International Journal of Advance Research in Science and Engineering
Vol. No.5, Special Issue No. (01), February 2016
www.ijarse.com
IJARSE
ISSN 2319 - 8354

# CONTEXT-DRIVEN PROCESS ORCHESTRATION METHOD (CODPOM)

**Mira Kajko Mattsson**

*School of Information and Communication Technology,*

*KTH Royal Institute of Technology, Sweden*

## ABSTRACT

*For years, software community has tried to find standard ways for designing optimal software development processes. It is however a very difficult task. Organizations, their businesses and contexts are multi-dimensional, diverse and very complex. Hence, using standard process models may not always be an optimal solution for all types of software engineering endeavors. Instead, organizations should be able to tailor their standard development processes to the specific software engineering contexts. Their design should be strongly dependent on many aspects such as type of a software system to be developed, available resources, regulatory constraints, system quality to be achieved and the like. In this paper, we suggest Context-Driven Process Orchestration Method (CoDPOM), based on the concept of practice composition. Our main goal is to provide basis for future research in practice composition.*

***Keywords: Activity Spaces, Practice Variant, Practice Composition, Process.***

## I. INTRODUCTION

There is no such thing as one process model fits all software development contexts. Development is too much diverse, multi-dimensional and multi-faceted and its methods strongly vary with respect to each individual company, the industrial domain to be applied in and many various reasons. The list of reasons is almost endless. To mention a few, software development process is dependent on financial resources, human resources, various regulatory constraints, stability of project requirements, supporting tools such as software and hardware tools, system quality to be achieved, delivery time, customer satisfaction, organizational knowledge and other reasons. [1, 2, 4]

One of the most important reasons leading to the diversity of software processes is the application domain of the system to be developed and quality goals. Development of an aircraft controller is different from development of a website informing about nutrition aspects of some food items. In case of the aircraft controller, a rigorous software process must be applied whereas a lightweight ad hoc-ish process may be satisfactory in case of the website development. [3, 6]

Even within one and the same company, processes vary depending on the type of project and its size, product to be developed, individuals involved and available resources. Organizational size is also an important aspect herein. It is easier for smaller organizations to maintain the uniformity of their process. It is however very difficult for large multi-national organizations to upkeep uniform processes due to diverse process and business needs of their different business units.

Understanding the scope of the process may also vary depending on the role involved in it and the context. For one developer, a software process may imply following some designer's instructions for how to code whereas for some other developer a process may imply all activities from specifying and analyzing requirements to designing, implementing and testing them. Something similar may apply to a team. For one team, a process may imply acquiring a product backlog and making sure that its items get implemented. For another team, it may mean handling both management and engineering tasks starting from gathering requirements to designing, implementing, testing and deploying the system. Finally, for high-level management, a process is understood as a totality of activities to be performed by an entire organization ranging from marketing to development, to continuous development, throughout maintenance to actual retirement and system replacement. For another management, a process may imply the same, however, in the context of a highly distributed multi-national organization.

One of the most obvious reasons aiding to the complexity and diversity of software processes is the fact that modern software-based systems are not developed from scratch. They are a mix of ingredients such as newly developed code, COTS, open source, legacy components and the like. These ingredients must all be considered while designing software development processes. In addition, processes must be designed in such a way so that organizations may smoothly and rapidly adapt them to changing business requirements and contexts.

The above-listed reasons to process diversity makes us realize that creating standard uniform processes within the organizations and forcing teams and individuals to follow them is not always the right solution. Standard processes may still be used, however, not as requirements for following them but as guidelines for designing optimal process variants that are appropriate for specific software development contexts.

In this paper, we suggest *Context-Driven Process Orchestration Method*, abbreviated as CoDPOM. CoDPOM addresses both managerial and engineering aspects and it is applicable within all types of organizations dealing with software development. It addresses process in the small and in the large. CoDPOM proposes a new way for designing software processes based on various project properties. It deals with all types of processes: software development, software evolution and maintenance, support processes, and various management processes [7].

The remainder of this paper is as follows. Section 2 describes current status within software organizations. Section 3 presents CoDPOM method. Finally, Section 5, rounds up this paper by claiming that CoDPOM constitutes a basis for defining future research and industry projects.

## II. STATUS TODAY

Companies have complex portfolios of business and engineering processes that are managed and performed by a complex portfolio of roles. Most of their processes are defined as organizational standards to be reused by various projects and roles. The standards are often complemented with guidelines to be reused in various project and non-project related contexts. The goal is to assure that all types of organizational activities are traced back to the organizational standards and that they provide feedback for analyzing and improving them.

Use of organizational standards has shown to be more difficult than expected. Instead of helping organizations have control over their processes, organizational standards hamper companies in adapting their work to their individual contexts and in drawing lessons learned from the adaptation efforts.

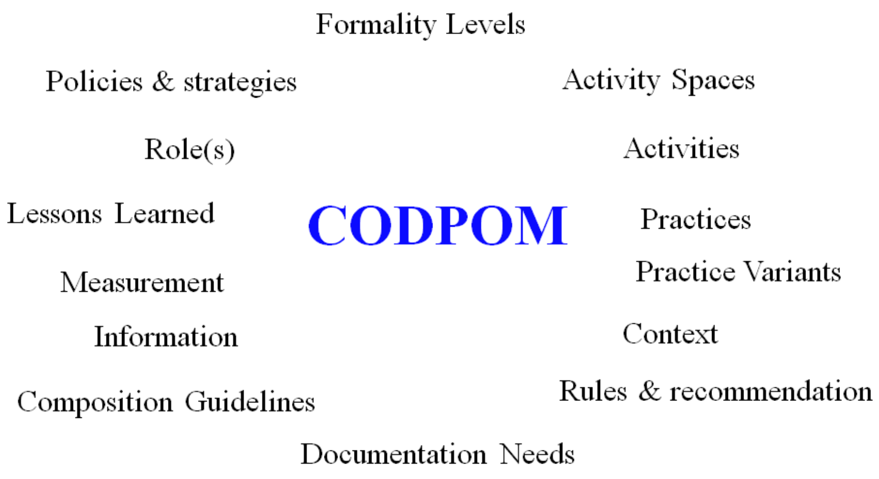


**Figure 1. Aspects considered by CoDPOM**

Today, companies have many ongoing projects. They run these projects with many processes. These projects are supposed to follow the standard organizational process models [7]. However, due to high discrepancy between the project needs and the standard process model, many of the projects have created their own process variants [2], [3]. This may be an optimal way of maximizing project productivity at hand as long as the discrepancy is managed in a controlled way. However, the variant's design should provide feedback for analyzing the organizational process standards and for creating process variants to be reused in the future.

The challenge does not stop on the fact that projects define their own process variants. The danger lies in the fact that projects may continue creating variants of variants and so on. In the end, organizations arrive at having many variants and no one has control over them. This in turn creates a challenge for high-level management, the challenge of how to analyze various process properties in a uniform manner from a strongly diversified bulk of process variants. Right now, management encounters the challenge of relating the variants to the standard processes, and the challenge of extracting knowledge and experience from them in order to effectively reuse them in future projects.

The existence of many process variants within organizations also creates a challenge for projects. They struggle with choosing the most suitable process variants and in mining and reusing knowledge and experience from them. All this hampers organizations from improving their processes and makes them continuously reinvent the wheel [10].

Imposing organizational process standards has also a negative influence on individual managers and software engineers. Forcing them to follow standardized and homogeneous process models may sometimes have an adverse impact on their creativity and productivity. Many times, attempts to make processes compliant with process models get in the way and slow down their production pace. They also strongly impact individuals' motivation for conducting their chores in their individual professional endeavors. [1]

## III. CoDPOM

To define optimal processes is a complex task [6, 10]. Processes should guide people in what to do and how. They should indicate what information should be managed and whether and how it should be documented and measured in specific contexts. They should promote effective communication and coordination, and most importantly, they should be convincing enough for its roles involved. We feel that CoDPOM is the right solution here. It fulfills all those requirements.

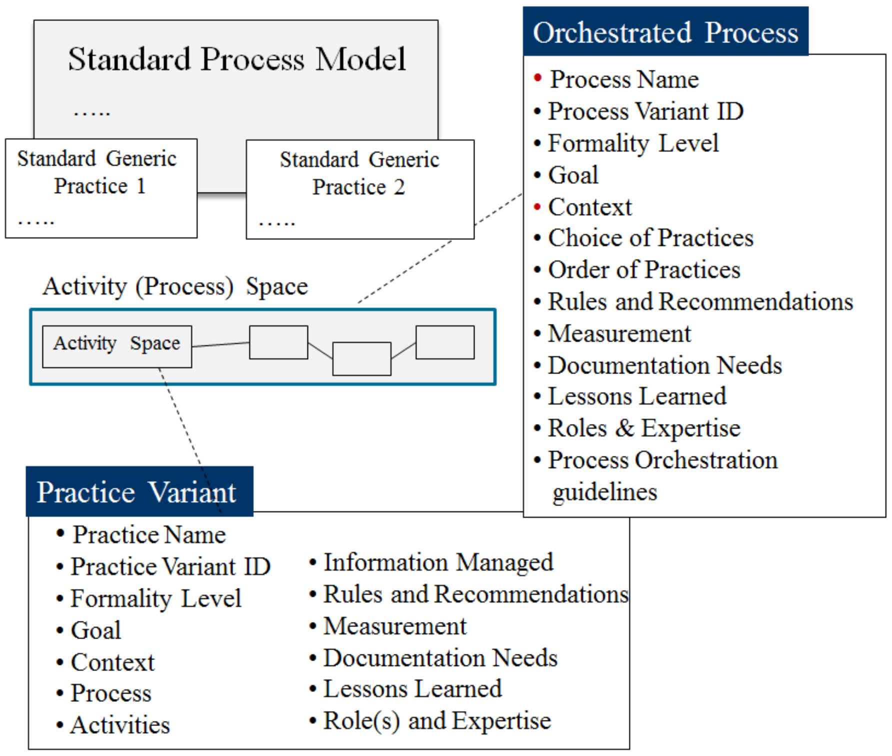


**Figure 2. The overall structure of CoDPOM**

CoDPOM, standing for ***Co**ntext **D**ependent **P**rocess **O**rchestration **M**ethod*, pays heed to the diversity of software processes. As illustrated in Fig. 1, it considers different properties of software development that have impact on the process design. Fig. 2 illustrates how they are organized. CoDPOM consists of the following building blocks (1) standard process model, (2) standard generic practice and practice variant, (2) activity spaces, and (4) orchestrated process. In this section, we describe these components and the properties to be regarded when orchestrating software processes.

### 3.1. Organizational Process Model

CoDPOM suggests that all organizations should define their own standard organizational process models and continuously maintain them. These standard models should provide a basis for tailoring processes to specific contexts. As indicated in Fig. 2, a standard process model includes a set of generic practices that may be reused in various contexts.

### 3.2. Activity Spaces

Activity spaces correspond to practice containers or place holders to include a set of activities that are appropriate for a particular practice variant to be then included in a specific orchestrated process. Activity spaces are method and practice neutral. There may be various relationships between the activity spaces alias practices implying the inclusion of practices within other practices or a sequence or an iteration between them. This is illustrated in Fig. 3. As can be seen there, there is a sequence between *Practice Variants 1- 4,* an iteration between *Practice Variants 2* and *3* and a composition of *Practice Variant 4* consisting of *Practice Variants 4.1* and *4.2*. These practice variants come from the standard generic activities inherent in the organizational standard process model whose contents has been adapted to the specific context of a particular software engineering endeavor.

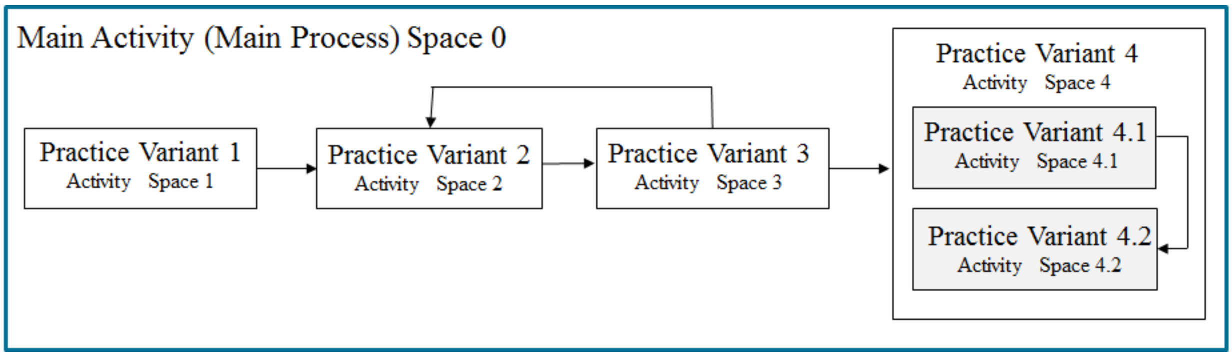


**Figure 3. Illustrating relationship between activity spaces and practices**

## 3.3. Practice and Practice Variants

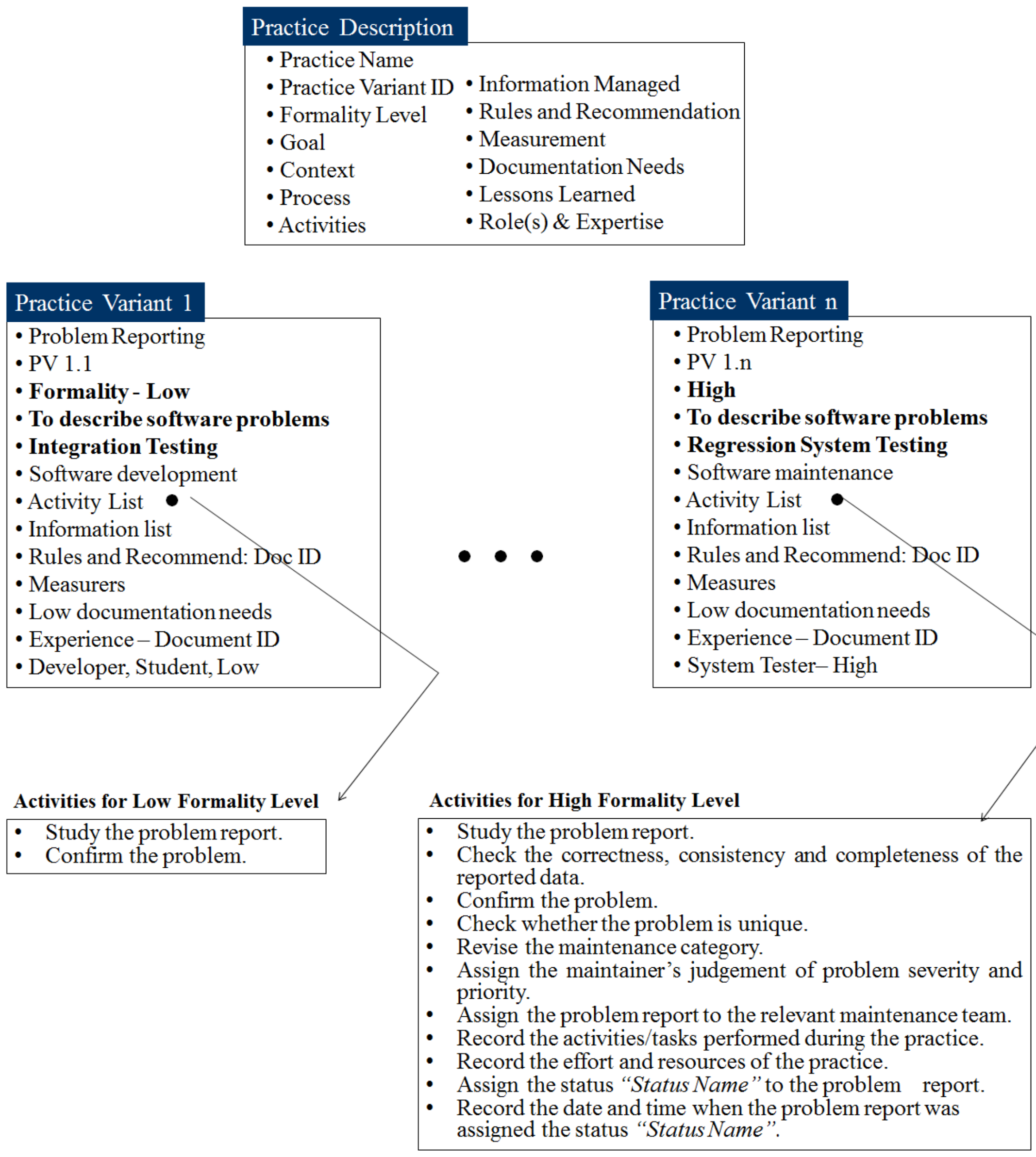
Specific process variants are appropriate only under certain conditions. They are a mix of a set of practices where a practice is a process in itself. It is difficult to define a difference between a process and practice. Both of them refer to a way of working that has been developed through knowledge and experience gained when developing and maintaining software systems. In this paper, we define practices as core elements in creating bigger practices, which we herein call processes. For instance, a mix of practices for managing requirements, design, implementation and testing may constitute a bigger practice, software development practice or software development process if our reader wishes to call it so. Or, an implementation practice is a mix of pair programming and test-first development practices.

As illustrated on the bottom part of Fig. 2, each practice variant is chosen according to thirteen properties. The contents of the majority of them are dependent on the specific context, goal and formality level of the process to be orchestrated. Below, we briefly describe them:

1. *Practice Name*: The name of the practice that is left intact for both the standard generic practice as well as its variants.
2. *Practice Variant ID*: The identifier of a specific practice variant. It is not enough to have a practice name. One practice may have many practice variants and they all need to be distinguished.
3. *Formality level*: The level of the formality of the practice. Here, CoDPOM does not impose any levels. Organizations are free to choose their own levels. In the least, CoDPOM suggests low, medium and high formality levels. These formality levels are determinants on the choice of the contents of the properties such as choice of activities to be included in the practice, subset of information to be managed when performing the practice, choice of roles to conduct the practice, choice of process and product aspects to be measured and determination of documentation needs.
4. *Goal*: The goal with the practice variant. For instance, the goal might be to unit test code.
5. *Context*: A set of circumstances that apply to the practice. Here, CoDPOM again leaves the floor open for choosing the parameters defining the practice context. An example of a context may be that a practice is

conducted in pre-determined process phases. For instance, the inspection practice is conducted after requirements gathering, design and implementation.

6. *Process*: The process, or bigger practice, in which this practice is included.
7. *Activities*: Activities to be part of the practice variant. For instance, a low formality level practice variant dealing with unit testing may include the following activities: (1) *choose code to be tested*, (2) *test code in an ad hoc manner*. A medium



Practice Description

- Practice Name
- Practice Variant ID
- Formality Level
- Goal
- Context
- Process
- Activities
- Information Managed
- Rules and Recommendation
- Measurement
- Documentation Needs
- Lessons Learned
- Role(s) & Expertise

Practice Variant 1

- Problem Reporting
- PV 1.1
- **Formality - Low**
- **To describe software problems**
- **Integration Testing**
- Software development
- Activity List ●
- Information list
- Rules and Recommend: Doc ID
- Measurers
- Low documentation needs
- Experience – Document ID
- Developer, Student, Low

● ● ●

Practice Variant n

- Problem Reporting
- PV 1.n
- **High**
- **To describe software problems**
- **Regression System Testing**
- Software maintenance
- Activity List ●
- Information list
- Rules and Recommend: Doc ID
- Measures
- Low documentation needs
- Experience – Document ID
- System Tester– High

**Activities for Low Formality Level**

- Study the problem report.
- Confirm the problem.

**Activities for High Formality Level**

- Study the problem report.
- Check the correctness, consistency and completeness of the reported data.
- Confirm the problem.
- Check whether the problem is unique.
- Revise the maintenance category.
- Assign the maintainer's judgement of problem severity and priority.
- Assign the problem report to the relevant maintenance team.
- Record the activities/tasks performed during the practice.
- Record the effort and resources of the practice.
- Assign the status *"Status Name"* to the problem report.
- Record the date and time when the problem report was assigned the status *"Status Name"*.

**Figure 4. Exemplifying generic practice and its variants. Activities are retrieved from *CM*$^3$*: Problem Management* [9]**

formality level practice might include the following activities: (1) *choose software component to be tested*, (2) *write test cases for the software component*, (3) *test code against the use cases*, (4) *document test results*.

8. *Information managed*: A set of information to be managed while performing the practice. For instance, within medium formality level unit testing practice, the information to be managed might be (1) Software Component ID, (2) Software Test ID, (3) Input, (4) Output, and (5) Testing Result.

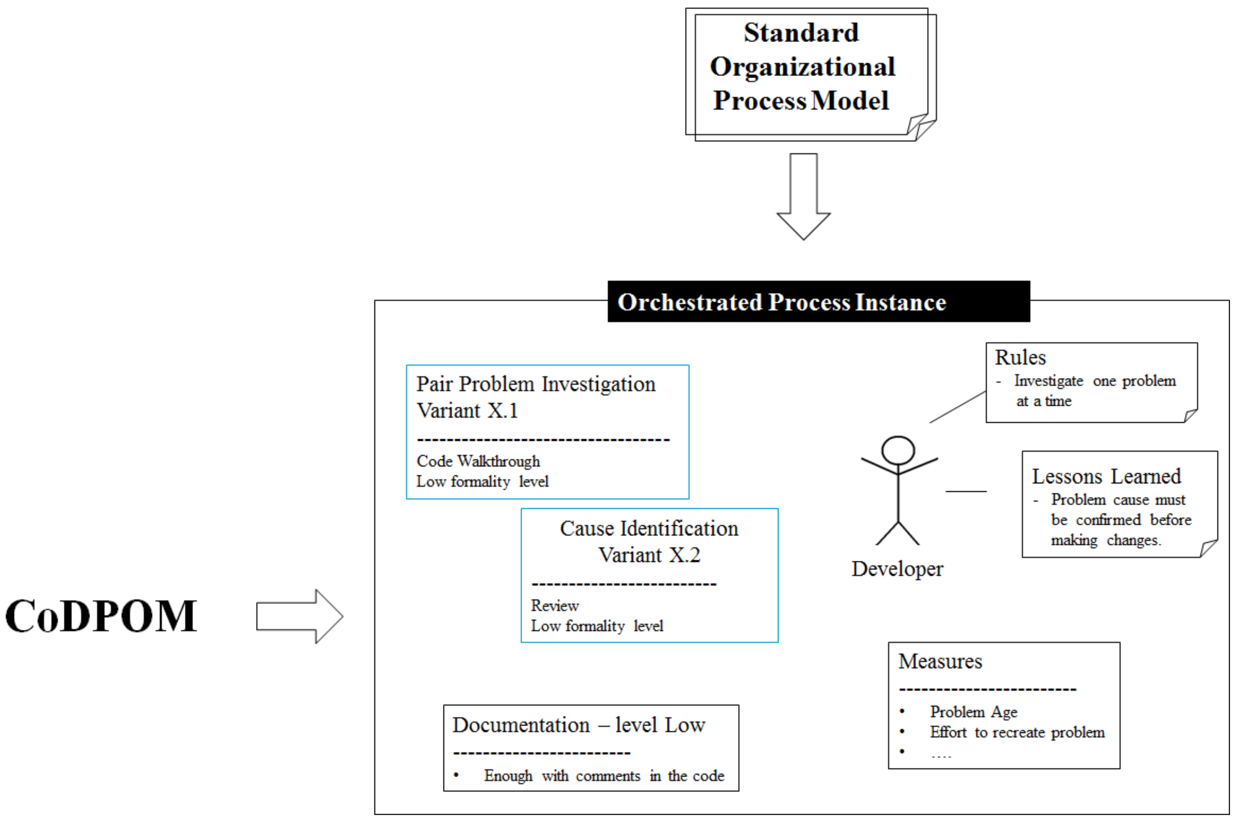

Standard Organizational Process Model
Orchestrated Process Instance
CoDPOM
Pair Problem Investigation Variant X.1
Code Walkthrough
Low formality level
Cause Identification Variant X.2
Review
Low formality level
Documentation – level Low
Enough with comments in the code
Rules
- Investigate one problem at a time
Developer
Lessons Learned
- Problem cause must be confirmed before making changes.
Measures
Problem Age
Effort to recreate problem
....

**Figure 5. A tentative illustration of a process instance**

9. *Rules and recommendations*: Policies and strategies for creating the practice. For instance, they may include advise to developers how to identify and use process patterns that work or a rule that each software component is to be separately tested first before it is tested with other components.
10. *Measurement*: A set of measures to be used within the practices. In the least, a practice may include no measurement at all. For instance, in one context a developer might perform unit testing, however, he may not measure anything. In his specific context and formality level, the organization may not require this from him.
11. *Documentation needs*: A specification of what needs to be documented. This may apply to documentation of what activities have been performed, what measurement has been done, what rules and recommendations have been applied, and the like. In the least, no documentation may be needed.
12. *Lessons Learned*: Experience reporting on knowledge, feedback or skills gained while being involved in or exposed to the practice. The practice user describes whether or not and why the practice worked, and generously shares the lessons learned from his/her experience as engineer, manager, consultant, or some other role.
13. *Role(s) and Expertise*: Expertise and roles required for performing the practice. In some cases, a practice may be conducted by anybody, such as for instance, a simple unit testing may be delegated to a software engineering student. His role would be to run software units and compare expected and achieved testing outputs. In other

cases, testing has to be conducted by an expert who while testing must understand the testing context and output and use it for specifying the next tests.

## 3.4. Process Orchestration

The main process instance, the one that has been orchestrated, is also put into an activity space. It is realized by combining (orchestrating) existing software practices that are placed in their own activity spaces [5]. It is created on an as-needed and context-driven basis. Depending on the context at hand and formality requirements, the orchestration may result in heavyweight, mediumweight or lightweight process instances or a mixture of those.

To orchestrate processes is not simple. To make them as optimal as possible, we suggest that the input to the process of orchestrating a specific process instance be (1) standard organizational process models that are local standards within each organization and (2) CoDPOM suggested herein. The local organizational standards should include a set of generic practices that may provide a basis for creating practice variants to be then included in the orchestrated process model.

When orchestrating processes, organizations should feel free to decide when to bind their practices. Both early and late bindings should be allowed. Late bindings are recommended in contexts dealing with many uncertainties and unknowns. Early bindings, on the other hand, may be conducted in obvious process cases. However, they should be easily unbound and rebound in cases when the process context changes. The reasons may be many. For instance, customer may have losen the quality requirements, and therefore, the process required for developing the system may be less formal. Being such a case, some more formal practice variants will have to be unbound and replaced by the less formal ones.

To orchestrate processes is not easy bearing in mind the fact that processes may comprise a great number of practices, they may need to follow specific organizational strategies and policies, they may have to consider the context and changes within it and they may have to adhere to specific formality levels. In order to obtain processes that are suitable for specific contexts and needs, the organizations should compose processes by extracting and assembling practices.

To maximize process results and to minimize process missteps, waste and failures, organizations need information supporting them in their process orchestration work. Such information is briefly presented on the right hand side of Fig. 2. It includes different properties whose contents, just as with practices, is driven by *Context* and *Formality Level*. The properties are the following:

1. *Process Name*: The name of a process, for instance, software development.
2. *Process Variant ID*: The identification of the composed software process variant. The identifier is very important in order to retrieve the instance of the process variant in cases one needs to analyze it, draw lessons learned and make suggestions for process improvement.
14. *Formality level*: The level of the formality of the process. Just as with the formality levels of practices, CoDPOM does not impose any levels. Organizations are free to choose their own levels. In the least, CoDPOM suggests low, medium and high formality levels.

    *Goal*: The goal with the process variant. For instance, the goal might be to develop a university administration application to be used by Swedish universities.

3. *Context*: A set of circumstances that apply to the process. Just as with the practice context, CoDPOM leaves the floor open for choosing the parameters defining the practice context. An example of a process context may be that the process is distributed across several nodes in the world such as India, Sweden, USA and Brazil.
4. *Choice of practices*: List of practices and their variants that have been chosen for orchestrating the process at hand.
5. *Order of practices*: A map of the order of activities that are included in the process.
6. *Rules and recommendations*: Policies, strategies and advice for orchestrating processes. They must adhere to the rules and recommendations of each individual practice that is part of the process.
7. *Measurement*: A set of measures to be used within the process. Here, the measures may include both measures for each practice involved and combined measures applying across the whole process.
8. *Documentation needs*: Specification of what needs to be documented. Just as with measures, documentation may apply to documentation of each individual inherent practice as well as documentation of the whole process.
9. *Lessons Learned*: Experience gained from using the orchestrated process instance.
10. *Roles & Expertise*: Expertise and roles required for performing the process. In contrast to roles and expertise required for performing the practice, roles and expertise for the process may strongly vary. Many roles may be involved and the expertise needs may be on different levels.
11. *Process Orchestration Guidelines*: Guidelines for how to combine practices.

## IV. EPILOGUE

It is difficult to find a universal way for choosing the right processes for the right contexts and formality levels. In this paper, we have suggested *Context-Driven Process Orchestration Method (CoDPOM).* CoDPOM addresses both managerial and engineering aspects and it is applicable within all types of organizations that are involved in software development, software evolution and maintenance, support processes, and various management processes [7]. CoDPOM considers many properties while designing software processes and adapting them to specific software engineering endeavors.

Right now, CoDPOM has only been suggested as a research idea and it has not been implemented yet. However, we regard CoDPOM as a realistic solution for relaxing the rigidness of current software engineering processes. Hence, we strongly suggest to the software community to use it as a guideline for defining future research and industry projects, and hopefully, for arriving at an optimal process orchestration method.

## REFERENCES

[1] S.W. Baker, Formalizing Agility, Part 2: How an Agile Organization Embraced the CMMI, Proc. AGILE Conference, 2006, IEEE Computer Society, 147-154.

[2] M. Beck, Managing Process Diversity while Improving Your Practices, IEEE Software, *IEEE Computer Society, Vol. 18, Issue 3, 2001, 21-27.*

[3] T. Bollinger, C. McGowan, A Critical Look at Software Capability Evaluations: An Update, IEEE Software, *IEEE Computer Society*, Vol. 26, Issue 5, 2009, 80-83.

[4] Carnegie Melon and SEI, Capability Maturity Model Integration (CMMI), http://www.sei.cmu.edu/ /tools/index.cfm, retrieved on January 6, 2017.

[5] A. Cockburn, Methodology Per Project, http://alistair.cockburn.us/Methodology+per+project, retrieved on January 8, 2016.

[6] S. Henninger, Software Process as a Means to Support Learning Software Organizations, Proc. Twenty-fifth Annual NASA Software Engineering Workshop, 1995.

[7] ISO/IEC 12207: 2008, Systems and Software Engineering - Software life cycle processes, 2008.

[8] M. Kajko-Mattsson, K. Sjökvist Söderström, DRiMaP - A Model of Distributed Risk Management Process, Proc. Fifth International Joint Conference on INC, IMS and IDC, ISBN: 978-1-4244-5209-5, IEEE, 2009.

[9] **M. Kajko-Mattsson, M,** Corrective Maintenance Maturity Model: Problem Management, PhD thesis, ISBN Nr 91-7265-311-6, ISSN 1101-8526, ISRN SU-KTH/DSV/R--01/15, Department of Computer and Systems Sciences (DSV), Stockholm University and Royal Institute of Technology, 2001.

[10] J. Nyfjord, M. Kajko-Mattsson, D. Wengelin, Exemplary System Development Framework Needed! Position Paper, http://www.semat.org/pub/Main/WorkshopPositions/SEMAT_position_SAAB.pdf, retrieved on January 6, 2016.