On a Concept Map for the Modelling of Controlled Flexibility in Software Processes

Ricardo Martinho
Dulce Domingos
João Varajão

DI–FCUL TR–2009–12

May 2009

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

On a Concept Map for the Modelling of Controlled Flexibility in Software Processes

Ricardo Martinho[1], Dulce Domingos[2], and João Varajão[3]

[1] School of Technology and Management, Polytechnic Institute of Leiria, Portugal
rmartin@estg.ipleiria.pt
[2] Department of Informatics, Faculty of Sciences, University of Lisboa, Portugal
dulce@di.fc.ul.pt
[3] Department of Engineering, University of Trás-os-Montes e Alto Douro, Portugal
jvarajao@utad.pt

Abstract. Software processes and corresponding models are commonly held as dynamic entities that are often changed and evolved by skillful knowledge workers such as the members of a software development team. Consequently, process flexibility has been identified as one of the most important features that both Process Modelling Languages (PMLs) and software tools that manage the processes should support. However, in the everyday practice, most software team members do not wish for total flexibility. They rather prefer to have controlled flexibility, i.e., to learn and follow advices previously modelled by a process engineer on which and how they can change the elements that compose a software process. Since process models constitute a preferred vehicle for sharing and communicating knowledge on software processes, the process engineer needs a PML that can express this controlled flexibility, along with other process perspectives. To achieve this enhanced PML, we firstly need a sound core set of concepts and relationships that defines the knowledge domain associated with the modelling of controlled flexibility. In this paper we capture and represent this domain by using Concept Maps (Cmaps). These include diagrams and descriptions that elicit the relationships between the concepts involved. The proposed Cmaps can then be used as input to extend a PML with modelling constructs to express controlled flexibility within software processes. Process engineers can use these constructs to define, in a process model, advices on changes that can be made to the model itself or to related instances. Software team members can then consult this controlled flexibility information within the process models, and perform changes accordingly.

1 Introduction

Software process modelling involves eliciting and capturing informal process descriptions, and converting them into process models. A process model is expressed by using a suitable Process Modelling Language (PML), and is best developed in conjunction with the people who participate in, or are affected by the process. Therefore, it is important for them to be familiar with the concepts

expressed by the PML. Most common ones include activities, artifacts, work products, roles, control flow elements (e.g., sequencing, fork, join, decision and merge nodes) and object flow elements (e.g., inputs and outputs to activities).

These process elements are specified in the PML's metamodel, and can be used to define process models. Models are instantiated to realise software projects. Therefore, process instances follow a certain model, and are complemented with specific (runtime) project data such as activities' duration, cost and resource assignments. Process instances are then executed along with project's real-world enactments.

Software systems that manage and execute these process models and instances are generally referred as Process-Aware Information Systems (PAIS) [1]. Particularly for software processes, these are often called Process-centred Software Engineering Environments (PSEE) [2].

Software process models and instances are commonly held as dynamic entities that often must be changed and evolved, in order to cope with alterations occurred in: the real-world software project (due to changing requirements or unforeseen project-specific circumstances); the software development organization; the market; and in the methodologies used to produce software [3]. Therefore, it should be possible to quickly design new process models, to enable on-the-fly adaptations of running instances, to defer decisions regarding the exact process modelling logic to runtime, and to evolve implemented process models over time.

Consequently, process flexibility has been identified as one of the most important features to consider in PMLs and supporting PSEEs [4]. In this context, it denotes the ability to change processes, without completely replacing them. Terms such as *adaptability* are also commonly used in literature. To clarify these, we adopted the distinction made in [5], which states that adaptability is *changeability* triggered by an internal *agent of change*, while *flexibility* involves changes made by an external one (human or software component).

Totally rigid software processes, PML and supporting PSEE have long proven to be not effective for the dynamic nature of software [3]. On the other hand, allowing for total process flexibility (i.e., 100% changeable) threatens the original purpose of process models, which is to provide guidance for software project planning and execution. In fact, enabling software team members to perform (and describe) unlimited and unclassified changes to software processes hampers seriously the learning and reuse of this information in the future. Moreover, in the everyday business practice, most people do not want to have much flexibility, but would like to follow very simple rules to complete their tasks, making as little decisions as possible [6, 7].

To corroborate this, case studies on flexibility in software processes (e.g., [8]) make evidence on the need of having (senior) process participants expressing and controlling the changes that other process participants are advised to make in software process models, instances and, consequently, real-world software projects. This *controlled flexibility* can be defined as *the ability to express and control, by means of a PML and resulting process models,* which*,* where *and*

how *certain parts of a software process can change, while keeping other parts stable* [7].

Therefore, the first step on deriving a controlled flexibility-aware PML is to provide a set of understandable concepts and relationships associated with this domain of knowledge. These should be easily learnt and shared among process engineers and software development team members. We provide, in this paper, a set of core concepts and relationships that pertain the modelling of controlled flexibility in software processes. We achieve this purpose by using Concept Maps (Cmaps) [9] that capture these concepts through diagrammatic representations of the relationships between them. We also provide detailed descriptions about these maps.

The purpose is to form a sound knowledge base on controlled flexibility, to be shared between process engineers and software team members. This knowledge is to be latter incorporated into a PML, in order to enable process engineers the modelling of advices on which, where and how changes can be made to software processes. Software team members should then follow those advices and perform changes accordingly.

This paper is organised as follows: section 2 introduces Cmaps and section 3 presents the main diagrams for the proposed maps, along with descriptions on the comprised concepts and relationships. Section 4 analyses most relevant related work on flexibility conceptual frameworks, taxonomies and corresponding applications. Finally, section 5 concludes the paper and presents future work.

2 Introduction to ontologies and Concept Maps

The use of ontologies pursuit a standard way of specifying content-specific agreements for the sharing and reuse of knowledge [10]. A body of formally represented knowledge is based on a *conceptualisation*: the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them [11]. An *ontology* is an explicit specification of a conceptualisation [10].

This specification can assume more or less formal representations, according to the purpose of the ontology, and to whom/what will be the sharing target of the knowledge it defines. If it is to be shared among humans, it can assume a less formal and human-centred representation, such as the diagrams and tools associated with Concept Maps (Cmaps) [9]. If the reasoner is automated (e.g., a software component), then a more formal and machine-centred language applies, such as the Resource Description Framework (RDF) and Web Ontology Language (OWL) [12].

According to Novak and Cañas in [9], a *Concept Map* is a graphical tool for organising and representing knowledge. It includes concepts that are graphically represented by a box or circle drawing shape, and relationships between concepts represented by a connecting line linking two concepts. Lines have also a textual representation (referred to as *linking words* or *linking phrases*), which specify the relationship between the two concepts.

Concept is defined as a perceived regularity in events or objects, or records of events or objects, designated by a label. This label is usually composed by one or more words, but can also assume other textual symbols, such as + or %.

Propositions are statements regarding objects or events in the universe, either naturally occurring or constructed. They form a meaningful statement about the relationships that exist between two or more concepts.

Relationships may be classified under *static* and *dynamic* [13]. Static relationships between concepts help to describe, define, and organise knowledge for a given domain. Classifications and hierarchies are usually captured in relationships that have a static nature and indicate belongingness, composition, and categorisation.

A dynamic relationship between two concepts reflects and emphasises the propagation of change in these concepts. It shows how change in quantity, quality, or state in one concept causes change in quantity, quality, or state of the other concept in a proposition. In other words, a dynamic relationship reflects the functional interdependency of the two concepts involved [14].

These are the basic foundations of Cmaps. We apply them in the next section to capture the knowledge domain of modelling controlled flexibility within software processes.

3 Capturing the knowledge involved in modelling controlled flexibility

We adopted Cmaps and CmapTools [15] as a less formal but proven method for explaining and communicating domain knowledge [16]. CmapTools has been developed over the previous decade as an intuitive, human-centred computer interface for creating and managing concept maps. We use it to create and manage the relationships between the proposed concepts. Our goal does not comply, for the time being, automated mechanisms as possible targets for sharing knowledge. We aim at enhancing the learning and sharing of process controlled flexibility-related concepts among process engineers and software team members. Nevertheless, CmapTools includes export and import features for ontologies represented in formal OWL-derived languages, which provides us the means for a latter and more formalised ontology approach.

To achieve our purpose, we begin by providing in the next section a focus question to which our Cmaps must be able to answer.

3.1 Focus question

Recent studies advocate that, when a Cmap is constructed with a particular purpose and for a certain audience, its objective is better achieved by providing a *focus question*, to which the Cmap will be designed to answer [14, 9]. Moreover, and to prompt dynamic thinking and dynamic relationships between concepts, the question should be of type *"How does the concept X work?"*, rather than *"What is concept X?"* [14].

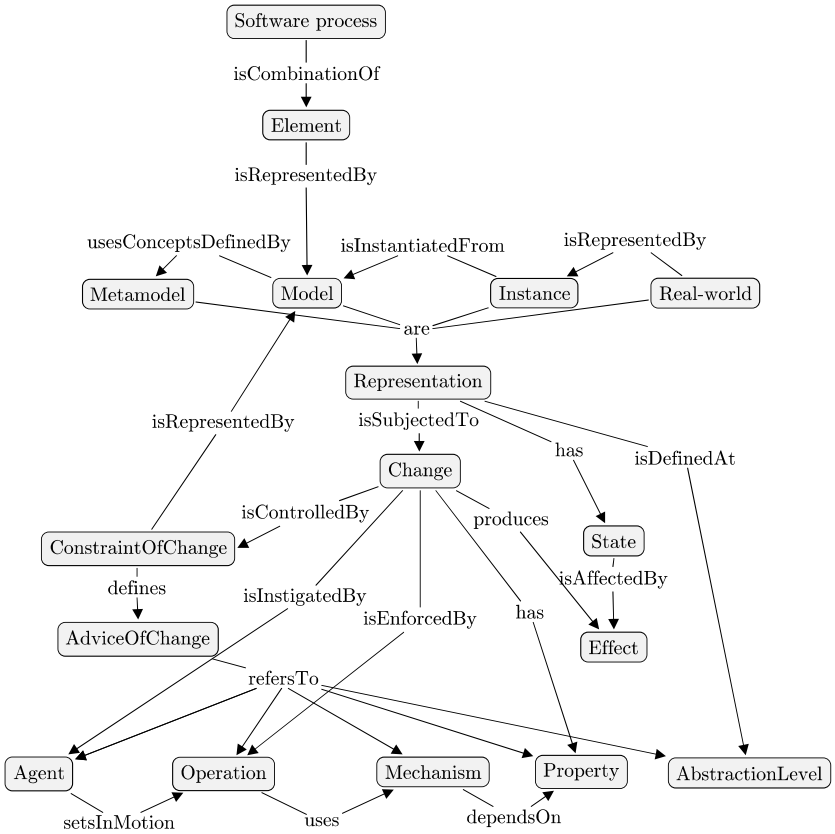

Fig. 1. Core Concept Map for sharing knowledge on process controlled flexibility

In the context of this work, we propose, in the next section, Cmaps to provide answers for the following focus question:

> *"How can software processes be subjected to changes in a controlled way?"*

3.2 Diagrams

In this section we present the diagrams for the concepts and relationships that provide answers to the focus question. We begin by illustrating in Figure 1 the main Cmap for this purpose. As we carry along describing those concepts, we also provide helpful descriptions on related contexts and examples (not represented in the diagrams, for not being within their focus).

Figure 1's diagram shows that a *software process* is a combination of *elements* represented by a *model*. These elements can be used to express correlated process perspectives, including [17]:

- *Functional* - what process elements are being performed (e.g. *phase*, *activity* and *step* elements);
- *Behavioural* - when process elements are performed (e.g. control flow nodes such as *fork*, *join*, *decision* and *merge* nodes, as also iterative/parallel region elements);
- *Organisational* - where and by whom (which agents) in the organisation process elements are performed (e.g. *role* elements, and resources like physical communication mechanisms, physical media and location used for storing entities); and
- *Informational* - informational entities produced or manipulated by a process (e.g. *data*, *artifact*, *work product* (intermediate and end) and *object* elements). It includes both structure of informational entities and the relationships among them.

Back to our diagram, a process element *model* depends on its *metamodel*, which establishes its structure of concepts, relationships and constraints. An overall process metamodel usually defines a Process Modelling Language (PML), where all process modelling elements are specified. The Software & Systems Process Engineering Metamodel (SPEM) [18] is an example of a software process metamodel which defines UML Activity Diagrams (AD) as the core PML.

Process models are then created as instances of the metamodel, and include as more or less specified arrangements of process element model representations such as activities, resources and resulting work products. Examples of software process models include the (textual and graphical/diagrammatic) process representations comprised by well known software methods such as the Unified Process [19].

Applying a process model for a specific software project is called process *instantiation*. An *instance* follows the model and needs to be provided with specific data for each distinct project, such as activities' duration, (human) resource assignments, cost estimations and monitoring/control data updates. Multiple process instances may share the same process model. On the contrary, *real-world* processes have a 1:1 multiplicity to process instances, as they reflect the activities, resources and products that are actually performed, used and produced by humans or tools. It describes what is really happening, and process participants may retrieve feedback which is used to update process running instances.

Metamodel, model, instance and real-world are process element *representations* defined at distinct but correlated *abstraction levels* of modelling. These representations are subjected to *changes*, which in turn have *effects* that can affect their *states*.

A *change* is characterised by *properties* and enforced by *operations*. Performing *change operations* includes creating, updating and deleting process elements, as well as moving them or realising element- and representation-specific operations such as *undo*, *skip* or *redo* an `Activity` in a process running *instance*. Actually, *change operations* are the actions that will change the state of process elements.

Properties of change are not dependent on a process element's type, but characterise multiple and general dimensions of a change. Concrete properties of change commonly referred in literature include [20, 21]:

- *Extent* of change, denotes whether a change is only introduced to an already existing process model (*incremental* change), or abolishes the existing process model and creates a completely new one (*revolutionary* change). Often experts are required to do revolutionary changes to the whole or part of a process model;
- *Duration* of change, states that changes can be *temporary* or *permanent* changes. Temporary changes are valid for a limited period of time, and permanent changes are valid until the next permanent change;
- *Swiftness* of change, expresses whether changes are to be applied *immediate*ly to all process model instances (also the running ones), or *deferred* only to new instances of the changed process model;
- *Anticipation* of change, defines whether the change is *planned* or *ad-hoc*. Ad-hoc changes are often made to tolerate exceptional situations, and planned changes are often part of a process redesign.

Operations use *mechanisms* in order to be enforced. For example, executing an *add* operation on a process model implies the use of a software tool that, besides supporting process editing features, also provides verification of conformance, consistency, and compliance rules associated with that operation. All these resources can be considered as mechanisms of change [5]. These mechanisms can depend on the properties that are desired for a change to have. For example, if the former add operation was to be valid during two weeks (duration property of change), the mechanism(s) used to support it would have also to comprise this property.

Changes are instigated (put into action) by *agents* of change. In the software process context, the agent of change is responsible for setting into motion *operations* that will result in an *effect* of change endured by one or more process element representations.

Agents of change may be software components that automatically change process element representations under some criteria, or humans such as software process engineers, project managers, analysts, designers, programmers and testers, that need to change/adjust software process representations.

A change can be controlled by *constraints*, which are also represented by *models* that the process engineer configures. Constraints define *advices of change*, which refer to the abstraction level, operations, properties, mechanisms and/or agents that should be considered when changing a certain process element representation. Advice on a change can be a *value*- or *text*-based attribute (for instance, *60%* or *recommended*), or any other combination of values that best fit the process element representation to which the advice is associated.

For example, a constraint of change may impose that *changes made at the process* model *are* recommended *to be reflected* immediately *to all corresponding running instances*. The modelling of this constraint can be made by composing

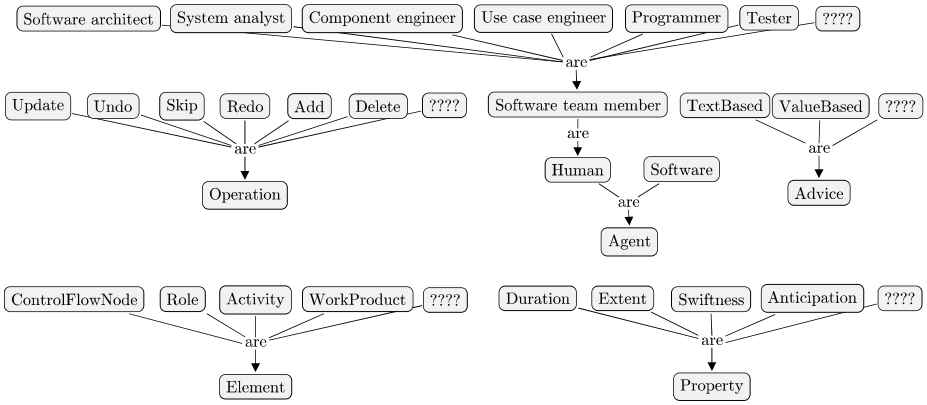

Fig. 2. Generalisation relationships for some of the concepts illustrated in Figure 1

a tuple with semicolon separated values of the form *(abstraction level representation: <String>; change property name: <String>, change property value: <ChangePropertyValue>; advice type: <AdviceType>, advice value: <AdviceValue>)*, with the values *(abstraction level representation:* `"model"`*; change property name:* `"swiftness"`*, change property value:* `immediate`*; advice type:* `TextBasedAdvice`*, advice value:* `"recommended"`).

Figure 2 represents some generalisation relationships pertaining concepts already presented in Figure 1 and referred above. Some concepts are filled with question marks (????). This means that the general concept can have more or different specialised elements beyond the examples represented, since they can depend on factors such as the modelling language, application domain, tool support and/or organisational context.

4 Related work

Since the early 1990s, there has been a shift from data orientation to process orientation, triggering the development of Process Aware Information Systems (PAIS): software systems that manage and execute operational processes involving people, applications, and/or information sources on the basis of process models [1]. Within the process modelling and enacting research areas, flexibility became a hype when several researchers observed that rigidity was not desirable in some specific areas, such as the ones involving software processes [3]. Thereafter, some research works contributed with conceptual frameworks, while others focused on tool support regarding on flexibility.

Although all these works are valuable for refining the process flexibility scope, they do not share a common set of inherent concepts. In fact, we can observe that there are different descriptions, perspectives and classifications for the concepts involved (see, e.g., the concept of *abstraction level of change*, which fall into different classifications in [22], [20] and [23]).

We also could not find any related work on concept maps for process flexibility, or any kind of specific (sub)taxonomy of concepts related with the control of that flexibility. Nevertheless, the taxonomies presented here as related work are a mix between flexibility concept descriptions and flexibility classifications, where relationships between the concepts described/classified are not always clear. Also, these works emphasise on flexibility within process models and instances, disregarding the possibility of controlling flexibility within metamodel and real-world representations as well.

One of the most prominent foundational research works on software process flexibility is the one presented by [23] and his Prism model of changes. Here, Madhavji has identified several items strongly related with the software development environment, which are subjected to continuous change. The basic items of change are people, policies, laws, resources, processes and results. For various predictable and unpredictable reasons, such items may need to be changed on an ongoing basis. Therefore, the Prism model of change included the following features:

- Separation between changes made to the described items and changes made to the environmental facilities encapsulating those items;
- A *dependency structure* that describes the various items and their interdependencies. It is used to identify the items affected by a given change;
- A *change structure* that classifies, records and analyses change-related data, and makes qualitative judgement of the consequences of a change;
- Identification of the many distinct properties of a change;
- A built-in mechanism for providing feedback on changes made.

Prism incorporates a two-level approach for designing, implementing and controlling changes to items in an environment. The first level (the *meta* level) deals with the identification of the set of items affected due to a certain change. The second level (the *normal* level) is where data related to the change is classified and recorded. These levels also include feedback mechanisms to instigate changes to items and provide projection for future changes.

The PRISM model of changes covers not only the identification of change concepts and relationships, but also the logging of changes made and corresponding analysis/feedback to improve the software process. In spite of being an important contribution to characterise change, it does not comply a way for process engineers to model some control of changes, based on feedback information from software project runs.

Figure 3 illustrates the Prism model of change with the two-level approach. Both dependency and change structures are to be use at the normal level. It is at the meta level that these structures can be added, refined and improved through meta feedback mechanisms that come from their use in the normal level.

Another earlier and often cited work is the one presented in [24], tagged as a reference framework for categorising process evolution. Based on six dimensions (origin, cause, type, how, when, and by whom), a change categorised by this framework is called a *change pattern*.

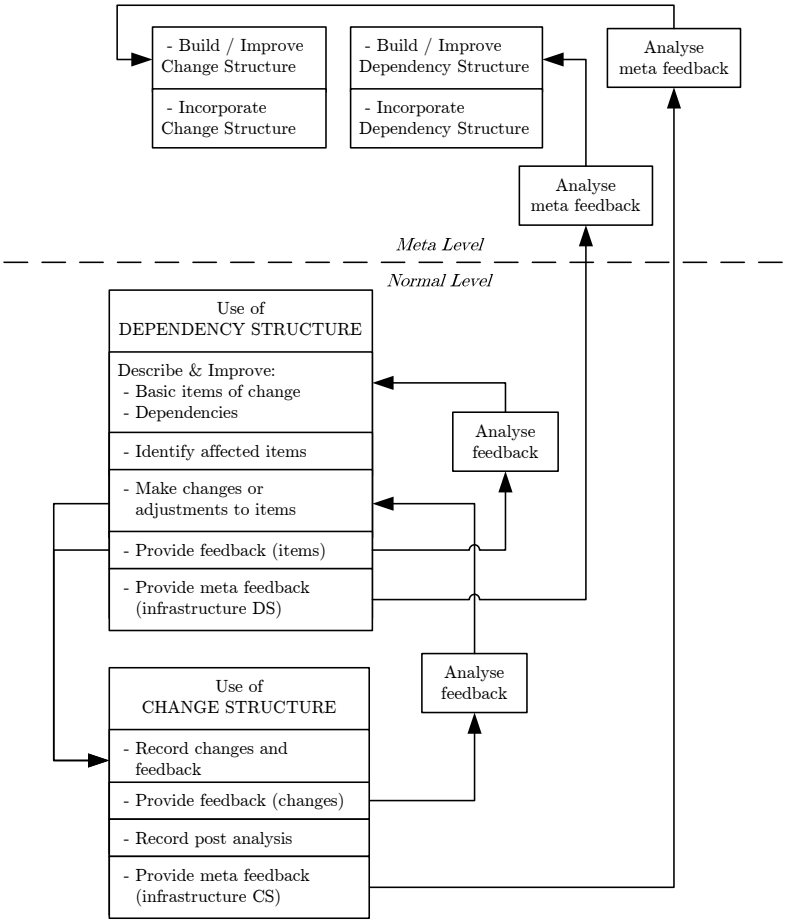

Fig. 3. The PRISM Model of changes (adapted from [23])

The change pattern, project characteristics, and product quality attributes are stored together so that they can be used for future projects. Data on the evolution of a software development project were collected in a case study performed in the software development department of a banking institution. With regard to the *origin*, i.e., the sources of process changes, 40% of the recorded changes were due to customer requests, and 60% were due to changes from senior or middle management. The most common observed reasons (*causes*) were the following:

- Misunderstanding originating from the customer;
- Resources and competence was not always available; and
- A new approach for solving the problem was adopted.

As the aforementioned work of Madhavji with the PRISM model, the authors focused on registering and categorising changes made to process runs. This information is then directly used to fine-tune process representations. Once again, it is not included as guidance or control for process participants that really change the way the process is executed against model representations.

According to a recent taxonomy proposed in [20], process flexibility can be classified in three orthogonal dimensions:

1. *Abstraction level of change*, that distinguishes *where* changes are to be made, i.e., if at the *type* or *instance* levels (or both). Changing the process model (*type* level) implies changing the defined standard way of working, as it will affect all instances created there forward. However, change can occur only for certain instances of a process (*instance* level), in order to accommodate exceptional situations;
2. *Subject of change*, representing *which* modelling elements are to be changed, and, consequently, which related perspective(s) (such as the ones in [17] mentioned above);
3. *Properties of change*, denoting *how* can a modelling element be changed. Examples include properties mentioned above, such as *extent*, *duration*, *swiftness* and *anticipation* of change.

This taxonomy was very useful to establish some of the concepts we used in our Cmaps, namely the concepts of *abstraction level of change* and *property of change*. These can be used to compose the tuple of a *constraint of change* to control the changes made to a process element representation. Process element representations correspond (although not explicitly), in this taxonomy, to the *subjects of change* concept.

In a sequence of also recent contributions, in [21] Schonenberg et. al propose another taxonomy on process flexibility. Four distinct approaches are identified, each having its own application area. They are *flexibility by*:

- *design* - for handling anticipated changes in the operating environment, where supporting strategies can be defined at design-time;
- *deviation* - for handling occasional unforeseen behaviour, where differences with the expected behaviour are minimal;
- *underspecification*: for handling anticipated changes in the operating environment, where strategies cannot be defined at design-time, because the final strategy is not known in advance or is not generally applicable;
- *change*: either for handling occasional unforeseen behaviour, where differences require process adaptations, or for handling permanent unforeseen behaviour.

Each flexibility type operates in different ways. Figure 4 provides an illustration of the distinction between these types in isolation, in terms of the time at which the specific flexibility options need to be configured - at design time, as part of the process model or at runtime via facilities in the process execution environment. It also shows the anticipated completeness of the process represen-

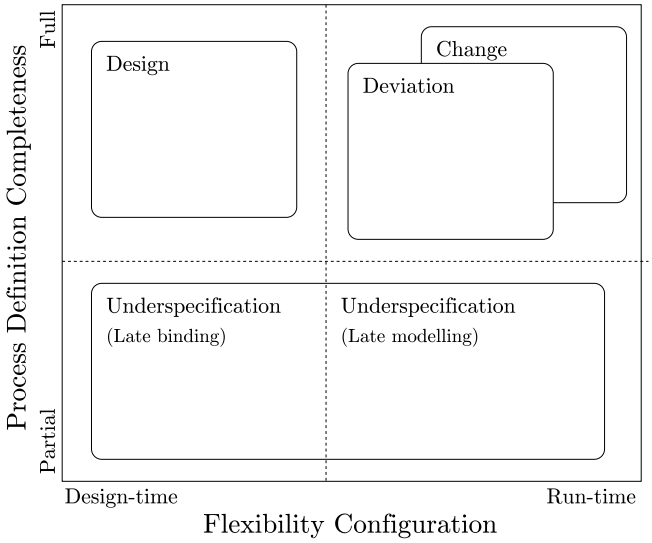

Fig. 4. Flexibility type spectrum (adapted from [21])

tation for each flexibility type.

Flexibility by *underspecification* works on the basis of an incomplete process model. Combined with late binding only, it just offers design-time configuration options, i.e., only the fragments that have been defined during design-time can be selected at runtime. Whereas, combined with late modelling, also runtime configuration options are offered by providing means to define and select fragments at runtime.

In the spectrum of options, flexibility by *design* distinguishes itself by being the flexibility type that works for full process definitions, whilst only being configurable at design-time. At runtime, only predefined paths can be chosen.

Both flexibility by *deviation* and *change* work with complete process representations. For both types, the configuration options are only available at runtime. Although very similar, only flexibility by change affects process representations from both instance and model levels of abstraction, whereas flexibility by deviation does not affect process models at all.

The same authors also survey contemporary approaches [25], and evaluate their taxonomy against a selection of PAIS, namely ADEPT1 [26], YAWL [27], FLOWer (version 3.0) [28] and Declare (version 1.0) [29].

Although our purpose does not include an explicit classification of types of flexibility, these works by Schonenberg et. al contributed to refine the scope of several change concepts and relationships of our Cmaps, namely *mechanism*, *property*, *operation* and *abstraction level*.

Constraints of change in process models are analysed in [30]. The authors propose modelling tool artifacts to support correctness, compliance and consistency constraint modelling and checking. They have developed a simple diagrammatic

language named Business Process Compliance Language (BPCL) which a process engineer can use to specify constraints to which process representations have to comply.

This work has contributed to develop our *constraint of change* concept. It also provides us several ideas for future work, namely regarding tool support for the (syntax and semantic) analysis on the modelling and checking of controlled changes.

5 Conclusions and future work

We provide in this paper concept maps for the modelling of controlled flexibility in software processes. The main purpose of this map is to establish a sound knowledge base to enhance process engineers and software team members' understandability on process flexibility and how to control it. Our approach was to define simple concepts that provide answers to the focus question *"How can software processes be subjected to changes in a controlled way?"*.

We provide no taxonomic classifications for the concepts. The intent is to develop a PML metamodel structure flexible enough to support these concepts and corresponding relationships, in order to reflect software organisations' distinct needs of controlling flexibility within their software processes.

For this matter, we are using the presented maps as a conceptual basis to extend UML Activity Diagrams (AD) as a flexibility-aware PML. We have been working on this through the use of a profile of our own called FlexUML (see [31] for further details). FlexUML defines a UML profile with a set of stereotypes and tagged values that extend AD to support the modelling of controlled flexibility in software process models. Being a standard *de facto* nowadays, the use of UML AD as a PML to extend with controlled flexibility provides a better adjustment for the modelling experience of software team members. They already use UML AD and stereotypes on a daily basis to develop software. They also recognise them as the PML used for modelling well known processes like RUP [32], XP [33], OpenUP [34] and Scrum [35].

We are also implementing the extended AD on a PSEE tool called Flex-EPFC (see [36] for further details), which is based on the IBM's Eclipse Process Framework Composer (EPFC[1]) tool. This enables process engineers to define and publish software process models with additional (textual/graphical) flexibility information through the use of the FlexUML profile stereotypes. Other software team members can then visualise and learn about this information, and change process model and/or instance representations accordingly.

References

1. Dumas, M., van der Aalst, W., ter Hofstede, A.H.M., eds.: Process-Aware Information Systems: Bridging People and Software Through Process Technology. John Wiley & Sons, Inc. (2005)

[1] http://www.eclipse.org/epf

2. Gruhn, V.: Process-centered software engineering environments, a brief history and future challenges. Ann. Softw. Eng. **14**(1-4) (2002) 363–382
3. Cugola, G.: Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models. IEEE Transactions on Software Engineering **24**(11) (1998) 982–1001
4. Reichert, M., Rinderle-Ma, S., Dadam, P.: Flexibility in process-aware information systems. LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC), Special Issue on Concurrency in Process-aware Information Systems **2** (2009) 115–135
5. Ross, A.M., Rhodes, D.H., Hastings, D.E.: Defining changeability: Reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value. Systems Engineering **11**(3) (2008) 246–262
6. Bider, I.: Masking Flexibility Behind Rigidity: Notes on How Much Flexibility People are Willing to Cope With. In: Proceedings of the 17th International Conference on Advanced Information Systems Engineering (CAiSE'05). (2005) 7–8
7. Borch, S.E., Stefansen, C.: On Controlled Flexibility. In: Proceedings of the 7th Workshop on Business Process Modeling, Development and Support (BPMDS'06) co-located with the 18th Conference on Advanced Information Systems Engineering (CAiSE'06). (2006) 121–126
8. Cass, A.G., Osterweil, L.J.: Process Support to Help Novices Design Software Faster and Better. In: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE '05). (2005) 295–299
9. Novak, J.D., Cañas, A.J.: The theory underlying concept maps and how to construct and use them. Technical report, IHMC CmapTools, 2006-01 Rev 2008-01, Florida Institute for Human and Machine Cognition (2008) available at: `http://cmap.ihmc.us/Publications/ResearchPapers/TheoryUnderlyingConceptMaps.pdf`.
10. Gruber, T.R.: Toward principles for the design of ontologies used for knowledge sharing. International Journal of Human-Computer Studies **43**(5-6) (1995) 907–928
11. Genesereth, M.R., Nilsson, N.J.: Logical foundations of artificial intelligence. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1987)
12. W3C: Owl web ontology language guide - w3c recommendation 10 february 2004. Technical report, W3C (MIT, ERCIM , Keio) (2004)
13. Safayeni, F., Derbentseva, N., Cañas, A.J.: A theoretical note on concepts and the need for cyclic concept maps. Journal of Research in Science Teaching **42**(7) (2005) 741–766
14. Derbentseva, N., Safayeni, F., Cañas, A.J.: Concept maps: Experiments on dynamic thinking. Journal of Research in Science Teaching **44** (2007) 448–465
15. Cañas, A.J., Hill, G., Carff, R., Suri, N., Lott, J., Gmez, G., Eskridge, T.C., Arroyo, M., Carvajal, R.: Cmaptools: A knowledge modeling and sharing environment. In: Proceedings of the 1st International Conference on Concept Mapping. (2004)
16. Hoffman, R.R., Woods, D.D.: Studying cognitive systems in context: Preface to the special section. Journal of the Human Factors and Ergonomics Society **42** (2000) 1–7
17. Curtis, B., Kellner, M.I., Over, J.: Process Modeling. Communications of the ACM **35**(9) (1992) 75–90
18. OMG: Software Process Engineering Metamodel Specification, v2.0. Technical report, Object Management Group (2007)
19. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1999)

20. Regev, G., Soffer, P., Schmidt, R.: Taxonomy of Flexibility in Business Processes. Input to the 7th Workshop on Business Process Modeling, Development and Support (BPMDS'06) co-located with the 18th Conference on Advanced Information Systems Engineering (CAiSE'06), Website (June 2006) available at: `http://lamswww.epfl.ch/conference/bpmds06/taxbpflex`.
21. Schonenberg, H., Mans, R., Russell, N., Mulyar, N., van der Aalst, W.M.P.: Towards a taxonomy of process flexibility (extended version). BPM Center Report BPM-07-11, BPMcenter.org (2007)
22. Schonenberg, H., Mans, R., Russell, N., Mulyar, N., van der Aalst, W.M.P.: Towards a taxonomy of process flexibility. In: Proceedings of the Forum at the CAiSE'08 Conference. (2008) 81–84
23. Madhavji, N.H.: Environment evolution: The prism model of changes. IEEE Trans. Softw. Eng. **18**(5) (1992) 380–392
24. Nguyen, M.N., Conradi, R.: Towards a rigorous approach for managing process evolution. In: Proceedings of the 5th European Workshop on Software Process Technology (EWSPT'96), London, UK, Springer-Verlag (1996) 18–35
25. Schonenberg, H., Mans, R., Russell, N., Mulyar, N., van der Aalst, W.M.P.: Process flexibility: A survey of contemporary approaches. In: Proceedings of Advances in Enterprise Engineering I, 4th International Workshop CIAO! and 4th International Workshop EOMAS, held at CAiSE 2008. (2008) 16–30
26. Reichert, M., Rinderle, S., Dadam, P.: Adept workflow management system. In: Proceedings of the International Conference on Business Process Management (BPM 2003). (2003) 370–379
27. van der Aalst, W.M.P., ter Hofstede, A.H.M.: Yawl: yet another workflow language. Information Systems **30**(4) (2005) 245–275
28. van der Aalst, W.M.P., Weske, M.: Case handling: a new paradigm for business process support. Data and Knowledge Engineering **53**(2) (2005) 129–162
29. Pesic, M., Schonenberg, M.H., Sidorova, N., van der Aalst, W.M.P.: Constraint-based workflow models: Change made easy. In: Proceedings of the OTM Conference on Cooperative information Systems (CoopIS 2007). (2007) 77–94
30. Wörzberger, R., Kurpick, T., Heer, T.: On correctness, compliance, and consistency of process models. In: Proceedings of the 17th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2008). (2008)
31. Martinho, R., Domingos, D., Varajão, J.: FlexUML: A UML Profile for Flexible Process Modelling. In: Proceedings of the 19th International Conference of Software Engineering and Knowledge Engineering (SEKE'2007). (2007) 215–220
32. Kruchten, P.: The Rational Unified Process: An Introduction. 3 edn. Addison-Wesley, Boston (2003)
33. Beck, K.: Extreme Programming Explained: Embrace Change. 1st edn. Addison-Wesley Professional (October 1999)
34. Eclipse Foundation: OpenUP. Wiki (October 2008) http://epf.eclipse.org/wikis/openup/index.htm, Accessed on April 6th, 2009.
35. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Prentice Hall PTR (2001)
36. Martinho, R., João Varajão, Domingos, D.: A two-step approach for modelling flexibility in software processes. In: Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'2008). (2008)