious design activities	The development process should contain dedicated design activities
A professional attitude	The development process should be performed by effective multidisciplinary teams
Usability champion	Usability experts should be involved early and continuously throughout the development lifecycle
Holistic design	All aspects that influence the future use situation should be developed in parallel
Processes customization	The UCSD process must be specified, adapted and/or implemented locally in each organization
A user-centred attitude should always be established	All people involved in the project must be aware of and committed to the importance of usability and user involvement

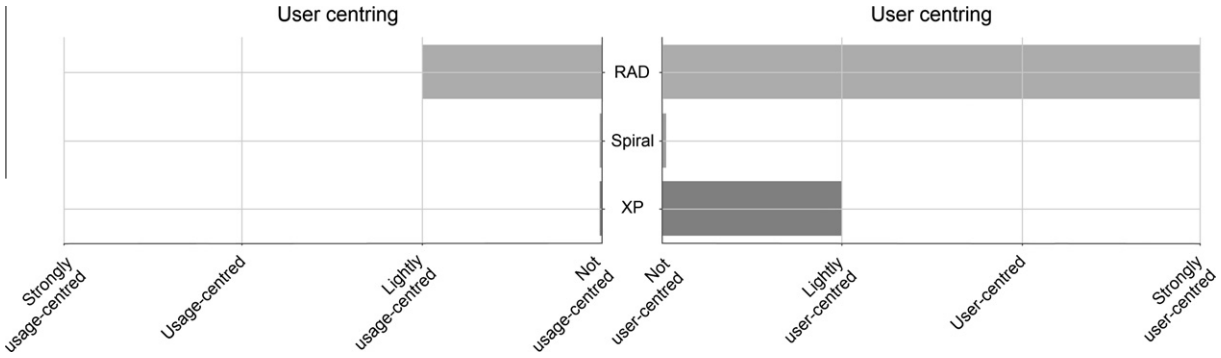

Fig. 9. Example of process models classification on user and usage-centring.

Constantine defines the main characteristics of a usage-centered approach: focus on usage, driven by models, selective user involvement in exploratory modeling, model validation, structured usability, inspections, models of user's relationships with system, abstract design models, design through modeling, systematic, fully specified process, derivation through engineering.

Ferreira et al. [86] study the main models of usage-centered design. According to them, the three core models are: the role model, that describes the relationships between users and the system, the task model, representing the structure of tasks that users will accomplish, and the content model, that abstracts the tools and materials to be supplied by the user interface. The use of these models constitutes an indication to evaluate if a process model is usage-centered.

This axis is suggested with the following possible values:

- **Not usage-centered** when the process model doesn't define any usage-centered principle or activity.
- **Lightly usage-centered** when it suggests some principles, without defining any concrete activity or usage-centered model.
- **Usage-centered** if it suggests a few usage-centered activities or models.
- **Strongly usage-centered** when it defines a large number of usage-centered activities or models.

Even if User-Centered Design and Usage-Centered Design suggest very different approaches and activities, Lingaard et al. [87] consider that "*usage-centered design can, in principle, integrate user-centered design and software engineering and support both sets of activities throughout the development process*". It is therefore possible to evaluate how much a process model is compliant with each of the both approaches. This explains why we have two axes with two gradations and not only one graduated from "pure User-Centered Design" up to "pure Usage-Centered Design".

In the Spiral model [16], the diagram shows activities about "simulations, models and benchmarks", but this is not detailed enough to define which models have to be created. XP [22] does not mention any use of models or diagrams in the process. It recommends a large users' involvement, which is in opposition to usage-centering. The Spiral model and XP are evaluated as not usage-centered. RAD [21] recommends also a large users' involvement and defines several types of models like entity-relations diagram, process decomposition diagram, data flow diagram and action diagram. However, none of these models match the set identified by Ferreira et al. The action diagram, which can be seen as a mix of the task and content models, is closer to an algorithm than to a model. Moreover, RAD is not driven by models. However, considering the importance of models in RAD activities, this process model is classified as lightly usage-centered. Fig. 9 illustrates these results.

7. Axis #3: Artifacts

The main goal of a design method is to provide help and guidance during the process of creating software. In this sense, every process leads to at least one deliverable: the system itself. But many methods recommend a lot of others productions, like requirements documentation, models, user manual and so on.

We have grouped together all these products under the name of artifacts. Some of them might be reserved for an internal use in the design/realization team, while others are dedicated to be finally given to the customer. We are not interested into the contractual terms of the project management: we consider here all what the process model recommends to provide to the customer for validation, whether or not under contract.

We name internal artifacts the artifacts that are not made to be given to the customer and deliverable artifacts those who are given to him. We separate internal and deliverable artifacts in two categories: executable or non-executable, and for each of these categories, when it makes sense, we study the quantity of artifacts and their formalization.

7.1. Quantities

The quantity of artifacts measures the amount of artifacts that the process model recommends to make. Following the categories we defined above, there are four sub-axes about quantity:

- Quantity of internal non-executable artifacts.
- Quantity of internal executable artifacts.
- Quantity of deliverable non-executable artifacts.
- Quantity of deliverable executable artifacts.

According to Sommerville [9], these quantities may be huge: "*If this were all to be printed, the documentation would probably fill several filing cabinets for moderately large systems*". But in fact, methods offer various recommendations about the quantity of artifacts to be produced. For instance, XP [22] and RAD [21] recommend many executable releases named prototypes, each of them being delivered at the end of an iteration. XP does not recommend non-executable artifacts, because it advocates avoiding all the artifacts that it says to be unuseful, i.e. all the artifacts that are not necessary for the delivery. RAD suggests a large number of artifacts in initialization phase – thirty kinds of them, like system functions, objectives, decomposition diagram, process flow diagram, but suggests only a small number of them during the iterations (some validation reviews). The Spiral model [16] defines several non-executable artifacts that are delivered during the iterations: risk analysis, simulations results, models, requirements, design, tests,

etc. It recommends delivering three kinds of prototypes at the end of cycles, but does not define internal artifacts.

These examples open a new question: does the quantity measure the number of artifacts types, or the number of artifacts themselves? If 5 artifacts are produced iteratively, the total amount of artifacts will not be 6 but (5*n*+ the final system), where n is the number of iterations. Therefore we need to measure the number of artifacts made before and after the iterative cycles, and within the iterations. Of course, we do not count individual parts of a document: a specification file divided into several documents, e.g. one per use case, will be counted as one document.

We do not think that we need a very precise scale for these axes. This is why we suggest evaluating both quantity of artifacts outside iterations and quantity of artifacts inside iterations as small, medium or big. According to the process models we have analyzed, a small number is lower than 5, medium number is lower than 20 and a large number is greater than 20. These values have been obtained by separating nine process models into three consistent subsets (Scrum [30], RAD [21], XP [22], V-Model [88], Waterfall [47], Merise [63], Spiral model [16], Rational Unified Process (RUP) [89], Feature Driven Development (FDD) [31]).

Two gradations are defined for measuring the quantities of these four kinds of artifacts: the first one measures the quantity of artifacts produced before or after the iterative stages of the process model, the second one focuses on the artifacts created during these iterative stages. The two of them are structured like:

- The process model does **not define artifact**.
- It suggests a **small number of artifacts**.
- It defines a **medium number of artifacts**.
- It recommends a **large number of artifacts**.

For instance, we focus on the numbers of deliverable non-executable artifacts during iterations. According to its recommendation, XP is rated as defining no such artifact. The Spiral model defines a large number of non-executable deliverable artifacts during iterations (like requirement plan, risk analysis, models and development plan) and is therefore classified as promoting a large number of non-executable artifacts during iterations. RAD suggests a large number of non-executable deliverable artifacts in initialization phase (thirty kinds, like objectives, decomposition diagram, process flow diagram, etc.), but a small number during iterations (some validation reviews). It is classified as proposing a small number of non-executable artifacts during iterations. These gradations are illustrated in Fig. 10.

7.2. Formalization

The formalization of artifacts evaluates if the process model suggests writing them in natural language, or if it suggests one or more languages to express them. For instance, a user scenario might be written in natural language, while a class diagram can be expressed using Unified Modeling Language (UML). A process model may recommend a natural language for some artifacts and a formal language for some others. The question here is to evaluate what part of the artifacts has to be formalized. Formalization is not always absolute: Chkolvski [90] explains that full formalization of all knowledge may be cost-uneffective and not practical and therefore suggests a semi-formal representation, combining machine-processable structures with free text statements. Executable artifacts are by nature formalized: they are written in a coding language and must respect the syntax of this language. Therefore it does not seem interesting to evaluate the formalization of the code itself. Anyway, the comments in programs can be free, fully written in natural language, semi-formalized or formalized. This is what we want to evaluate in the formalization of executable artifacts.

Process models suggest therefore different degrees of formalization according to the different needs. For instance, XP [22] defines a single rule for the formalization: programmers have to write all code in accordance with rules emphasizing communication through the code. The coding standards have to be defined locally, according to the rule "Once and Only Once", meaning that redundant code has to be avoided. The Spiral model [16] and RAD [21] say that non-executable deliverables are based on diagrams and free text, but do not give indications for other artifacts.

The gradations for axes about formalization of artifacts are the following:

- The process model gives **no indication** about the formalization of artifacts.
- It suggests **informal artifacts only**.
- It suggests **informal and semi-formal artifacts**.
- It suggests **semi-formal artifacts only**.
- It suggests **informal and formal artifacts**.
- It suggests **formalized artifacts only**.

Fig. 10 shows some results for our three examples: XP is classified as proposing informal non-executable deliverables, while RAD and the Spiral model are rated as proposing a mix of semi-formalized (the models and diagrams) and unformalized (the documents in natural language) non-executable deliverable artifacts.

8. Axis #4: Recommended use

We study here the recommendations that the process model gives about its own application. For instance, we question either it defines some criteria that would impact its elicitation – does it match the needs? – or some advice about the way it has to be applied.

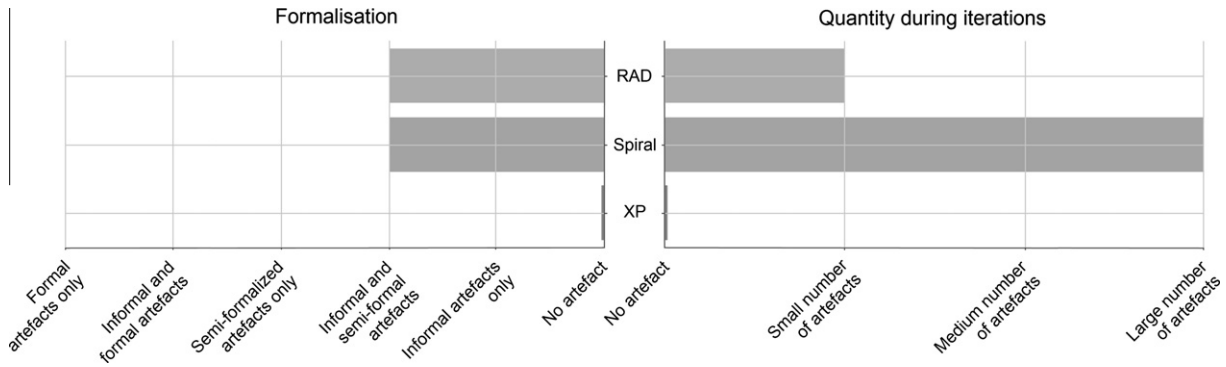

Fig. 10. Example of formalization and quantity of non-executable deliverable artefacts.

8.1. Project size

Pérez et al. [18] estimate that managing a small project with a process model that recommends a large number of produced documents can lead to an incongruent process. Therefore the process model should either suggest different quantities of documents according to the project size or explicitly declare the project size it is made for. The central question of these two options is then the project size. As will be detailed below, this question can be divided into two dimensions: what has to be measured to evaluate the size of the project and what is the limit of each size grade.

The first dimension is what has to be measured to evaluate this size. Several aspects of the project can be measured. For instance, when Charrette [91] talks about large-scale projects, he mentions sometimes the budget and sometimes the number of lines of codes. Kim [92] refers only to the number of lines of code. Boehm [93] measures the number of persons implied in the realization team and Pérez et al. [18] look at the budget allocated to the project. Method123 Project Management Methodology [94] evaluates the project size according to several dimensions: the financial resources, the number of team members, the number and size of develirable artifacts, the timeframes and the complexity of the project. These examples show that there is no consensus about what to measure.

The second dimension concerns the limits between two sizes and how to graduate the size of a project. One again, authors give various – and often vague – indications. For instance, XP [22] expresses that it is made for small to medium-sized projects but does not define clearly what this means. The Spiral model [16] claims to be well suited for "large, expensive and complicated projects" but it does not give a clear definition of what is a large and complicated project. Method123 [94] does not give any information on how evaluating the various dimensions it defines nor on how consolidating them. As a result, it appears that all these resulting sizes are totally arbitrary, depending on the feelings and beliefs of the evaluator.

To answer these two questions, Boehm [95] has introduced the Cocomo (COnstructive COst MOdel) model. Cocomo relies on the evaluation of "function points", which are units measuring delivered functionality and depending on the number of user inputs and outputs, user inquiries, files, and external interfaces. Each of these elements is categorized as simple, average or complex, and a formula gives the number of function points. The formula itself depends on other parameters, like the need of reliable backup and recovery or the criticality of performances. The number of function points is then converted into a number of source lines of code (SLOC). This SLOC number is used to evaluate the project size. The project size evaluated by Cocomo is the measure of the effort to be made to produce the final system. Dimensions like available human or financial resources do not impact the project size in itself.

Cocomo appears to be an interesting tool to evaluate the size of a project. Measured dimensions are well defined and the way to consolidate them into a single number is clearly specified. However, there is no proof that these definitions do actually give a "good" measure and Rollo [96] contests the reliability of the conversion of function points into SLOC number, estimating that two similar projects can finally have different estimated sizes.

Another problem is that there is no systematic matching between the size evaluated by Cocomo and the size expected by a process model: would XP medium projects be evaluated as medium by Cocomo? Extrapolating this remark, we cannot use a standard measuring method to identify the size of projects managed by process models. Therefore, we consider that the semantics and the concrete gradations associated to the sizes mentioned in process models have to be defined by the process models themselves: the considered sizes of projects are those defined by method authors.

The gradation is based on the casual small, medium and large grades and reflects the value defined in each process model:

- The process model gives **no information about the project size**.
- It recommends itself for **small projects, without giving an evaluation procedure** to evaluate if the actual project matches this category.
- It suggests to handle **medium-sized projects, without any evaluation procedure**.
- It claims to manage **large projects, without any evaluation procedure**.
- It claims to handle **various sizes of projects, without any evaluation procedure**.
- It suggests to handle **small projects and gives an evaluation procedure** to check the match with the actual project.
- It recommends itself for **medium-sized project, and gives an evaluation procedure**.
- It suggests to handle **large projects, and gives an evaluation procedure**.
- It claims to handle **various sizes of projects and gives a procedure** to determine which options are adapted to the actual project.

For instance, RAD [21] does no mention a clear expected size of the project. It requires "scalable projects with sufficient and skilled human resources". RAD is graduated as giving no specific information about the expected project size. XP [22] is classified as expecting medium-sized project without giving an evaluation procedure. The Spiral model is graduated as managing large projects without any evaluation procedure. Fig. 11 illustrates these classifications.

8.2. Team size

The team size, i.e. the number of people involved in the project, is not directly connected to the number of roles (internal or external) that we presented in Sections 6.1.1 and 6.2.1. Indeed, several persons may have the same role and a single person can fulfil many roles. A process model can define independently the number of roles and the expected size.

The team size is also lightly connected to the project size, because the required number of people depends on the project duration. Anyway, whatever the project size is, it appears that the team size impacts the implemented process: Laanti [97] notices that synchronizing activities and decisions for large teams differs from what has to be done for small teams. He illustrates this on Agile methods: these process models, originally designed for small teams, cannot easily handle the coordination of large teams. This suggests that either a process model is made for one team size either it has to suggest adaptations corresponding to the different sizes it handles. This implies that a process model should explicitly indicate the size(s) of the team(s) to which it suggests allocating activities and the adaptations if a team is smaller or bigger.

This is not often made clear in process models. For instance, the Spiral model [16] gives no indication about the recommended team size. RAD [21] avoids large teams, and counsels to create as many teams as needed, limiting each of them to three or four people. It defines clear coordination procedures. XP [22] recommends teams with five to twenty people, including four to ten programmers. It advises starting a new team with four to six programmers.

The process models we studied categorize the sizes as small, medium and large. A few authors give a concrete definition of the number of people corresponding to each of these grades. A small team is quantified as 1–5 persons by Martin [21], and less

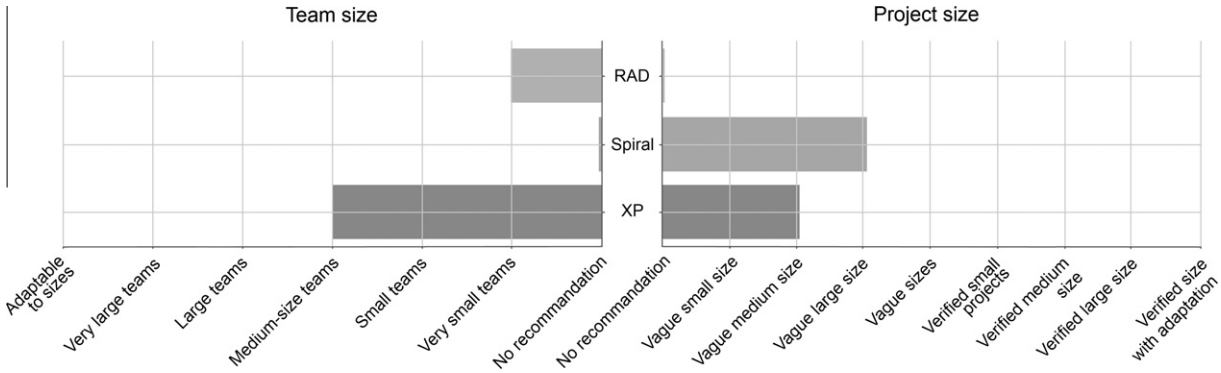

Fig. 11. Example of team and project sizes recommended by process models.

than ten people in several Agile methods as mentioned by Rising and Janoff [98]. Spencer [99], from Microsoft, says that more than 50 persons make a large team and Cusumano and Selby [100] mention teams of more than 100 persons. These indications give us a gradation for this axis:

- **No recommendation**: the process model does not mention a recommended team size.
- It recommends **very small teams** (1–5 persons).
- It recommends **small teams** (6–10 persons).
- It recommends **medium-size teams** (11–49 persons).
- It recommends **large teams** (50–99 persons).
- It recommends **very large teams** (more than 100 persons).
- **Adaptable size**: the process model recommends one team size and defines procedures to adapt to other teams sizes.

Other grades could have been suggested, like a customizable size, when the process model does not define a favorite team size and gives indications to adapt the project to its effective team size. But these cases did not appear in the process models we studied and we choose to not indicate them, in order to simplify the use of this gradation.

Fig. 11 shows the classification of the three examples: the Spiral model [16], which gives no indication, corresponds to the "no recommendation" grade. RAD is recommending very small teams including three persons. The 20 people teams of XP match our medium-size teams.

8.3. Risks

According to Alexander and Davis [17], Boehm suggested the Spiral model [16] so as to focus on the risk as the selection criterion for adapting the process model. Indeed, Boehm identifies several kinds of risks in software development (personnel shortfalls, unrealistic schedules and budget, shortfalls in externally performed tasks and so on). He suggests series of milestones, checkpoints and metrics to identify, evaluate and manage these risks and he gives some key points about the corresponding adaptations of the process model like cross training of personnel or use of prototypes.

All process models do not define a detailed risk analysis. For instance, RAD [21] details the risks linked to the use of the prototypes it recommends but does not specify which risks it suggests to handle. Beck [22] lists a set of risks that XP helps addressing like schedule slips, project cancelation, defect rate. There is no procedure for evaluating risks. The process model is sensed to avoid them by construction. For instance, schedule slips are addressed by the short release cycles, where the most valuable functionalities are implemented. Some process models claim to be able to handle highly risked projects. For instance, Yourdon [101] suggests a development method for "mission impossible projects", which he delimitates as projects where the allowed schedule, staff or budget are reduced to at least half of the need or where the functionalities are twice what could be expected. But the chapter about risk management is reduced to six pages where the author gives general considerations about what should be done, like anticipate, priories, evaluate or follow-up. This appears to be quite flimsy regarding the kind of projects this method manages.

As mentioned by Boehm or Yourdon, there are several kinds of risks. Westfall [102] classifies them into six categories. **Technical risks** include problems with languages, project size, project functionality, platforms, methods, standards, or processes. **Management risks** include lack of planning, lack of management experience and training, communications problems, organizational issues, lack of authority, and control problems. **Financial risks** include cash flow, capital and budgetary issues, and return on investment constraints. **Contractual and legal risks** include requirements changes, market-driven schedules, health and safety issues, government regulation, and product warranty issues. **Personnel risks** include staffing lags, experience and training problems, ethical and moral issues, staff conflicts, and productivity issues. **Other resource risks** include unavailability or late delivery of equipment and supplies, inadequate tools, inadequate facilities, distributed locations, unavailability of computer resources, and slow response times.

Several risks mentioned before do not concern directly the design or the development of a system. For instance, health issues do not appear at first as having an impact on the design and development cycle. But the process model can give indications on how to manage their possible consequences. For instance, the process model can specify, as XP [22] does, that developers have to work in pairs, reducing the impact of a health problem. This is the scope of this axis that we formulate as: what indications, what management procedures are given by the process about the risks that can occur during the project lifecycle?

The corresponding gradation is:

- The process model does **not mention any form of risk management**.
- The process model defines some **procedures for risk evaluation**.
- The process model mentions **some principles for risk management**.
- The process model defines **procedures for evaluating and managing risks**.

As shown in Fig. 12, the Spiral model [16] specifies many activities for monitoring and managing risks. RAD [21] gives no

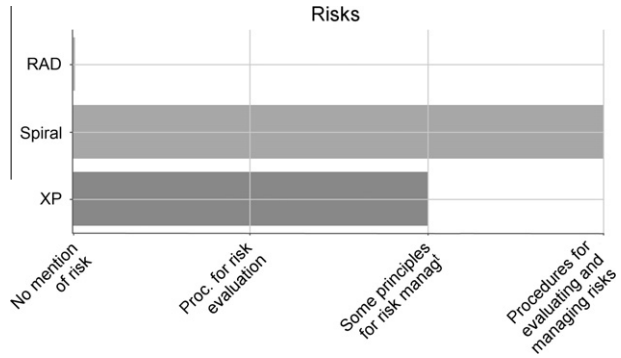

Fig. 12. Example of risk levels managed by process models.

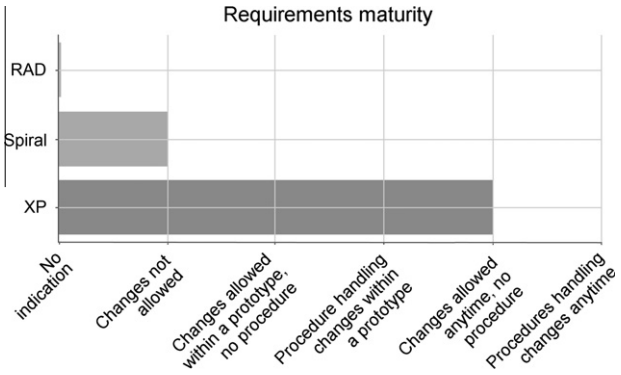

Fig. 13. Example of requirement maturity management in process models.

indication about risk evaluation or management. XP [22] claims to address some risks in its organization, but there is no specific procedure for monitoring them nor for evaluating risks that are not taken into consideration in the model structure. This process model is graduated as proposing some principles for risk management.

8.4. Requirements maturity

The stability of the requirements impacts directly the development process: when a late change occurs, the design and development activities have to be adapted, reworked or restructured. To ensure this adaptation, the process model has to be organized to be able to manage the changes.

For instance, business changes are one of the risks that XP [22] claims to help handling but this process model gives only a vague procedure to adapt to it: the customer is "*welcome to substitute new functionality for functionality not yet completed*" at the end of each iteration and the new goals are taken into account in the planning. The Spiral model [16] does not suggest a procedure to manage requirements changes but explicits that there should be a "*substantial participation of the user-project personnel*" particularly in early phases of the project so as to avoid this risk. RAD [21] does not mention any expectation about requirements maturity, nor about requirements changes.

According to these various situations, we suggest the following gradation for this axis:

- The process model suggests to study the requirements but gives **no indication** about their maturity.
- The process model **does not allow requirement changes**.
- The process model **allows requirements changes during the development of a prototype**, but does not define the procedure to achieve this.
- The process model **defines a procedure handling requirements changes during the development of a prototype**.
- The process model **allows requirements changes at any time**, but does not define the procedure to manage this.
- The process model **defines a procedure handling requirements changes at any moment**.

Requirements changes are welcome in XP [22] within a release, but there is no clear procedure to handle this. For instance, the impact of these changes is not defined as having to be studied. This is why XP is rated as allowing requirements changes at any time, without defining the procedure that manages these changes. The Spiral model [16] expects that requirements are studied before the development phase and are then stable. This model is rated as not allowing changes. RAD [21] does not mention what to do when requirements change. This model is rated as giving no indication, as illustrated in Fig. 13.

8.5. Applications types

This axis is focused on the kinds of applications that the process model claims to be able to handle, and how it suggests specifically handling each of them. For instance, it measures how a process model manages applications with or without plastic user interface (UI) [103], i.e. interfaces adapting themselves to the user characteristics, the environment of use and the platform on which they are used. Does the process model suggest specific activities dedicated to create such UIs and other activities for classical UIs? The same question can be asked for the kinds of persistency, cloud-computing, data mining, multi-core processes and so on.

Only a few process models give these indications, the application kind it manages remains implicit, most of the time corresponding to a "classical" information system. But when a designer has a specific need, it is hard to know which method can help designing and developing it. This is the case of XP [22] and the Spiral Model [16], which do not mention the kind of application they are dedicated to. Martin [21] specifies that RAD applies to the building of "*Information System applications that are essential for every large enterprise, including some hundreds of thousands of lines of code, developed with small teams in 6 months or so*". He makes precise that RAD does not apply to building a chess-playing program, complex operating systems, software tools like Lotus' 1–2–3 or the software of the space shuttle.

The gradation of this axis measures how well the process model defines the kinds of applications it should be applied to. For this information be fully useful, the applications kinds will also have to be recorded and available, which will be possible in the comments associated to the rate (see chapter 12 p. 37 about the classification website).

- The process model does **not mention the kinds of applications** it is made for.
- The process model **mentions vague kinds of applications**, with an informal textual description.
- It **defines the concrete kinds of applications** it can help designing and developing, with criteria to evaluate if the needs are matching its abilities.
- It **defines the concrete kinds of applications** it can help designing and developing, and helps choosing an adapted variant.

Due to the indications of Martin, RAD [21] is graduated as giving a vague mention of application kind. As XP [22] and the Spiral

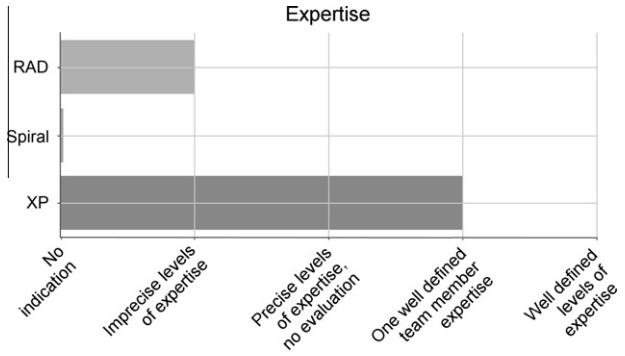

Fig. 14. Example of team expertise required by process models.

model [16] do not mention their expected types of applications, they are evaluated as giving no indication.

8.6. Team expertise

Some methods express the need of experimented developers while some others say that it is not required that all team members are experts as long as there is a "guru" or some experts in the team. For instance, RAD [21] suggests to choose a Workshop Leader, chosen for his experience and having several characteristics, like being impartial, unbiased, neutral, good negotiator, diplomatic, good at conducting meetings and so on. The end users involved in the definition of the system should be intelligent, creative, good communicators, able to understand information-systems techniques, highly knowledgeable about their own business areas. The Information Systems professionals should be familiar with chosen tools and techniques. In XP [22], there can be a Coach, guiding and mentoring the team, and his presence is recommended for teams who are adopting this method. He has to be "a good communicator, not easily panicked, technically skilled (although this is not an absolute requirement), and confident", and his role is to guide programmers with individual technical skills, taking responsibilities or achieving difficult technical tasks, watch on long-term refactoring goals, be a contact for upper-level managers and promote the communication between the team members. It recommends developers to discover technologies by themselves and requires the help of a consultant if deep knowledge is needed.

According to all these situations, we suggest to evaluate the process model with the following gradations:

- The process model does not **mention the required level of team expertise**.
- The process model **suggests imprecise levels of expertise**.
- The process model **suggests precise levels of expertise** without detailing how to know who can be considered or not as experimented.
- It **defines the expected level of expertise for one team member**, for instance a "guru", with concrete criteria to evaluate his competencies, but does not define what is waited from the other members of the team.
- It **defines the expected levels of expertise** with concrete criteria to evaluate the knowledge of (mostly) all team members.

Fig. 14 shows the classification of RAD [21], XP [22] and the Spiral model [16]. RAD defines several characteristics for the people involved in the project, but remains incomplete and imprecise: these abilities are not clearly defined and there is for instance no information about how evaluating the creativity or the intelligence of users, and no mention about what being familiar with a tool means. This process model is classified as giving imprecise levels of expertise. XP definition of expertise is vague and incomplete. It suggests that any expertise level is acceptable for developers, as they will have to learn what they need. The role of the Coach is well defined, even if his competencies remain imprecise. This matches our "one well defined 'guru'" grade. The Spiral model does not mention any expected experience. This process model is graduated at the lowest level.

9. Axis #5: Maturity

The maturity of the process model is awaited to reflect its ability to guide efficiently the creation of a system. It is casual for engineers to consider that a new technology – for instance a new language or a new framework – suffers from its lack of maturity and needs to be validated and improved. This is the question that this axis asks about process models.

9.1. Validation of the process model

In Physics, a theory needs to be validated before being adopted. For instance, the Standard Model is the name given to the current theory of fundamental particles and their interactions, and is considered as valid because its predictions have matched experimental results.

The question here is: has the process model been validated? A process model can be considered as validated when it has proved matching its intended requirements in terms of the process employed and the results obtained and being useful in the sense that it addresses the right problem and provides accurate information about the construction of the system.

We do not intend to develop a standard validation procedure to which process models would be compared. The goal is to measure if and how the process model has been validated. Anyway, if seems interesting to present an example of validation method, suggested by Moody ([104,105]). Moody evaluates design methods through the measurement of several characteristics like the Syntactic Quality (the relationship between the model and the language used to write it), the Semantic Quality (the relationship between the model and the modeled domain, linked to the validity and the completeness of the model), the Actual Efficacy (representing whether a method improves the performance of a task) or the Adoption In Practice (measuring whether the method is used in practice). According to the author, the Actual Efficacy and the Adoption in Practice appear to be easier to measure than the other characteristics. Adoption in Practice will be focused in the following chapter that deals with the process model diffusion. Actual Efficacy measures the benefits of using a method and the limits of the method. For instance, the Waterfall model [47] has been mentioned for its rigidity due to the fact that each stage depends on the complete achievement of the previous one. Thereby, the definition of the complete set of requirements is needed first, leading to early decisions about the system and to hardness of backwarding. The actual efficacy of this method is now regarded as limited [106].

Graduating the evaluation of the process model validation is complex, because it means that the validation procedure itself has to be evaluated. The purpose of this taxonomy is to enlighten the characteristics of process models and not to suggest a complex system to evaluate validation procedures. The need is here to measure whether the process model claims to have been evaluated. Therefore, the gradation of this axis is:

- The process model indicates **no validation** results.
- The process model indicates **vague validation** results.
- The process model details the **validation procedure and its results**.

In the preface, Martin says that RAD [21] has been demonstrated to be superior to traditional development in many projects but gives no further information about the validation procedure. XP [22] and the Spiral model [16] do not mention any information about this subject. The three of them indicate no concrete validation results. Other works have been made for evaluating methods. For instance, Gong and Kieras [107] evaluated the GOMS model methodology, concluding that there is a clear benefit when implementing this process model and that "it can be applied, with a reasonable amount of effort, identify significant usability bottlenecks, provide guidance for design solutions, and produce useful quantitative predictions for design alternatives". Another example is the Symphony method. After it has been extended for User Interfaces concerns [108], some aspects of this extension (e.g. the coupling between the resulting components) have been evaluated using metrics [109] leading to the conclusion that, when the process model is actually implemented, the resulting business and interactional components are reusable and modular. This method would therefore be classified as including a validation procedure with a special note detailing that only some aspects have been evaluated.

9.2. Diffusion

This axis evaluates if the method is actually used. It is different from other axis because all the others evaluate what the process model is while this one evaluates how the process model is perceived and adopted. Moody et al. [105] estimate that there is a little agreement on what makes a "good" model and that we do not have the elements to evaluate directly the quality of the process models. This is why Moody suggests [104] the measure of the 'adoption in practice', that evaluates whether or not the method is used in practice. The same concept, named User Acceptance, is presented by Davis in [110]. These approaches rely on the hypothesis that a satisfied user will encourage other people to use a satisfying method, leading to a larger and larger diffusion and that an unsatisfied designer will no longer implement the method, leading to a reducing diffusion. Therefore, a method with a high presence on the Internet or in the professional newspapers, with dynamic web sites and lots of suggested trainings has chances to be considered as a "winning method" by designers.

We suggest two metrics to measure the diffusion of the method: the number of books or research articles published about the method and the number of available Internet pages. Ideally, the number of sold tools sustaining the method and the number of trainings suggested at present for each method should also be taken into consideration, but these two values are Extremely hard to measure: on the Internet, the archived pages about a training would be found as well as the current training propositions, and there would be no guaranty that the resulting number is representative.

The two metrics have been calculated using common search engines, especially Google and Yahoo, which have been ranked as the two biggest in the United States of America by the comScore classification in May 2011 [111]. Both sets of metric results have been distributed in four sets, according to their rank given by the quartile formula. Thereby, every process model is associated to two values: a first rank on its papers and books number, and a second on the number of web pages. The consolidated rank is the average of these two values. The full results are available on the web site dedicated to Promote (see chapter 10.3 page 32), including the number of researches, the number of books and the number of web pages. For instance, Scrum is ranked 4 (greatest diffusion), due to 37,800 articles, 308,000 books and 34,150,000 web pages. Unified Modeling Methodology is ranked 1 (the lowest diffusion) with 36 articles, 13 books and 4659 pages. The results show that methods have a very different diffusion: some have a few dozen articles and books, and some have millions of web pages and publications. This study enlightens also the fact that certain methods have a good diffusion in academic community but are not in fact used in industry: for example, the Spiral model [16] has a great number of articles (12,800, fourth quartile for articles only), but a relatively small number of Internet pages (484,000, second quartile).

We have restrained the research to the two most representative engines: Google and Yahoo.

We suggest basing the gradation on the rank computed from the quartiles:

- The process model is **very poorly diffused** (rank $\leqslant 0.50$).
- The process model is **poorly diffused** ($0.50 < \text{rank} \leqslant 1.50$).
- The process model has a **medium diffusion** ($1.50 < \text{rank} \leqslant 2.50$).
- The process model is **well diffused** ($2.50 < \text{rank} \leqslant 3.50$).
- The process model is **very well diffused** (rank $\geqslant 3.50$).

We measured the diffusion of XP [22], RAD [21] and the Spiral model [16] in June 2011. 10,400 papers, 15,900 books and two millions web pages talk about RAD. The calculated rank is 3.33, graduating RAD as well diffused. XP is mentioned in 18,300 papers, 19,400 books and 4.7 million pages, giving a rank of 4. XP is therefore considered as very well diffused. The Spiral model is also well known, as it has the exact same rank than RAD, but for different reasons: if it is more often mentioned in papers and books (respectively 12,800 and 17,200, a fourth quartile for each of them), it is much less present on the Internet, where 484,000 pages are mentioning it. The high number of publications and the relatively little number of web pages mentioning the Spiral model make it possible to think that this model is quite present in academic papers, but has a relatively little presence in industrial usages, while the opposite situation applies to RAD, which is less present in publications but much more mentioned on the Internet. Fig. 15 illustrates these classifications.

9.3. Formalization of the process model

This axis questions the formalization of the process model. The question of formalization has already been discussed in the chapter about artifacts (p. 20). The discussion about process models would recall the same arguments. This is why we will just discuss its application to process models.

Several process models have been written in natural language. For instance, the practioner's guide to the RUP written by Kroll and Kruchten [89], or the Fountain model described by Henderson-Sellers [48] are written in free text. Some methods include formal or semi-formal parts, e.g. algorithms or sequence diagrams to determine the process. For instance, Martin [21] suggests 23

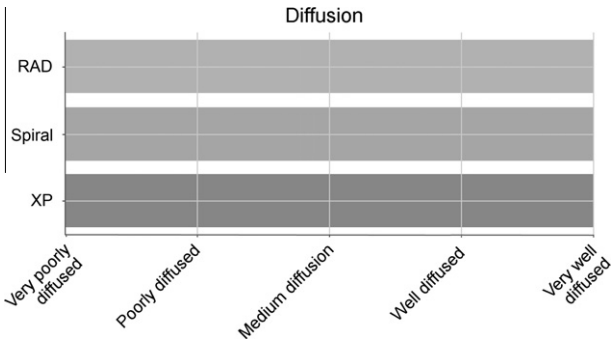

Fig. 15. Example of diffusion of process models.

algorithms – named methodology charts – which link together the activities to be performed, indicating the optional tasks and the cycles underlying the iterations. This is a fine guide for the team leaders, designers and developers. Many stages of the process model are thereby defined: requirements planning, procedure to create a process model, identification of reusable modules, data modeling, etc.

According to these examples, the gradation we suggest is:

- The process model is **described in natural language**.
- The process **model is mostly described in natural language**, a few parts are semi-formal.
- The process model contains **large semi-formal parts**.
- The process model contains **some formal parts**.
- The process model contains **large formal parts**.

XP [22] and the Spiral model [16] are written in natural language. RAD [21] is mostly described in natural language but has many methodological algorithms. This process model is graduated as having some formal parts.

9.4. Metrics measuring how well the model has been applied

The usefulness of a process model depends largely on its ability to predict what will be done in the effective design process. And reciprocally the validity of the actual design process depends on its accurate implementation of the guidance given by the process model. Therefore, it might be interesting for process models to suggest some way to validate the good application of their recommendations. This notion is very different from the process model validation presented in Section 9.1: the validation of the process model estimates how well it can lead to achieve a project and the metric evaluates how well the model is applied in a specific project.

Several approaches can be found in literature to implement such a validation. For instance, Cook and Wolf [112] present a formal verification and validation technique for identifying and measuring the discrepancies between process models and actual executions. They suggest to measure the "string distance" between the process and its model, i.e. the number of modifications needed to transform one string (the process representation) into another (the model representation).

It is also possible, for example, to explicit a set of code metrics and the awaited ranges for each of them. The Symphony method claims to separate interactional and business concerns; thus interactional and business objects in the application should be decoupled. The measure of the coupling between them is a good indicator about this recommendation [109].

The gradation associated to this axis is:

- The process model suggests **no validation** of its own implementation.
- It suggests an **informal validation**.
- It suggests a **formal validation** procedure (model application metrics, checkpoints, etc.).

XP [22] defines a set of metrics measuring various aspects of the project: for instance, the Team Velocity, which is the ratio of estimated time tasks to actual time spent on them, or the "Big Visible Chart" that enlightens a specific point at a time of the project, e.g. the number of realized tests compared to expected number. It recommends following three or four measures at a time. But, as noticed by O'Reilly [113], these metrics are dedicated to tracking the schedule and not how XP is implemented. RAD [21] contains large parts about metrics, which measure the expected effort and the subdivision of the project in parallel developments. But this is not a measure of the process model application. The Spiral model [16] does not suggest any form of validation. These three models are classified as proposing no validation.

10. Axis #6: Flexibility

Agerfalk [114] said, about distributed process development methods, that "*flexibility and the ability to adapt to different circumstances are necessary for successful software development*". This axis evaluates how the process models offer this flexibility, i.e. the possibilities of adaptation to the project context and needs.

According to Harmsen et al. [115], flexibility can be measured on a single axis. Fig. 16 shows the diagram of increasing flexibility, from the lowest level, where methods are rigid, through methods proposing choice selection up to the highest flexibility with modular construction of methods. This is a method engineer point of view.

But we are here focusing on designers' point of view. A few surveys have studied their needs and expectations, like Garzotto and Perrone [1], Fitzgerald [116,3], Barry and Lang [2]. We have analyzed and classified the designers' expectations and needs mentioned in these surveys. This gave us seven categories: adaptation, simplicity, efficiency, tooling, realism, validity and completeness. Adaptation includes dimensions like possibility of customizing the process, need of different design languages and guidelines for addressing the different design tasks and experts, adaptation to the project situation, type and client. Realism includes expectations like focusing on the product under construction and not on the approach, managing real projects rather than an "orderly rational process as most methodologies do" or adaptation to the project size, meaning that methods are perceived as requiring too much documentation effort for small projects and become cumbersome and time consumers. When abstracting all these data, it appears that each of these needs can be reduced to two dimensions: variants and extensions. As an example of variant, the documentation effort should be great for a large project but little for a small one. The customization of the process can be seen as an extension (or a reduction) of the process model: some stages can be added or avoided when needed. For instance some designers would like to add an unexpected benefit and costs analysis, some others estimate that modeling is not always necessary.

These two dimensions, variants and extensions, are included in Harmsen's diagram. Rigid methods offer neither variants nor extension possibilities. Selection of paths in methods offers variants but no extension possibilities. Construction of modular methods offers both variants and extension possibilities, with the condition that a method fragment corresponding to the need can be found.

10.1. Variability

In 1989, Potts and Colin [59] noticed that design methods were "*never adopted exactly as its author conceived it*": lots of projects and

Fig. 16. Flexibility levels in process models according to Harmsen et al.

human factors caused the method to be modified. According to them, existing design methods and models – like the Waterfall model [47], the Spiral model [16] or the V-model [117] – could be considered as fixed procedures, this fixity making it hard to adapt to local conditions. Defining variability as the capacity of a design method to offer different possibilities to realize an activity, we would say that the methods mentioned by Potts and Colin were not variable: they did not suggest the variants that would have enabled the designers to choose the most adapted path in the method.

Some methods offer some forms of variability. For instance, RAD [21] suggests explicitly to customize the method by selecting an appropriate toolset, by choosing the timebox option, by combining some phases from the requirements planning and the user design, or by using or not a coordinating model for parallel development, etc. All these possibilities are presented in methodological charts with well defined criteria and procedures.

Offering variants has been studied by other approaches, for instance while focusing on decisions or goals. Rolland [60] suggested an approach for requirements engineering based on a 4-uple <situation, decision, argument, action> and presented a few years later the MAP [61], a labeled directed graph representing tasks and strategies alternatives. Potts and Colin [59] suggested a "*paradigm for constructing method-defining models*". This paradigm is generic and has to be instantiated for a specific design method. The basic concept is to manage design state information, in order to refine and revise what has to be done next and to choose the following design steps. The authors mention that one limit of this model is that autonomy and responsibility of designers are not represented and that, in large projects, it would be necessary to manage parallel works and their coordination. We also guess that implementing this model to a particular design method requires knowing very well the generic model, the handled method and its steps. This induces a high threshold of use. The same remark applies to goal oriented methods: they require identifying all the intentions, which appears hard to achieve. Overmore, according to Rolland and Salinesi [118], "*experience shows that it is difficult for domain experts to deal with the fuzzy concept of a goal*".

Another approach is to compose methods fragments, as studied by several works like [119–121]. The basic concept is that, as different methods suggest different ways to achieve a task, it is possible to choose the most adapted fragments of each method and to combine them so as to recreate a specific method. Harmsen [6] said that these so named Situational Methods are thereby providers of choices because they allow choosing between "*method fragments having the same purpose*". But in fact, situational methods are not design methods: they are meta-methods leading to the construction of a method. The variants they offer are not selected by designers but by method engineers, who are sensed to take into consideration designers' needs and requirements. Design and development choices are made at a high level of abstraction in an early stage of the project.

This quick overview shows that embedding variants in methods is an actual need that has been worked a lot, but also that it is not so obvious to suggest a full process model with variants in stages. Some methods offer choices in some parts, like the requirement analysis in the MAP, and a few methods suggest choices in several stages. Therefore, this axis is graduated with a five degrees scale:

- The process model suggests **no variant**.
- It suggests **some informal variants** in some parts of the process.
- It suggests **some well defined variants** in some parts of the process with tools and guidance for each option.
- It suggest **informal variants** in most parts of the process.
- It suggests **well defined variants** in most parts of the process, with tools and guidance for each option.

RAD [21] variants are numerous and formalized into methodological charts, but lots of stages contain none of them. This process model is rated as offering some well defined choices. XP [22] and the Spiral model [16] do not suggest any variant. Fig. 17 illustrates these values.

10.2. Distensibility

The goal here is to evaluate if the process model suggests explicit ways to extend (or reduce) the activities it includes basically. For instance, we saw in user centering axis that some process models suggest activities involving the final user. If ever a designer wants to implement such activities in a method that does not suggest them, does the process model explain how to extend itself with such activities?

We name distensibility the ability of a design method to be extended or reduced. This word is used by analogy with the medical meaning that describes for instance this capability of the heart.

Most of the process models do not suggest such possibilities, as the Spiral model [16] and RAD [21], which do not mention any extension capacities. XP [22] suggests adding stages or tools if needed (e.g. define needed metrics), but does not define where it is possible nor how this can be achieved.

However, many design methods have been extended. For instance, Kolodner et al. [122] suggest the Case-Based Reasoning Process and make it explicit that it is "*an extension of what Schank (1982) refers to as intentional reminding, and is a component of Wilensky's (1983) Projector*". Laanti [97] presents a work which suggests a procedure for scaling up the Scrum method to make it able to manage several teams working on the same project. This is an extension of the Scrum method. But as we said, the question here is to evaluate how the process model suggests ways to extend itself if needed. Even if these authors found a way to extend existing process models, these models did not anticipate these extensions. For instance, Scrum has been refined over time (Schwaber and Sutherland have written a few versions of the Scrum Guide) but suggests no way to extend its procedures or activities.

As it has been said in the previous chapter, situational methods are built by composition of methodological fragments. Building a situational method is therefore a continuous extension process, adding new fragments to those which have already been chosen. Building a situational method is thereby a way to offer distensibility.

We suggest the following gradation for the process model distensibility:

- The process model suggests **no extension/reduction procedure**.
- It suggests **informal extension/reduction possibilities** without explicit procedures to integrate new features.
- It suggests **well defined extension/reduction procedures** with explicit manners, but without well defined limits and validation checkpoints.
- It defines **extension/reduction procedures** with explicit manners, limits and validation checkpoints.

As shown in Fig. 17, XP [22] propositions make this process model being classified as offering some informal extension possibilities. The Spiral model [16] and RAD [21] are classified as offering no possibility of extension.

10.3. Completeness

According to Barry and Lang [2] or Fitzgerald [3], some designers estimate that design methods can be cumbersome. Many

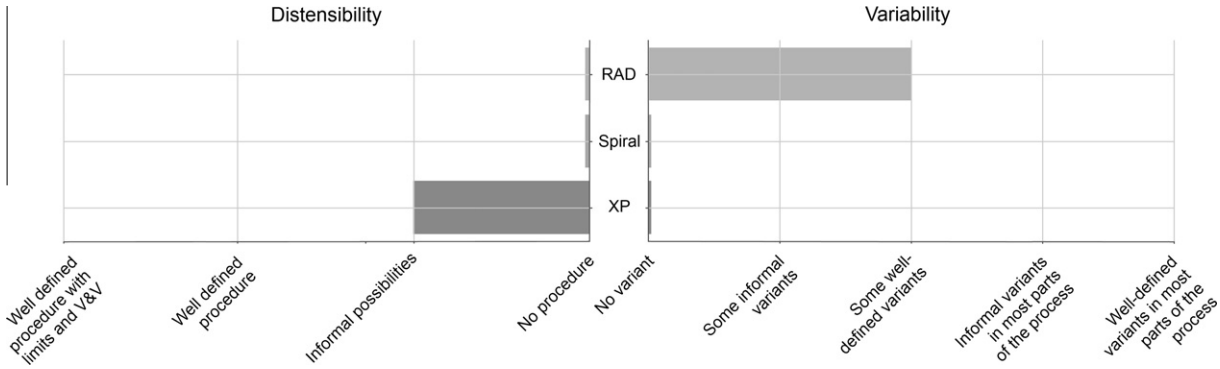

Fig. 17. Example of variability and distensibility of process models.

examples can be found in the literature. For instance, some designers think that writing design documentation is expensive and unadapted to small projects [1]; Paulk [123] notices that risk analysis promoted by the Spiral Model [16] is not always worth the effort.

This axis measures how design methods handle the fact that some of the stages they recommend can sometimes be useless: do they suggest some path to avoid them?

We define the completeness of a process model as the amount of stages that it requires to necessarily achieve. This has to be distinguished from variability, which is the capacity of achieving a task using different ways.

For instance, RAD [21] defines several required stages, like Data Modeling or Cutover. It also defines some optional stages, like Prototyping or Usability Testing. Prototyping is mentioned as being "*of limited use in some logic-intensive systems although, in these cases, partial prototypes can check the human interaction*". RAD also gives a list of situations in which prototypes are particularly valuable. According to this, RAD recommends developing prototypes in some situations and, even if not explicitly mentioned, to avoid this stage in the other cases.

However, completeness is not only the non-cancelation of full stages. As noticed by Gamma et al. [124], Design Patterns enable designers to reuse tested and proven solutions to commonly occurring problems within given contexts. But a Design Pattern is not a finished design, it has to be adapted to the local needs. In this sense, when a process model suggests using Design Patterns, it helps saving designers' time and effort. This corresponds to a non-complete work.

It is also possible, as it can be seen nowadays in some development frameworks, to suggest full default design – and software – components. For instance, Django [125] suggests a default user model and a corresponding built-in user and authentication system. Designers can then use directly this model instead of designing their own solution. If ever they need to adapt this model, Django suggests procedures to achieve this. This kind of solution – even if suggested by a development framework and not a process model – gives rise to a lower required completeness: designers only have to identify their own need for a user model and take directly the Django default user model off the shelf.

This axis evaluates the process model flexibility. By construction, flexibility is higher when full completeness is not required. We wish the gradation to be representative of this key point. Two solutions can be studied: ordering the gradation on the **decreasing** completeness or naming this sub-axis 'incompleteness'. We rejected the second solution, because it induces an ambiguity: the process model could be understood as incomplete, which would be in turn perceived as a weakness. According to these considerations, we suggest the following gradation for completeness:

- The process model gives **no indication** about its own completeness.
- The process model has to be **fully completed**, i.e. almost all stages have to be completed.
- It can be **partially uncompleted**, i.e. it specifies some avoidable stages and/or several some Design Patterns and/or several default design solutions.
- It can be **highly uncompleted**, i.e. it specifies several avoidable stages and/or several Design Patterns and/or several default design solutions.

As mentioned before, some stages are optional in RAD [21]. However, even if RAD gives guidelines to produce reusable design and code, it does not include patterns or built-in solutions. RAD can therefore be partially uncompleted. Beck [126] recommends the use of patterns in XP [22], but there is no concrete Design Pattern or built-in design solution defined in this process model. All activities are presented as necessary. XP is therefore rated as requiring to be fully completed. The Spiral Model [16] suggests a series of activities that are all required, and mentions no form of effort savings. It is also rated as having to be fully completed.

10.4. Granularability

Several criticisms are made by designers on the detail level of process models. For instance, according to Barry and Lang [2], 41% of the designers involved in their survey estimate that method "*has a high level of details*". On the hand, Garzotto [1] shows that designers reproach "*the lack of documentation that is targeted to practitioner and that offers real examples and complete real case studies*". As a conclusion of these two – apparently – opposed remarks, we can say that designers expect various granularities in process models: sometimes they need very detailed information, including examples and sometimes, depending on their expertise level, they need high level guidance.

Most often, methods offer more than one granularity. For instance, XP can be studied at different levels: this method is presented online on a single web page [127], is detailed in O'Reilly's pocket guide [113] and is even more detailed in Beck's original book [126]. At last, some aspects of this method are presented in dedicated papers or books, like '*Extreme Programming for Web Projects*' [128] or '*Planning Extreme Programming*' [129]. If this set of documents enlightens several facets of the method and offers the different detail levels, it seems difficult to use it. First, designers have to buy lots of documents and then they have to know exactly which document gives what kind of information. This appears to

be far from the easiness of learning and use that designers are expecting [1,2].

Some methods have been thought considering this specific need. For instance, RAD [21] suggests four integrated visions of the same process model. Chapter 1 presents in a few pages a quick overview of the method. Chapter 4 refines this description over 20 pages. Chapters 7–24 present all aspects with many details and discussions. At last, 23 'methodology charts' present detailed algorithms guiding finely over each stage. Compared to other methods, a synthetical and graphical vision of the life cycle is lacking in RAD initial description. However, the various granularities make it possible for designers to find the information adapted to their needs and knowledge.

This axis questions the capacity of a process model to present directly different granularities. The ability of a system to offer various levels of detail is named granularability. We suggest the following gradation to evaluate the process model granularability:

- **One main document presents one or two granularities** (e.g. a global overview and some detailed aspects).
- **One main document presenting one or two granularities, completed by other documents** for specific situations.
- **One main document presents more than two granularities** (e.g. a graphical synthesis and/or a global overview and/or some detailed aspects and/or some algorithms).
- One **main document presenting more than two granularities, completed by other documents**.

XP has originally been described by Beck [126], a book in which a first section presents an overview of the method, followed by several chapters detailing each methodological prescription. As mentioned before, XP description has been completed by many other books and studies. This matches the second rate of the gradation. Martin's book [21] about RAD contains four granularities, and there are several hundreds of books and scientific papers about this method. This reaches the highest grade. The Spiral Model has been described at first in a publication [16] that presents a graphical view of the lifecycle and a short description of involved activities. This paper has been widely completed by papers and books. This corresponds to the second grade. These results are illustrated in Fig. 18.

After this large description of the axes defined by Promote, the next section presents the results of an evaluation that we have made on this taxonomy.

11. Evaluation

The evaluation aimed to test the following hypothesis: Promote is easy to learn (even for designers unfamiliar to concepts considered in the taxonomy); the axes and their gradations are usable and complete.

Two evaluation sessions were conducted in 2010 and in 2011. Two groups of students (eleven students in total) of a second year Master, who can be considered as novice designers, participated to evaluation. Therefore, we were very interested in their evaluation of the learnability of the taxonomy. Participants also had to put the taxonomy in action and thus to confront the concepts and gradations to the reality of some methods. This provides us interesting feedbacks about the pertinence of the concepts and gradations.

The two groups followed the same protocol. First, the taxonomy was presented to participants through an already known method, namely Scrum. During this presentation, the students had to fill a first questionnaire asking them to note definitions and gradations on the basis of four possibilities: "not clear", "not very clear", "quite clear" or "clear". After this presentation, the two groups were asked to study and classify another method. Then, they had to present a summary of their results and to justify them. In the second group, the students had to work by pair in order to compare their results, enabling us to evaluate whether the taxonomy could facilitate methods comparison or not. An interview was conducted at the end of the presentation in order to collect participants' opinions, difficulties and remarks. A few days later, an anonymous questionnaire was sent to each participant in order to collect their free feedbacks about the classification. This questionnaire was indented to evaluate the clarity of the definitions as perceived after having applied the taxonomy. The questionnaire offered the opportunity to participants to add comments. Another part of the questionnaire was intended to gather participants' opinions regarding the completeness, the pertinence or the unusefulness of the axes. Finally, the questionnaire asked participants to evaluate profits of the classification, and their intention to reuse it. The presentations, reports, interviews and questionnaires have been recorded and analyzed. Results are presented below.

After having worked on a method classification, the students estimated that they had very well understood all the definitions, and 92% of the definitions were evaluated as clear or quite clear, as shown on Table 2. We can notice that the second group estimated that the definitions are clear more often than the first group, with a score of 55.50% vs. 34.50%. We consider that the evolution of the definitions due to the feedbacks of the first group led to this better score.

As shown in Table 3, the students estimated that classifying a method implies studying it deeply, analysing the description to interpret and understanding what is defined, or eventually what is missing. All of them agreed that the use of the taxonomy raised questions about what they would have had otherwise, and most of them thought that after classifying a method, they had a better idea about its key points (8 students over 9, the last one estimating

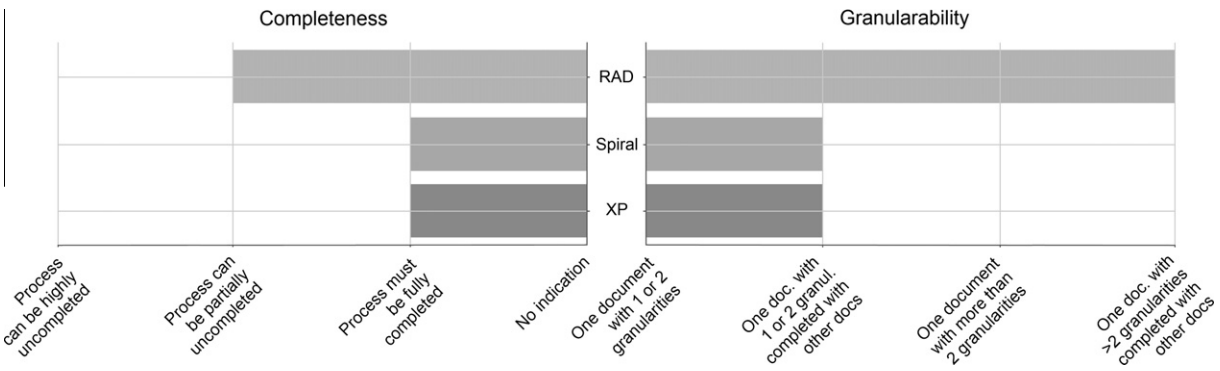

Fig. 18. Example of completeness and granularability of process models.

Table 2
Evaluation of the clarity of the definitions after using the taxonomy.

	Not clear	Not very clear	Quite clear	Clear	Total
Group 1	0	2	48	66	116
Group 2	3	18	57	111	189
Total	3	20	105	177	305
Percents (%)	0.98	6.56	34.43	58.03	100

that critical software development processes are not well enough evaluated) and untreated notions (9 students over 9). Consistently with these evaluations, the students conclude that using the taxonomy could help them verifying if a method suits well a project (8 students over 9).

We also evaluated their difficulties or problems with the taxonomy, especially with the second group, thanks to the feedbacks of the first evaluation. We first asked if they thought that some axis were superfluous or unnecessary. Four students said that they agree with this idea and, when questioned, they mention three axes: distensibility, variability and artifacts formalization. They argue that a method should be complete (and therefore there is no need to extend it), that variants are not needed because they believe that there a "good way" to do things and that evaluating artifacts formalization is too complex and does not give pertinent information about the process model. 5 of the students in the second group also agree that classifying a process model is a long and subtle work, especially because there are too many definitions to learn. They did not feel that they would be interested into classifying other process models: according to five of them, the taxonomy appears to be too complex to be used "in the real world". However, they all agreed with one of them who concluded: "it is complicated in somehow to do all the classification. But it is really an important thing to do before choosing a method for a project. It lets us analyse clearly details about each method. So, we can know which method is the most suitable for the project".

These results helped us to improve the taxonomy and to suggest a new version that has been presented above. We identified which axes were problematic and so led us to improve the definitions when possible, or to remove some axes that they estimated not so interesting. For instance, the flexibility of the method was perceived as the hardest to understand. This axis was at this time divided into two dimensions: the adaptability (including the model predefinition and its adaptability degree) and the engineering of the method (evaluated as monolithic, situational and dynamic). One student remarked that "even if the process model does not suggest variants, it can still be adapted", showing that the underlying notions were not understood. We clarified that we want to classify exactly what is actually said in the process model and not what people are used to do with it. Another student said that she felt a distortion between the classification she had made and the 'reality' of the method. According to the classification, FDD [31] did not make adaptation procedures explicit and was therefore evaluated as not flexible. According to the forums on the Internet, one of the great benefits given by the use of this method was its flexibility. This remark led us to reconsider this axis and to restructure it as it was presented above. But this leads us to an opened question: would it be interesting to also classify the regular use of the method, thereby enlightening the differences between the theorical and the actual processes. Evaluating the actual use of a process model is a very complex task, because it is imaginable that there are as many uses as there are users: how would it be possible to reflect such diversity?

The axes about the approach, the parallelism, the forms of internal collaboration, the user or usage centering, the requirement maturity and the risk appeared to be quite difficult to understand. For instance, the students understood that parallelism includes parallel development and classified unparallelized process models as parallel because they included a chapter about the distribution of development tasks to several programmers. The same mistake has been made about risk. A common remark about this axis has been "*it is impossible to predict all the risks and to manage them*". We have improved these axes, by clarifying the definitions and the gradations.

The axes about the risk or the metric were considered as unuseful by a student and we gave more substance to their explanations. The students also indicated that some information was hard to find, like the number of versions of the method or its origin (academic, industrial and so on). These remarks helped us to realize that these axes did not reflect the process model in itself and we have removed them from the taxonomy.

We were also interested in the feelings of the students about the usefulness of the taxonomy. All their remarks and evaluations show that the students estimated that classifying a method leads to a more indepth discovery than what they have had without this conceptual tool. According to them, each axis was a (good) question to answer to. Using the taxonomy led them to be more exhaustive in the analysis and in the searching for documents, what they would never have done otherwise. This feedback was not expected before this experiment. In previous years, some students were asked to study and describe methods and to apply it to a concrete small project, but the lack of documents or the need of seeking more information never appeared before. It seems that

Table 3
Evaluation of some benefits of the use of the taxonomy.

	Strongly agree	Somewhat agree	Somewhat disagree	Strongly disagree	Total
The taxonomy led you to ask questions you would not have thought to					
Group 1	2	1	0	0	3
Group 2	4	2	0	0	6
The taxonomy helped you identifying methods key points					
Group 1	2	0	1	0	3
Group 2	4	2	0	0	6
The taxonomy helped you identifying the points not treated by a method					
Group 1	3	0	0	0	3
Group 2	5	1	0	0	6
The taxonomy helped you comparing methods					
Group 1	3	0	1	0	3
Group 2	3	1	0	1	6
The taxonomy helped you verifying if a method suits well a project					
Group 1	1	1	1	0	3
Group 2	4	2	0	0	6

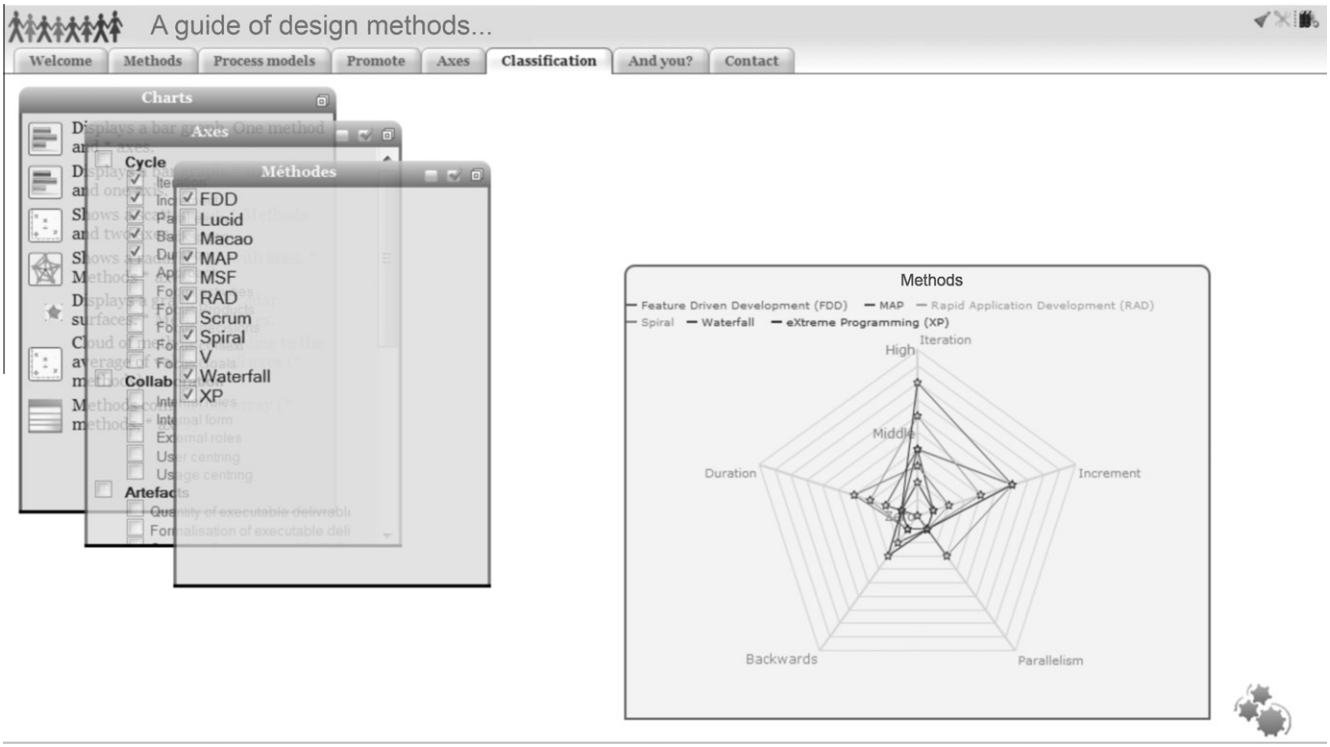

Fig. 19. Comparison of methods on design-methods.net.

approaching a process model with the taxonomy leads to studying it in more depth.

The students also pointed out that the taxonomy can be a useful tool to compare methods, because the comparison criteria are then made objective. They evaluated Promote as useful for choosing a method, with a little restriction: all the axes did not appear to have the same importance, and they expected having some kind of ponderation of their relative weights. The students concluded that they were convinced that Promote is a useful tool and are ready to reuse it when needed.

Two other points have not been mentioned directly by the students, but have emerged from the analysis of their documents and presentations. First, they expressed in their feedbacks that the taxonomy has appeared useful to highlight "key points" and "weaknesses" in methods, leading to the possibility of evaluating their adaptation to the indented use of the designers. One of them gave as an example that methods that do not talk about the customer's role will not be well suited to a project in which this customer wants to be deeply involved. Thereby, the students made us realize that when a method omits to mention a point, it can also be very interesting. This led us to refine the gradations so as to be able to express this. For instance, most of the gradations contain a value like "the method does not mention any validation procedure".

This evaluation has been very useful to us, because the qualitative analysis of students' reactions has enlightened an actual use of the taxonomy, bringing out the axes that needed refinement and rework. Of course, the results are limited by the small number of participants, which does not make it possible to draw firm conclusions on the taxonomy. Therefore, it would be interesting to complete this evaluation by studying the reaction of designers looking for a method that matches the real needs of a concrete project. The scope of the evaluation was also limited because the students only addressed the evaluative power of the taxonomy. The two other dimensions – descriptive and generative powers – would also require to be assessed. This could be achieved by interviewing methods engineers, asking them to estimate if the description given by Promote is a good overview of a process model and if the weaknesses pointed by the taxonomy help improving a method.

12. Tooling for the taxonomy

Classifying a process model is long work, for it requires studying deeply all it deals with. Obviously, a project team would not have time to study several methods and to understand them well enough to achieve this task. Therefore it seems very interesting to suggest a tool where knowledge about process models can be centralized and made accessible to everyone. This is why we have developed a specific web site[3] presenting this taxonomy and the classification of some process models.

The web site presents all the theoretical aspects of the process models and the axes. Each axis is described with its bibliography and presents a table for comparing the results over all classified methods. 12 methods (or process models) are yet described and classified on the web site: Extreme Programming [22], Feature Driven Development [31], (Lucid/Star)* [130], Macao [131], MAP [61], Microsoft Solution Framework [132], Rapid Application Development [21], Rational Unified Programming [89], Scrum [30], the Spiral model [16], the V-Model [88] and the Waterfall model [47]. A quick overview of each method is offered, including the bibliographical references and the classification of the method, with explanations about each grade. The web site also offers graphical comparisons of process models: the visitor can choose the methods he wants to compare, the axis he wants to include in the comparison and the kind of chart (bar charts, scatter plots, radar) he wants to get. We have used the functionalities of this web site to illustrate this paper. All the classifications pictured in this paper are extracted from the web site. Fig. 19 shows the comparison of a few methods on 5 axis (iteration, increment, parallelism, backwards, duration) using a Kiviat chart. It is also possible to obtain the results in a textual form. If ever the user wants, he can suggest his

[3] http://www.design-methods.net.

own classification of methods. These complementary classifications will be validated (are they consistent, pertinent, argumented?), and made available to the public. A visitor can enter comments on each page of the site, or answer to any previous comment.

13. Conclusion and perspectives

We present Promote, a taxonomy for process models based on six main axes: cycle, collaboration, artifacts, recommended use, maturity and flexibility, which are, in turn, divided into 34 subaxes in order to evaluate each single characteristic of process models. We prove the applicability of Promote through its axes description in classifying: XP, RAD and the Spiral Model. These illustrations highlight the differences and the similarities of these process models, and thus confirm that such a taxonomy is useful for understanding recommendations specific to a method and for comparing a process model to another. These examples illustrations also show how Promote supports discovery of methods as well as elicitation of their strengths, weaknesses and originalities. Finally, we presented a web site dedicated to Promote and reports the taxonomy evaluation conducted with two groups of students.

In a very near future, we will be working on making this web site a knowledge database about process models. We will also enrich the web site that by adding more methods and classify them. Some of the methods are currently being classified and will be available soon. These are: Diane+[133], Dynamic Systems Development Method [134], and Symphony [36]. We will suggest visitors to leave comments and to fill out questionnaires, in order to collect feedback about the usefulness and the completeness of Promote axes. Visitors will have the possibility of suggesting methods with their classification. The web site visitors will be able to add comments and will be asked to fill out questionnaires in order to collect feedback on usefulness and completeness of Promote axes. In addition, the web site visitors will be able to extend the classification by suggesting new axis. We also would like to add the possibility of adding keywords to every method grade. For instance, when measuring the applications types, the gradation does not explicit which concrete kinds of applications are handled by a method. A mechanism based on keywords could make this possible. The feedback will also influence the evolution of the web site functionalities, and thus help improve of the web site ease of use, presented examples as well as the shared knowledge.

References

[1] F. Garzotto, V. Perrone, Industrial acceptability of web design methods: an empirical study, Journal of Web Engineering 6 (2007) 73–96.
[2] C. Barry, M. Lang, A survey of multimedia and web development techniques and methodology usage, IEEE Multimedia 8 (2001) 52–60.
[3] B. Fitzgerald, An empirical investigation into the adoption of systems development methodologies, Information and Management 34 (1998) 317–328.
[4] N. Jayaratna, Understanding and Evaluating Methodologies: NIMSAD, a Systematic Framework, McGraw-Hill, Inc., New York, NY, USA, 1994.
[5] G. Booch, Object-Oriented Analysis and Design with Applications, second ed., Addison-Wesley Professional, 1993.
[6] Harmsen, Situational Method Engineering, University of Twente, Moret Ernst & Young Management Consultants, 1997.
[7] W.S. Humphrey, The software engineering process: definition and scope, SIGSOFT Software Engineering Notes 14 (1988) 82–83.
[8] W. Scacchi, Process Models in Software Engineering, in: Encyclopedia of Software Engineering, second ed., John Wiley and Sons, 2001.
[9] I. Sommerville, Software engineering, Pearson Education (2001) 10.
[10] M. Beaudouin-Lafon, Designing interaction, not interfaces, in: Proceedings of the Working Conference on Advanced Visual Interfaces, ACM, New York, NY, USA, 2004, pp. 15–22.
[11] I. Sommerville, Software process models, ACM Computing Surveys 28 (1996) 269–271.
[12] C. Hug, A. Front, D. Rieu, A Process Engineering Method based on a Process Domain Model and Patterns, in: International Workshop MoDISE-EUS 2008 (Model Driven Information Systems Engineering: Enterprise, User and System Models), Held in Conjunction with CAISE 2008, Montpellier, France, 2008.
[13] A. Seffah, J. Gulliksen, M.C. Desmarais, Human-Centered Software Engineering - Integrating Usability in the Software Development Lifecycle, first ed., Springer, 2005.
[14] J. Carter, J. Liu, K. Schneider, D. Fourney, Transforming usability engineering requirements into software engineering specifications: from PUF to UML, in: A. Seffah, J. Gulliksen, M.C. Desmarais (Eds.), Human-Centered Software Engineering—Integrating Usability in the Software Development Lifecycle, Springer, Netherlands, 2005, pp. 147–169.
[15] X. Ferre, N. Juristo, A. Moreno, Which, when and how usability techniques and activities should be integrated, in: A. Seffah, J. Gulliksen, M.C. Desmarais (Eds.), Human-Centered Software Engineering—Integrating Usability in the Software Development Lifecycle, Springer, Netherlands, 2005, pp. 173–200.
[16] B. Boehm, A spiral model of software development and enhancement, SIGSOFT Software Engineering Notes 11 (1986) 14–24.
[17] L.C. Alexander, A.M. Davis, Criteria for selecting software process models, in: Computer Software and Applications Conference, 1991, COMPSAC '91., Proceedings of the Fifteenth Annual International, 1991, pp. 521 –528.
[18] G. Pérez, K. El Emam, N. Madhavji, Customising software process models, in: W. Schäfer (Ed.), Software Process Technology, Springer, Berlin/Heidelberg, 1995, pp. 70–78.
[19] I. Sharon, M. dos S. Soares, J. Barjis, J. van den Berg, J.L.M. Vrancken, A Decision Framework for Selecting a Suitable Software Development Process, in: ICEIS'10, Funchal, Madeira, Portugal, 2010, pp. 34–43.
[20] S.K. Card, Pioneers and settlers: methods used in successful user interface design, in: Proceedings of a Workshop on Human-computer Interface Design: Success Stories, Emerging Methods, and Real-world Context: Success Stories, Emerging Methods, and Real-world Context, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995, pp. 122–169.
[21] J. Martin, Rapid Application Development, Macmillan Publishing Co., Inc., Indianapolis, IN, USA, 1991.
[22] K. Beck, Embracing change with extreme programming, IEEE Computer 32 (1999) 70–77.
[23] D.J. Anderson, Feature-Driven Development: Towards a TOC, Lean and Six Sigma Solution for, Software Engineering, 2004.
[24] M. Fowler, J. Highsmith, The Agile Manifesto, Software Development, Miller Freeman, Inc, 2001. pp. 28–32.
[25] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
[26] A. Cockburn, Unraveling incremental development, Object Magazine (1995) 49–51.
[27] O. Benediktsson, D. Dalcher, K. Reed, M. Woodman, COCOMO-based effort estimation for iterative and incremental software development, Software Quality Journal 11 (2003) 265–281.
[28] D. Graham, Incremental development: review of nonmonolithic life-cycle development models, Information and Software Technology 31 (1989) 7–20.
[29] P. Kruchten, The Rational Unified Process: An Introduction, third ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
[30] K. Schwaber, SCRUM Development Process, in: Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA), 1995, pp. 117–134.
[31] P. Coad, E. Lefebvre, E. DeLuca, Java Modeling in Color with UML: Enterprise Components and Process, Prentice Hall PTR, Upper Saddle River, NJ, 1999.
[32] R.F. Graf, Modern dictionary of electronics, Newnes (1999).
[33] J.-P. Stromboni, Analyse de la granularité des applications régulières, in: 16° Colloque sur le traitement du signal et des images, GRETSI, Groupe d'Etudes du Traitement du Signal et des Images, Grenoble, 1997, pp. 1299–1302.
[34] V. Messager Rota, Gestion de projet: vers les méthodes agiles, Eyrolles, 2009.
[35] P. Roques, F. Vallée, UML en action, Eyrolles, Paris, France, 2002.
[36] I. Hassine, D. Rieu, F. Bounaas, O. Segrouchni, Symphony: a conceptual model based on business components, in: SMC'02, IEEE International Conference, Hammamet, Tunisia, 2002.
[37] W.S. Humphrey, M.I. Kellner, Software process modeling: principles of entity process models, in: Proceedings of the 11th International Conference on Software Engineering, ACM, New York, NY, USA, 1989, pp. 331–342.
[38] C. Morley, Gestion d'un projet système d'information: principes, techniques, mise en œuvre, Masson, 1998.
[39] R. Descartes, Discours de la méthode, 1861.
[40] R. Feynman, Personal Observations on the Reliability of the SHUTTLE, Washington, DC, USA, 1986.
[41] M. Pizka, A. Bauer, A brief top-down and bottom-up philosophy on software evolution, in: International Workshop On Principles of Software Evolution, 2004, pp. 131–136.
[42] M. Dowson, Iteration in the software process; review of the 3rd International Software Process Workshop, in: Proceedings of the 9th International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, USA, 1987, pp. 36–41.
[43] C. Rolland, L'ingénierie des méthodes: une visite guidée, E-revue en Technologies de l'Information (e-TI), 2005.
[44] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini, Tropos: An Agent-Oriented Software Development Methodology, 2003.

[45] S. Nurcan, A survey on the flexibility requirements related to business processes and modeling artifacts, in: HICSS, 2008, pp. 378–387.
[46] C. Rolland, From Conceptual Modeling to Requirements Engineering, in: D. Embley, A. Olivé, S. Ram (Eds.), Conceptual Modeling – ER 2006, Springer, Berlin/Heidelberg, 2006, pp. 5–11.
[47] W.W. Royce, Managing the development of large software systems: concepts and techniques, in: Proceedings of the 9th International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, USA, 1987, pp. 328–338.
[48] B. Henderson-Sellers, J.M. Edwards, The object-oriented systems life cycle, Communications of the ACM 33 (1990) 142–159.
[49] M. Benabdellatif, Process modelling analysis: comparison between activity-oriented models, product-oriented models and decision-oriented models, in: Data Mining II, A. Zanasi, C.A. Brebbia, N.F.F. Ebecken, P. Melli (Editors), Southampton, SO40 7AA, UK, 2002, pp. 251–258.
[50] J. Börstler, Experience with work-product oriented software development projects, Journal of Computer Science Education 11 (2001) 111–133.
[51] M. Jarke, J. Bubenko, C. Rolland, A. Sutcliffe, Y. Vassilou, Theories underlying requirements engineering: an overview of NATURE at Genesis, in: Proceedings of the IEEE International Symposium on Requirements Engineering, San Diego, CA, USA, 1993, pp. 19–31.
[52] C. Rolland, A Comprehensive View of Process Engineering, in: Proceedings of the 10th International Conference on Advanced Information Systems Engineering, Springer-Verlag, London, UK, 1998, pp. 1–24.
[53] K. Pohl, P. Assenova, R. Doemges, P. Johannesson, N. Maiden, V. Plihon, et al., Applying AI Techniques to Requirements Engineering: The NATURE Prototype, in: Proceedings of the Workshop on Research Issues in the Intersection Between Software Engineering and, Artificial Intelligence, 1994.
[54] C. Rolland, N. Prakash, A. Benjamen, A multi-model view of process modelling, Requirements Engineering 4 (1999) 169–187.
[55] S. Brinkkemper, M. Saeki, F. Harmsen, Assembly Techniques for Method Engineering, in: Proceedings of CAISE'98, Pisa, Italy, 1998.
[56] J. Ralyté, R. Deneckère, C. Rolland, Towards a Generic Model for Situational Method Engineering, in: J. Eder, M. Missikoff (Eds.), Advanced Information Systems Engineering, Springer, Berlin/Heidelberg, 2003, p. 1029.
[57] G. Guzelian, Conception de systèmes d'information, une approche orientée service, PhD dissertation, University Paul Cézanne, 2007.
[58] W.S. Humphrey, Managing the Software Process, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
[59] C. Potts, A generic model for representing design methods, in: Proceedings of the 11th International Conference on Software Engineering, ACM, New York, NY, USA, 1989, pp. 217–226.
[60] C. Rolland, A contextual approach for the requirements engineering process, in: SEKE, 1994, pp. 28–35.
[61] C. Rolland, N. Prakash, A. Benjamen, A multi-model view of process modelling, Requirements Engineering 4 (1999) 169–187.
[62] J. Eckstein, Roles and Responsibilities in Feature Teams, in: Agility Across Time and Space: Implementing Agile Methods in Global Software Projects, Springer, 2010, pp. 289–297.
[63] A. Rochfeld, R. Colletti, H. Tardieu, La méthode Merise, tome 2: Démarche et pratiques, Editions d'Organisation, 1985.
[64] H. Williams, A. Ferguson, The UCD Perspective: Before and After Agile, in: Proceedings of the AGILE 2007, IEEE Computer Society, Washington, DC, USA, 2007, pp. 285–290.
[65] K. Phalp, M. Shepperd, Quantitative analysis of static models of processes, Journal of Systems and Software 52 (2000) 105–112.
[66] N.B. Harrison, J.O. Coplien, Patterns of productive software organizations, Bell Labs Technical Journal 1 (1) (1996) 138–145.
[67] B.G. Cain, J.O. Coplien, N.B. Harrison, Social patterns in productive software development organizations, Annals of Software Engineering 2 (1996) 259–286.
[68] H. Amblard, Les nouvelles approches sociologiques des organisations, Seuil, 1996.
[69] J.W. Little, The Persistence of Privacy: Autonomy and Initiative in Teacher's Professional Relations, Teacher's College Record, 1990, pp. 509–36.
[70] C.B. Handy, Understanding Organizations, Oxford University Press, Inc., New York, 1993.
[71] R. Cerón, J. Dueñas, J. de la Puente, A first assessment of development processes with respect to product lines and component based development, in: F. van der Linden (Ed.), Software Architectures for Product Families, Springer, Berlin/Heidelberg, 2000, pp. 158–167.
[72] J. Gulliksen, B. Göransson, I. Boivie, J. Persson, S. Blomkvist, Å. Cajander, Key Principles for User-Centred Systems Design, in: A. Seffah, J. Gulliksen, M.C. Desmarais (Eds.), Human-Centered Software Engineering—Integrating Usability in the Software Development Lifecycle, Springer, Netherlands, 2005, pp. 17–36.
[73] D.A. Norman, S.W. Draper, User Centered System Design: New Perspectives on Human–Computer Interaction, Lawrence Erlbaum Associates, 1986.
[74] ISO, Human-Centred Design Processes for Interactive Systems, 1999.
[75] J. Karat, User centered design: quality or quackery?, Interactions 3 (1996) 18–20
[76] J.D. Gould, in: R.M. Baecker, J. Grudin, W.A.S. Buxton, S. Greenberg (Eds.), Human-Computer Interaction, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995, pp. 93–121.
[77] J.M. Greenbaum, M. Kyng, Design at Work: Cooperative Design of Computer Systems, Routledge, 1991.
[78] J. Nielsen, Iterative user-interface design, Computer 26 (1993) 32–41.
[79] M. Maguire, Methods to support human-centred design, International Journal Human-Computer Studies 55 (2001) 587–634.
[80] J.-Y. Mao, K. Vredenburg, P.W. Smith, T. Carey, User-centered design methods in practice: a survey of the state of the art, in: Proceedings of the 2001 Conference of the Centre for Advanced Studies on Collaborative Research, IBM Press, 2001, pp. 12–25.
[81] M. Pieper, K. Stroetmann, Participatory insight to universal access, in: Universal Access in Health Telematics: a Design Code of Practice, Stephanidis, Constantine, 2005, pp. 271–296.
[82] D.L. Morgan, The Focus Group Guidebook, SAGE, 1998.
[83] R.W. Root, S. Draper, Questionnaires as a software evaluation tool, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA, 1983, pp. 83–87.
[84] M. Maguire, N. Bevan, User requirements analysis: a review of supporting methods, in: Proceedings of the IFIP 17th World Computer Congress – TC13 Stream on Usability: Gaining a Competitive Edge, Kluwer, B.V., Deventer, The Netherlands, The Netherlands, 2002, pp. 133–148.
[85] L.L. Constantine, R. Biddle, J. Noble, Usage-centered design and software engineering: models for integration, in: ICSE Workshop on SE-HCI'03, 2003, pp. 106–113.
[86] J. Ferreira, J. Noble, R. Biddle, The semiotics of usage-centred design, in: P.-J. Charrel, D. Galarreta (Eds.), Project Management and Risk Management in Complex Projects, Springer, Netherlands, 2007, pp. 211–229.
[87] G. Lindgaard, R. Dillon, P. Trbovich, R. White, G. Fernandes, S. Lundahl, et al., User needs analysis and requirements engineering: Theory and practice, Interacting with Computers 18 (2006) 47–70.
[88] J. McDermid, K. Ripken, Life cycle support in the ADA environment, ADA Letter III (1983) 57–62.
[89] P. Kroll, P. Kruchten, The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP, Addison Wesley, 2003.
[90] T. Chklovski, User interfaces with semi-formal representations: a study of designing argumentation structures, in: Under Review for the Intelligent User Interfaces Conference, 2005, pp. 130–136.
[91] R.N. Charette, Large-scale project management is risk management, IEEE Software 13 (1996) 110–117.
[92] Y.W. Kim, Efficient use of code coverage in large-scale software development, in: Proceedings of the 2003 Conference of the Centre for Advanced Studies on Collaborative Research, IBM Press, 2003, pp. 145–155.
[93] B.W. Boehm, R.K. McClean, D.B. Urfrig, Some experience with automated aids to the design of large-scale reliable software, in: Proceedings of the International Conference on Reliable Software, ACM, New York, NY, USA, 1975, pp. 105–113.
[94] Project sizes – small, medium and large, Method123 Project Management, Methodology. (n.d.).
[95] B.W. Boehm, B. Clark, E. Horowitz, J.C. Westland, R.J. Madachy, R.W. Selby, Cost Models for Future Software Life Cycle Processes: COCOMO 2.0, Annals of Software Engineering 1 (1995) 57–94.
[96] A.L. Rollo, Functional size measurement and COCOMO—a synergistic approach, in: Proceedings, Rome, Italy, 2006, pp. p. 259–267.
[97] M. Laanti, Implementing program model with agile principles in a large software development organization, in: Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International, 2008, pp. 1383–1391.
[98] L. Rising, N.S. Janoff, The Scrum software development process for small teams, Software, IEEE 17 (2000) 26–32.
[99] R. Spencer, The streamlined cognitive walkthrough method, working around social constraints encountered in a software development company, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA, 2000, pp. 353–359.
[100] M.A. Cusumano, R.W. Selby, How Microsoft builds software, Communications of the ACM 40 (1997) 53–61.
[101] E. Yourdon, Death March: The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects, Prentice Hall PTR, 1999.
[102] L. Westfall, Software risk management, in: Annual Quality Congress Proceedings, Washington, DC, USA, 2000.
[103] G. Calvary, J. Coutaz, D. Thevenin, A Unifying Reference Framework for the Development of Plastic User Interfaces, in: M. Little, L. Nigay (Eds.), Engineering for Human-Computer Interaction, Springer, Berlin/Heidelberg, 2001, pp. 173–192.
[104] D.L. Moody, The method evaluation model: a theoretical model for validating information systems design methods, in: ECIS, 2003.
[105] D. Moody, G. Sindre, T. Brasethvik, A. Sølvberg, Evaluating the Quality of Process Models: Empirical Testing of a Quality Framework, in: S. Spaccapietra, S. March, Y. Kambayashi (Eds.), Conceptual Modeling — ER 2002, Springer, Berlin/Heidelberg, 2003, pp. 380–396.
[106] S. Blomkvist, Towards a Model for Bridging Agile Development and User-Centered Design, in: A. Seffah, J. Gulliksen, M.C. Desmarais (Eds.), Human-Centered Software Engineering—Integrating Usability in the Software Development Lifecycle, Springer, Netherlands, 2005, pp. 219–244.
[107] R. Gong, D. Kieras, A validation of the GOMS model methodology in the development of a specialized, commercial software application, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence, ACM, New York, NY, USA, 1994, pp. 351–357.
[108] S. Dupuy-Chessa, G. Godet-Bar, J.-L. Pérez-Medina, D. Rieu, D. Juras, A Design process integrating mixed system into information systems, in: E. Dubois, P.

Gray, L. Nigay (Eds.), The Engineering of Mixed Reality Systems, Springer. Chapter 15.
[109] E. Ceret, S. Dupuy-Chessa, G. Godet-Bar, Using software metrics in the evaluation of a conceptual component model, in: RCIS, 2010, pp. 507–514.
[110] F.D. Davis, User acceptance of information technology: system characteristics, user perceptions and behavioral impacts, International Journal of Man-Machine Studies 38 (1993) 475–487.
[111] comScore Releases May 2011 U.S. Search Engine Rankings – comScore, Inc., (n.d.).
[112] J.E. Cook, A.L. Wolf, Software process validation: quantitatively measuring the correspondence of a process to a model, ACM Transactions on Software Engineering and Methodology 8 (1999) 147–176.
[113] M. O'Reilly, Extreme Programming Pocket Guide, first ed., O'Reilly Media Inc, Sebastopol, CA, USA, 2003.
[114] P.J. Agerfalk, B. Fitzgerald, Flexible and distributed software processes: old petunias in new bowls?, Communications of the ACM 49 (2006) 26–34
[115] F. Harmsen, S. Brinkkemper, J.L.H. Oei, Situational method engineering for informational system project approaches, in: Methods and Associated Tools for the Information Systems Life, Cycle, 1994, pp. 169–194.
[116] B. Fitzgerald, The use of systems development methodologies in practice: a field study, Information Systems Journal 7 (1997) 201–212.
[117] J. McDermid, K. Ripken, Life Cycle Support in the ADA Environment, Cambridge University Press, New York, NY, USA, 1984.
[118] C. Rolland, C. Salinesi, Supporting requirements elicitation through goal/scenario coupling, in: Conceptual Modeling: Foundations and Applications, 2009, pp. 398–416.
[119] N. Cross, Engineering Design Methods, John Wiley and Sons Ltd., Chichester, West Sussex, England, 1989.
[120] S. Brinkkemper, M. Saeki, F. Harmsen, Meta-modelling based assembly techniques for situational method engineering, Information Systems 24 (1999) 209–228.
[121] J. Ralyté, Ingénierie des méthodes à base de composants, PhD thesis in Computer Science, Université Paris 1 - Sorbonne, 2001.
[122] J.L. Kolodner, R.L.S. Jr, K. Sycara-Cyranski, A process model of cased-based reasoning in problem solving, in: IJCAI, 1985, pp. 284–290.
[123] M. Paulk, Using the Software CMM with Good Judgment, ASQ Software Quality Professional 1 (3) (1999) 19–29.
[124] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, 1995.
[125] Django Software Foundation, Django, The Web Framework for Professionals with Deadlines, 2005.
[126] K. Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley Professional, 1999.
[127] D. Wells, Extreme Programming: A Gentle Introduction., (n.d.).
[128] D. Wallace, I. Raggett, J. Aufgang, Extreme Programming for Web Projects, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
[129] K. Beck, M. Fowler, Planning Extreme Programming, first ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.
[130] J.W. Helms, Developing and Evaluating the (LUCID/Star)∗Usability Engineering Process Model, Virginia Polytechnic Institute and State University, 2001.
[131] N. Aussenac-Gilles, N. Matta, Making a method of problem solving explicit with MACAO, International Journal of Human-Computer Studies 40 (1994) 193–219.
[132] M.S.V. Turner, Microsoft® Solutions Framework Essentials: Building Successful Technology Solutions, Microsoft Press, 2006.
[133] J.-C. Tarby, M.-F. Barthet, The DIANE+ Method, in: CADUI, 1996, pp. 95–120.
[134] J. Stapleton, DSDM, Dynamic Systems Development Method: the Method in Practice. Addison-Wesley, Reading, MA, 1997, p. 163.